Spline-based manifolds for combustion and thermodynamic closure

A Spline fitting method with adaptive domain optimization and its application on combustion chemistry manifolds

Master thesis by Isaac Lucarelli Elias

Date of submission: March 21, 2025

1. Review: Prof. Dr.-Ing. Christian Hasse
2. Review: Prof. Dr.-Ing. Flavius Portella Ribas Martins
3. Review: Tim Jeremy Patrick Karpowski, M.Sc.

Darmstadt

TECHNISCHE UNIVERSITÄT DARMSTADT

Spline-based manifolds for combustion and thermodynamic closure
A Spline fitting method with adaptive domain optimization and its application on combustion chemistry manifolds

Master thesis by Isaac Lucarelli Elias

Date of submission: March 21, 2025

Darmstadt

To my mother and father, Helda and Jorge, which provided me with love and care even in the presence of this great distance. To my fiancée, Luiza, who makes me overflow with the passion that drives me through life.

Aufgabenstellung für eine Masterarbeit (Double Degree with São Paulo)

TECHNISCHE UNIVERSITÄT DARMSTADT

Fachgebietsleiter:	Prof. Dr.-Ing. Christian Hasse
Betreuer San-Paulo:	Prof. Dr.-Ing. Flavius Portella Ribas Martins
Studierender:	Isaac Lucarelli Elias
Mat-Nr.:	2930518
Betreuende:	Jeremy Karpowski, M.Sc

Titel der Arbeit:
Spline-based manifolds for combustion and thermodynamic closure
Spline-basierte Mannigfaltigkeit für Verbrennungs- und Thermodynamik-Schließung

Simulation reaktiver Thermo-Fluid-Systeme

Simulation of reactive Thermo-Fluid Systems

Otto-Berndt-Straße 2
64287 Darmstadt

T. Jeremy P. Karpowski, M.Sc.
L1|01 286
karpowski@stfs.tu-darmstadt.de

17. Dezember 2024

To reduce greenhouse emissions, combustion systems are transitioning to new carbon-free or carbon-neutral fuels. These exhibit additional complexity due to differential diffusion. At the same time, to increase the predictability of numerical simulations, additional physics, such as heat losses and their effect on pollutant formation, are increasingly included in simulations. Simulations of technical systems often require simplifications to reduce the computational cost of reactive simulations while retaining most of these physics.

One such tool is the utilization of tabulated manifolds. However, these have a large memory footprint, which limits their applicability and hinders the inclusion of additional physics into these models.

Here, spline-based methods promise to reduce memory footprints and enable manifold-based methods for more complex simulations. Their lower memory but higher computational requirements suggest that they could also profit from accelerator technology, thereby allowing the usage of manifolds on novel solver architectures.

Thus, in this thesis, the utilization of spline-based methods for chemistry and thermodynamic closure in CFD applications is to be evaluated. After a thorough literature review, adequate spline-fitting algorithms shall be implemented. The spline-based manifolds are to be compared to tabulated manifolds in an a-priori and a-posteriori manner for test cases of increasing complexity, starting from 1D flames, later including heat-losses and potentially differential diffusion. To this end, also adaptive fitting methods (h-, r-, k-refinement) and their inclusion in the fitting procedure shall be assessed. Furthermore, the potential of GPU acceleration shall be evaluated. Afterwards, novel tabulation strategies can be explored.
All results are to be presented, discussed, and documented in a suitable form.
Timeframe for the thesis preparation: 24 weeks.

Betreuer | Betreuer São Paulo | Fachgebietsleiter

Erklärung zur Abschlussarbeit gemäß § 22 Abs. 7 APB TU Darmstadt

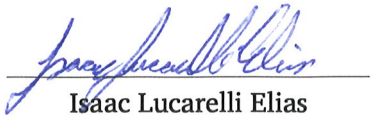
Hiermit erkläre ich, Isaac Lucarelli Elias, dass ich die vorliegende Arbeit gemäß § 22 Abs. 7 APB der TU Darmstadt selbstständig, ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt habe. Ich habe mit Ausnahme der zitierten Literatur und anderer in der Arbeit genannter Quellen keine fremden Hilfsmittel benutzt. Die von mir bei der Anfertigung dieser wissenschaftlichen Arbeit wörtlich oder inhaltlich benutzte Literatur und alle anderen Quellen habe ich im Text deutlich gekennzeichnet und gesondert aufgeführt. Dies gilt auch für Quellen oder Hilfsmittel aus dem Internet.

Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Mir ist bekannt, dass im Falle eines Plagiats (§ 38 Abs. 2 APB) ein Täuschungsversuch vorliegt, der dazu führt, dass die Arbeit mit 5,0 bewertet und damit ein Prüfungsversuch verbraucht wird. Abschlussarbeiten dürfen nur einmal wiederholt werden.

Bei einer Thesis des Fachbereichs Architektur entspricht die eingereichte elektronische Fassung dem vorgestellten Modell und den vorgelegten Plänen.

Darmstadt, March 21, 2025



Isaac Lucarelli Elias

Abstract

The aim of the present work is to assess the suitability of using Splines as a higher-order tabulation method for chemistry manifolds. In the context of Computational Fluid Dynamics, tabulated manifolds are a important tool to accelerate simulations. They can be generated with the simulation of a one-dimensional flamelet, which provides thermochemical data that is stored and later reused as tabulated data for higher-dimension simulations. In that direction, the study and implementation of an algorithm to calculate and fit Splines using techniques associated with machine learning was developed. Normalization and batching methods are discussed for a flamelet dataset in order to improve training stability. Additionally h- and r-refinement are also implemented in an effort to capture the shape of the manifold using less allocated memory, and afterwards have their results evaluated a priori *against this flame. For each of the refinement methods, different criteria for enhancing the domain discretization are chosen and compared quantitatively. The results show that higher-order tabulation methods may be able to increase accuracy without further memory allocations as the degree of the Spline grows, presenting a trade-off between memory allocation and processing time to retrieve data which is discussed. As the importance of reactive flow simulation become bigger when evaluating new fuels to power a green economy, Splines present themselves as a possible solution to accelerate the computations.*

Das Ziel der vorliegenden Arbeit ist, die Eignung von Splines als Tabellierungsmethode höherer Ordnung für chemische Mannigfaltigkeiten zu bewerten. Im Rahmen der numerischen Strömungsmechanik sind tabellierte Manifolds ein wichtiges Instrument zur Beschleunigung von Simulationen. Sie können aus eindimensionalen Flamelet Simulationen erzeugt werden, das thermochemische Daten liefert, die gespeichert und später als tabellierte Daten für höherdimensionale Simulationen wiederverwendet werden. In diesem Zusammenhang wurde ein Algorithmus zur Berechnung und Anpassung von Splines mit Hilfe von Techniken des maschinellen Lernens entwickelt und implementiert. Es werden Normalisierungs- und Batching-Methoden auf Basis eines Flamlet-Datensatzes diskutiert, um die Trainingsstabilität zu verbessern. Zusätzlich werden h- und r-Verfeinerung implementiert, um die Form der Mannigfaltigkeit mit weniger zugewiesenem Speicher zu erfassen, und anschließend werden die Ergebnisse anhand dieser Flamme bewertet. Für jede der Verfeinerungsmethoden werden verschiedene Kriterien zur Verbesserung der Bereichsdiskretisierung gewählt und quantitativ verglichen. Die Ergebnisse zeigen, dass Tabellierungsmethoden höherer Ordnung in der Lage sind, die Genauigkeit ohne weitere Speicherzuweisung zu erhöhen, wenn der Grad des Splines wächst, was einen Kompromiss zwischen Speicherzuweisung und Verarbeitungszeit zum Abrufen von Daten darstellt. Da die Bedeutung der reaktiven Strömungssimulation bei der Bewertung neuer Brennstoffe für eine umweltfreundliche Wirtschaft immer größer wird, bieten sich Splines als mögliche Lösung zur Beschleunigung der Berechnungen an.

Table of contents

List of figures

List of tables

Notation

0.1. Abbreviations

- ADAM: Adaptive Moment Estimation
- ANN: artificial neural network
- Auto-diff: automatic differentiation
- CFD: Computational Fluid Dynamics
- CPU: central processing unit
- FGM: Flamelet-Generated Manifolds
- FLUT: Flamelet Lookup Table
- GHG: greenhouse gases
- GPU: graphics processing unit
- HOQ: head-on quenching
- ICE: internal combustion engine
- LHV: lower heating value
- MIMO: multiple-input multiple-output
- NMAE: normalized maximum absolute error
- NURBS: non-uniform rational B-Spline
- PCA: principal component analysis
- QGM: Quenching Flamelet-Generated Manifolds
- RE: relative error
- REDIM: Reaction-Diffusion Manifolds
- SIMD: single instruction, multiple data
- SIMO: single-input multiple-output
- SISO: single-input single-output

- TCI: turbulence-Chemistry Interaction

0.2. Notation

- A: Helmholz free energy
- $B_{i,K}$: Spline basis
- C_i: set that denote the region of the domain governed by a weight w_i
- I: number of Spline weights
- $ID_{r,m}$: set of indexes for data points inside region r and output dimension m
- K: Spline degree
- M: number of Spline outputs
- N: number of Spline inputs, number of particles
- N_s: number of chemical species
- P: pressure
- P_i: set of data points x_d contained within a region C_i
- R_i: Region enclosed between two nodes
- S: entropy
- S_K: Spline
- T: temperature
- U: internal energy of a system
- α_h: learning rate for h-refinement
- α_r: relaxation parameter for r-refinement
- α_{opt}: learning rate for Adam method
- β_1 and β_2: hyper-parameter for Adam method
- $\dot{\omega}'_T$: temperature source term
- $\dot{\omega}_{Y_{YC}}$: progress variable source term
- ϵ: hyper-parameter for Adam method
- λ: specific heat capacity
- μ: dynamic viscosity
- ϕ: Spline parameters

- ρ: density

1. Introduction

The importance of mitigating emissions is a matter of survival. The world is getting warmer, the seas are rising and extreme climate events are becoming more common as this is written. This conjuncture of effects caused by the unbound release of CO_2 in the atmosphere already present consequences in agricultural productivity [1], and studies show that the human hand is indeed the main cause driving the number of extreme weather events up [2]. As of 2020, 73% of the emissions come from the energy sector, followed by agriculture, forestry and land use sector with 18.4% [3]. This indicates that the energy sector would be the most promising one for policies that aim to reduce Greenhouse Gas Emissions. In Europe, the share of petroleum and products, solid fuels and gas corresponds to 81.3% of the energy production [4], thus finding new fuels to power the European economy would be of great interest to diminish the amount of GHG emissions.

In this context, Computational Fluid Dynamics is a powerful method to assess the behavior of fuels when reacting, being a important design tool in the effort of developing efficient burners that present acceptable level of emissions of different pollutants, i.e. nitrogen oxides NO_x, carbon monoxide CO and fine particulate material, and GHG, as carbon dioxide CO_2 and unburnt methane CH_4 [5]. It's based on the numerical solution of the Navier-Stokes equations to predict the flow behavior, alongside balance equations to account for the chemical taking place.

However, the costs for simulating the complete set of reactions can be prohibitive when dealing with three-dimensional simulations, due to the high computational cost involved in running this numerical simulations. To solve for a reactive flow, one must solve $N_s + 5$ transport equations, with N_s here being the number of chemical species involved in the reaction. The GRI 3.0 mechanism [6] used to generate the manifolds used here used includes 53 different species and computes 325 reactions, leaving us with a already large system of partial differential equations to be solved numerically, whereas more complex mechanisms may include hundreds to thousands of different chemical species. As a matter of fact, there are many works in the direction of diminishing this computational costs, and to address this problem tabulated manifold methods have been introduced [7]. These tabulations consist of representing reaction rates and the species mass fractions as function of a set of fewer suitable control variables ψ, such as a progress variable and the mixture fraction, greatly reducing the number of transport equations to be solved for. When using tabulated manifold approaches, only the transport equations of the control variables need to be considered to account for the chemical reaction, vastly reducing the number of equations to be integrated. Furthermore, when using these approaches the source terms are not calculated using the chemical mechanism, but the tabulated values serve as closure model for the chemistry.

Tabulation methods for thermodynamic closure have been studied for a while, at least from 2000s [8]. The present work looks at various sources that have discussed the theme to come up with

an approach to fit Splines against chemistry manifolds in the aim to serve as tabulated chemistry for acceleration in reactive flow simulations. Various areas of study were consulted in an effort to come up with a reasonable way of fitting the Splines based on the current knowledge from the field of machine learning, which mostly focus its efforts nowadays in the study of neural networks, with focus on fitting a very different structure from NNs, but with the same effort of bringing low memory consumption and cheapness of evaluation that they provide. Not only works in the realm of combustion simulation were assessed, but also in the fields of signal processing and representation and image processing, alongside with machine learning techniques that can have analogies for application on Splines.

In 1992, Maas and Pope published a work developing a simplified chemistry method using methods from dynamical systems, named Intrinsic Low-Dimensional Manifolds. Their method takes as input the detailed chemical kinetics and the number of degrees of freedom in the simplified scheme. Based in the fact that the mathematical model for reactive flows is described by a set of differential equations, it possible to calculate the local Jacobian of the chemical source terms, and identify which are the fastest chemical time scales. With that in hand, one can choose to only calculate the n_g source terms, assuming that the n_f fastest reactions happens negligibly fast and therefore not needed to be included in the set of partial differential equations that govern the process. From a system with n_s species, each one with its own source term, we are now left with a n_g-dimensional manifold, with $n_g = n_s - n_f$, to represent a system approximately equivalent to the otherwise n_s-dimensional manifold representing the detailed chemistry mechanism [9]. Nevertheless, this approach effectively acts as a low pass filter, throwing out of the scenario important transport phenomenon, that usually occur in the colder regions of the flame.

An advance in representing reduced chemistry using lower order manifolds was made when Flamelet-Generated Manifold method was developed by Oijen and de Goey in 2000 by combining the laminar flamelet model, which states that a turbulent flame front behaves as a laminar one-dimensional flame when observed locally. Aligning this model with manifold approaches, they were able to successfully capture the behavior of a 2D burner-stabilized laminar premixed flame. This method turned out superior than ILDM for representing colder regions of the flame, where diffusion processes may disturb the balance between chemical production and consumption. As a result, this approach requires less control variables in order to capture the combustion phenomenon with a suitable precision. For simple fuels, two controlling variables were found to be sufficient, but the number of control variables may be increased if the perturbations caused by the simplification cannot be neglected [8]. Whereas ILDM threw away the slowest reactions, FGM projects all of the reactions in the same lower-dimensional space, and captures this lower-dimensional representation in terms of a selected number of control variabes.

Various kinds of tabulation methods have been tried in the effort of better representing tabulated chemistry manifolds, such as Flatkernels [10], Neural Networks [[11], [12]] and also Splines [13]. As combustion is a very nuanced phenomena, that involves often three-dimensional turbulent fluid flows, a elevated number of source term equations if the detailed chemistry is included, and can be affected by stratification and heat losses, presenting important phenomena that happens in different scales, capturing all the desired interactions can be a challenge. Current research aims to extend manifold based methods to include such effects at a reduced computational cost compared to detailed chemistry.

When storing the tabulated results, however, the size of the manifold database grows exponentially with the number of control variables if a constant resolution of the table is kept. One way of addressing that issue is to store only the most ”meaningful” values, focusing the resolution of the grid in the regions that are most important to capture the phenomena. The values obtained from the simulation are stored inside a computational structure, that must balance the trade-off between the memory that it allocates, the delay for retrieving the stored values and precision. This leads us to the need of developing more efficient methods of tabulations, using different techniques to representing the data, i.e. representing it as a collection of polynomials or linear interpolations, or using a artificial neural network [[10], [12], [11]].

As to balance the characteristics of different approaches, starting with piecewise linear functions, which may be assembled to interpolate between the points of data generated by the one-dimensional simulation, to provide a continuous representation of this discrete data, without having to store all the points obtained from the simulation. The data points provided by the simulation may be very unevenly distributed, which can be seen in Figure 4.3, and there is no much sense to store a high resolution representation containing all this points, as the majority of them end up being far away from the flame inside the domain. This may be achieved by selecting points from the FGM and setting them as control points for the interpolation to pass through. Another possible way to achieve this linear interpolation is to choose this control points by minimizing some cost function, i.e. a mean squared error. Although, linear interpolations cannot capture curvature if there is not enough resolution in the table of values, and are of class C^0 with no continuous derivative everywhere in the domain.

Polynomials may be fitted to the data, in order to account for the higher order derivatives of the function, and hopefully better represent the data using less computer memory, in exchange for an overhead in calculation. This is always a possibility, as guaranteed by the Stone-Weierstrass theorem [14], that gives us the result that any continuous function may be uniformly approximated by a polynomial with a arbitrary precision. However, high order polynomials suffer from the Runge’s phenomenon when constrained to pass through more and more fixed positions the degree grows [15], presenting in some cases increasing oscillations of the function value with higher oscillation on its boundaries.

One such approach to avoid this oscillations is to fit lesser degree polynomials locally, and switch between different polynomials definitions depending on the region of the input variables ψ. This approach was developed by Popp in 2017 [10], where an algorithm to generate Flatkernels was created, making it possible to have different polynomials defined in arbitrary regions of the domain, as seen in Figure 1.1.

Neural networks can also be used as a tabulation method for manifolds, as these structures fall under the universal approximation theorem [16], which states that there exists given a family of neural networks with parameters ϕ_n for which there exists a sequence of parameters $\phi_1, \phi_2, \ldots$ that approximate a function f if there are enough neurons inside the network. To find such sequences, it has been shown in practical cases that this sequence may be approximated using gradient descend, despite the theorem being about the existence of such sequences, and not offering a guarantee that indeed the sequence will be achieved with such method.

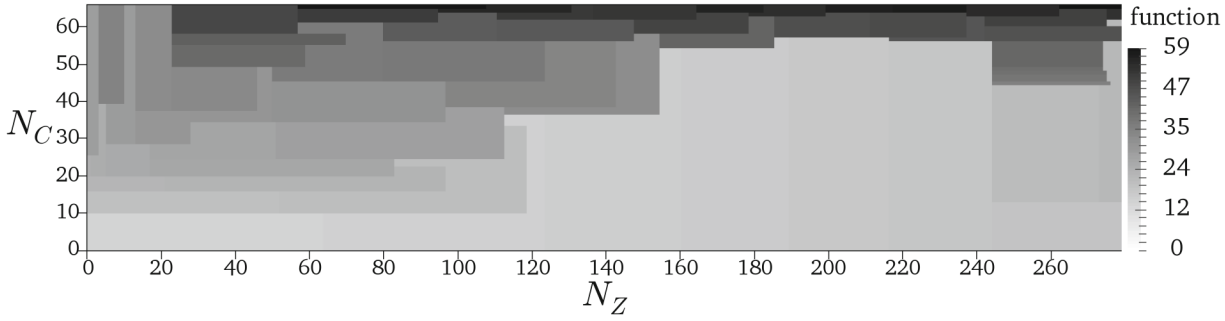


Fig. 1.1.: Domain discretization using Flatkernels. N_C stands for the point count for the progress variable, and N_Z for the mixture fraction. (From Popp 2017[10])

In recent work from 2022, Zhou [12] classifies the use of machine learning techniques in the reactive flow simulation scenario into: 1) chemical kinetics calculation acceleration, 2) combustion kinetic model uncertainty quantification, 3) discovery of unknown reaction pathways, 4) sub-grid combustion modeling and 5) building surrogate solvers for simulation. The present work aims to tackle the first class of problems proposed in the mentioned work, using Splines instead of ANNs. The acceleration strategy of Zhou's work consists in tabulating the results of a flamelet model using a ANN to serve as lookup table, which is then read from during subsequent simulations. That is aligned with using scalars to index into the lookup table, such as the mixture fraction or a progress variable, which were used as inputs to train a NN to output thermochemical quantities, such as temperature, pressure, species fractions, and their source term. Neural Networks have the benefit of being able to represent functions in higher-dimensions with lower memory consumption because the number of weights changes very slightly with the number of inputs, and also they take advantage when being ran in specific hardware [11].

In 2023, Bissantz and Karpowski [11] developed a more specific work, assessing the use of dense neural networks to tabulate chemistry manifolds which are able to capture flame-wall interactions, namely here 1D head-on quenching (HOQ) and 2D side-wall quenching (SWQ). The scheme for the neural network used in this tabulation approach can be seen in Figure 1.2. The variables chosen as inputs for the NN were chosen in an effort to capture the behavior of both a freely propagating flame (captured by the progress variable PV) and the effects of quenching that happens near walls due to the presence of heat losses (captured by the temperature T). This work shows the applicability of data-driven approaches to achieve this tabulation.

Here, the aim is to use Splines and assess the feasibility of using this kind of curves, which are going to be defined section 2.1. The name *Spline* was originally coined by Isaac Schoenberg in his 1946 publication [17]. In this work, it's addressed how Splines are a suitable way of using piece-wise high-order degree polynomials to interpolate functions while avoiding Runge's phenomenon, which is illustrated in Figure 1.3. Splines suffer less from this effect, since the weights of the Spline are *not* interpolating. There is no guarantee, and it's not true in most majority of cases, that the Spline will assume the weights assigned for each region.

The data used in the present work is the same used by Bissantz and Karpowski [11] to train the ANN, and comes from a detailed chemistry simulation using the GRI 3.0 mechanism [6] performed

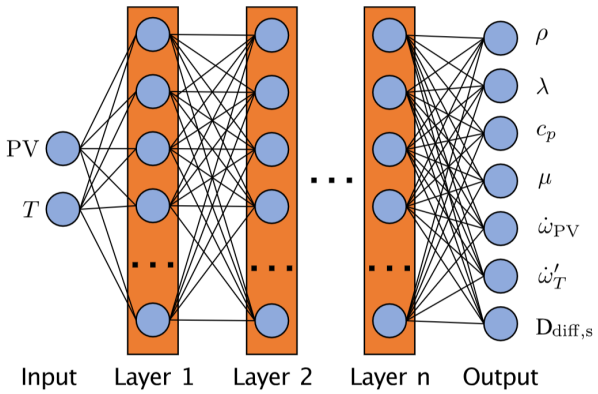


Fig. 1.2.: Schema of a ANN for use as tabulated chemistry for FGM and QGM. (From Bissantz 2023 [11])

by Steinhausen [18] and published in 2020 of a methane-air flame undergoing side-wall quenching. This manifold is going to be described here in function of the progress variable PV, which was defined as the principal component that explain the variance in the output data in terms of a linear combination of species mass fractions.

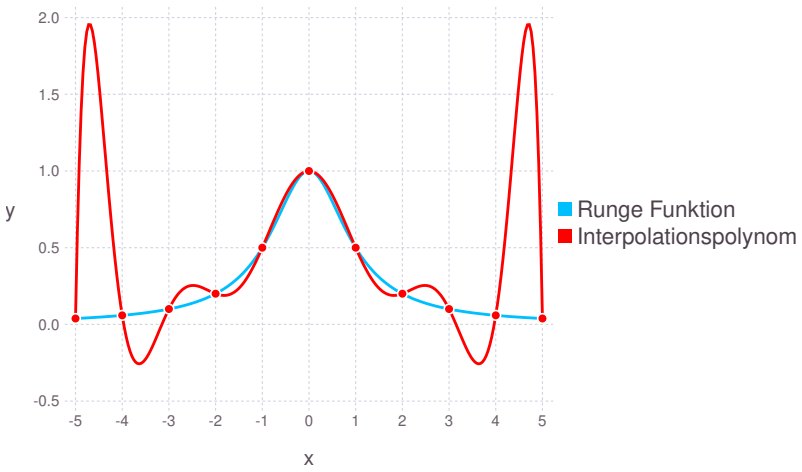


Fig. 1.3.: Illustration of the Runge's phenomenon. (Cuvwb, CC BY-SA 4.0, via Wikimedia)

De Boor [19] developed what is now the standard way of evaluating Splines. The method developed in this work allows one to calculate Splines using only the shape functions that indeed are responsible for the evaluated region. This allows reducing the number of shape functions to be evaluated to a minimum, vastly eliminating the evaluations when evaluating Splines. It's worth noticing that the notation used in the aforementioned work differs from the one used here. Also achieved by De Boor, a method to completely eliminate the need to explicitly evaluate the basis functions was developed and distributed as a FORTRAN package, that contained the sub-routines to evaluate

Splines [20].

Briand [21] developed a theoretical framework in 2018 in order to solve for the weights to be used in order to properly capture the main features of an image. For that, he defines a transformation that brings the values from an arbitrary finite signal to what is defined as the B-Spline space. After accomplishing this did, the work proceeds to show in practice the suitability of B-Splines for representing 2-d signals. One of the conclusions of this work is that Splines may be able to carry more information if the number of weights is kept the same, and the degree of the Spline is increased. This is the ground that allows us to seek reduced memory consumption when tabulating chemistry with the hope that higher-order representations may be able to capture the shape of the manifold using less weights than linear interpolations would.

In 2005, Gu [22] published a work defining Splines over a planar domain with an arbitrary topology, for example a domain discretized as triangular regions, and then parametrize it to be able to represent a manifold space, where the surface may intersect itself. This work shows that there is more to be explored when dealing with Splines than the rectangular divided domain that the present work will use. Surfaces that are not directly function of a given variable may be parameterized in a way that Splines are able to represent its eventual self-intersections using the developed formulation.

From the collection of stochastic optimization methods that were developed in recent years, the *Adam* method was chosen as the go to algorithm for finding inputs that minimize a given function. Just as normal gradient descend, it's based of taking steps with the parameters in the opposite direction that the gradient of the loss function points to. Further than that, the Adam method expands on the idea of gradient descend by incorporating momentum into the equations [23] as if the parameters where effectively a mass rolling down an N-dimensional field of mountains, and a local point of minimum may not retain the parameters in its valley, but these parameters may scape the local minimum by climbing up the hills described by the surface that they live in, with their now assigned mass and momentum, such as a mass would behave when traversing hilly surfaces when subjected to a gravitational field.

Additionally, to be able to perform gradient descend to search for solutions, we need to be able to evaluate the gradient of the given function. To solve that, here a method called *automatic differentiation* will be used, as it can provide fast and precise evaluations of a large class of computational routines [24]. This method works by differentiating the instructions followed by the computer, which can all be described as a composition of basic mathematical functions, by accumulating the derivative of the function by applying the chain rule for every operations performed using the intermediate values stored in the memory. This approach is not the same as numerical differentiation or differentiation using computer algebra systems, as will be seen in section 2.2. With this approach, it's possible to achieve theoretically "exact" results withing the floating point precision.

As a digression, there is a Hermetic text which circulated among the later alchemists. Three Latin versions versions of the scripts of the mythological figure Hermes Trismegistus, prince, poet and philosopher, that are known to be part of the arabic alchemical set of texts *Hermetica*, translated into Latin in the twelfth century. According to a 1926 publication in the Journal of Chemical Education, one of the versions of the text that circulated within the community of alchemists

allegedly states ”that which is below is as that which is above, and that which is above is as that which is below, for accomplishing the miracles of a single thing.” [25]. It’s not known by our society whether this text from the hermetic tradition effectively once represented nature in a somehow useful form for humanity, but there is a straightforward relationship between that which is bellow in dimension terms (i.e. the one-dimensional laminar flamelet) can represent what is above (i.e. the 3-dimensional turbulent flame), as what is above (i.e. the detailed chemistry model) may be represented as something that which is below (i.e. the ILDM or FGM methods). As a matter of fact, there is an amazing album by Jorge Ben released in 1970 named *A Tábua de Esmeralda* that enthusiastically sings about the most characteristic hermetic traditions, with an energetic guitar played by the energetic composer that makes the passion for chemistry and it sciences - or arts - blossom by possibilities that emerge from such a unique way to face reality.

During the course of this work, the present report is written to serve - hopefully - as a guide through the mathematical concepts that underlies the construction of such curves and the properties that arise from the definition; through the ways of transforming a relatively old type of curve, which has analytical solutions for some fitting cases [21], into a problem that can be tackled by techniques currently associated with the field of machine learning. The Splines used in this work were implemented from the ground up, as also were the refinement techniques, in an effort from the author to have complete control on the behavior of the algorithm and to better understand the particularities present when dealing with this curves, as they are not being used here with the standard fitting methods, but with a mix between classical approaches and machine learning. In the end, the extra effort put in implementing the core part of the work without the help of external Spline libraries, resulting in a 5500 lines codebase, was of the utmost importance when developing the sampling strategy and the refinement criteria, as well as other formulations.

2. Methods

This work aims to put together methods used in machine learning to fit a Spline against manifolds generated by a freely propagating methane-air flame. It starts by defining the Splines and highlighting some of its most important characteristics for the work, starting from the most simple case of a one-dimensional Spline, going afterwards to define them to represent a single-input multiple-output function. These SIMO Splines, are the candidates for a possible high order tabulation method, that hopefully leads to less memory consumption, while retaining a suitable precision and a reasonable evaluation time. In that direction, a framework to use Splines with gradient descend will be created to adjust these curves against a manifold parametrized by a control variable $\psi = [PV]$ that output desired thermodynamic and chemical quantities φ.

In addition to this work, the bi-dimensional Spline manifold present in a book by Kunoth released in 2018 [26] is rewritten with a generalization for more than two inputs. This is done in the direction of foreseeing the eventual challenges that may arise when elevating the number of inputs from 1 to N, as it results in rapidly growing number of weights, as the order of the tensor that includes the weights also rises with N, which is going to be discussed in more detail in section 3.10. The focus here is to provide a concise and hopefully coherent mathematical framework to guide the given task, as mathematical modeling and computational implementation work hand in hand in finding solutions to real world problems

Automatic differentiation and gradient descend will be explained shortly, as they are key methods used in this work to search for solutions of the optimization problem that will be defined in section 3.1. Both methods can be combined to approximate the solutions by finding a suitable set of Spline parameters by relaxing the problem, where instead of searching for exact solutions, we take a series of steps that guide the Spline parameters in the direction of better solutions, as it's done in ANNs training.

The mathematical modeling from the work that generated the dataset here used is also going to be discussed, in order to point out the importance of CFD simulations when fighting climate change and show how Splines may be a suitable tool when dealing with thermochemical manifolds, as they can serve as a surrogate closure model for the quantities that need to be solved for when simulating reactive flows. There it will be assumed that the manifold generated by a one-dimensional laminar flame can be parametrized in fewer control variables than the total amount needed to solve a detailed chemistry mechanism, and be used to account for the chemical reactions in the flow. The flamelet model, which states that a three-dimensional flame can be considered as an ensemble of one-dimensional laminar flames, is assumed. This is allows us to use the data from single one-dimensional flame and tabulate it's results containing thermodynamic state of the mixture and the transport equations for the quantities used to index into the table, to calculate solutions

for 2- or 3-dimensional domains, while avoiding to perform a detailed chemistry simulation for a domain with higher dimension.

2.1. Splines

The term *Spline* was coined by the Romanian-American mathematician Isaac Jacob Schoenberg in his 1946 paper *Contributions to the problem of approximation of equidistant data by analytic functions. Part B. On the problem of oscillatory interpolation. A second class of analytic approximation formulae* [17]. They are piece-wise polynomials with class C^{K-1} globally and C^{∞} locally, which present an alternative to common polynomial fitting, where a polynomial of degree K is constrained to pass through $K+1$ data-points. Here, the piecewise polynomials are ”attached” in a way to preserve class of differentiability. This approach suffers less from the Runge's phenomenon [15], on which the values for the interpolation result can greatly differ from what is intuitively expected. Splines are a way of using polynomials for interpolation, but instead of elevating the polynomial order to be able to accommodate more data points, the Splines can have their domain further divided, so the amount of information that a region must represent is less than if a single polynomial was fitted to the whole curve.

Splines - and their more restrained cousins, the Bezier curves - have a big importance in computer graphics, such as in image manipulation software that relies on Splines and NURBS (Non-uniform rational B-Splines) to give the user an easy way of defining smooth curves by changing control points that, although are not interpolating, can be intuitively manipulated to obtain a desired curve. This intuition that comes naturally when manipulating Splines arises from their locality, as each weight influences a fixed number of neighboring regions. The goal here is to take advantage of this locality, that may be useful when dealing with problems which involve big scale separation, by defining a Spline in a way suitable for machine learning use, more specifficaly, for fitting parameters using gradient descend, taking advantages of r- and h-refinement methods.

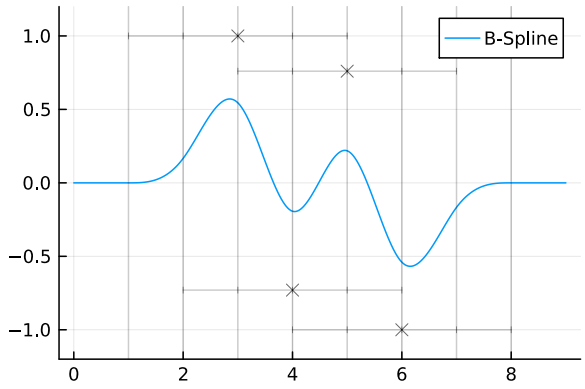


Fig. 2.1.: Representation of a Spline with degree $K = 3$ highlighting the value of the weights and the discretization of the domain.

In the notation used here to represent the Splines in the graphs tries to represent the formulations that will be given for the Spline. The vertical lines (in Figure 2.1 spanning throughout the picture, but later graphed just near the weights for simplicity) represent the Spline nodes ξ_i positioning in their respective axes; and the regions R_r that lies in between these two nodes. The crosses are used to indicate the weights w_i values, and their vertical position is the meaningful part, as their horizontal position in the SISO and SIMO cases is there to represent their center of mass in respect to the weights they represent. At last, the horizontal bars that crosses each cross are there to represent where their respective weight influences. Moving a given weight is guaranteed to not influence regions it doesn't control, and this region is always well defined.

The weights of a Spline are *not* interpolating. There is no guarantee, and it's not true in most majority of cases, that the Spline will assume the weights assigned for each region, - as there are multiple weights that govern a single region. That said, it can be proved that the Spline will never assume a value lesser or greater than the minimum and maximum weights, respectively. This can be used as an easy argument to ensure that the Spline output values won't explode to unreasonable values. That differs from the ANNs, which are black boxes with (currently) no easy way of assessing their behavior.

Important functions to this work, such as the basis Spline $B_{K_{i_1}}$ and the Spline S_K - whose will be defined in short -, are here parameterized explicitly after a semicolon. So, in the name of simplicity, a arbitrary function $f : X \to Y$ is going to be denoted as

$$f(x;\theta) = y \tag{2.1}$$

, omitting V from its definition, even though only $x \in X$, but $\theta \in V$. This is useful to express explicitly *where* the learnt function is calculated - it's input arguments -, namely x, and *which* are the parameters ϕ that define wheter the funtion parameters represents or not well reality, but without cluttering the notation with the parameter dimensions every time, since they will vary when adopting r- and h-refinement and should be implicitly defined by the context.

Here the same function name, for e.g. the Spline S_K may be called in different ways:

$$S_K\left(u;\vec{w},\vec{\xi}\right) \neq \vec{S}_K\left(u;W,\vec{\xi}\right) \neq \vec{S}_K\left(\vec{u};\mathbf{w},\Xi\right) \tag{2.2}$$

as $S_K(u) : \mathbb{R} \to \mathbb{R}$, $\vec{S}_K(u) : \mathbb{R} \to \mathbb{R}^M$ and $\vec{S}_K(\vec{u}) : \mathbb{R}^N \to \mathbb{R}^M$.

2.1.1. Single-input single-output

The first framework to experiment with was chosen to be the simplest case, namely the single-input single output case, which have a vast literature associated with, and it's simple to implement with the definition. Understanding the construction of the Spline and its following properties is useful to, in the future, have insights on how the local support works - and also how local it is. This understanding will serve as base for the justification of the chosen sampling method, that involves

introspecting in how our optimization target function (here called *loss function*). Later, being able to index every quantity will be useful to define the h-refinement criteria here proposed. This case is not going to be used to tabulate chemistry manifold, instead it will be used solely to serve as a base to construct the SIMO case, which is the main focus of this work, and the MIMO case, which is here defined as basis for future work. Results for the SISO Spline, such as the sampling method, are believed to generalize to more intricate cases.

To uniquely define a SISO Spline, it suffices a vector $\vec{w} \in \mathbb{R}^I$ with the weights

$$\vec{w} = [w_1, \ldots, \ w_I]^T \tag{2.3}$$

, another vector $\vec{\xi} \in \mathbb{R}^{I+K+1}$ with the node positions

$$\vec{\xi} = [\xi_1, \ldots, \xi_{I+K+1}]^T \quad , \quad \xi_i < \xi_{i+1} \quad , \quad \forall i \in [1..I+K] \tag{2.4}$$

and a degree $K \in \mathbb{N}_0$. Both vectors in the SISO context can be easily visualized in the plots, as explained in section 2.1. In the sake of remembrance (or the lack of it), each weight value is represented by a cross with a horizontal line passing though that expresses its zone of influence, and vertical lines that section the horizontal one to denote the Spline nodes.

For the SISO case, the basis Spline $B : \mathbb{R} \to \mathbb{R}$ is defined recursively (here with a slight change in notation) as done by De Boor [19]

$$\begin{aligned} B_{i,0}\left(u;\vec{\xi}\right) &= \begin{cases} 1 & , \quad \xi_i \leq u < \xi_{i+1} \\ 0 & , \quad \text{otherwise} \end{cases} \\ B_{i,k}\left(u;\vec{\xi}\right) &= \frac{u-\xi_i}{\xi_{i+k}-\xi_i} B_{\boxed{i,k-1}}\left(u;\vec{\xi}\right) + \frac{\xi_{i+k+1}-u}{\xi_{i+k+1}-\xi_{i+1}} B_{\boxed{i+1,k-1}}\left(u;\vec{\xi}\right) \end{aligned} \tag{2.5}$$

. In Figure 2.2, it's possible to see the shape of the basis Spline for different degrees K. The boxed subscripts are the responsible for the recursive nature of the basis Spline evaluation. From this definition, it's possible to notice that the degree K of the Spline is the sole responsible for determining the depth of the recursion, and as a consequence, a Spline of this degree will generate piece-wise polynomials of degree K. This definition is general enough to describe piecewise constant functions as a Spline with degree $K = 0$, and also for degree 1, with which the Spline generate a piece-wise linear interpolation, or any piece-wise polynomial of degree K.

With this recursive definition of the basis Spline in hand, the Spline $S_K : \mathbb{R} \to \mathbb{R}$ is defined as a summation over all the weights multiplied by the basis function $B_{i,K}$ in a given location u

$$\boxed{S_K\left(u;\vec{w},\vec{\xi}\right) = \sum_{i=1}^{I} w_i B_{i,K}\left(u;\vec{\xi}\right)} \tag{2.6}$$

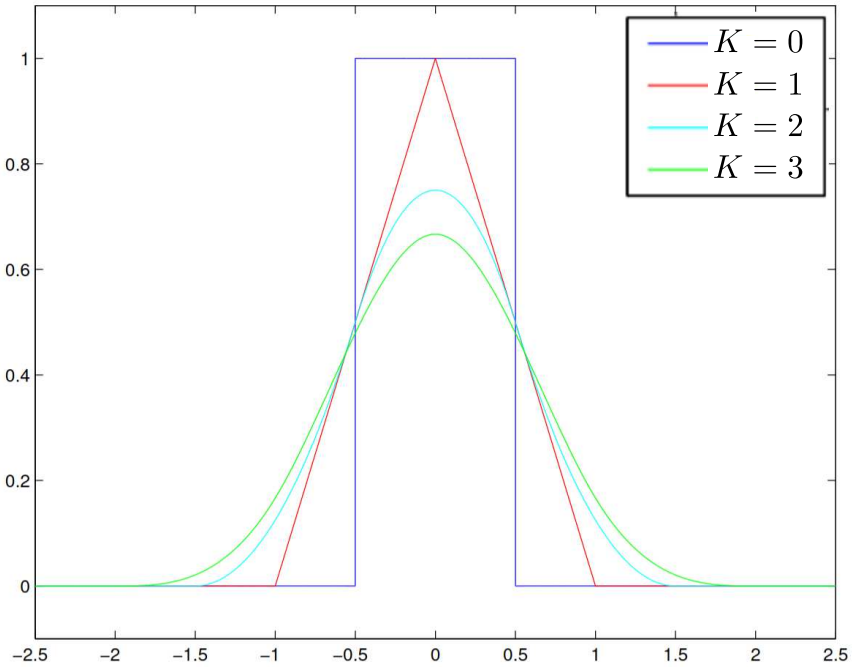


Fig. 2.2.: Basis Splines for degrees $0 \leq K \leq 3$. (Modified from Brian 2018 [21])

In order to speed up calculation using the optimizations provided by linear algebra libraries (in the present case LAPACK [27] through Julia wrappers), defining a basis Spline in the form of a vector function function $\vec{B}_K : \mathbb{R} \to \mathbb{R}^I$

$$\vec{B}_K\left(u;\vec{\xi}\right) = \begin{bmatrix} B_{1,K}\left(u;\vec{\xi}\right) \\ \vdots \\ B_{I,K}\left(u;\vec{\xi}\right) \end{bmatrix} \tag{2.7}$$

is useful to express the Spline definition as a scalar product between the weights and the basis function. This allows for SIMD optimizations and fearless parallelization of this piece of code, that could be used in the future to implement the use of GPU acceleration. The Spline representation as a scalar product is then

$$\boxed{S_K\left(u;\vec{w},\vec{\xi}\right) = \vec{w}\cdot\vec{B}_K\left(u;\vec{\xi}\right)} \tag{2.8}$$

The Spline defined until here presents the issue of having to calculate the basis functions for the whole domain. This means that the evaluation time of the function will grow, at least, linearly with the size of the weights/nodes vector. We can avoid many evaluations by using De Boor's algorithm [19], which allows us to reduce the processing evaluation cost by previously assessing which weights will govern the region that the input u lies within.

The non-null basis functions for a given weight can be visualized as a triangle in Table 2.1. This is the schema that is currently used to minimize the number of basis functions that are evaluated for each Spline evaluation. After expanding the basis Spline recursive function in its depth, it is possible to notice that when evaluating for a given u, $K+1$ basis functions are active (have value greater than zero).

		$B_{i,K}$			
	$B_{i,K-1}$		$B_{i+1,K-1}$		
	$B_{i,K-2}$	$B_{i+1,K-2}$		$B_{i+2,K-2}$	
		$\vdots$			
$B_{i,0}$	$B_{i+1,0}$	$\cdots$		$B_{i+K-1,0}$	$B_{i+K,0}$

Tab. 2.1.: De Boor's triangle.

Now let's quickly define a concept for region when dealing with Splines. Here, a region of the domain is defined as

$$R_r = \{x \in \mathbb{R} | \xi_r \leq x < \xi_{r+1}\} \tag{2.9}$$

which is the set of points enclosed by a node and its successor, as shown in Figure 2.3. A suitable definition for higher-dimension approaches could be easily defined by considering that in that case we must consider N subscripts for each region $R_{i_1,\ldots,i_N}$, one for each dimension, instead of only R_r. This regions serve now as a basis to explain the more efficient algorithm proposed by De Boor, but will be later used as a central construct when developing the refinement algorithms and criteria.

Putting together the expansion of a basis function and the concept of a region, we can say that if $u \in R_r$, then

$$\boxed{S_K\left(u; \vec{w}, \vec{\xi}\right) = \sum_{i=r-K}^{r} w_i B_{i,K}\left(u; \vec{\xi}\right)} \tag{2.10}$$

[19]. In respect to computation cost, this cuts the evaluation for the whole domain, replacing it for a *find first* search - given the nodes vector is ordered from lowest to highest - and the evaluation of $K+1$ basis functions. This can also be intuitively understand by thinking about how many quantities are necessary to define a polynomial of degree K, namely $K+1$ quantities.

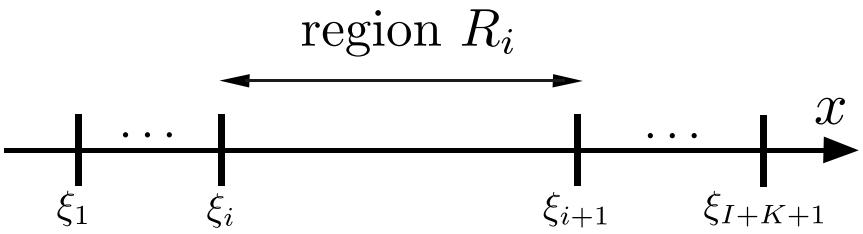


Fig. 2.3.: Illustration of the definition of a region within the Spline domain.

Further than that, it's shown also by De Boor [20] that the calculation may be achieved without the need to evaluate any basis function explicitly, with great savings in computation time and memory consumption, by defining

$$\begin{aligned} d_{i,k} &= (1-\alpha_{i,r})d_{i-1,r-1} + \alpha_{i,r}d_{i,r-1} \quad , \quad i \in [r-K..r] \quad , \quad k \in [1..K] \\ \alpha_{i,k} &= \frac{u-\xi_i}{\xi_{i+1+K-k}} \end{aligned} \tag{2.11}$$

, with which the desired result becomes

$$S_K(u) = d_{r,K} \tag{2.12}$$

This efficient Spline evaluation algorithm can be used when retrieving the values interpolated by the Spline during coupled CFD simulations, with a FORTRAN library developed by De Boor and released in 1977 [20]. Although, this algorithm in its most optimized form could not be used in this work, as we are restrained by the limitations of automatic differentiation, as will discussed in section 2.2, to find the derivatives of the Spline in respect to the weights and nodes. This form developed by makes a very heavy reuse of memory addresses, which are constantly overwritten, in order to spare allocations, which is not allowed by the current Auto-diff libraries. Nevertheless, this restriction only applies during training, when the weights and nodes that define the Spline are being determined, and does not imposes restrictions on the algorithms used to evaluate it afterwards.

2.1.2. On the authority of the parameters over the Spline domain

Before advancing to the SIMO case, it's worth to point out how the Splines govern the domain as defined by its nodes. As a direct consequence from the definition given in Equation 2.5, and also as can be deduced by the De Boor's triangle shown in Table 2.1, the first regions, namely $r \in [1..K]$, and last ones, where $r \in [I+1..I+K+1]$, won't have a complete set of weights defined. This leads to regions that cannot have the output values fully controlled by changing the weights. In these regions, it's not interesting to have data points, since the Spline behaves necessarily as if one or more of its weights is zero (the limit being for $u \notin [\xi_1, \xi_{I+K+1}]$, where the Spline behaves as if all its weights were zero) to ensure a differentiability class. The implementations assumed these partially controlled regions not to be suitable to represent data, and do not enforce any extrapolation methods to deal with regions outside the controlled zone.

To sum up before moving on to the next definitions, the Figure 2.5 is given here to display in one image the content up to here using a Spline of order $K = 3$. It can be seen in the image the non-zero part of the basis Spline shape functions that characteristic the Spline, defined by the curves shown in the lower part of the graph, that here have the nodes evenly placed, which should not be assumed the case here on in this work. On the bottom graph, the non-zero only happens in the parts that the upper graph marks as being part of the region that a given weight controls. Following a single region from bottom up, we can easily see how many weights influence a given region, and most importantly what weights have influence there.

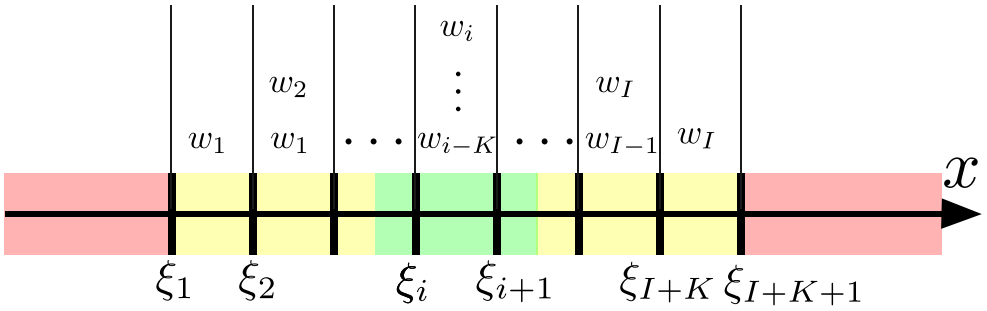


Fig. 2.4.: Illustration of how each region is influenced by the weights.

2.1.3. Single-input multiple-output

The SIMO case is the main focus of this work, with it we'll be able to represent the thermochemical states $\varphi = [\rho, \lambda, c_p, \mu, \dot{\omega}_c, \dot{\omega}'_T]$ based on $\psi = [PV]$, quantities that will be defined in section 2.4. This method is going to undergo a *a priori* analysis to evaluate how to different fitting methods perform in recreating the structure of a freely propagating one-dimensional flame. Although a very simple case, this is the ideal scenario to test a variety of hypothesis in a close-to-production scenario. The single-input multiple-output case makes it possible to assess how Splines will work when being constrained with individual weight vectors $\vec{w}_m$, but a shared node vector $\vec{\xi}$, which is specially important when applying the refinements; this without having the hassle of implementing h-refinement for the MIMO case, which involves indexing and iterating through a infinite dimension vector space. This combination of presenting the restrictions of the multiple-output cases, and without the complexity that multi-input brings, allows for fast prototyping of refinement criteria, error metrics and relaxation strategies.

In this case, will elevate the order of the weights from a vector to a matrix, and at first moment, the nodes in the same way. The main goal of explicitly defining initially the Spline with this elevated order also in the nodes is to show afterwards the benefits that we collect from restricting ourselves to work with every vector of nodes being equal instead of using a matrix, which would provide a bigger freedom to choose the nodes individually for each output, namely a reduction in the cost of calculating the basis functions.

Now, let the matrix $W \in \mathbb{R}^{I \times M}$ containing the weights be defined as

$$W = [\vec{w}_1, \ldots, \vec{w}_M] = [w_{i,m}] \quad , \quad \vec{w}_m \in \mathbb{R}^I \tag{2.13}$$

and a matrix $\Xi \in \mathbb{R}^{(I+K+1) \times M}$

$$\Xi = \left[\vec{\xi}_1, \ldots, \vec{\xi}_M\right] = [\xi] \ , \ \vec{\xi}_m \in \mathbb{R}^{I+K+1} \tag{2.14}$$

with the node positions for each of the M curves. The degree K of the Spline will not undergo any change during training, which must not necessarily be the case, as when dealing with Spline, there is a modality of refinement called in the literature as p-refinement, in which the order of the Spline can be increased locally, without affecting the order in regions away from it. Here K will

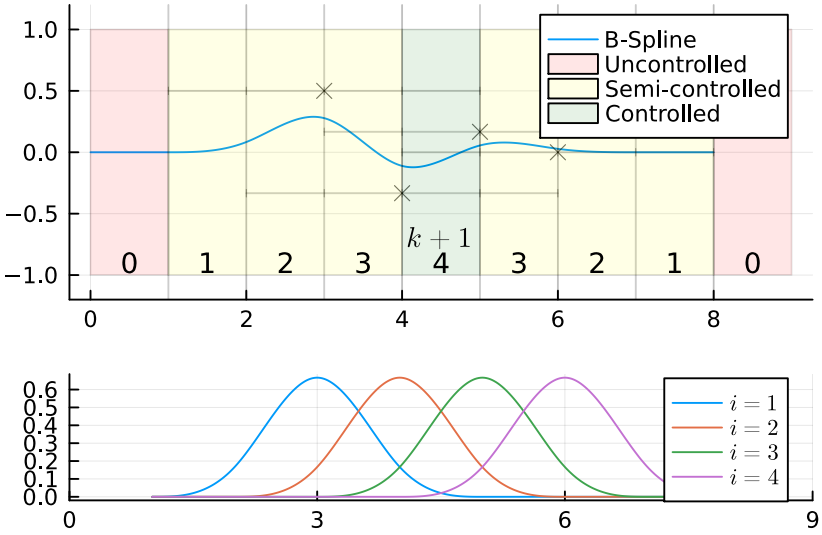


Fig. 2.5.: Spline representation with the resulting curve in the upper graph showing all important notation to be used when plotting Splines, and the bottom graph representing the four basis functions that provide local support to it. The numbers on each region represent the number of weights that are active for each.

stay constant in a single simulation for all nodes within a single run, but a variety of K are going to be tested in different training runs.

Having defined the matrices for weights and nodes, we can preliminarily define the SIMO Spline $\vec{S}_K : \mathbb{R} \to \mathbb{R}^M$ as

$$\vec{S}_K\left(u; W, \Xi\right) = \begin{bmatrix} S_{K1}\left(u; \vec{w}_1, \vec{\xi}_1\right) \\ \vdots \\ S_{KM}\left(u; \vec{w}_M, \vec{\xi}_M\right) \end{bmatrix} = \begin{bmatrix} \vec{w}_1 \cdot \vec{B}_K\left(u; \xi_1\right) \\ \vdots \\ \vec{w}_M \cdot \vec{B}_K\left(u; \xi_M\right) \end{bmatrix} \tag{2.15}$$

. This version could be used to undergo the training algorithm here described, but it would be no different of calculating M independent SISO Splines. The main idea of the work is to provide a reasonably fast way to calculate this Splines that does not prohibit the computation, while retaining the ability to perform automatic differentiation. For that, an approach to reduce this cost was to restrict these ensemble of Splines to have the same nodes vector, and consequently the same domain grid.

Now let's apply the aforementioned restriction that $\vec{\xi}_m = \vec{\xi} \in \mathbb{R}^{I+K+1}$, concretely transforming the ensemble of SISO Splines that would be calculated with the definition given in Equation 2.5, in a more reasonable definition that encapsulates these Splines using a single vector of nodes that will dictate which weights are going to influence each region in the output space. This allows us to reduce by $1/M$ the computational cost of evaluating the Spline, and a direct consequence of

this is that the r- and h-refinement algorithms must be able to place the nodes in position that are simultaneously advantageous for every output dimension. The nodes can be interpreted as a way to control where the weights are going to exert their influence, being directly associated with the number of input variables as will be seen in subsection 2.1.4. This was inspired in the work from 2017 by Popp [10], where he develops the Flatkernel method, where the quantities are tabulated using a N-dimension orthogonal grid that is feed into a code generator, so it can take further advantage of optimizations provided by the GNU Compiler Collection. Here we keep the position of the nodes equal for every output Spline with the aim of benefiting from LLVM compiler optimizations and hardware accelerations such as SIMD.

Now we can define a Spline for the single-input multiple-output case as its going to be used for training during this work $\vec{S}_K : \mathbb{R} \to \mathbb{R}^M$

$$\begin{aligned} \underline{\vec{S}}_K\left(u; W, \underline{\vec{\xi}}\right) &= \begin{bmatrix} \vec{w}_1 \\ \vdots \\ \vec{w}_M \cdot \end{bmatrix} \cdot \left[\vec{B}_K\left(u; \underline{\vec{\xi}}\right)\right] \\ &= W^T \cdot \left[\vec{B}_K\left(u; \underline{\vec{\xi}}\right)\right] \end{aligned} \tag{2.16}$$

. The definition used here for training will not take advantage of the full potential of De Boor's algorithm. Using the evaluation method discussed in [20] would be prohibitive during training, as the method called *automatic differentiation* here requires that no value that is present within the calculation of the functions to be differentiated is mutated in place. Nevertheless, this doesn't mean that the evaluation of the Spline to retrieve results and be used in coupled CFD simulations needs to be the same as the stated here. Due to the training process happening in a separate moment then the evaluations, which could be interpreted as using the tabulation method *in development* or *in production*, we may choose different formulations to evaluate in these different scenarios. For training, the formulation provided above is well suited, given that it's adequate for use with automatic differentiation, and also doesn't present a prohibitive cost for evaluation. After the training procedure, the nodes and weights obtained may be used alongside the De Boor's algorithm for optimal performance, as in this scenario there is no need to differentiate a loss function with respect to the Spline parameters. Although, this doesn't mean that no differentiation can be performed: calculating the derivative regarding only the input variable may be achieved easily using analytical formulations well described in literature that don't depend on automatic differentiation to work, meaning that these Splines could be used to tabulate thermodynamic potentials, such as the Helmholtz free energy, and be later used to retrieve quantities by differentiating the potential in respect to different variables.

During the training process and also when using the Spline to retrieve the tabulated data, we may want to calculate it in lots of different positions inside the domain, lets say $Z \in \mathbb{N}_1$ positions. To speed up that process, it's here implemented a computational routine to evaluate the Spline for many positions simultaneously. So let the index $z \in [1..Z]$. Let's also define a vectorized version of the Spline $\uparrow S_K : \mathbb{R}^Z \to \mathbb{R}^{M \times Z}$ which will operate in a vector of positions $\uparrow u = [u_1, \ldots, u_Z]^T$, $u_z \in \mathbb{R}$. We are going to also define a vectorized basis Spline $\uparrow B_K : \mathbb{R}^Z \to \mathbb{R}^{Z \times M}$

$$
\begin{aligned}
\uparrow \underline{S}_K \left(\uparrow u; \vec{w}, \vec{\xi} \right) &= \begin{bmatrix} S_{K1} \left(u_1; \vec{w}_1, \vec{\underline{\xi}} \right) & \cdots & S_K \left(u_Z; \vec{w}_1, \vec{\underline{\xi}} \right) \\ \vdots & \ddots & \vdots \\ S_{KM} \left(u_1; \vec{w}_M, \vec{\underline{\xi}} \right) & \cdots & S_K \left(u_Z; \vec{w}_M, \vec{\underline{\xi}} \right) \end{bmatrix} \\
&= \underbrace{\begin{bmatrix} \vec{w}_1 \\ \vdots \\ \vec{w}_M \end{bmatrix}}_{W^T} \underbrace{\begin{bmatrix} \vec{B}_K \left(u_1; \vec{\underline{\xi}} \right) & \cdots & \vec{B}_K \left(u_Z; \vec{\underline{\xi}} \right) \end{bmatrix}}_{\uparrow B_K \left(\uparrow u; \vec{\underline{\xi}} \right)}
\end{aligned}
\tag{2.17}
$$

Then we can finally have the version that is going to be evaluated in this work

$$
\boxed{\uparrow \underline{S}_K \left(\uparrow u; \vec{w}, \vec{\xi} \right) = W^T \cdot \uparrow B_K \left(\uparrow u; \vec{\underline{\xi}} \right)} \tag{2.18}
$$

Although it's not the most optimal implementation of Splines, it served useful to guide this work through the caveats involved in dealing with them in this scenario.

2.1.4. Multiple-input single-output

This case leaves us with the task of elevating the order of the Spline inputs to a vector instead of a scalar. We want to define formulation that allows us to pass a vector of inputs $\vec{u}$ to the Spline and get a scalar as result. With this formulation, we are going to follow the same path in subsection 2.1.4 as used for going from the SISO to the SIMO case in subsection 2.1.3, by concatenating the results of an ensemble of Splines inside a single vector of outputs, and afterwards restricting the formulation to use the same domain discretization for all curves, more effectively transforming a variety of uniquely defined Splines into a formulation that better resembles what we look for when started to look for such formulations.

The case for a MISO Spline is useful to assess the complexity with which the memory allocations grow with increasing dimensions. For the multiple-input single-output case, we are going to elevate the order of the weights from a matrix to a tensor of order N, with weights indexed as $i_1, \dots i_N$. Here it can be seen the exponential growth in complexity presented with the addition of further dimensions. Each new input value, for which the Spline is a function of, causes the total amount of weights to be multiplied by the size I_n of this new dimension. This is a problem that occurs with linear interpolation methods when used to represent high-dimensional data, and is still a problem when dealing with Splines, as we still need a tensor with an order that scales with the number of input parameters to represent the weights for such case.

$$
\mathbf{w} = [w_{i_1, \dots, i_N}] \in \mathbb{R}^{I_1 \times \cdots \times I_N} \tag{2.19}
$$

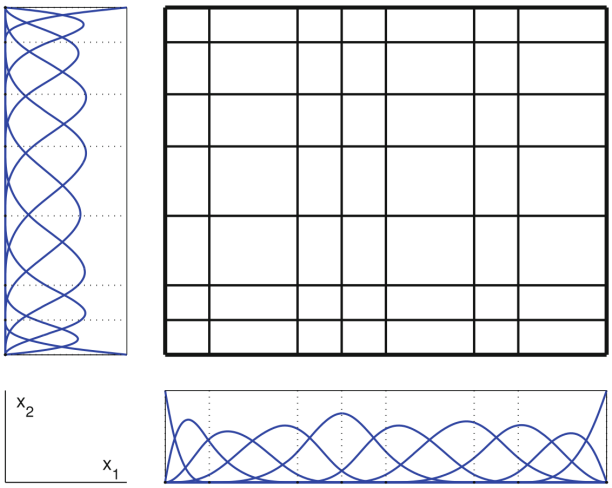


Fig. 2.6.: Orthogonal domain discretization (From Kunoth 2018 [26])

However, elevating the order of the nodes vector is not as straightforward as the procedure for weights. Instead of going from a vector to a matrix, due the fact that here each input dimension may have a different number of nodes, it is not possible to concatenate the different node vectors to form a rectangular matrix in the general case, with the only possibility for this to happen is if all the input variables were discretized with the same amount of nodes, and consequently the same amount of weights $I_n = I$. As a result, we would have a rectangular matrix belonging to $\mathbb{R}^{(I+K+1)\times N}$ containing the nodes. Despite the simplicity it would bring, the formulation for the MISO case is here done to allow a greater freedom, which forces us to group the node vectors inside a set Ξ. Individually, the node vectors are defined as

$$\vec{\xi}_n \in \mathbb{R}^{I_n+K+1} \quad , \quad n \in [1..N] \tag{2.20}$$

, and we may group them together as $\Xi = \{\vec{\xi}_1, \cdots, \vec{\xi}_N\}$.

Having now defined new orders for the weights tensor $\mathbf{w}$ and also for the set of vectors containing the nodes for each input dimension, we can follow to define the multiple-input single-output Spline $S_K : \mathbb{R}^N \to \mathbb{R}$

$$\boxed{S_K\left(\vec{u}; \mathbf{w}, \Xi\right) = \sum_{i_1=1}^{I_1} \cdots \sum_{i_N=1}^{I_N} \left(w_{i_1,\ldots,i_N} \cdot \prod_{n=1}^{N} B_{i_n,K}\left(u_n; \vec{\xi}_n\right) \right)} \tag{2.21}$$

The definition given here results in a domain divided orthogonally, as seen in Figure 2.6. Other domain discretizations are also possible, as shown by Gu in 2025 [22], where Splines are constructed over a planar domain triangularly divided, and as orthogonal regional hierarchical divisions, as will be discussed in section 5.1 when Hierarchical Basis Splines are briefly introduced.

2.1.5. Multiple-input multiple output

In order to go from the MISO to the MIMO case, the same brief discussion contained in subsection 2.1.3 giving the motivation to constrain the discretization of the domain, which presents itself as a N-dimensional grid for both this and the MISO cases, is also valid. The order of the weights tensor is elevated from N to $N+1$, to accommodate another index m, responsible for representing the sets of weights for each output, as did in subsection 2.1.3, and the nodes are represented by a set of vectors that define the grid for each output as did in subsection 2.1.4. The definition using the distinct discretizations that would be used here to perform the same discussion will be omitted, as the benefits that comes with it are analogous to the ones already discussed when dealing with a single input dimension, and we will jump straight to the case where the Splines are all restrained to the same nodes. That said, in this case we assume from the beginning that $\Xi_n = \underline{\Xi}$, so the set of nodes is defined as it was in the MISO case. We need now only to elevate again the order of the weights tensor to accommodate a different set of weights for each of the output variables

$$\mathbf{w} = [\mathbf{w}_1, \dots, \mathbf{w}_M] \in \mathbb{R}^{I_1 \times \cdots \times I_N \times M} \tag{2.22}$$

, with each of the weight tensors $\mathbf{w}_m \in \mathbb{R}^{I_1 \times \cdots \times I_N}$ being defined in the same way as for MISO, which let's us define the MIMO Spline $\vec{\underline{S}}_K(\vec{u}) : \mathbb{R}^N \to \mathbb{R}^M$

$$\vec{\underline{S}}_K\left(\vec{u}; \mathbf{w}, \underline{\Xi}\right) = \begin{bmatrix} \underline{S}_K\left(\vec{u}; \mathbf{w}_1, \underline{\Xi}\right) \\ \vdots \\ \underline{S}_K\left(\vec{u}, \mathbf{w}_M, \underline{\Xi}\right) \end{bmatrix} \tag{2.23}$$

. If we set $N = M = 1$, this definition drives us back to the SISO Spline, which is a fundamental property that such generalization to higher-dimensions should present.

2.2. Automatic differentiation

The methods for evaluating derivatives of computational functions can be divided in four different approaches: solving the derivatives by hand and implementing them; using numerical differentiation methods that performs finite difference approximations by evaluating the function at two distinct but appropriately close by points; performing symbolic differentiation using libraries that implement expression manipulation in computer algebra systems; and, finally, employing automatic differentiation [24], which is the technique used here.

Using the first method presents the clear disadvantage of demanding a heavy algebra exercise that gets heavier with the increasing complexity of the function to be differentiated. The second approach of performing finite difference calculations presents the advantages of being evaluated completely by the machine and being applicable to all computational functions that can be implemented to return a floating point value as result. Nevertheless, it presents precision problems, as the precision of the floating point quantities is limited to the amount of memory allocated for

it, as defined by the IEEE 754 standard, and the method presents two problematic things while calculating with floating point quantities: subtracting two numbers very close in the real line, and dividing by a quantity near zero. Following in the path of using a computer to calculate derivatives, one can use the computer to evaluate the symbolic expression for the derivative of the function by providing it with a analytical representation of it, which yields an exact representation of the function's derivative; although, this presents limitations when dealing with more complicated functions that present control flow or recursion, as it may be hard to describe them purely by mathematical symbols. Automatic differentiation comes in to solve the problem of precisely and efficiently evaluating derivatives of a large class of computational functions.

Automatic differentiation is a recent development that mixes characteristics of numerical differentiation, as it provides values for the derivative instead of a symbolic representation, and symbolic differentiation, as it uses differentiation rules to manipulate computation instructions. It's based on the fact that every instruction performed by the computer to evaluate any function that returns a number consists on composition of basic mathematical operations (such as addition, multiplication and trigonometric functions) that have a known derivatives. Auto-diff, as it's also called, may be performed in two different modes: forward and reverse. Forward mode is the most simple, and consists in accumulating the value of the derivative using dual numbers as the function is evaluated, where one will return the function value and the other the derivative, resulting in one "pass" through the function instructions. This mode is more performant when evaluating the derivative for a function $f : \mathbb{R}^N \to \mathbb{R}^M$, where $N \ll M$. In the other direction, the backward mode consists of accumulating the value of the derivative by passing through the memory addresses that contain the computation that was performed to evaluate the function, and performing the chain rule on every instruction responsible for the calculation, and it's more efficient for functions where where $N \gg M$ [28].

This method presents the advantages of being performed without user intervention ("automatic"), while also being able to deal with arbitrarily complicated computer functions without a symbolic representation of them. It is also very efficient, imposing an overhead of, at most, 6 times higher than evaluating the original function. Additionally, it is, in theory, an exact derivative of the computational implementation of the symbolic representation, in the sense that it uses the analytical results to calculate each step of the chain rule [29], by storing the intermediate values for a computation

$$\begin{aligned} y &= f(g(h(x))) = f(g(h(w_0))) = f(g(w_1)) = f(w_2) = w_3 \\ w_0 &= x \\ w_1 &= h(w_0) \\ w_2 &= g(w_1) \\ w_3 &= f(w_2) = y \end{aligned} \tag{2.24}$$

and afterwards calculating the derivative with a backwards pass on all instructions performed

$$\underleftarrow{\frac{\partial y}{\partial x} = \frac{\partial y}{\partial w_2}\frac{\partial w_2}{\partial w_1}\frac{\partial w_1}{\partial x} = \frac{\partial f(w_2)}{\partial w_2}\frac{\partial g(w_1)}{\partial w_1}\frac{\partial h(w_0)}{\partial w_0}} \tag{2.25}$$

. This leads to a restriction imposed by many Auto-diff libraries that no memory in the path of the chain rule can be mutated, in order to conserve the intermediate values used to perform the backwards pass. Naturally it leads to implementations with more memory consumption than it would be needed to evaluate a function that reuses memory, but this process of automatic differentiation would otherwise be slower and more memory consuming.

Contrary to the Splines, where no external packages where used, this work used various Julia Auto-diff libraries, mainly Zygote, but also ForwardDiff and ReverseDiff during most of the work, which seemed to work interchangeably for normal situations, and FiniteDiff when debugging, as even though less precise, works without the aforementioned restriction of immutable memory. More modern widespread tools, such as Enzyme, may be used, which consist in a tool that differentiates LLMV Intermediate Representation, thus being able of differentiating any language that can be converted to it, such as C, C++, Julia, FORTRAN and much more.

2.3. Gradient descend

Gradient descend is a method to solve unconstrained optimization problems. It consists in taking the gradient of a function, which happens to point to the direction of maximum growth in the calculated local, and then using this direction in order to take a step in the opposite direction in order to seek a local minimum. It is a iterative method that performs a number of iterations until some stopping criteria is met, such as a absolute or relative tolerance, or a specified maximum number of iterations. Due to its nature, the results achieved are an approximation of the true result, that may be influenced by the presence of local minima in the space formed by all possible parameters. The simplest form of gradient descend-based optimization consists in taking this steps proportionally to the gradient of the loss function in respect to the parameters multiplied by a learning rate here called α_{opt}, which could be represented as the iterative formula

$$\vec{\phi}_i = \vec{\phi}_{i-1} - \alpha_{opt} \nabla_{\vec{\phi}} \mathcal{L}\left(\vec{\phi}_{i-1}\right) \tag{2.26}$$

that is then iterated until the stop criteria is satisfied. This simple method, although sufficient to locate a local minimum in the vicinities of the starting point, will stop at the first valley, refusing to further explore more possible parameter sets.

Other methods to accomplish gradient descend were developed in the recent years that aim to find strategies that lead to a further exploration of the space, which may result in a better local minimum to be found, mainly taking advantage of adopting definitions of momentum that are inspired in a ball rolling down a hill [30]. This can be useful to further explore the space of possible parameters, since the ”ball” won’t blindly stop in the first local minima it finds if there is enough momentum to keep that ball rolling up the hill again, as it’s illustrated in Figure 2.7, and will keep the parameters changing for longer, and consequently leading to more regions of the space of all possible parameters being explored.

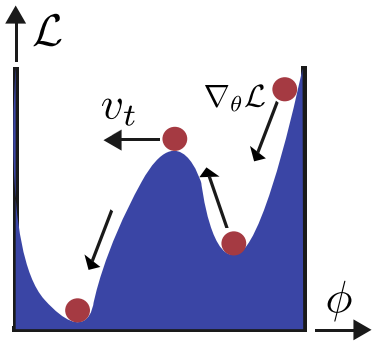


Fig. 2.7.: Illustration of the principle behind gradient descend and how methods that implement momentum avoid local minima.

One of such methods is the Adaptive Moment Estimation (Adam) [23], which calculate adaptive learning rates for the gradient descend problem by storing an exponentially decaying average of the past squared gradients v_t based on the second moment (uncentered variance) estimation, as many other methods do, but in addition to that by also storing an exponentially decaying average for the past gradients m_t with an estimation of the first moment (mean) of the loss function. The algorithm is described as

Algorithm 1 Adam method (from Diederik 2024 [23])

Require: α_{opt}: Learning rate
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
Require: $f(\phi)$: Stochastic objective function with parameters θ
Require: ϕ_0: Initial parameter vector
Ensure: ϕ_t: Resulting parameters
  $m_0 \leftarrow 0$ (Initialize first moment vector)
  $v_0 \leftarrow 0$ (Initialize second moment vector)
  $t \leftarrow 0$
  while ϕ_t not converged **do**
    $t \leftarrow t + 1$
    $g_t \leftarrow \nabla_\theta f_t(\phi_{t-1})$ (Get gradients w.r.t stochastic objective at time-step t)
    $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
    $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
    $\hat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
    $\hat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
    $\phi_t \leftarrow \phi_{t-1} - \alpha_{opt} \cdot \hat{m}_t / (\sqrt{\hat{h}_t} + \epsilon)$ (Update parameters)
  end while

The authors that developed this method propose in their work default values for the hyper-parameters as 0.9 for β_1, 0.999 for β_2 and 10^{-8} for ϵ. This proposed values were used in this present work, but the learning rate α_{opt} (in the original work called only α) was varied between the earlier and later parts of this work, with $\alpha_{opt} = 0.05$ for the studies on sampling and batching methods discussed in section 3.7, and $\alpha_{opt} = 0.025$ for fitting the Splines in the studies performed

in section 4.3 onwards. In Figure 3.4, it's possible to see the effects of the momentum making the error oscillate after the tenth iteration instead of going directly down, as the parameters climb a hill in the surface generated by the loss function before going down again. This extra time spent climbing up is naturally added to the total time of training, increasing the requirements of computational time, but may lead to better solutions being found.

The presence of momentum however showed itself a problem during runs with r-refinement turned on, as the nodes would keep getting closer to each other and consequently "colliding" afterwards, causing both regions that are so small that end up containing no data points, leading to lack of control of the parameters that govern such region as they are thus not represented inside the loss function, and nodes that end up in almost the same position, causing a division by near zero. To mitigate that, as discussed in section 3.8, a relaxation coefficient α_r is adopted to slow down the movement of the nodes, thus reducing the instabilities that arise from their movement towards each other.

2.4. Reactive flows simulation

Computational Fluid Dynamics is a powerful tool used to simulate the behavior a flow by numerically solving the conservation equations that describe it. The first attempts in predicting fluid movement numerically were performed using the Electronic Numerical Integrator and Computer, mostly known as ENIAC, the first multi-purpose programmable digital computer, that was completed in 1945 and first used to run calculations to assess the feasibility of the nuclear bomb, which it took 6 weeks to complete [31]. Despite this first infamous use, it was later used by J. von Neumann and J. Charney to perform calculations for weather forecasting, in one of the earliest computations involving fluid dynamics, which took 24 hours to solve the computation for a 24 hour forecast [32].

In order to simulate a flow, a set of equations comprised by mass and momentum conservation must be solved, alongside with thermodynamic state equations relating temperature, pressure and density, which may be assumed constant when the flow of interest allows [33]. Additionally, when simulating reactive flows, one must account for the chemical reactions taking place, either by solving a detailed or reduced chemical kinetics mechanism or by using automatic reduction methods, i.e. ILDM or FGM. This set of formulations regarding the flow were developed through several decades starting from 1822, firstly by Claude-Louis Navier and afterwards carried on by George Gabriel Stokes, who deduced what today are known as the Navier-Stokes equations.

Starting with the conservation of mass for a compressible flow, this formulation states

$$\frac{\partial \rho}{\partial t} + \frac{\partial \rho u_j}{\partial x_j} = 0 \tag{2.27}$$

, where ρ denotes the average density of the mixture, t is the temporal coordinate, x_j the spacial one, and u_j is the flow velocity. It's the same for reactive and non-reactive flows, since combustion does not create or destroy mass. As for the conservation of momentum, which is also the same for both flow cases, it's given by

$$\frac{\rho u_i}{\partial t} + \frac{\partial \rho u_i u_j}{\partial x_j} = \frac{\partial}{\partial x_j}\left[\rho\nu\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right) - \frac{2}{3}\rho\nu\frac{\partial u_k}{\partial x_k}\delta_{ij}\right] - \frac{\partial p}{\partial x_i} \tag{2.28}$$

, where ν is the momentum diffusivity, p is the pressure and δ_{ij} is the Kronecker delta, which equals one when $i = j$.

Given that we are dealing with a reactive flow, we must also take into account the chemical reactions, starting by writing a conservation mass for each species $k = 1, \ldots, N$

$$\frac{\partial \boxed{\rho} Y_k}{\partial t} + \frac{\partial \boxed{\rho} u_j Y_k}{\partial x_j} = \frac{\partial}{\partial x_j}\left(\frac{\boxed{\lambda}}{\boxed{c_p}}\frac{\partial Y_k}{\partial x_j}\right) + \boxed{\dot{\omega}_{Y_k}} \tag{2.29}$$

, where $Y_k = m_k/m$ is the mass fraction of the k-th species, λ is the heat conductivity, c_p is the specific heat capacity and $\dot{\omega}_{Y_k}$ is the source term for species k. We also need a balance equation for the temperate T, given by

$$\boxed{\rho c_p}\frac{\partial T}{\partial t} + \boxed{\rho c_p}\frac{\partial u_j T}{\partial x_j} = \boxed{\dot{\omega}'_T} + \frac{\partial}{\partial x_j}\left(\boxed{\lambda}\frac{\partial T}{\partial x_j}\right) - \boxed{\rho}\frac{\partial T}{\partial x_j}\underbrace{\left[\sum_{k=1}^{N} c_{p,k}\left(-D\frac{\partial Y_k}{\partial x_j}\right)\right]}_{\boxed{D_{diff}}} \tag{2.30}$$

, where D is the diffusion coefficient and D_{diff} stands for a measure of the temperature diffusion caused by species diffusion.

The quantities that were highlighted in equations Equation 2.29 and Equation 2.30, if known, are enough to make a closure model of the chemistry and thermodynamic state, allowing one to solve the equations numerically.

Even though the available computing power have rapidly increased since the first CFD simulations performed with the ENIAC, performing direct numerical simulations accounting for a detailed chemistry mechanism in a three-dimensional flow can be still prohibitively expensive, despite the amount of computational power present today. To mitigate that, one could simplify the representation of the chemical reactions by assuming that a progress variable Y_{PV} may be used to collapse all the chemical equations into a single balance equation for this control variable, allowing us to rewrite Equation 2.29 as

$$\frac{\partial \rho Y_{PV}}{\partial t} + \frac{\partial \rho u_j Y_{PV}}{\partial x_j} = \frac{\partial}{\partial x_j}\left(\frac{\lambda}{c_p}\frac{\partial Y_{PV}}{\partial x_j}\right) + \boxed{\dot{\omega}_{Y_{PV}}} \tag{2.31}$$

and Equation 2.30 as

$$\rho c_p\frac{\partial T}{\partial t} + \rho c_p\frac{\partial u_j T}{\partial x_j} = \dot{\omega}'_T + \frac{\partial}{\partial x_j}\left(\lambda\frac{\partial T}{\partial x_j}\right) - \rho\frac{\partial T}{\partial x_j}\left[\sum_{k=1}^{N} c_p\left(-D\frac{\partial Y_{PV}}{\partial x_j}\right)\right] \tag{2.32}$$

, where $\dot{\omega}_{Y_{PV}}$ is the source term for this progress variable, which, if known, closes the model. As a result, the number of equations needed to solve the chemical species balance equations is drastically reduced. From the initial scenario, on which a balance equation for each species had to be solved, we end up in a scenario that only one balance equation has to be solved to account for the chemical reactions.

In order to achieve this closure model, the laminar flamelet model is used to generate the data using a one-dimensional laminar flame using a detailed chemistry. Afterwards, it's possible to use the data obtained from this manifold as a chemistry closure model for higher-dimensional turbulent reactive flows, as the turbulent flame behaves locally as the laminar flamelet, and the manifold approach allows us to express all the closure variables in term of a selected control variable.

3. Spline fitting

The kernel of this thesis is to apply machine learning techniques to fit Spline parameters that are suitable for representing a FGM of a freely propagating methane flame, and with that to foresee the path and the challenges that may arise when fitting higher-dimensional data generated by phenomena that can only be reasonably represented using more than one input variable, such as quenching, which demands the parametrization also to take into account the temperature of the mixture, in order to capture the effects of heat transfer to a wall. The methods addressed here are mainly the ones that make it possible to transform the Spline parameter fitting problem into a optimization problem that can be approximated by gradient descend. By relaxing the problem to an approximation of the parameters that bring the loss function to a minimum, we can use techniques that exchange laborious analytical work for automated gradient calculations and parameter updates that will navigate through the solution space until a suitable set of parameters is found. When formulating such problems, it's important to keep in mind that the minima found is local, not global; so the problem must be formulated to provide a reasonable approximation for the vast majority of local minima that can be found.

Of course, not always the local minima found is going to be an accurate representation of the target data. To tackle that problem, r- and h- refinement methods are going to be implemented and assessed to verify their suitability to replace or insert new nodes respectively in such a way that make it possible to find new local minima for the loss function. When dealing with h-refinement, new nodes are inserted according to criteria, which leads to a domain topology change, which makes it tricky to say precisely that a new local minima was found, but whether if a new local minima was found for the newly parametrized function. The criteria for both types of refinement were created *ad-hoc*, and were though to attack the problems that arouse from each of them.

Usually, machine learning refers to techniques involved in fitting artificial neural networks. These are universal function approximators [16] that are currently being developed both by academia and industry, as it does not cease to show new potentials to be applied in vastly different fields of knowledge, with the capability of finding the correlation between inputs and outputs with a surprising generalization potential. These will not be the subject of this work. Here, the techniques normally applied to train ANNs are revisited in the context of Spline fitting, most importantly the relaxation of the problem, modifying it from a formulation which demands exact solutions into a optimization problem that minimizes the value of a given loss function in terms of the parameters that define the output model; and the solution of this new formulation of the problem using gradient descend, that performs a search through the parameter space with each step - hopefully - leading us to a better and better set of weights as the training process advances.

3.1. Optimization problem formulation

As defined earlier, the Splines are defined by their parameters, namely weight values and node positions and also by their degree. In order to search for a suitable set of these parameters that approximates the Spline, we are going to relax the problem of finding the set of values that minimize a given loss function, i.e. the mean squared error, by turning it into a gradient descend problem. This moves us from complex analytical solutions that present restrictions to be valid, but would yield exact solutions, to a problem with fewer constraints to be solved and gives us approximated solutions.

To represent the weights and nodes, a vector of parameters $\vec{\phi}$ is going to be used. Inside this vector, we can place all the quantities that we may want to be solved for using gradient descend. There are two distinct cases for which we could define this vector, namely with and without r-refinement, distinguished by including or not the nodes in the optimization. In this case, let's define a vector $\vec{\phi}$ containing the trainable parameters

$$\vec{\phi} = [\vec{w}_1^T, \ldots, \vec{w}_M^T, \underbrace{\xi_1, \ldots, \xi_{I+K+1}}_{\text{Only in } r\text{-refinement}}]^T \tag{3.1}$$

After having defined how the parameters vector is going to look like, we must define the structure that the dataset must obey when using the current approach. For that, let's define a function $\vec{\varphi} : \mathbb{R} \to \mathbb{R}^M$ with $\vec{\varphi} = [\varphi_1, \cdots, \varphi_M]$ to be the target which we would like to approximate with a Spline $\vec{S}_K \approx \vec{\varphi}$ by finding a suitable vector of trained parameters $\vec{\phi}_t$. In this work, the functions to be approximated are

$$\begin{array}{rcl} \rho & = & \varphi_1(PV) \\ \lambda & . & \\ c_p & . & \\ \mu & . & \\ \dot{\omega}_{Y_{PV}} & . & \\ \dot{\omega}_T' & = & \varphi_6(PV) \end{array} \tag{3.2}$$

For the present study, we have M outputs in function of a single input, so the data to be represented has the form

$$y_{d,m} = \varphi_m(x_{d,m}) \quad , \quad \forall d \in [1..D_m] \quad , \quad \forall m \in [1..M]$$

, with D_m being the number of entries in the training dataset for the m-th curve, $x_{d,m}, y_{d,m} \in \mathbb{R}^{D_m}$ being the inputs and outputs present in the dataset and D being the total number of data points. No assumption is made regarding the number of samples of each given curve, so differently sized sets of points can also be used. The dataset $\mathcal{D} \in \mathbb{R}^{D \times 2}$ can be defined as

$$\mathcal{D} = \begin{bmatrix} \vec{x}_1^T & \vec{y}_1^T \\ \vdots & \vdots \\ \vec{x}_M^T & \vec{y}_M^T \end{bmatrix} \tag{3.3}$$

with

$$\begin{aligned} \vec{x}_m &= [x_{1,m}, \ldots, x_{D_m,m}]^T \\ \vec{y}_m &= [\vec{y}_{1,m}, \ldots, \vec{y}_{D_m,m}]^T \end{aligned} \tag{3.4}$$

Performing a normalization on the dataset is important for any machine learning algorithm in order to bring the data into a space where patterns are more recognizable. This allows the model to abstract away differences in magnitude across different outputs, which is essential for the current problem, where the tabulated quantities have very distinct ranges, as will be seen in section 4.2. To perform normalization, we will use the minimum and maximum values present in the dataset. This is possible, because as discussed in the same mentioned section, the dataset under study does not contain any outliers. So let's define the input extrema $x_{min}\ ,\ x_{max} \in \mathbb{R}$ as

$$\begin{aligned} x_{min} &= \min_{m\in[1..M]} \min_{d\in[1..D_m]} x_{d,m} \\ x_{max} &= \max_{m\in[1..M]} \max_{d\in[1..D_m]} x_{d,m} \end{aligned} \tag{3.5}$$

and the output extrema $\vec{y}_{min}\ ,\ \vec{y}_{max} \in \mathbb{R}^M$ as

$$\begin{aligned} \vec{y}_{min} &= [y_{min1}, \ldots, y_{min_M}]^T \quad , \quad y_{min_m} = \min_{d\in[1..D_m]} x_{d,m} \\ \vec{y}_{max} &= [y_{max_1}, \ldots, y_{max_M}]^T \quad , \quad y_{max_m} = \max_{d\in[1..D_m]} x_{d,m} \end{aligned} \tag{3.6}$$

With that, we can define normalized versions of the inputs and outputs contained in the dataset as

$$\begin{aligned} \hat{y}_{d,m} &= norm(y_{d,m}; y_{min_m}, y_{max_m}) \\ \hat{x}_{d,m} &= norm(x_{d,m}; x_{min}, x_{max}) \end{aligned} \tag{3.7}$$

which allows us to define a normalized version of the dataset $\hat{\mathcal{D}}$. The methods used for normalization will be further discussed in section 3.6, whereas here $norm$ represents any suitable normalization scheme. As the dataset $\mathcal{D}$ can be arbitrarily large and unevenly distributed, a sampling strategy will be discussed in section 3.7, which will alow us to batch the data for each training iteration in order to reduce the computational cost at each epoch, by using only a subset $\tilde{\hat{\mathcal{D}}}$ of the total data.

Additionally in section 3.4 and section 3.5, the loss function $\mathcal{L}$ will be defined for the SISO and the SIMO case respectively. This loss function encapsulates the trained parameters $\vec{\phi}_t$ that provide the approximation $\vec{S}_K\left(PV; W, \vec{\xi}\right) \approx \vec{\varphi}(PV)$, as it is defined to decrease in value when a given error metric goes down (here mostly the mean squared error) in a way that the trained parameters can be expressed as

$$\boxed{\hat{\vec{\phi}}_t = \arg\min_{\hat{\vec{\phi}}} \mathcal{L}\left(\hat{\vec{\phi}}, \tilde{\hat{\mathcal{D}}}\right)} \tag{3.8}$$

To search for the parameters that minimize the loss function, a iterative process will take place, where at each iteration the weights (and nodes, if r-refinement is on) get updated according to the gradient descend method discussed in section 2.3. Moreover, after a given amount of iterations, h-refinement is performed to insert new nodes to the Spline with methods to be discussed in Figure 3.6. The algorithm that performs this iterations until convergence or other stopping criteria being match is the following:

Algorithm 2 Training loop

Require: $\mathcal{D}$: dataset
Require: $atol \geq 0$: absolute tolerance
Require: $itr_{max} \in \mathbb{N}_1$: maximum number of iterations
Require: $itr_h \in \mathbb{N}_1$: number of iterations between h-refinements
Ensure: $\vec{\phi}_t$: trained parameters vector
  $\vec{\phi} \leftarrow$ INITIALIZE($\mathcal{D}$) (Initialize the parameters)
  $\hat{\mathcal{D}} \leftarrow$ NORM($\mathcal{D}$) (Normalize the dataset)
  $\hat{\vec{\phi}} \leftarrow$ NORM($\vec{\phi}$) (Normalize the parameters)
  $itr \leftarrow 0$ (Initialize iteration counter)
  while $MAE \geq atol$ and $itr < itr_{max}$ **do** (Repeat until convergence or stopping criteria is met)
    $itr \leftarrow itr + 1$ (Increase iteration counter)
    $\tilde{\hat{\mathcal{D}}} \leftarrow$ SAMPLE($\hat{\mathcal{D}}$) (Select a sample from the dataset)
    $\mathcal{L}_{cur} \leftarrow \mathcal{L}(\hat{\vec{\phi}}, \tilde{\hat{\mathcal{D}}})$ (Evaluate the current loss)
    $\mathcal{L}'_{cur} \leftarrow \nabla_{\hat{\vec{\phi}}} \mathcal{L}(\hat{\vec{\phi}}, \tilde{\hat{\mathcal{D}}})$ (Evaluate the current loss gradient w.r.t the parameters)
    $\hat{\vec{\phi}} \leftarrow$ UPDATE($\hat{\vec{\phi}}$, $\mathcal{L}_{cur}$, $\mathcal{L}'_{cur}$) (Update the parameters following the gradient descend)
    if itr mod $itr_h = 0$ **then** (Evaluate if it's h-refinement iteration)
      $\hat{\vec{\phi}} \leftarrow h$-REFINE($\hat{\vec{\phi}}, \hat{\mathcal{D}}$) (Insert new weights and nodes)
    end if
  end while
  $\vec{\phi}_t \leftarrow$ DENORM($\hat{\vec{\phi}}$) (Denormalize the parameters)

3.2. Weights initialization

For the SIMO case, two criteria were considered to initialize the Spline before advancing to the training loop. The methods tried were first to initialize the weights with random values between y_{min} and y_{max}. This method is used in neural networks training under the - very straightforward - name *Random initialization*. Both constant and Gaussian probability distributions were tried on this approach. Despite working well in this kind of scenario, it quickly showed itself not to be the ideal one when training Splines, as the Spline would usually be initialized with weights that resulted in a initial Spline with relative error (as defined in Equation 4.1) greater than 50%.

In order to diminish this problem, a technique called *Data-driven initialization* [34] is used. Although very complicated for neural networks, it was easy to come up with an initialization criteria for Spline weights based on the given data. The method used here is to initialize all the weights with the same value, with this value being $(y_{min} + y_{max})/2$. This ensures that the initial Spline will present a maximum relative error of 50% or less, which in practice turned out to be a good enough initialization criteria, as it was observed that, with a proper learning rate for the gradient descend, this error rapidly diminishes and the Spline gets close enough to the original curve, where the initial values don't matter anymore. More precise ways of initializing the weights may be achieved using analytical solutions for the weight values, to achieve an even better starting guess before passing the problem to be solved using gradient descend. This would be very advantageous, as it can be seen during the training process that time is spent bringing the weights from a far away position close to the Spline before the real fine-tuning of the parameters begin, where the Spline really starts to take advantage of the higher degrees it can achieve, as the weights are guided not only to represent the value of the given curve, but also its derivative.

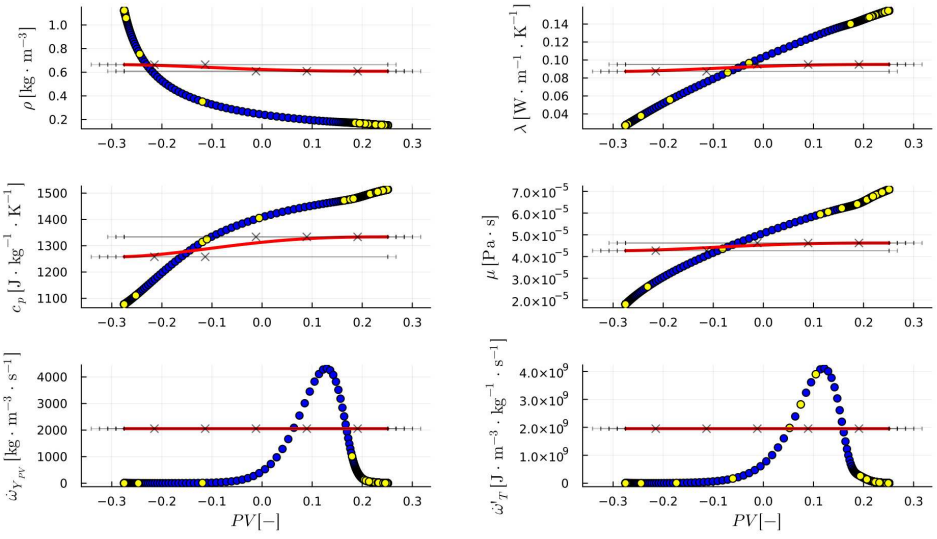


Fig. 3.1.: Spline initialized with all weights at $(y_{min} + y_{max})/2$ after one training iteration.

In Figure 3.1 we can observe an image of a single-input multiple-output Spline in its early stages of

training. This Spline was initialized with the strategy of placing all the weights at $(y_{min} + y_{max})/2$, limiting the initial relative error to $\leq 50\%$, afterwards the recently initialized Spline underwent one iteration of training. Both progress variable source term $\dot{\omega}_{Y_{PV}}$ and temperature source term $\dot{\omega}'_T$ remained barely untouched after this first iteration, so the initialization strategy can be clearly seen. The beginnings of how gradient descend brings the weights closer to the Spline can be seen in the curve for the specific heat capacity c_p. This method showed itself in practice to be sufficient in the task of choosing weights close enough that they lie within the radius of convergence of the Adam method [23].

3.3. Nodes initialization

The nodes initialization can be divided in two different steps: initializing the nodes that will lie outside the domain, that are only there to ensure that the Spline is fully controlled within the dataset span, alongside the ones that lie right on the boundaries; and initializing the nodes that lie strictly within the domain. For this second step, there are also differences in the initialization for the training with and without h-refinement. Here, the nodes that lie outside the domain are located in the region marked as yellow in Figure 3.2, and the ones inside the domain reside in the green area.

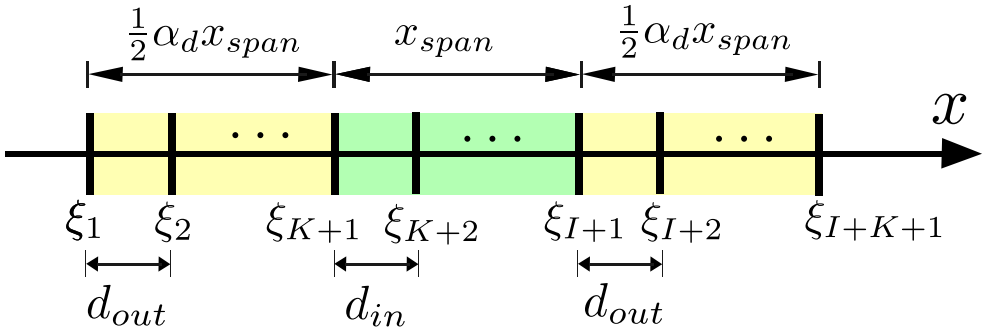


Fig. 3.2.: Initial node positioning for case without h-refinement.

In the scenario without h-refinement, the nodes for both regions are initialized equidistantly as shown in Figure 3.2, each with a distinct distance d_{out} or d_{in} in respect whether that node marks the beginning of an authorized region or not as defined

$$\begin{aligned} \xi_{i+1} - \xi_i = d_{out} &\iff R_i \text{ is unauthorized} \\ \xi_{i+1} - \xi_i = d_{in} &\iff R_i \text{ is authorized} \end{aligned} \tag{3.9}$$

, where the distances are defined as

$$
\begin{aligned}
d_{out} &= \frac{\alpha_d x_{span}}{2K} = \frac{x_{min} - \xi_1}{K} = \frac{\xi_{I+K+1} - x_{max}}{K} \\
d_{in} &= \frac{x_{span}}{n_{reg}}
\end{aligned}
\tag{3.10}
$$

with x_{span} being the span distance for the values that the input variable may assume, defined as

$$
x_{span} = x_{min} - x_{max} \tag{3.11}
$$

Additionally, four specific nodes are set in fixed positions, with the extreme nodes being placed accordingly to a parameter α_d meant to represent how further apart from the extrema of the data domain the end nodes should be positioned, and the nodes on the boundaries of the dataset domain being placed at the minimum and maximum values of the input variable, as defined by

$$
\begin{aligned}
\xi_1 &= x_{min} - \frac{\alpha_d x_{span}}{2} \\
\xi_{K+1} &= x_{min} \\
\xi_{I+1} &= x_{max} \\
\xi_{I+K+1} &= x_{max} + \frac{\alpha_d x_{span}}{2}
\end{aligned}
\tag{3.12}
$$

For the initialization with h-refinement the same strategy is used to initialize the nodes outside the dataset domain and the ones right on the boundaries, and the strategy adopted to initialize the nodes inside the domain is based on keeping them in a minimum quantity, as the node and weight vectors are going to be modified to include the new topology of the grid. Following this philosophy of keeping the regions to a minimum during initialization in the case that h-refinement is present, it was chosen to start the training runs with $n_{reg} = 1$, which means that the whole domain is represented by a single region in the beginning. This situation is analogous to fitting a single polynomial of degree K using the same data, a situation that leads us to what here is considered to be the tiniest well-defined Spline, with $I = K + 1$ weights and $2K + 2$ nodes.

3.4. Loss function for single-input single-output case

One of the most important parts when dealing with machine learning and optimization is to craft a loss function that captures what we expect to be the *best model*. The core idea in this part involves crafting the loss function in such a way that minimizing the error that the model presents against the target data decreases its value, but may also involve other techniques, such as bringing other analysis inside the loss function, to include symmetries that the resulting model is expected to respect, or even enforcing energy and mass conservation. The loss function should also be sufficiently inexpensive to calculate to allow us to perform thousands of iterations before the algorithm converges, which stops us from using every method available to enhance the loss function at the same time. In the present case, a loss function consisting solely of the mean squared

error was sufficient to capture in a optimization problem the nuances of what kind of model is being searched for with gradient descend.

The single-input single-output case served as a laboratory to experiment with regularization techniques, which are methods that include inside the loss function information about the distribution of the weights. When dealing with ANNs, one can make use of L^p regularization [35] to prevent the issue of vanishing or exploding gradients by penalizing the magnitude of the weights value alongside the magnitude of the MSE, leading to smaller valued weights, to increase training stability and to diminish unexplainable behavior in the model outputs. Here, regularization was experimented as a strategy to fight the problem of regions becoming unpopulated brought by r-refinement. A term was inserted inside the loss function to penalize the proximity between two nodes, so the gradient descend looks for solutions that present nodes more spread apart. The loss function was then defined as

$$\begin{aligned}\mathcal{L}_{SISO}\left(\tilde{\hat{\mathcal{D}}}, \hat{\vec{\phi}}\right) =& \alpha_1 \underbrace{\frac{1}{\tilde{D}} \sum_{d=1}^{\tilde{D}} \left(S_K\left(\hat{x}_d; \hat{\vec{w}}, \hat{\vec{\xi}}\right) - \hat{y}_d\right)^2}_{MSE} \\ &+ \alpha_2 \underbrace{\frac{1}{I+K+1} \sum_{i=1}^{I+K} \frac{1}{(\xi_{i+1} - \xi_i)^2}}_{\text{Measure of nodes proximity}}\end{aligned} \tag{3.13}$$

, where the quantities marked by a hat are normalized quantities (such as the dataset $\tilde{\hat{\mathcal{D}}}$, the nodes $\hat{\vec{\xi}}$ and weights $\hat{\vec{w}}$), and the tilde marking the dataset $\tilde{\hat{\mathcal{D}}}$ indicates that it it a subset of the whole set of available data, with $\tilde{D}$ being the cardinality of this subset. The normalization technique is discussed in section 3.6 and the sampling strategy to assemble $\tilde{\hat{\mathcal{D}}}$ is discussed in section 3.7.

Since the proximity of the nodes and the mean squared error may have different magnitudes, two coefficients α_1 and α_2 were inserted in order to make it possible to tune the loss function in terms of this two considered quantities. This coefficients are vital in regularization strategies, and present a new challenge to determine suitable values for they, increasing the complexity of the problem by adding new hyper-parameters that must be tuned in order to lead to a stable training run. A poorly chosen α_2 could be either too high, leading to the optimization process going to a direction were the loss function decreases further by spreading the weights apart instead of by diminishing the MSE, or too low, in a scenario that the benefits of such regularization would not be felt, leaving us with a problem that is more computationally intense without properly adding new qualities to the final solution.

After these experimentation, it was decided to stick with the mean squared error as sole term inside the loss function, as there was no perceivable improvement in stability observed in the early stages of this work, and the addition of new terms in the loss function always brings up another coefficient α responsible for tuning the importance of this newly added term, and if not chosen wisely, may introduce instabilities in the training process. The author believes that such regularization methods may work in the efforts to bring more stability for the training with r-refinement, although these benefits probably did not blossom in the single-input single-output case possibly due to the lack of

other important features that were implemented in the single-input multiple-output case, such as the relaxation of the step taken in the nodes vector.

As a side comment, the main challenge while crafting a loss function is to avoid what was described by Asimov in his book from 1950 titled *I, Robot* containing nine short tales about what he calls the alignment problem. This is an example in literature that clearly exemplifies how the optimization of parameters can lead to unexpected results, with so called *artificial intelligence* acting against the will of their creators leading to undesired consequences. In the same direction, we can interpret that the previously seen misbehavior when optimizing parameters to fit Splines while penalizing proximity of the nodes as sort of a misalignment error, where the loss function was further minimized by spreading the weights apart then by getting the resulting Spline closer to the target curves.

3.5. Loss function for single-input multiple-output case

After experimenting with the loss function in the SISO case, the definition for the SIMO case was chosen to contain only the mean squared error. The aim here is to make a single loss function that can capture the mean squared error of every output curve in a balanced manner. This is important to ensure that the optimizer will take care of each weight giving to them all the same importance, because if it was not the case, some of the curves could end well fitted to the data, while others would be practically ignored by the process. This is very prone to happen in the current scenario, given that the extreme values for the outputs y_{min_d} and y_{max_d} are very different for the different output curves φ_m. The loss function for this case is defined as

$$\mathcal{L}_{SIMO}\left(\vec{\hat{\phi}}, \tilde{\hat{\mathcal{D}}}\right) = \text{MSE}\left(\vec{\hat{\phi}}, \tilde{\hat{\mathcal{D}}}\right) = \frac{1}{M}\frac{1}{\tilde{D}}\sum_{m=1}^{M}\sum_{d=1}^{\tilde{D}}\left(S_{K_m}\left(\hat{x}_{d,m}; \vec{\hat{\phi}}\right) - \hat{y}_{d,m}\right)^2 \tag{3.14}$$

When using the sampling strategy discussed in section 3.7, this loss function here defined leaded to a very stable training, always converging towards a solution that was, at least by eye, what is intuitively searched when it was first proposed to use gradient descend to find the adequate weights. As the Splines showed a tendency to sit closer to the target curves for every training step, it is assumed here that this loss function is suitable to capture the shape of the FGM that we aim to represent.

3.6. Normalization

When working with many subjects, normalizing the data based on some reasonable factor often comes pretty handy in order to deal with discrepancies. Normalization is a pre-processing method also used to improve the performance of machine learning [36], enhancing the prediction accuracy mainly by bringing the standard deviation of all the features onto a reasonable base for comparison, which is acknowledged to be a very important factor when determining the most elucidative features when performing a PCA.

As discussed in section 3.7, the Spline optimization process showed itself to be very sensitive regarding its symmetries. Naturally, if its sensitive to having the symmetries respected when batching, one could foresee that the process will respond well to being performed in normalized quantities.

Here, two methods were used to normalize both the datasets and the Spline weights and nodes as well.

The first normalization $norm_{0,1} : \mathbb{R} \to [0, 1]$ used is defined as

$$norm_{0,1}(x; x_{min}, x_{max}) = \frac{x - x_{min}}{x_{max} - x_{min}} \tag{3.15}$$

. This definition has the property that

$$\begin{aligned} norm_{0,1}(x; x_{min}, x_{max}) = 0 &\iff x = x_{min} \\ norm_{0,1}(x; x_{min}, x_{max}) = 1 &\iff x = x_{max} \end{aligned} \tag{3.16}$$

, this meaning that the boundaries of the normalization function image will always coincide with 0 and 1.

The second normalization $norm_{-1,1} : \mathbb{R} \to [-1, 1]$, which is the one that effectively delivered results is defined after as

$$norm_{-1,1}(x; x_{min}, x_{max}) = \frac{x}{max(\{|x_{min}|, |x_{max}|\})} \tag{3.17}$$

, and presents the property that

$$\begin{aligned} norm_{-1,1}(x; x_{min}, x_{max}) = -1 &\iff x = x_{min} \quad , \quad |x_{min}| \geq |x_{max}| \\ norm_{-1,1}(x; x_{min}, x_{max}) = 1 &\iff x = x_{max} \quad , \quad |x_{min}| \leq |x_{max}| \end{aligned} \tag{3.18}$$

, as the boundaries of the function image are not necessarily attached to both 1 and -1

This normalization probably worked because somewhere in the code, it may be assumed that

$$norm(ax; x_{max}, x_{min}) = a \cdot norm(x; x_{max}, x_{min}) \tag{3.19}$$

, a property that only $norm_{-1,1}$ satisfies.

3.7. Batching and sampling strategy

Batching is a technique used in machine learning to reduce the total amount of computation used for each training epoch by selecting only subset of the dataset to calculate the loss function. This technique not only makes possible to deal with big collections of data without dealing with the struggle of allocating memory for the calculation of derivatives of every point available, but also can be useful to prevent overfitting by preventing that the underlying model ”memorizes” the data

instead of achieving a fit by learning the patterns present in it. As a consequence, implementing a way of performing the loss calculation via data batches opens the door to use distributed computing [37], which could be useful when training the Splines for higher-dimensional datasets.

A proper sampling method to assemble the batches is also effective for dealing with imbalanced data, where some of the important phenomena under study may be only a small subset of the available data, mainly with two strategies: by undersampling majorities and/or oversampling minorities [38]. This problem was faced during the development of the present work, given that the data points density greatly varies through the dataset as explained in section 4.2, due to the flame-front occupying a small part of the domain at any given instant.

In order to tackle this problem, let's study how the derivative of the loss function in respect to the weights behaves. For that, we'll consider, without loss of generality, the SISO loss function as defined in Equation 3.13 with $\alpha_1 = 1$ and $\alpha_2 = 0$, so we do not account the behavior of the regularization

$$\mathcal{L}_{SISO}\left(\tilde{\hat{\mathcal{D}}}, \hat{\vec{\phi}}\right) = \frac{1}{\tilde{D}} \sum_{d=1}^{\tilde{D}} \left(S_K\left(\hat{x}_d; \hat{\vec{w}}, \hat{\vec{\xi}}\right) - \hat{y}_d\right)^2 \tag{3.20}$$

. We want to assess the behavior of its gradient in respect to $\hat{\vec{\phi}}$

$$\nabla_{\hat{\vec{\phi}}} \mathcal{L}_{SISO} = \left[\frac{\partial \mathcal{L}}{\partial \hat{w}_1}, \cdots, \frac{\partial \mathcal{L}}{\partial \hat{w}_I}, \frac{\partial \mathcal{L}}{\partial \hat{\xi}_1}, \cdots, \frac{\partial \mathcal{L}}{\partial \hat{\xi}_{I+K+1}}\right]^T \tag{3.21}$$

, more precisely, how the individual derivatives with respect to the weights behave. If we consider the derivative of the function only in respect to one arbitrary weight w_i we get

$$\begin{aligned} \frac{\partial \mathcal{L}_{SISO}}{\partial \hat{w}_i} &= \frac{\partial}{\partial \hat{w}_i}\left[\frac{1}{\tilde{D}} \sum_{d=1}^{\tilde{D}} \left(S_K\left(\hat{x}_d; \hat{\vec{w}}, \hat{\vec{\xi}}\right) - \hat{y}_d\right)^2\right] \\ &= \frac{1}{\tilde{D}} \sum_{d=1}^{\tilde{D}} \frac{\partial}{\partial \hat{w}_i}\left(S_K\left(\hat{x}_d; \hat{\vec{w}}, \hat{\vec{\xi}}\right) - \hat{y}_d\right)^2 \end{aligned} \tag{3.22}$$

Since the Splines present local support, a weight $\hat{w}_i$ only governs the a subset of the domain C_i

$$C_i = \bigcup_{r=i-K}^{i+K} R_r \tag{3.23}$$

, with R_i being defined as a Spline region in Equation 2.9, it follows that

$$\hat{x}_d \notin C_i \implies \frac{\partial}{\partial \hat{w}_i}\left(S_K\left(\hat{x}_d; \hat{\vec{w}}, \hat{\vec{\xi}}\right) - \hat{y}_d\right)^2 = 0 \tag{3.24}$$

Now we may assume that when $\hat{x}_d \in C_i$, we can approximate the squared error as SE

$$\frac{\partial}{\partial \hat{w}_i}\left(S_K\left(\hat{x}_d; \hat{\vec{w}}, \hat{\vec{\xi}}\right) - \hat{y}_d\right)^2 \approx \frac{\partial}{\partial \hat{w}_i} SE = \text{cte.} \quad , \quad \hat{x}_d \in C_i \tag{3.25}$$

, which is inserted into equation Equation 3.22

$$\frac{\partial \mathcal{L}_{SISO}}{\partial \hat{w}_i} \approx \frac{1}{\tilde{D}} \sum_{\substack{d=1 \\ \hat{x}_d \in C_i}}^{\tilde{D}} \frac{\partial}{\partial \hat{w}_i} SE \tag{3.26}$$

, and leads us into defining the set of data points contained within the region governed by $\hat{w}_i$

$$P_i = \{\hat{x}_d | \hat{x}_d \in C_i\} \tag{3.27}$$

, so we can express Equation 3.26 as

$$\boxed{\frac{\partial \mathcal{L}_{SISO}}{\partial \hat{w}_i} \approx \frac{1}{\tilde{D}} \frac{\partial}{\partial \hat{w}_i} SE \cdot n(P_i)} \tag{3.28}$$

, where $n(P_i)$ denotes the cardinality of the set P_i.

The result in Equation 3.28 is very important to steer the crafting of the sampling strategy, as is states that the derivative of the loss function in respect to a single weight $\hat{w}_i$ is proportional to the number of data points present in the region governed by it. As a result, the gradient descend, which is performed based on the gradient as discussed in section 2.3, will take larger steps on the parameters that govern a very populated region of the domain, while leaving the least populated regions barely untouched. During the initial stages of the project, the initial sampling strategy of randomly picking data points throughout the domain was not sufficient to diminish this problem, as the random sampling did not consider the data imbalance, so although is was a valid sampling method to form a batch in order to speed up calculations, the resulting training runs were unstable and prone to overfit.

As a possible solution for this problem, it was chosen to define the sampling strategy, here called *regional*, in a way that the resulting sets of data points P_i that lie withing a region have the same cardinality $n(P_i) = P$ for $i \in [K+1..I]$, removing the variations between the derivatives in respect to different weights in function of the number of points. The chosen strategy was to choose randomly the same amount of data points from each region and afterwards construct the sampled dataset $\hat{\tilde{D}}$ with this points. This is done by shuffling a vector containing the data points in P_i and then cyclically appending them to a new vector until the amount of desired points per region is achieved, making it possible to add repeat data points when dealing with unpopulated regions if they do not present this desired amount.

As a result, we can observe a comparison between the random and regional sampling strategies in Figure 3.3. This comparison indicates that the proposed strategy lead to a smother training, with fewer spikes in the loss function, which could mean that the difference in error for each batch varies less, indicating that the sample better represent the whole population. It's also possible

to notice that the loss function decreased faster for the random sampling without bringing the error of the model down in the same manner as the regional sampling. This may be understood as overfitting, as the loss function decreases without properly leading to a better model.

After defining this regional sampling strategy and observing that it indeed presented itself as a better approach than only sampling the points randomly, it's worth asking how the amount of points P chosen per region affect the training run, as the computational cost of training grows proportionally with this P. For that, different training settings were assembled and varying only the amount of points that were chosen to represent each region.

As seen in Figure 3.4, increasing the number of points per region lead both to a more stable training process, with less spikes in the loss function, and to a faster decrease of the error. In this figure, the normalized maximum absolute error NMAE is calculated with the formula given in Equation 4.1, but using the normalized Spline against the normalized datset. During the training performed in chapter 4, the number of points chosen for each region are in the order of 100, in an effort to achieve a sufficiently stable training process, while still keeping the computations not prohibitively expensive.

The sudden increases both in the error and in the loss functions in Figure 3.3 are explained due to the high learning rate of $\alpha_{opt} = 0.05$ used during these runs. As discussed in section 2.3, the Adam method implements momentum for the parameters, making it possible to further explore the space of possible parameters without getting stuck in the first found local minimum. As the parameters continue moving beyond this first local minimum, the values for the loss function and the error are expected to increase by a short number of iterations, until they find another way to a local minimum.

3.8. r-refinement

r-refinement is a technique for improving the precision of numerical solutions of partial differential equations by reducing the error associated with space discretization [39]. It consists in performing relocations of the mesh nodes based in some criteria that operates without user intervention and leads to a grid that calculate the solution more precisely. Still according to McRae[39], this improvement that blossomed from the refinement should be quantifiable. Since we can expresses the Spline with its parameters, including the node positions, explicitly defined and we are also able to take the gradient of the loss function $\mathcal{L}$ not only in respect to the weights $\vec{w}$ but also in respect to these node positions $\vec{\xi}$, r-refinement can be implemented by having them to undergo the same process of having their values determined by gradient descend that the weights also go.

In the present case, the implementation of r-refinement consists in deriving the optimization loss function also in respect to the nodes, and take steps to bring them in the direction that the gradient descend indicates to minimize it. More precisely for the SISO case, this refinement strategy is achieved by concatenating the nodes vector $\vec{\xi}$ inside the parameters vector $\vec{\phi} \in \mathbb{R}^{2I+K+1}$

$$\vec{\phi} = \left[\vec{w}^T, \vec{\xi}^T\right]^T \tag{3.29}$$

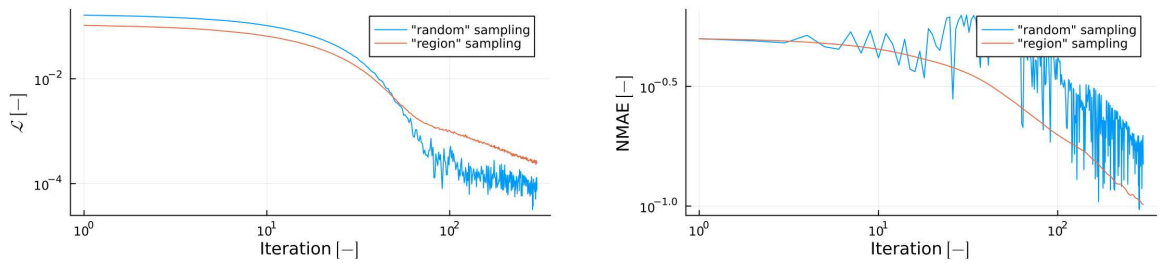


Fig. 3.3.: Evolution of the loss function and normalized error along the training epochs for different sampling approaches. Order $K = 5$, number of regions $n_{reg} = 9$ and learning rate $\alpha_{opt} = 0.05$.

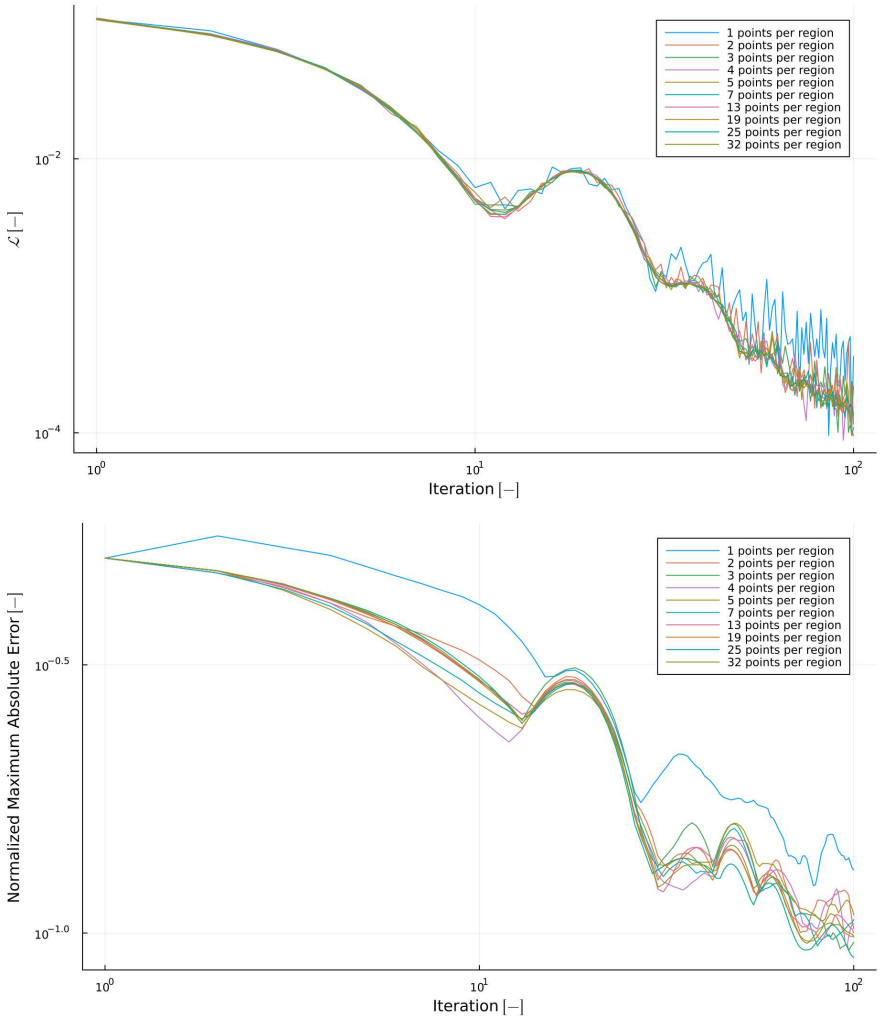


Fig. 3.4.: Evolution of the loss function and normalized error along the training epochs for different batch sizes. Order $K = 5$, number of regions $n_{reg} = 9$ and learning rate $\alpha_{opt} = 0.05$.

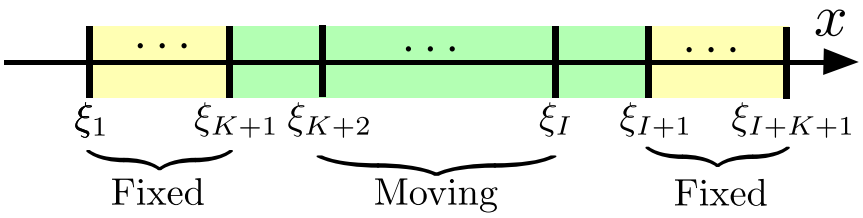


Fig. 3.5.: Fixed and moving nodes during r-refinement procedure represented on the domain.

. Afterwards, the loss function $\mathcal{L}$ has its gradient in respect to ϕ taken, and the value of the parameters will be updated according to a optimization algorithm, as discussed in section 2.3. Notwithstanding, relying on the same learning ratio α_{opt} to lead the steps of both quantities can lead to instabilities. Having the nodes to change mean, for the Spline, a very different kind of change than the weights. The distance between nodes appear in the denominator of the Spline basis, meaning that the derivative is influenced by this distance, not properly by the values of the nodes themselves. When nodes ξ_i and ξ_{i+1} get closer to each other, the slope of the Spline changes sharply in the region R_i enclosed by them; as observed, sharper than the resulting change in slope that happens when moving the weights using the same learning hate.

To solve this situation, another *ad-hoc* parameter is used to provide relaxation for the steps taken in the space that the nodes live in. A coefficient α_r is inserted inside the gradient of the loss function, that leads us to calculate the gradient $\nabla_{\hat{\phi}}\mathcal{L}_{SISO}$ as

$$\nabla_{\hat{\phi}}\mathcal{L}_{SISO} = \left[\frac{\partial\mathcal{L}}{\partial\hat{w}_1}, \cdots, \frac{\partial\mathcal{L}}{\partial\hat{w}_I}, \alpha_r\frac{\partial\mathcal{L}}{\partial\hat{\xi}_1}, \cdots, \alpha_r\frac{\partial\mathcal{L}}{\partial\hat{\xi}_{I+K+1}}\right]^T \tag{3.30}$$

. The values that α_r may assume are intended to be inside $[0, 1]$, since we want the steps taken by the nodes to be smaller than the taken by the weights, although nothing impedes any real value to be used.

The last problem faced was that the nodes sitting outside the domain were moving inwards the domain, which creates unauthorized Spline regions inside the data domain. As seen in subsection 2.1.2, these kind of partially controlled regions are chosen to be unsuitable for representing inputs inside $[x_{min}, x_{max}]$, since they don't have enough weights to uniquely define a order K polynomial, namely $K+1$ values. To avoid this situation, it was tried in the SISO case to insert new nodes outside of the domain whenever some region of the domain became partially or unauthorized, but this method did not show good stability and was ditched in favor of fixing the position of the outside nodes. By not allowing these very important nodes to move, as illustrated on Figure 3.5, the problem is avoided and the training runs go more smoothly.

3.9. h-refinement

h-refinement is a technique used to enhance the potential of the domain discretization to yield a solution with better precision by modifying the topology of the domain by inserting new divisions in the discretization, here represented by the nodes $\vec{\xi}$. The problem of implementing this type of refinement for Splines was faced in three different fronts, those being finding a criteria to choose which regions should be marked to be refined, with a learning rate α_h to control the amount of new nodes that are going to be inserted in every refinement iteration; deciding where in the regions marked for refinement the nodes should be placed; and finally interpolating the value of the new weight to be inserted alongside the node.

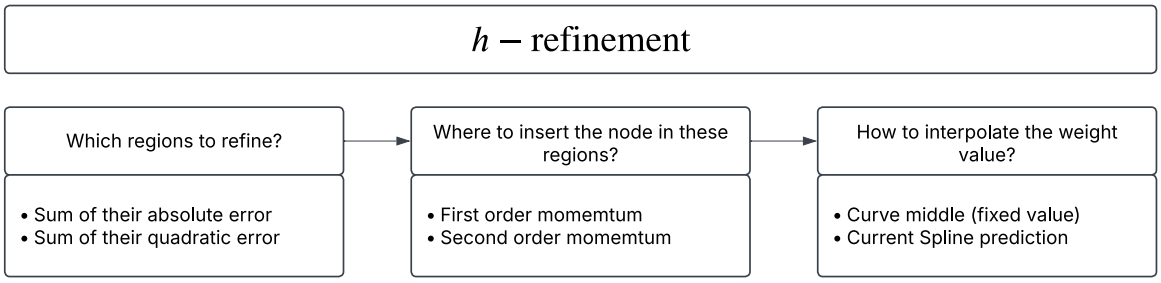


Fig. 3.6.: Schema of the h-refinement steps and the criteria used for each.

To make sense of which regions would be the most promising to receive a new node, it was assumed that if a Spline doesn't have enough resolution in a region to get close to the curve, sub-dividing this region would lower the amount of "shape information" that each of the newly created regions now have to represent. With that in mind, it was reasonable to assume that the region that presented the highest absolute error would be the most benefited by this further subdivision. In that same direction, the regions in the neighborhood of the one containing the new node are also benefited by the its insertion, due to the amount of different regions that the weight controls beside the one that was subject to refinement.

In order to rank which regions should first be refined, let's define the error E_r for a given region R_r to be

$$E_r = \frac{1}{M}\sum_{m=1}^{M}\frac{1}{n(ID_{r,m})}\sum_{d\in ID_{r,m}}\left|S_K\left(\hat{x}_{d,m};\hat{\vec{w}}_m,\hat{\vec{\xi}}\right)-\hat{y}_{d,m}\right| \tag{3.31}$$

, and the squared error E_i^2 for the same region as

$$E_r^2 = \frac{1}{M}\sum_{m=1}^{M}\frac{1}{n(ID_{r,m})}\sum_{d\in ID_{r,m}}\left[S_K\left(\hat{x}_{d,m};\hat{\vec{w}}_m,\hat{\vec{\xi}}\right)-\hat{y}_{d,m}\right]^2 \tag{3.32}$$

, where $ID_{r,m}$ is the set of all indexes d that provide a $\hat{x}_{d,m}$ within the region R_i

$$ID_{r,m} = \{d | \hat{x}_{d,m} \in R_r\} \tag{3.33}$$

. This definition allows us to assess how far apart from representing the curve the Spline is in the given region while accounting for the differences in the region's population.

It's worth to point out that during h-refinement, the errors are calculated using the whole dataset, without any sampling strategies, as the error is not prone to be disturbed by the data imbalance, due to the summation of errors being divided by the number of data points that lie within. As a matter of fact, those strategies could be implemented also during the refinement process to diminish the computational costs as the number of points grow exponentially with the number of input control variables N, where calculating the error with the whole population could become prohibitively expensive. Although, while dealing with the data generated by the methane flame, and using only a subset of the data that represents the regions of the domain that are not affected by heat losses, it was possible not to use batching for calculating the rankings of the regions.

A learning rate for the h-refinement α_h is introduced to make it possible to define how many new nodes are inserted in each refinement step. This number of new nodes n_{new} is given by

$$n_{new} = \lceil \alpha_h \cdot n_{reg} \rceil \tag{3.34}$$

After defining two different approaches for ranking the regions, it's now time to define approaches to insert a new node inside these regions according to a criteria that specifies where this new node will be inserted. The first approach used was to insert the new node in the exact middle of the selected regions when dealing with the SISO case, but later on two more approaches were developed to perform the insertion based in the moments of the error quantification, namely the first and second moments, which are calculated based on the representation shown in Figure 3.7.

Using the first order moment of the error, a new node $\xi_{r_{FOM}}$ inside a region R_r is going to be placed at

$$\xi_{r_{FOM}} = \frac{\sum_{m=1}^{M} \sum_{d \in ID_{r,m}} \overbrace{A_{d,m}}^{\text{Area}} \cdot \overbrace{PO_{d,m}}^{\text{Position}}}{\sum_{m=1}^{M} \sum_{d \in ID_{r,m}} A_{d,m}} \tag{3.35}$$

, where the area $A_{d,m}$ that represents the magnitude of the error for a given data point is defined as

$$A_{d,m} = \left| S_K \left(\hat{x}_{d,m}; \vec{\hat{w}}_m, \hat{\vec{\xi}} \right) - \hat{y}_{d,m} \right| \cdot (\hat{x}_{d+1,m} - \hat{x}_{d,m}) \tag{3.36}$$

, and the position $PO_{d,m}$ (as P is already taken) is defined as

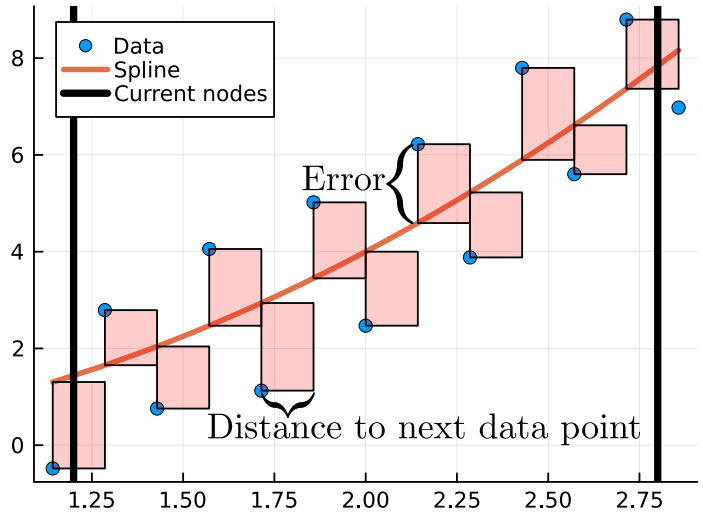


Fig. 3.7.: Illustration of how the error areas between a Spline and the data are defined.

$$PO_{d,m} = \frac{\hat{x}_{d,m} + \hat{x}_{d+1,m}}{2} \tag{3.37}$$

The criteria to insert the new node based on the second order momentum of the error of the new node is to insert it at the position

$$\xi_{r_{SOM}} = \frac{\sum_{m=1}^{M} \sum_{d \in ID_{r,m}} A_{d,m}^2 \cdot PO_{d,m}}{\sum_{m=1}^{M} \sum_{d \in ID_{r,m}} A_{d,m}^2} \tag{3.38}$$

More precise approaches could be used to calculate this same criteria, for example, using Gaussian quadrature, which would yield exact results in this scenario, given that the Spline is a polynomial within two distinct nodes. Here, this method underwent some development, but was left aside for the sake of simplicity.

3.10. Computational budget

This sections brings a brief discussion about the memory requirements of storing a Spline and the expected growth in complexity that it's expected when elevating the number of weights or the degree. When dealing with Splines, there are two main approaches to increase the complexity of the curves that it can represent, namely increasing the number of weights, which increases the amount of memory that must be allocated to store them, and also by elevating the degree of the Spline, which allows it to represent more nuanced curves for the price of increasing the evaluation time, as can be seen in Figure 3.8. Both approaches are going to be assessed here.

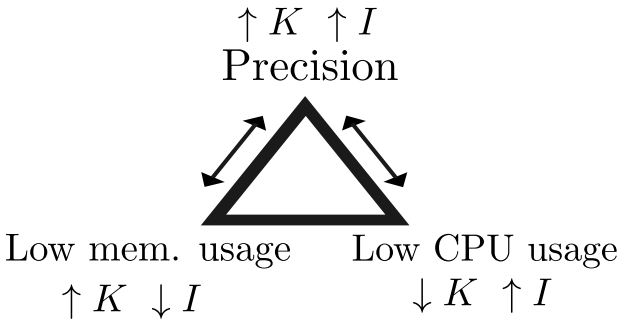


Fig. 3.8.: Different forms of achieving precision when using Splines: elevating the number of weights or elevating the degree.

We can assess the amount of memory expected to be used to store a structure containing information to define a Spline by counting how many floating point numbers must be allocated, as each of the allocations is of constant and known size. During this work, the quantities used are mainly 64-bit floating point numbers, this was chosen as a sufficient precise initial guess, as it would mean trouble to incur firstly in issues related to floating point operations truncation before hitting a precision limit with the Splines. A 64-bit representation should be more than enough to represent the quantities dealt with. For a SISO Spline $S_K : \mathbb{R} \to \mathbb{R}$, the number a_{SISO} of floating point quantities allocated is

$$a_{SISO} = \underbrace{I}_{\text{weights}} + \underbrace{I + K + 1}_{\text{nodes}} \tag{3.39}$$

. The degree K of the Spline is a integer value, and its allocation has a negligible memory cost, thus it will be omitted. SISO is the only scenario where the nodes occupy more memory space than the weights.

In the case of the SIMO Spline $\underline{S}_K : \mathbb{R} \to \mathbb{R}^M$, the number of allocated weights grows linearly with the number of outputs M. It is a case way cheaper to be evaluated than the generalization for multiple inputs, as the number of weights do not grow exponentially.

$$a_{SIMO} = \underbrace{I \cdot M}_{\text{weights}} + \underbrace{I + K + 1}_{\text{nodes}} \tag{3.40}$$

The last case here to be discussed is the MIMO Spline $\vec{\underline{S}}_K : \mathbb{R}^N \to \mathbb{R}^M$, which consists of the most expensive case both in memory allocation and in processing time for evaluation. This is mainly due to the rapidly growing size of the tensor that contains the weights. The number of allocations for this case is given by

$$a_{MIMO} = \underbrace{M \cdot \prod_{n=1}^{N} I_n}_{\text{weights}} + \underbrace{\sum_{n=1}^{N} I_n + K + 1}_{\text{nodes}} \tag{3.41}$$

. So it can be seen that in this scenario, the memory requirements to store a Spline grows with the same complexity as a common multidimensional table does. At the first glance, this could automatically lead one into thinking that using Splines for tabulation would have no meaningful advantage over conventional tables, but as the degree of the Spline K increases, the more accurate is the result at the end [21].

The main task of the current work was trying to find ways of adjusting the Spline against chemistry manifolds in a form that the phenomena that happen in a flame is captured and represented in as few as possible weights per dimension, as performing h-refinement in higher dimensions is prone to insert a significant amount of weights each time it's performed. As for example, inserting a new entry in a matrix necessarily means adding another row or another column, never a single value, a behavior that is exacerbated in higher dimensions.

The trade-off that presents itself now is between precision, CPU time and memory consumption, being memory consumption the current bottleneck when representing combustion that take account a variety of phenomena. With low requirements of precision, a very coarse Spline can be fit without demanding many values to represent it, and also won't need to have high-order representations, saving computational time. As the requirements for precision become more strict, more information may be stored to generate the representation, leading to increased memory consumption. To mitigate this memory consumption, here we adopt a higher order representation that may be able to capture the shape of the chemistry manifold while using less allocated memory.

Regarding the CPU time for each evaluation, the numbers may vary, mainly because of the restriction used here that the function implemented must be suitable to undergo automatic differentiation. Consequently, a sub-optimal version was implemented to be used during training that satisfies the criteria for Auto-diff, and may be only used in such circumstance, as there is no advantage in using the sub-optimal formulations when evaluating the Spline during a coupled CFD simulation to retrieve thermochemical quantities. The intention is to obtain the weights and node positions during the training process, that may be slow and time consuming due to the used formulation, and afterwards using the resulting parameters for evaluation with fast specialized implementations that are not subject to the same restrictions, such as the FORTRAN library developed by De Boor [20].

4. Results

To assess the results of the proposed method to fit Splines, firstly it will be considered a simple case, where known functions are sampled and a SISO Spline is fitted with and without refinements in order to check whether it's such techniques show themselves feasible for the job. This part is mainly done by visually inspecting the fitting process, and assuring that at every training step, the Spline better represents the curve in this qualitative criteria. Afterwards, comes the main corpus of results, where the techniques are applied to generate Spline-based manifolds, that will be also visually compared to the data used to generate them, as well as evaluated using a relative error RE defined as

$$RE = \frac{y_{tab} - y_{true}}{y_{max} - y_{min}} \tag{4.1}$$

, and which will be used to measure the difference between the Spline and the FGM it tries to represent, as well as to evaluate the error in a *a priori* analysis where the comparison is graphed into physical space, and the resulting flame is also quantitatively analyzed. When refereed as normalized maximum absolute error (NMAE), the error is calculated using the formula above, but using the normalized quantities. To tabulate the results of the study, the maximum relative error will be analyzed between the different methods proposed.

4.1. Single-input single-output with r- and h-refinement

Before tackling the more complex cases, it's reasonable to assess the simplest one. The chosen case to be done as a proof of concept was to fit a Spline using both kinds of refinement, namely h- and r-refinement against two different datasets. This simple case makes it possible for us to assess the suitability of the strategies that are later going to be used in the single-input single output case. Most of the decisions taken in the implementation were only preliminary, such as doing h-refinement by inserting the new node in the exact middle of the region marked for refinement, instead of refining with some criteria based on the error distribution within the region. This initial approach also served to determine whether it's plausible to also make the nodes vector undergo r-refinement, and the possible problems that can arise when doing this, i.e. regions becoming so small that they become unpopulated, here meaning that there are sub-datasets $\mathcal{D}_i = \varnothing$ that causes problems regarding control of the weight values by the optimizer.

In Figure 4.1 we can observe two different moments in the training of a single-input single-output Spline against a cosine curve $f(x) = \cos(x)$. The first figure shows the Spline after some few training steps, in a situation that it doesn't have enough weights to be able to properly capture the shape of the functions in all regions, a situation that often leads to a Spline stuck in a position roughly at the middle of the function that allows it to minimize the MSE even when being defined with fewer weights than needed [1]. In the second image we can observe how the nodes were successfully positioned in places suitable to capture the oscillating character of the cosine function, and the weights were also moved to values that, at least to the view, seem to suit reasonably well the targeted curve.

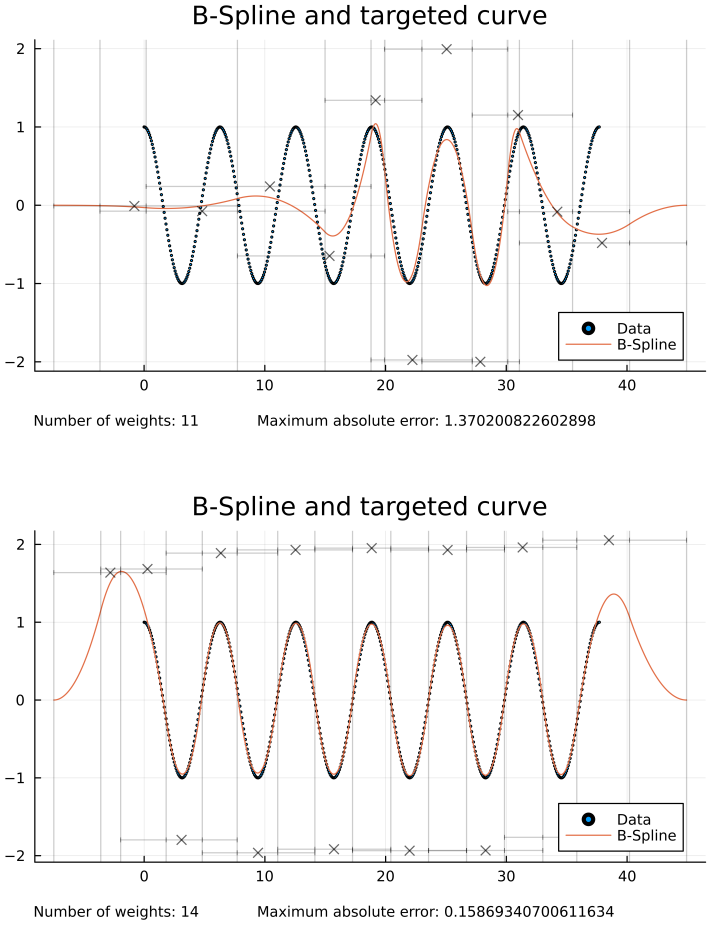


Fig. 4.1.: Two different stages of the single-input single-output Spline fitting with r- and h-refinement with $K = 2$.

After fitting the Spline for the simple cosine curve, a exponentially separated cosine $f(x) = 10^{-x}\cos(10^x) + 1$ was used to assess whether the first chosen criteria for r- and h-refinement were

[1] The number of weights needed to represent a given signal can be approximated using the Nyquist-Shannon theorem [21]

suitable for problems involving scale separation.

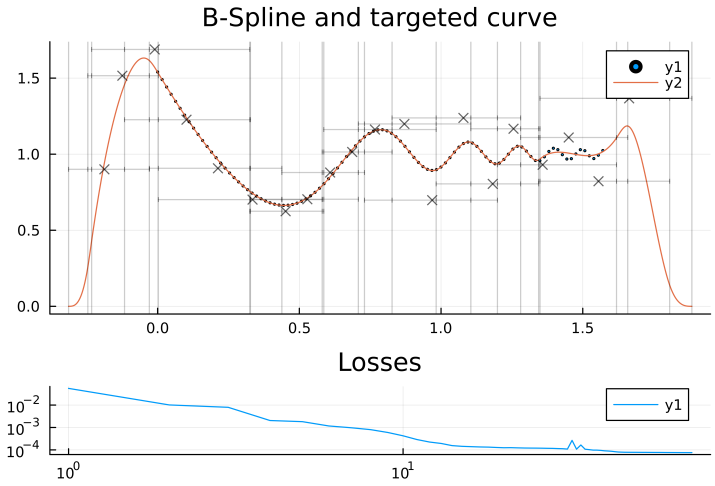


Fig. 4.2.: Exponentially separated cosine fitting.

At this early stage, it was noticed that r-refinement with Splines can be very unstable. The majority of training sessions were finished because the process of gradient descent placed two nodes too close to each other, leaving the region enclosed between them with no data points. This is problematic, because the gradient descend process becomes blind to such regions as the derivative of the loss function inside the region is no more calculated. This leads to a weight that has no criteria to be optimized, and therefore is stuck in its current position.

To deal with this problem, two approaches were used. The first was to halt the training as soon as the gradient descent becomes blind for any region - used in the current case already-, and the second one was to implement a relaxation parameter for the r-refinement, so the gradient descent goes slower for the nodes than it does for the weights - which was discussed in section 3.8.

4.2. 1D head-on quenching dataset and training assumptions

Before showing the results for the single-input multiple output Spline, it is worth to examine how the dataset used for the following cases was generated, and the assumptions that where used in presence of the current data. The Spline generated here is based on a one-dimensional laminar methane-air flame that propagates freely towards a wall, where heat-losses takes place, causing the flame to undergo quenching and extinguish. It comes from a study performed by Steinhausen in 2020 [18], where the flame-wall interactions are studied, in this case by solving a one-dimensional detailed chemistry simulation, which is validated by comparing the results with the ones gotten from a burner by comparing the shape of the distribution of the Heat-Release Rate. This data is then used to assemble three different chemistry tabulation methods: Flamelet-Generated Manifolds (FGM), Quenching Flamelet-Generated Manifolds (QFM) and Reaction-Diffusion Manifolds (REDIM); these last two being able to capture the interactions under study.

Methane is a very important fuel as it is the main component of natural gas, corresponding to at least 85% of it's composition in volume [40]. It is the simplest hydrocarbon, of formula CH_4 and presents a higher LHV than other more complex hydrocarbons, namely 47.79 [kJ/g], thus giving it the technically interesting property of releasing less CO_2 mass when burnt per unity of energy released compared to other carbon-based fuels.

Even tough natural gas currently being obtained from the large reserves available in different countries [41], it is not a reasonable activity to expand the exploration of such reserves, as the release of more CO_2 in the atmosphere enhance climate change. Whereas, expanding green energy production may generate new jobs in the effort of developing the infrastructure needed for the energy transition, and mitigating poverty among the population is closely related to expanding access to energy [42]. One main approach to produce Synthetic Natural Gas, which methane is the most prominent component, is through the Sabatier reaction, where CO_2 and H_2 are combined to form methane and water in the exothermic reaction

$$4H_2 + CO_2 \longrightarrow CH_4 + 2H_2O \tag{4.2}$$

, where the H_2 can be produced through alkaline electrolysis by using energy available from a given source, most commonly from the power grid [43]. Another process to produce methane is to use anaerobic digestion to process food waste and vegetables, presenting a viable option to disposal or incineration, as the first can release methane generated from microbial digestion in atmosphere, while the second one releases the carbon contained within the waste in atmosphere [44]. Releasing methane in the atmosphere is specially problematic, as it presents a Global Warming Potential of 28, which means that per kilo it can cause this times much the increase in global temperature when compared to carbon dioxide [45]. Methane is also a promising alternative to power the vehicle fleet, that is already being considered in Latin America and Asia [46].

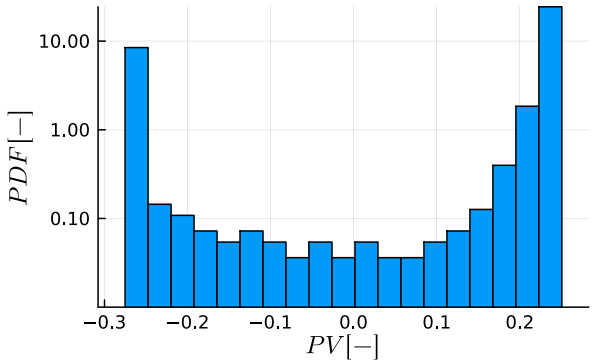


Fig. 4.3.: Histogram of the probability density function for the data points distribution. Logarithmic y-axis, 20 bins.

The techniques used here to fit Splines against data are very adjusted to the dataset used for the present study. This present dataset consists of the thermochemical data generated by a freely propagating methane-air flame, which is going to be used to create a manifold parametrized by the

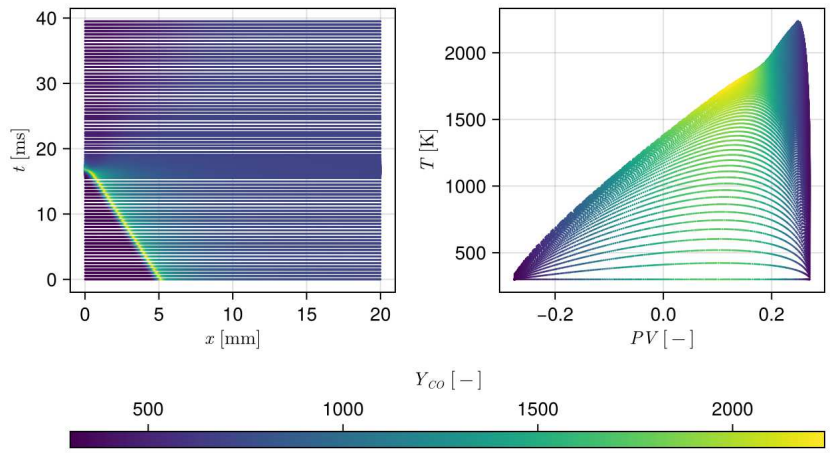


Fig. 4.4.: Head-on quenching flame in physical space versus time (left) and in PV space versus temperature (right), both colored by CO mass fraction. (Reproduction from Bissantz 2023 [11])

progress variable $Y_{PV} = -0.5Y_{O_2} - 0.5Y_{CH_4} + 0.5Y_{H_2O} + 0.51Y_{CO_2}$, which was found by Bissantz and Karpowski in 2023 [11] to be the first principal component that describes the variance observed present in the output variables φ in the methane-air flame undergoing head-on quenching.

In Figure 4.5, it's possible to see how the dataset is distributed, where all of the quantities to be tabulated are displayed. As we can see in this graph, the distribution of the quantities varies widely along the ordinate, with quantities in the order of 10^{-5}, as the dynamic viscosity μ, up to the order of 10^{9}, as the the temperature source term. Although all quantities during the training procedure being normalized, the resulting Spline is denormalized, so the outputs are directly the desired variables, and the calculations to denormalize the outputs do not have to be performed at every data retrieval. Additionally, in Figure 4.4, we can see the flame propagating in space as time advances, until it's extinguished in the wall's vicinity. The study is going to be performed using the data of the first time step, as a highlighted in Figure 4.5, with a dataset containing 2000 for each of the $M = 6$ variables to be represented. In this data section, the temperature diffusion caused by mass diffusion D_{diff} is always equal to zero, so it won't be included.

With the present dataset, there is no noise and no outliers present in the data. So along this work, some possibly problematic situations were not faced, and thus not dealt with, as the Spline parameters that are searched for in this work are the ones that fits the data assuming that every data point is equally trustworthy. The performance of the algorithm developed here was not assessed for noisy datasets, tough it may work, as the fitting procedure involves minimizing a mean squared error, which is broadly used with ANN and provides good results when dealing with real world data containing noise.

Another assumption made when defining the formulations for the Spline to be fit is that a single vector of nodes is capable of describing how every output dimension should be discretized. This assumption is expected to hold true, due to the region in PV space where the combustion happens

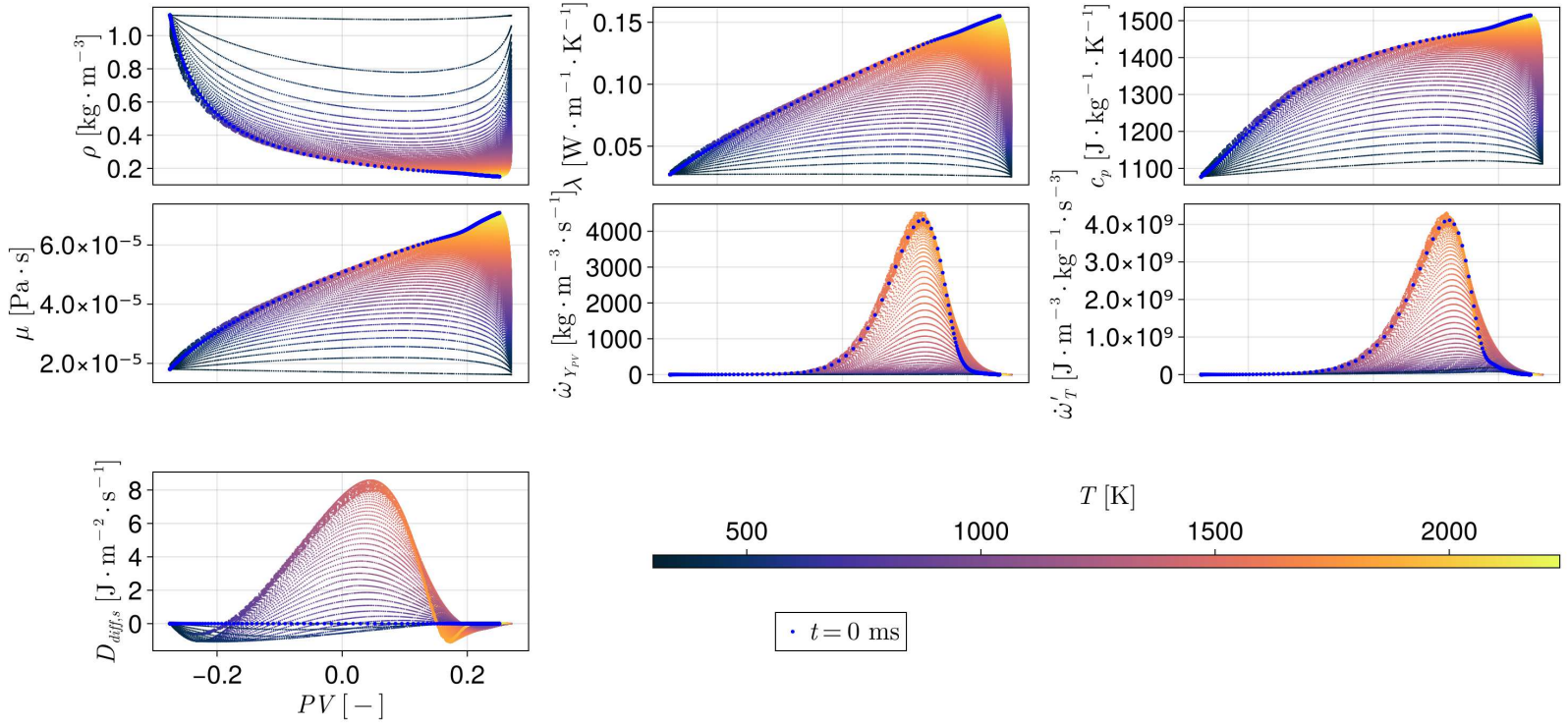


Fig. 4.5.: Scatter plot of the methane-air flame simulation dataset with a highlight in $t = 0$. (Data from Steinhausen 2021 [18])

being the same for every output curve, as they are thermochemical information. The fact that the most intricate parts of the curves inside φ for each output dimension lie roughly within the same region of the domain where the flame occurs is useful when performing refinement, as inserting a node in this region will benefit all the curves.

Besides the problems that were not dealt with, other problems have to be tackled, such as data unbalance. The form that the data was generated, via a numerical simulation of the flame, leads to points being unevenly distributed in the domain, as there are many points that captures the state of the mixture in regions of the simulation domain where not much is happening. This unbalance can be seen in the probability density function for PV in Figure 4.3. Here it was assumed that one may want to directly plug in the data resulted from a simulation into the Spline fitting algorithm without pre-processing it (assuming that the data doesn't contain noise), so a sampling method was developed in section 3.7 to ensure that the training process would remain stable even in the presence of such imbalances.

It's also not assumed that the curves provided for each output are given already in the form of a single-input multiple-output function. Here it is fact that all the curves are parametrized for all output quantities by the same points in the PV space. Nevertheless this was not assumed during development, so as the most remote example, one could use this algorithm to fit a function

$$f(x) \approx [f_1(x), f_2(x)] \tag{4.3}$$

as if both outputs where functions of the exact same domain, and most importantly having their points in the training dataset given as a function of the same $\vec{x}$, where in reality they are a function of different inputs

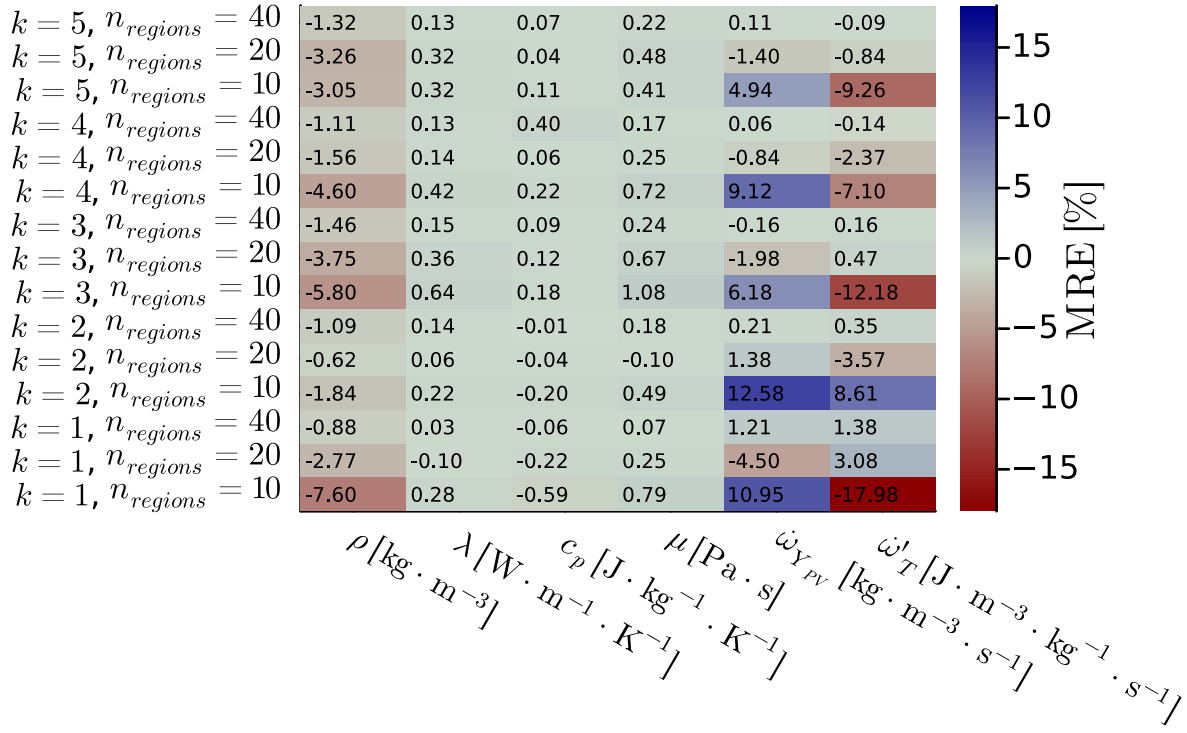
$$\begin{aligned} f_1(x_1) &= y_1 \\ f_2(x_2) &= y_2 \end{aligned} \tag{4.4}$$

, as long as the domain range is similar enough. This simplification is not made in order to accommodate datasets that do not present all of their entries calculate with the exact same values for the control variables ψ, although it causes the drawback of M times more evaluations of the Spline when evaluating the loss function.

4.3. Single-input multiple-output without refinements

During this phase, as discussed in the section 3.3, the nodes inside the controlled region were set to be equidistant, and the node positions remain the same during the whole training process.

The results for the single-input multiple-output Spline without refinements can be seen in the Table 4.1. It can be seen that the information that the Spline can represent after a given number of training iterations with the same learning rate grows faster with the number of regions inside the domain than it does for the value of K. This training run was performed with 1000 iterations with



Tab. 4.1.: Relative error achieved by SIMO Splines with different degree K and number of regions $n_{regions}$ after 1000 iterations with a optimizer learning ratio of $\alpha_{opt} = 1 \cdot 10^{-2}$. No refinements.

a learning ratio of $\alpha_{opt} = 0.025$. To train all the 16 Splines used in this error calculation, roughly 12 hours of computation where used, taking advantage of parallelization.

We can infer from the graph Figure 4.6 that the Splines with higher degrees tested had more benefits than those with lower degrees for extended training runs. This is one of the results shown experimentally by Briand in 2018 [21]. Splines with higher degree present less locality, in the sense that the overlap of the sets of weights that influence each region grows with K, which leads to the trade-off between being able to represent more intricate functions by having a higher degree for each piece of polynomial, and the fact that the weights that give shape to regions, must now capture the characteristics of more regions at the same time.

Even with that trade-off, it's inferrable by the graph that indeed there was a increase in precision by increasing the order. This now brings us to the trade-offs that appear in the computational context: in one hand, by elevating the order of the representation of the manifold, we are able to get closer to the targeted curve with fewer regions; but in the other hand, these higher order representations become more costly. A naive way here proposed for dealing with this trade-off is to determine the acceptable delay for data retrieval from the Spline-based manifold and compare it with the known execution times for each Spline degree, and choose the Spline degree based on the maximum acceptable time for data retrieval. This is thought to be a good strategy, as we

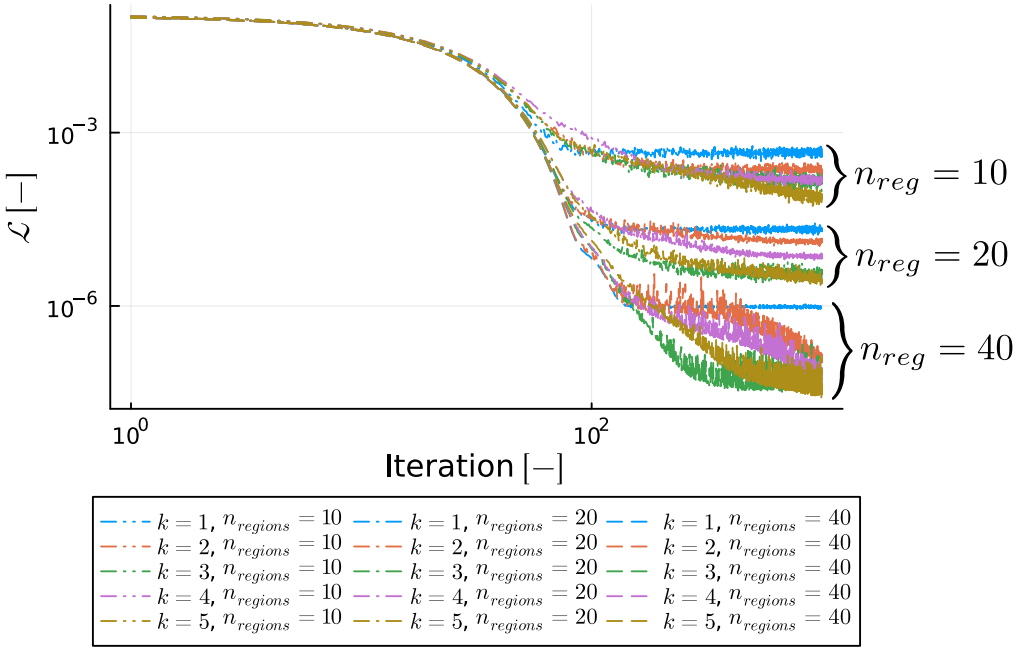


Fig. 4.6.: Loss function evolution for multiple Splines with variate K and n_{reg} after 1000 iterations. No refinements.

are trading between CPU time and allocated memory, being the allocated memory the current bottleneck to represent higher-dimensional look up tables for thermochemical data.

From the *a priori* analysis, a possible new way of informing the optimization process with physics: the analysis of the flame structure in the physical domain could also be included in the loss function. This is a possible approach to better fine tune the loss function that undergoes gradient descend, as we can be able to better quantify in a function *what a good model is?* For example, during the development of this work, it was observed that the density was presenting its maximum error in a very small region when parametrized by PV, but this error caused an offset in the density in the burnt mixture region in physical space.

From all the Splines trained in this section, the one with $K = 4$ and $n_{reg} = 20$ was chosen arbitrarily to undergo the analysis. The results of the fit directly against the thermochemical manifold can be seen in Figure 4.7. At a first glance, the curve looks fit to to the eye. A more in depth quantitative look may notice that the density, which presents the maximum relative error spikes in the flame front, and source terms for temperature and progress variable oscillate their error in the flame, as seen in Figure 4.8. This could be expected, as this regions contain more complex reactions happening, thus making them more difficult to capture. Also can be seen that the strategy of equidistantly placing the nodes is not the most efficient, as regions that remain barely constant in PV space get the same discretization as the regions near the flame front. For the present case, the maximum relative error presented by the Spline was an undershoot of 2.37% in the temperature

source term in the flame region.

4.4. Single-input multiple-output with r-refinement

This section brings a short stability study for the r-refinement relaxation coefficient α_r. As mentioned in section 3.8, when using this kind of refinement, it's advantageous to insert this coefficient in order to decelerate the movement of the nodes and avoid generating unpopulated regions. Different values for the coefficient and the number of iterations performed before such a region is generated can be seen in Table 4.2, which indicates that using $\alpha_r \leq 10^{-6}$ may be a reasonable choice of value, since the training didn't come to a halt because of the mentioned problem when such values were used.

α_r	Number of iterations before exit
10^{0}	18
10^{-1}	13
10^{-5}	25
10^{-6}	> 100
10^{-7}	> 100

Tab. 4.2.: Stability study for r-refinement relaxation coefficient α_r using emergence of unpopulated regions as stop criteria. $K = 3$, $n_{reg} = 10$, $\alpha_{opt} = 0.025$.

As a complement, when training the Spline with both r- and h-refinement present, which will be discussed in section 4.6, this coefficient was set to $\alpha_r = 10^{-8}$, and 3000 iterations could be achieved.

4.5. Single-input multiple-output with h-refinement

When training Splines with h-refinement, it was here chosen to begin with a single region inside the domain, so the capability of the criteria to choose good coordinates for the new nodes could be assessed in situations that the Spline is still not a good fit for the curve (as happens in the beginning of the process), and in situations that the error is not large and the Spline already sits relatively close to the curve.

For all the Splines trained, a single new node was introduced every 200 iterations, and at the end an additional of 2000 iterations were performed after the last node insertion in an effort to avoid returning recently refined Splines, as their error spikes right after refinement. This strategy causes Splines with more regions to undergo more training iterations as the number of regions in the end goes up.

h-refinement also presented a similar problem to r-refinement, as it constantly tried to refine small regions with already few data points, resulting in unpopulated and thus uncontrolled regions. This

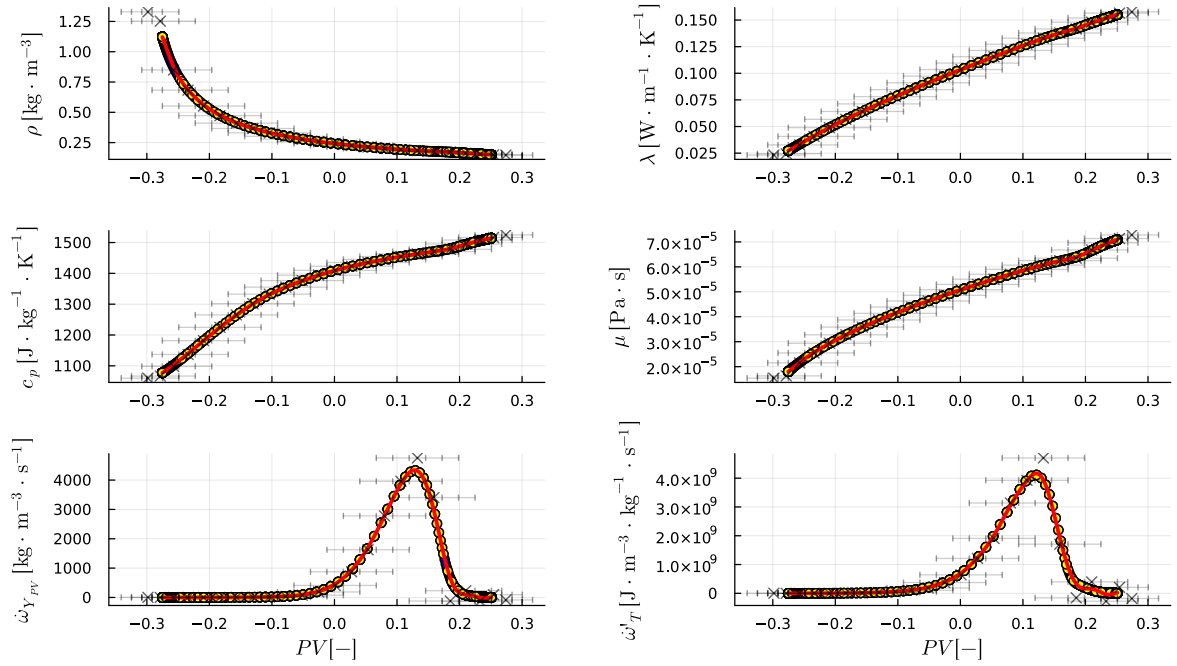


Fig. 4.7.: Comparison between the original data for φ (discrete blue and yellow dots) and the Spline representation achieved to tabulate it (continuous red lines, with weights and regions). No refinements. $K = 4$, $n_{reg} = 20$

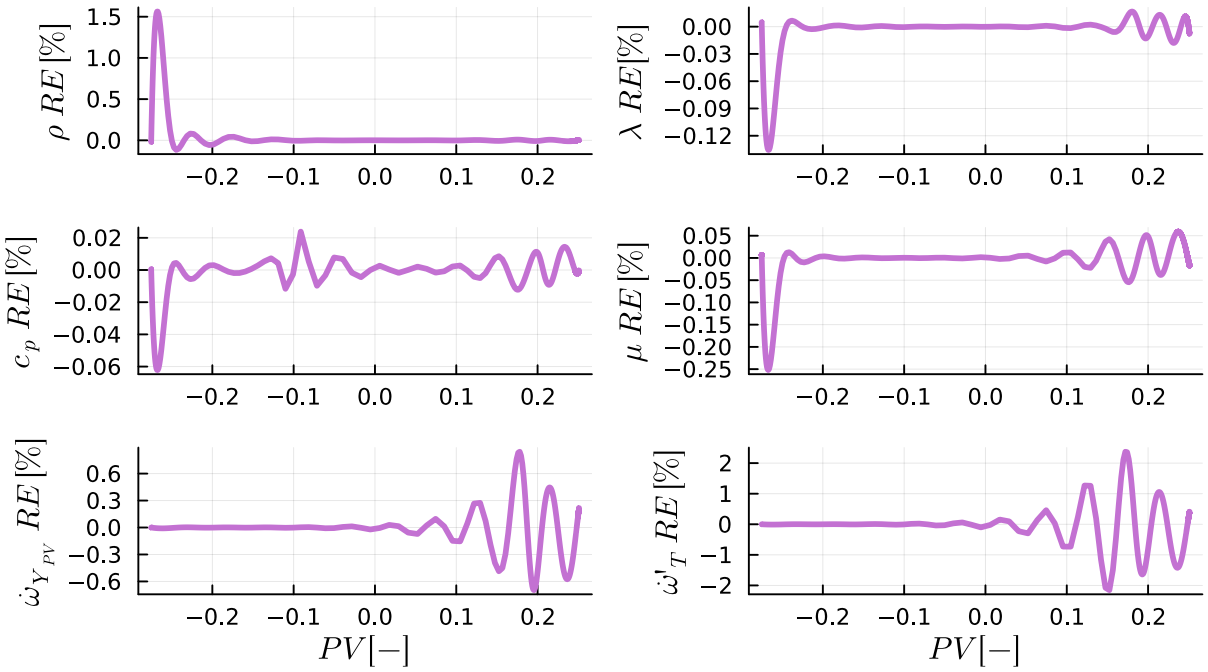


Fig. 4.8.: Relative error in PV space for the SIMO Spline. No refinements. $K = 4$, $n_{reg} = 20$.

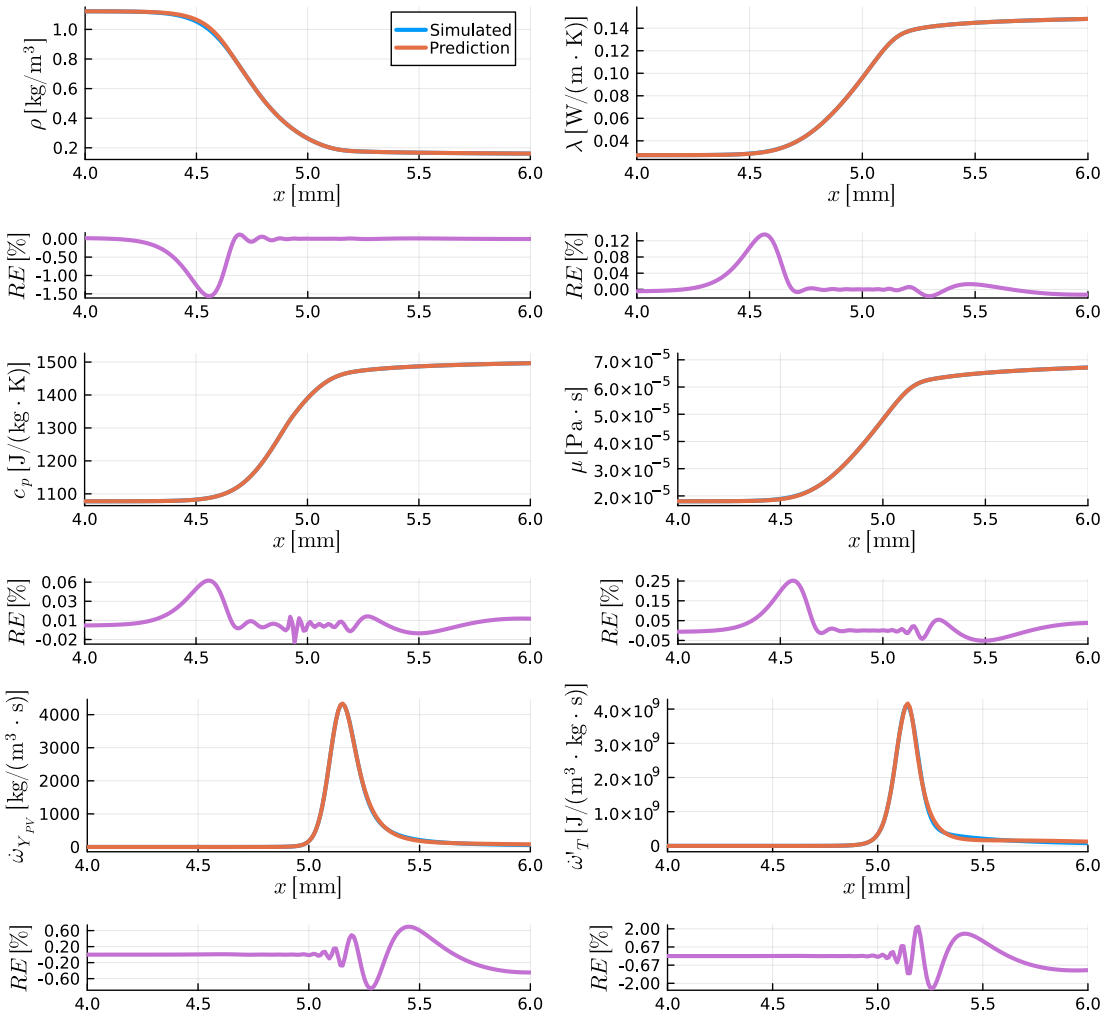


Fig. 4.9.: Flame structure in physical generated by a Spline of order $K = 4$ and $n_{reg} = 20$ fitted without refinements.

regions happen to sit on the flame, where the phenomena that one may want to capture takes place. As these regions have more intricate shapes, but happen at the same time to present a lesser resolution in PV space, they often are classified as the most suitable for training following the criteria stipulated on section 3.9. To avoid that, a minimum number of points that a region must have in order to be eligible for refinement was defined *ad-hoc* as 10. As a result, the maximum resolution that a Spline can achieve in any given portion of the domain is limited by the dataset provided, and methods to determine this maximum resolution may be developed.

The results for four different combinations of refinement criteria are shown in Table 4.4. The highlighted relative errors stand for the criteria combination that was best able to capture the output quantity behavior. As a comparison with the Spline analyzed in section 4.3, the h-refinement methods were able to use the same 20 regions more efficiently, bringing the maximum relative error to a overshoot of 1.15% in the density, less than half when compared with the one achieved without refinement. As for the temperature source term, this method was able to bring the error from 2.43% to a undershoot of 0.04%, a 60 times improvement.

In Table 4.3, we can see which criteria performed the best to achieve the lowest maximum relative error for each final number of regions, and the quantity that presented the biggest error. We can notice that for all cases, the first order moment was the most effective criteria to select where to insert a new node in the refined region, whereas there was no clear best criteria to select which of the regions to refine.

n_{reg}	Best performing criteria	Maximum relative error [%]	Quantity
5	Quadratic error First order moment	11.01	$\dot{\omega}'_T$
10	Error First order moment	2.43	$\dot{\omega}'_T$
15	Error First order moment	1.03	ρ
20	Quadratic error First order moment	1.15	ρ

Tab. 4.3.: Stability study for r-refinement relaxation coefficient α_r using emergence of unpopulated regions as stop criteria.

The decrease in maximum relative error as we increase the number of regions can be seen in Figure 4.10. It's possible to notice in this graph that the improvement of the error stagnated after 15 regions. This may be explained by the imposed maximum resolution that the Spline may have introduced by the minimum amount of points that a region must have that was imposed. Apart from that, we can see that increasing the number of regions was effective to bring the error down up to two orders of magnitude down, as happened with the temperature source term.

After comparing the 20 regions Spline here with the case without refinement, we can move on to the Spline with 15 regions trained with the Error and First order moment criteria, highlighted in Table 4.4, which was the one that presented the lowest maximum relative error between all the Splines generated in the current study. The relative error in PV space can be seen in Figure 4.11,

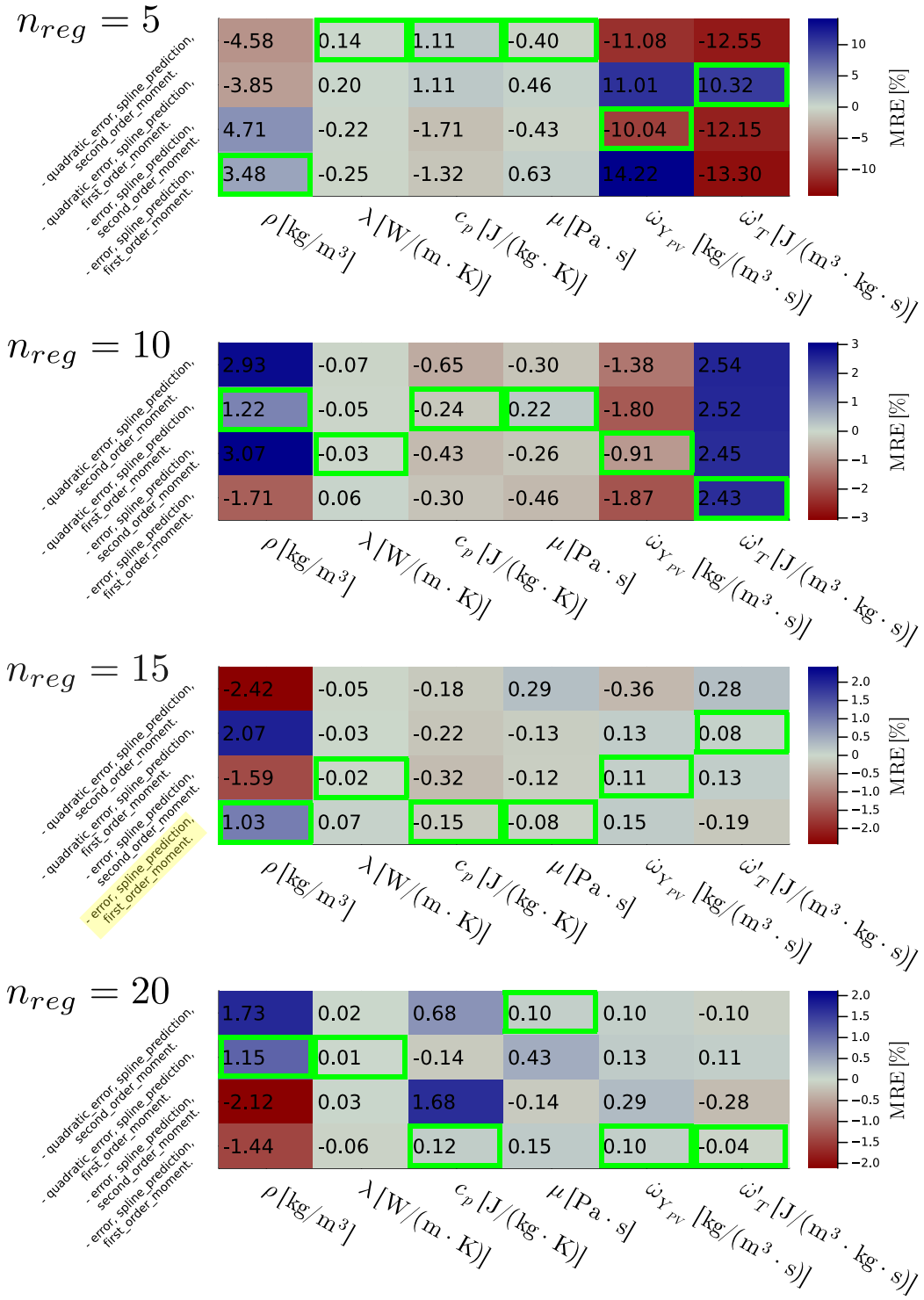
where we can notice a lower error magnitude and a more distributed error profile than the one achieved without refinement, whereas the error for the source terms continue to oscillate near the flame front.

Even with a sampling method that tries to avoid the problem of data imbalance, choosing points randomly per region works just like choosing points randomly in the case that there is only one region. This made explicit again that the capability of this method to find Spline representations for data does suffer from problems caused by data imbalance. As the training run progresses and more nodes get inserted via h-refinement, this problem becomes less and less noticeable, as we have more regions dividing the domain, so the sampled data becomes more evenly distributed. This problem could be further attacked by expanding the current sampling method to adjust the probability of a data point showing up in the sampled set by normalizing it based on the density of points in its neighborhood, allowing the whole domain of the dataset to express itself in the loss function in a more even manner, while also ensuring that every region have the same number of data points.

4.6. Single-input multiple-output with r- and h-refinement

At last, we can see in Figure 4.12 that the combination of r- and h-refinement was successful in further bringing the error down. The same formula for the relative error is used, but it is calculates with the normalized values during training. A small number of 5 regions was chosen, in order to keep them apart enough to avoid unpopulated regions. The graph shown accounts for the moving average of the error, since it's very noisy, as can be seen in Figure 1 in the appendix. The noisy nature of this data comes from it being calculated using a sample of the population, differently than the relative error, which is calculated with the whole population.

The r-refinement methods could be further developed to implement measures to avoid instabilities. It was not suitable for long training runs with a higher region count, but it has shown itself to be a valid method to be used alongside with h-refinement in the effort of achieving a better discretization for the Spline domain. The measures taken in the SISO case, as explained in section 3.4, namely to penalize nodes proximity did not show satisfactory results, and other approaches, such as applying the discussed regularization only when nodes are within a stipulated distance from each other could be tried.



Tab. 4.4.: Comparison of the maximum relative error achieved by the different h-refinement methods for varying

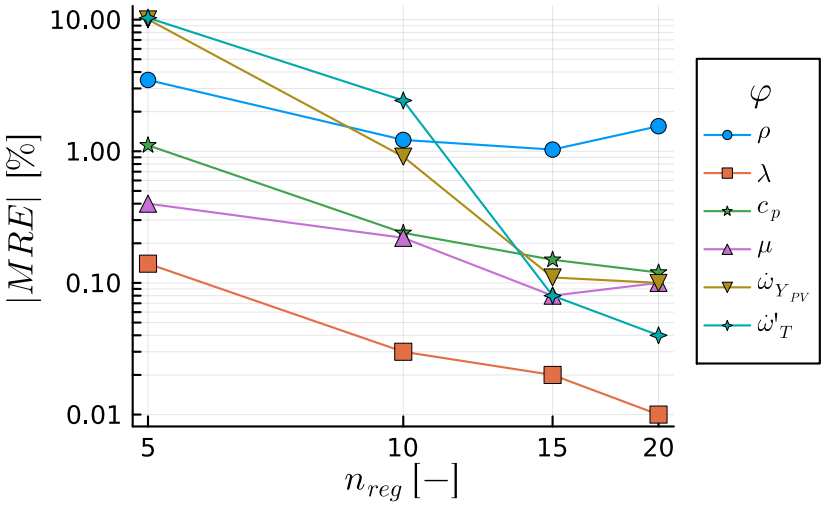


Fig. 4.10.: Maximum relative error for each quantity in terms of n_{reg}. Logarithmic axes. $K = 4$, $\alpha_{opt} = 0.025$.

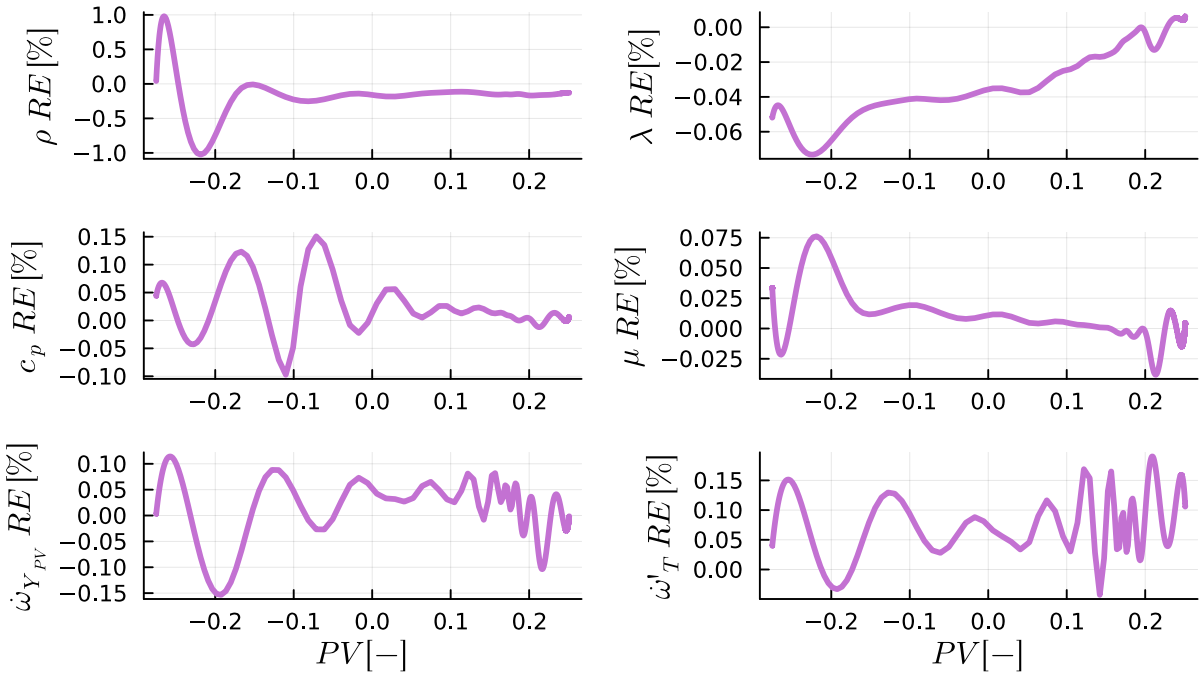


Fig. 4.11.: Relative error in PV space for $K = 4$ Spline with $n_{reg} = 15$ refined with Error and First order moment criteria.

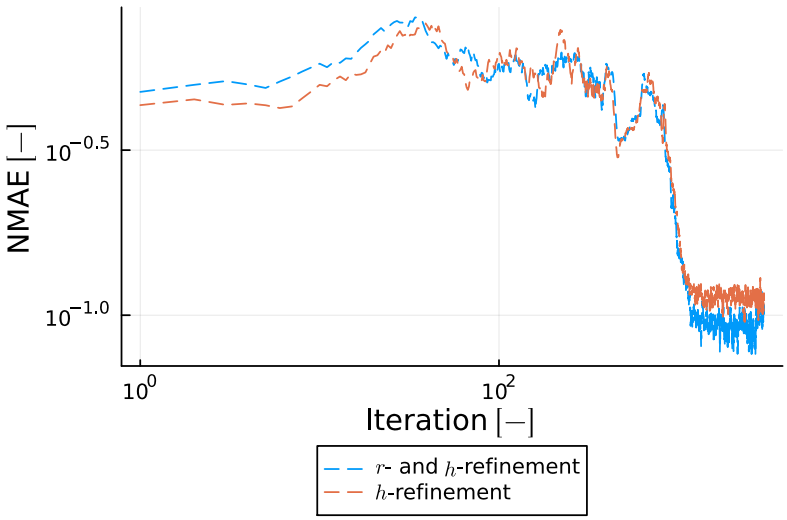


Fig. 4.12.: Comparison for the normalized maximum absolute error between r- and h-refinement and only h-refinement. $K = 4$, $n_{reg} = 5$, $\alpha_r = 10^{-8}$, $\alpha_{opt} = 0.025$. Both curves underwent a moving mean average filter with 20 points in its window.

5. Conclusion

The power of machine learning techniques for curve fitting span from simple function interpolation to complex generative artificial intelligence. Even thought here the work steered away from the - nowadays almost unanimous - path of artificial neural networks, the value of these techniques presented itself to be of extreme usefulness in order to represent chemistry manifolds using higher-order Splines. This ability to represent manifolds is a very important one when speeding up reactive flow simulations by using tabulated quantities to retrieve a set of thermochemical states φ parametrized with quantities ψ that are suitable to describe the phenomena under study.

Here, the Splines were investigated as suitable candidates as higher-order representation for chemistry and thermodynamical manifolds. Their mathematical construction was made having in sight the future use as models to be fitted using gradient descend. This construction went from the single-input single-output form, which was used to demonstrate the inner workings of the Spline, like its weights, nodes, regions and degree, and to show mathematically why some decisions, such as the sampling method, were made. Afterwards, the Splines were generalized for the single-input multiple-output case, which is the most used during this work, as the formulation for the machine learning problem is made having it as objective. The SIMO case is evaluated as a candidate for a suitable representation that captures the behavior of a freely propagating pre-mixed methane air flame, presenting a worst precision of 2.43% of relative error when using Splines with 20 regions and degree 4 for the case without any refinement. When dealing with the case that h-refinement is used, the worst maximum relative error drops to 1.03% for a Spline with same degree and 15 regions. The different criteria to perform this kind of refinement were also discussed. For this both cases, the errors were assessed, and shown to decrease with the application of the discussed refinement techniques. r-refinement is experimented with, and shown as capable of enhancing the grid generated by h-refinement by bringing the error further down. Higher-dimensional Splines were defined, but not further developed, leaving room for further work alongside more refinement techniques.

The challenges faced during development where addressed, mainly of choosing a suitable sampling strategy for the data, and also avoiding the emergence of unpopulated regions. For both problems, suitable solutions were proposed, which worked well in the case with only h-refinement, but could be further developed for r-refinement, as it still lacks stability. The solution for the sampling strategy was backed up by mathematical results, but could be further developed to sample points more equally inside each region.

5.1. Further work

Seen that Splines can be packed with enough information to represent a manifold, as seen in [22], and the current method was able to adjust Splines against a manifold in term of a single input variable PV with an error that decreases as one increases the number of regions or the degree of the representation, it would be a natural next step to implement the MIMO case and assess their capabilities of tabulating higher-dimensional data with a generalization of the methods developed here. This increase in the dimension would make possible to assess the performance of Splines in capturing a wider variety of combustion phenomena, by representing Quenching Flamelet-Generated Manifolds, and possibly also to represent Reaction-Diffusion Manifolds.

It would be possible to calculate the flame structure in physical space generated by the Spline model, evaluate the error in that domain and insert it inside the loss function. This would allow the Splines to be optimized also to minimize the mean squared error of the physical flame structure alongside the current optimization in PV. Using these approach, errors that may contribute a little to the loss function in PV, but present huge effects when visualized in physical space can also be penalized, in the direction of a higher-fidelity representation of the data in both spaces, while keeping the Spline in function of PV and not x. A possible challenge that may come with this approach is to define how the error in physical space will influence h-refinement, which takes place in PV space.

In this same direction, we could formulate the problem with physics-aware machine learning, where physical conservation laws are inserted in the loss function to steer the gradient descend to models that tend to better respect they.

$$\begin{aligned}\mathcal{L}(\phi, \mathcal{D}) = \quad & \alpha_1 \cdot MSE(\phi, \mathcal{D}) \\ & + \alpha_2 \cdot Energy\ Violation(\phi) \\ & + \alpha_3 \cdot Mass\ Violation(\phi)\end{aligned} \tag{5.1}$$

Writing the codebase in Julia provide some advantages. One that could be further exploited is the flexible line between compile and runtime. This allows to compile a specialized version of the Spline with information that may only be known at runtime, such as number of weights I and nodes $I + K + 1$ and also the degree K, unfolding the recursion at compile time. With just-in-time compilation, a specialized version for the Spline can be compiled as the new sets of parameters appear. This could be very useful for functions that are evaluated multiple times, as the compilation overhead at runtime is worth to generate a specialized code that is faster to evaluate afterwards, precisely the case for the Splines.

More mathematical techniques to deal with Splines could also be applied. Here, as the Splines where assumed to have their nodes to be in distinct positions, it's not possible to use local order elevation techniques based on the first derivative discontinuities that nodes placed in the same coordinate introduce. This allowed to deal with the definition for the base Spline shown in Equation 3.22 without having to deal the division by zero that this technique bring. Here, the main reason for it

was simplicity, which indicates that more refinement techniques could be explored in this endeavor of fitting Splines with the least amount of points.

Given the ease that these type of curves can be differentiated in respect to the input values, MIMO Splines could be also used to tabulate thermodynamic potentials, such as the Helmholtz energy $A = U - TS$, which can be used to calculate the thermodynamic properties with a single mathematical relation that was shown to achieve a good precision [47]:

$$\begin{aligned} S &= -\left(\frac{\partial A}{\partial T}\right)\Big|_{V,N} \\ P &= -\left(\frac{\partial A}{\partial V}\right)\Big|_{T,N} \\ \mu &= \left(\frac{\partial A}{\partial N}\right)\Big|_{T,V} \end{aligned} \tag{5.2}$$

In the case of tabulating the Helmholtz energy, Sobolev training could be used. It is a technique that consists in incorporating the error of subsequent derivatives of the model to the loss function formulation, in a way that not only the direct output of the trained model is fitted to adjust the data, but also its derivatives.

Regarding domain, as seen in subsection 2.1.4, the grid as defined here spans through the whole space, but could be restricted to certain regions, as seen in Figure 5.1, by applying a hierarchical grid [48], which could help to further reduce the amount of weights needed to represent a higher-dimensional curve.

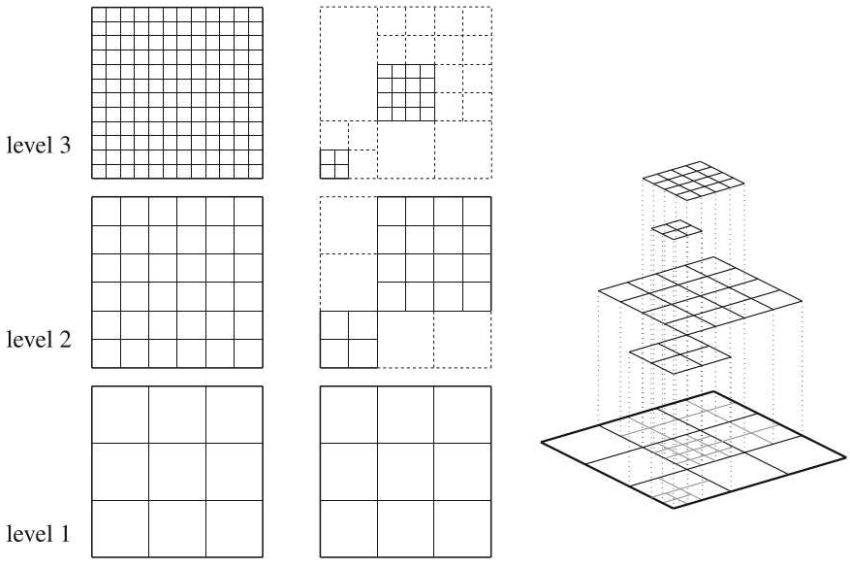


Fig. 5.1.: Hierarchical Spline grid. (From Kunoth 2018 [26])

Bibliography

[1] C. E. P. Cerri, G. Sparovek, M. Bernoux, W. E. Easterling, J. M. Melillo, and C. C. Cerri. "Tropical agriculture and global warming: impacts and mitigation options". In: *Scientia Agricola* 64 (2007), pp. 83–99.

[2] P. Stott. "How climate change affects extreme weather events". In: *Science* 352.6293 (2016), pp. 1517–1518.

[3] H. Ritchie. "Sector by sector: where do global greenhouse gas emissions come from?" In: *Our World in Data* (2020). https://ourworldindata.org/ghg-emissions-by-sector.

[4] E. Kommission and G. Energie. *EU energy in figures : statistical pocketbook 2024*. Amt für Veröffentlichungen der Europäischen Union, 2024. DOI: doi/10.2833/802460.

[5] M. Noor, A. P. Wandel, and T. Yusaf. "Design and development of mild combustion burner". In: *Journal of Mechanical Engineering and Sciences* 5 (2013), pp. 662–676.

[6] D. M. G. Gregory P. Smith. *GRI-Mech 3.0*. 1999. URL: http://www.me.berkeley.edu/gri_mech/ (visited on 1999).

[7] T. Poinsot and D. Veynante. *Theoretical and numerical combustion*. RT Edwards, Inc., 2005.

[8] J. Van Oijen and L. De Goey. "Modelling of premixed laminar flames using flamelet-generated manifolds". In: *Combustion science and technology* 161.1 (2000), pp. 113–137.

[9] U. Maas and S. B. Pope. "Simplifying chemical kinetics: intrinsic low-dimensional manifolds in composition space". In: *Combustion and flame* 88.3-4 (1992), pp. 239–264.

[10] S. Popp, S. Weise, and C. Hasse. "A novel approach for efficient storage and retrieval of tabulated chemistry in reactive flow simulations". In: *High-Performance Scientific Computing: First JARA-HPC Symposium, JHPCS 2016, Aachen, Germany, October 4–5, 2016, Revised Selected Papers 1*. Springer. 2017, pp. 82–95.

[11] J. Bissantz, J. Karpowski, M. Steinhausen, Y. Luo, F. Ferraro, A. Scholtissek, C. Hasse, and L. Vervisch. "Application of dense neural networks for manifold-based modeling of flame-wall interactions". In: *Applications in Energy and Combustion Science* 13 (2023), p. 100113.

[12] L. Zhou, Y. Song, W. Ji, and H. Wei. "Machine learning for combustion". In: *Energy and AI* 7 (2022), p. 100128.

[13] M. Bode, N. Collier, F. Bisetti, and H. Pitsch. "Adaptive chemistry lookup tables for combustion simulations using optimal B-spline interpolants". In: *Combustion Theory and Modelling* 23.4 (2019), pp. 674–699.

[14] M. H. Stone. "The generalized Weierstrass approximation theorem". In: *Mathematics Magazine* 21.5 (1948), pp. 237–254.

[15] C. Runge et al. “Über empirische Funktionen und die Interpolation zwischen äquidistanten Ordinaten”. In: *Zeitschrift für Mathematik und Physik* 46.224-243 (1901), p. 20.

[16] K. Hornik, M. Stinchcombe, and H. White. “Multilayer feedforward networks are universal approximators”. In: *Neural networks* 2.5 (1989), pp. 359–366.

[17] I. J. Schoenberg. “Contributions to the problem of approximation of equidistant data by analytic functions. Part B. On the problem of osculatory interpolation. A second class of analytic approximation formulae”. In: *Quarterly of Applied Mathematics* 4.2 (1946), pp. 112–141.

[18] M. Steinhausen, Y. Luo, S. Popp, C. Strassacker, T. Zirwes, H. Kosaka, F. Zentgraf, U. Maas, A. Sadiki, A. Dreizler, et al. “Numerical investigation of local heat-release rates and thermochemical states in side-wall quenching of laminar methane and dimethyl ether flames”. In: *Flow, Turbulence and Combustion* 106 (2021), pp. 681–700.

[19] C. De Boor. “On calculating with B-splines”. In: *Journal of Approximation theory* 6.1 (1972), pp. 50–62.

[20] C. De Boor. “Package for calculating with B-splines”. In: *SIAM Journal on Numerical Analysis* 14.3 (1977), pp. 441–472.

[21] T. Briand and P. Monasse. “Theory and practice of image B-spline interpolation”. In: *Image Processing On Line* 8 (2018), pp. 99–141.

[22] X. Gu, Y. He, and H. Qin. “Manifold splines”. In: *Proceedings of the 2005 ACM symposium on Solid and physical modeling*. 2005, pp. 27–38.

[23] P. K. Diederik. “Adam: A method for stochastic optimization”. In: *(No Title)* (2014). URL: https://doi.org/10.48550/arXiv.1412.6980.

[24] A. G. Baydin, B. A. Pearlmutter, A. A. Radul, and J. M. Siskind. “Automatic differentiation in machine learning: a survey”. In: *Journal of machine learning research* 18.153 (2018), pp. 1–43.

[25] T. L. Davis. “The emerald table of Hermes Trismegistus. Three Latin versions which were current among later alchemists”. In: *Journal of Chemical Education* 3.8 (1926), p. 863.

[26] A. Kunoth, T. Lyche, G. Sangalli, S. Serra-Capizzano, C. Manni, and H. Speleers. *Splines and PDEs: From approximation theory to numerical linear algebra*. Springer, 2018.

[27] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen. *LAPACK Users’ Guide*. Third. Philadelphia, PA: Society for Industrial and Applied Mathematics, 1999.

[28] L. B. Rall and G. F. Corliss. “An introduction to automatic differentiation”. In: *Computational Differentiation: Techniques, Applications, and Tools* 89 (1996), pp. 1–18.

[29] H. Dawood and N. Megahed. “Automatic differentiation of uncertainties: an interval computational differentiation for first and higher derivatives with implementation”. In: *PeerJ Computer Science* 9 (2023), e1301.

[30] S. Ruder. “An overview of gradient descent optimization algorithms”. In: *arXiv preprint arXiv:1609.04747* (2016).

[31] R. Rhodes. *Dark sun: the making of the hydrogen bomb*. Simon and Schuster, 2012.

[32] J. C. Hunt. “Lewis Fry Richardson and his contributions to mathematics, meteorology, and models of conflict”. In: *Annual Review of Fluid Mechanics* 30.1 (1998), pp. xiii–xxxvi.

[33] D. McLean. “Continuum fluid mechanics and the navier-stokes equations”. In: *Understanding Aerodynamics: Arguing from the Real Physics* (2012), pp. 13–78.

[34] J. Y. Yam and T. W. Chow. “A weight initialization method for improving training speed in feedforward neural network”. In: *Neurocomputing* 30.1-4 (2000), pp. 219–232.

[35] J. A. McCulloch, S. R. St. Pierre, K. Linka, and E. Kuhl. “On sparse regression, Lp-regularization, and automated model discovery”. In: *International Journal for Numerical Methods in Engineering* 125.14 (2024), e7481.

[36] JoJun-Mo. “Effectiveness of Normalization Pre-Processing of Big Data to the Machine Learning Performance”. In: *The Journal of the Korea institute of electronic communication sciences* 14.3 (June 2019), pp. 547–552.

[37] S. Tyagi and P. Sharma. “Taming resource heterogeneity in distributed ml training with dynamic batching”. In: *2020 IEEE International Conference on Autonomic Computing and Self-Organizing Systems (ACSOS)*. IEEE. 2020, pp. 188–194.

[38] S. Tyagi and S. Mittal. “Sampling approaches for imbalanced data classification problem in machine learning”. In: *Proceedings of ICRIC 2019: Recent innovations in computing*. Springer, 2019, pp. 209–221.

[39] D. S. McRae. “r-Refinement grid adaptation algorithms and issues”. In: *Computer Methods in Applied Mechanics and Engineering* 189.4 (2000), pp. 1161–1182.

[40] R. H. Crabtree. “Aspects of methane chemistry”. In: *Chemical Reviews* 95.4 (1995), pp. 987–1007.

[41] H. E. Curry-Hyde and R. Howe. *Natural gas conversion II*. Vol. 81. Elsevier, 1994.

[42] N. Abas, A. Kalair, and N. Khan. “Review of fossil fuels and future energy technologies”. In: *Futures* 69 (2015), pp. 31–49.

[43] G. Leonzio. “Process analysis of biological Sabatier reaction for bio-methane production”. In: *Chemical Engineering Journal* 290 (2016), pp. 490–498.

[44] J. Byun and J. Han. “Economically feasible production of green methane from vegetable and fruit-rich food waste”. In: *Energy* 235 (2021), p. 121397.

[45] A. Nemmour, A. Inayat, I. Janajreh, and C. Ghenai. “Green hydrogen-based E-fuels (E-methane, E-methanol, E-ammonia) to support clean energy transition: A literature review”. In: *International Journal of Hydrogen Energy* 48.75 (2023), pp. 29011–29033.

[46] H. Engerer and M. Horn. “Natural gas vehicles: An option for Europe”. In: *Energy Policy* 38.2 (2010), pp. 1017–1029.

[47] E. W. Lemmon and R. Tillner-Roth. “A Helmholtz energy equation of state for calculating the thermodynamic properties of fluid mixtures”. In: *Fluid phase equilibria* 165.1 (1999), pp. 1–21.

[48] C. Giannelli, B. Jüttler, and H. Speleers. “THB-splines: The truncated basis for hierarchical splines”. In: *Computer Aided Geometric Design* 29.7 (2012), pp. 485–498.

A. Appendix

A.1. Non-filtered normalized maximum absolute error for the SIMO case with r- and h-refinement.

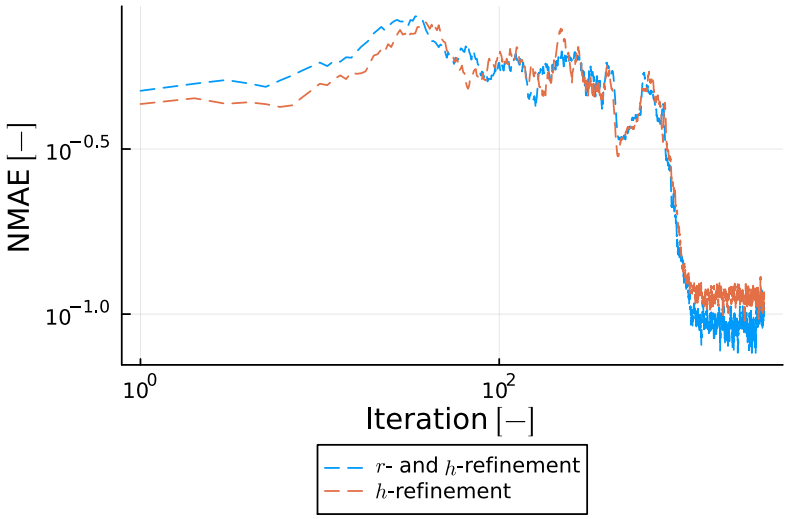


Fig. 1.: Comparison for the normalized maximum absolute error between r- and h-refinement and only h-refinement. $K = 4$, $n_{reg} = 5$, $\alpha_r = 10^{-8}$, $\alpha_{opt} = 0.025$.