UNIVERSIDADE FEDERAL DE UBERLÂNDIA
PÓS-GRADUAÇÃO FEELT
Faculdade de Engenharia Elétrica
Laboratório de Inteligência Computacional

UNIVERSITÉ DE STRASBOURG
ÉCOLE DOCTORALE 269
Mathématiques, Sciences de l'Information et de l'Ingénieur
Laboratoire ICUBE

TESE EM COTUTELA ◇ THÈSE EN CO-TUTELLE

apresentada por ◇ présentée par :

Igor Santos Peretta

defesa em ◇ soutenue le : **21/09/2015**

para obtenção do título de : **Doutor em Ciências**
Área de concentração : Processamento da Informação, Inteligência Artificial

pour obtenir le grade de : **Docteur de l'Université de Strasbourg**
Discipline/ Spécialité : Informatique

Evolution of differential models for concrete complex systems through Genetic Programming

◇

Evolução de modelos diferenciais para sistemas complexos concretos por Programação Genética

◇

Évolution de modèles différentiels de systèmes complexes concrets par Programmation Génétique

TESE orientada por ◇ **THÈSE dirigée par :**	Prof. Dr. Keiji Yamanaka UFU Prof. Dr. Pierre Collet, UNISTRA
REVISORES ◇ **RAPPORTEURS :**	Prof. Dr. Domingos Alves Rade, ITA Prof. Dr. Gilberto Arantes Carrijo, UFU
OUTROS MEMBROS DA BANCA ◇ **AUTRES MEMBRES DU JURY :**	Dr. Frederico Gadelha Guimarães, UMFG Dr. Welsey Pacheco Calixto, IFG Prof. Dr. José Roberto Camacho, UFU

Dados Internacionais de Catalogação na Publicação (CIP)
Sistema de Bibliotecas da UFU, MG, Brasil.

P437e
2015

Peretta, Igor Santos, 1974-
Evolution of differential models for concrete complex systems through genetic programming / Igor Santos Peretta. - 2015.
104 f. : il.

Orientadores: Keiji Yamanaka e Pierre Collet.
Tese (doutorado) - Universidade Federal de Uberlândia, Programa de Pós-Graduação em Engenharia Elétrica e Université de Strasborg, École Doctorale 269.
Inclui bibliografia.

1. Engenharia elétrica - Teses. 2. Análise de sistemas - Teses. 3. Equações diferenciais ordinárias - Teses. 4. Equacões diferenciais parciais - Teses. I. Yamanaka, Keiji. II. Collet, Pierre. III. Universidade Federal de Uberlândia, Programa de Pós-Graduação em Engenharia Elétrica. IV. Université de Strasborg, École Doctorale 269. V. Título.

CDU: 621.3

Abstract

A system is defined by its entities and their interrelations in an environment which is determined by an arbitrary boundary. Complex systems exhibit emergent behaviour without a central controller. Concrete systems designate the ones observable in reality. A model allows us to understand, to control and to predict behaviour of the system. A differential model from a system could be understood as some sort of underlying physical law depicted by either one or a set of differential equations. This work aims to investigate and implement methods to perform computer-automated system modelling. This thesis could be divided into three main stages: (1) developments of a computer-automated numerical solver for linear differential equations, partial or ordinary, based on the matrix formulation for an own customization of the Ritz-Galerkin method; (2) proposition of a fitness evaluation scheme which benefits from the developed numerical solver to guide evolution of differential models for concrete complex systems; (3) preliminary implementations of a genetic programming application to perform computer-automated system modelling. In the first stage, it is shown how the proposed solver uses Jacobi orthogonal polynomials as a complete basis for the Galerkin method and how the solver deals with auxiliary conditions of several types. Polynomial approximate solutions are achieved for several types of linear partial differential equations, including hyperbolic, parabolic and elliptic problems. In the second stage, the proposed fitness evaluation scheme is developed to exploit some characteristics from the proposed solver and to perform piecewise polynomial approximations in order to evaluate differential individuals from a given evolutionary algorithm population. Finally, a preliminary implementation of a genetic programming application is presented and some issues are discussed to enable a better understanding of computer-automated system modelling. Indications for some promising subjects for future continuation researches are also addressed here, as how to expand this work to some classes of non-linear partial differential equations.

Keywords: Computer-Automated System Modelling; Differential Models; Linear Ordinary Differential Equations; Linear Partial Differential Equations; Fitness Evaluation; Genetic Programming.

Resumo

Um sistema é definido por suas entidades e respectivas interrelações em um ambiente que é determinado por uma fronteira arbitrária. Sistemas complexos exibem comportamento sem um controlador central. Sistemas concretos é como são designados aqueles que são observáveis nesta realidade. Um modelo permite com que possamos compreender, controlar e predizer o comportamento de um sistema. Um modelo diferencial de um sistema pode ser compreendido como sendo uma lei física subjacente descrita por uma ou mais equações diferenciais. O objetivo desse trabalho é investigar e implementar métodos para possibilitar modelamento de sistemas automatizado por computador. Esta tese é dividida em três etapas principais: (1) o desenvolvimento de um solucionador automatizado para equações diferenciais lineares, parciais ou ordinárias, baseado na formulação de matriz de uma customização do método de Ritz-Galerkin; (2) a proposição de um esquema de avaliação de aptidão que se beneficie do solucionador numérico desenvolvido para guiar a evolução de modelos diferenciais para sistemas complexos concretos; (3) investigações preliminares de uma aplicação de programação genética para atuar em modelamento de sistemas automatizado por computador. Na primeira etapa, é demonstrado como o solucionador proposto utiza polinômios ortogonais de Jacobi como uma base completa para o método de Galerkin e como o solucionador trata condições auxiliares de diversos tipos. Soluções polinomiais aproximadas são obtidas para diversos tipos de equações diferenciais parciais lineares, incluindo problemas hiperbólicos, parabólicos e elípticos. Na segunda etapa, o esquema proposto para avaliação de aptidão é desenvolvido para explorar algumas características do solucionador proposto e para obter aproximações polinomiais por partes a fim de avaliar indivíduos diferenciais de uma população de dado algoritmo evolucionário. Finalmente, uma implementação preliminar de uma aplicação de programação genética é apresentada e algumas questões são discutidas para uma melhor compreensão de modelamento de sistemas automatizado por computador. Indicações de assuntos promissores para continuação de futuras pesquisas também são abordadas, bem como a expansão deste trabalho para algumas classes de equações diferenciais parciais não-lineares.

Palavras-chave: Modelamento de Sistemas Automatizado por Computador; Modelos Diferenciais; Equações Diferenciais Ordinárias Lineares; Equações Diferenciais Parciais Lineares; Avaliação de Aptidão; Programação Genética.

Résumé

Un système est défini par les entités et leurs interrelations dans un environnement qui est déterminé par une limite arbitraire. Les systèmes complexes présentent un comportement émergent sans un contrôleur central. Les systèmes concrets désignent ceux qui sont observables dans la réalité. Un modèle nous permet de comprendre, de contrôler et de prédire le comportement du système. Un modèle différentiel à partir d'un système pourrait être compris comme une sorte de loi physique sous-jacent représenté par l'un ou d'un ensemble d'équations différentielles. Ce travail vise à étudier et mettre en œuvre des méthodes pour effectuer la modélisation des systèmes automatisée par l'ordinateur. Cette thèse pourrait être divisée en trois étapes principales, ainsi: (1) le développement d'un solveur numérique automatisé par l'ordinateur pour les équations différentielles linéaires, partielles ou ordinaires, sur la base de la formulation de matrice pour une personnalisation propre de la méthode Ritz-Galerkin; (2) la proposition d'un schème de score d'adaptation qui bénéficie du solveur numérique développé pour guider l'évolution des modèles différentiels pour les systèmes complexes concrets; (3) une implémentation préliminaire d'une application de programmation génétique pour effectuer la modélisation des systèmes automatisée par l'ordinateur. Dans la première étape, il est montré comment le solveur proposé utilise les polynômes de Jacobi orthogonaux comme base complète pour la méthode de Galerkin et comment le solveur traite des conditions auxiliaires de plusieurs types. Solutions à approximations polynomiales sont ensuite réalisés pour plusieurs types des équations différentielles partielles linéaires, y compris les problèmes hyperboliques, paraboliques et elliptiques. Dans la deuxième étape, le schème de score d'adaptation proposé est conçu pour exploiter certaines caractéristiques du solveur proposé et d'effectuer l'approximation polynômiale par morceaux afin d'évaluer les individus différentiels à partir d'une population fournie par l'algorithme évolutionnaire. Enfin, une mise en œuvre préliminaire d'une application GP est présentée et certaines questions sont discutées afin de permettre une meilleure compréhension de la modélisation des systèmes automatisée par l'ordinateur. Indications pour certains sujets prometteurs pour la continuation de futures recherches sont également abordées dans ce travail, y compris la façon d'étendre ce travail à certaines classes d'équations différentielles partielles non-linéaires.

Mots-clés: Modélisation des Systèmes Automatisée par l'Ordinateur; Modèles Différentiels; Équations Différentielles Ordinaires Linéaires; Équations Différentielles Partielles Linéaires; Score d'Adaptation; Programmation Génétique.

Para Anabela & Isis,
com todo o meu amor

Agradecimentos

O processo intenso de realizar uma pesquisa e a posterior redação de uma tese não é um processo individual, mas sim conta com um grande número de pessoas ligadas direta ou indiretamente. Nessas páginas, gostaria de agradecer a todos aqueles com que me relacionei na minha vida de doutorando no Brasil e os quais, próximos ou distantes, contribuiram para a finalização dos meus trabalhos de pesquisa. Infelizmente, esta lista não é conclusiva e não foi possível incluir a todos. Assim, gostaria de exprimir meus agradecimentos ...

À minha esposa Anabela, pela cumplicidade e paciência, e à minha filha Isis, pela sua subjetiva compreensão, e à ambas pelo amor, apoio em momentos tão difíceis e por estarem sempre ao meu lado me acompanhando em todos os destinos necessários para a conclusão deste trabalho. À minha família, em especial meu pai Vitor, minha mãe Miriam, meus irmãos Érico e Éden, sempre referências em tempos de desorientação, pelo amor, suporte e incentivo incondicionais. À Priscilla, por ajudar a catalizar reflexões de doutorado, e à Jussara, pelas conversas sobre rumos de vida. Ao meu sobrinho Iuri, por ter chegado a tempo.

Ao Professor Keiji Yamanaka, meu orientador no Brasil, pela confiança em mim depositada, pela presteza em vir ao auxílio, pelas conversas e divisão de angústias, além da grande oportunidade de trabalharmos mais uma vez juntos.

Aos Professores Domingos Alves Rade e Gilberto Arantes Carrijo, pela presteza e pontualidade apresentadas para a árdua tarefa de serem pareceristas preliminares desta tese. Ao Professor José Roberto Camacho, pela contagiante paixão, pelo incentivo e pelas muitas discussões. Aos Professores Frederico Guadelha Guimarães e Wesley Pacheco Calixto, pelo entusiasmo e pelo interesse demonstrado neste trabalho. A todos esses, obrigado pelas considerações discutidas em tempos de qualificação e também por aceitarem o convite para participar da banca de defesa.

Ao grande Júlio Cesar Ferreira, por compartilhar angústias, receios e conhecimentos, além de ter sido um porto seguro em tempos de aventuras além-mar.

Aos meus digníssimos companheiros em armas: Hugo Xavier Rocha, Gerson Flávio Mendes de Lima, Ricardo Boaventura do Amaral, Monica Sakurai Pais e Josimeire Tavares, pelas conversas, discussões, incentivos e pela amizade duradoura. Aos amigos e preciosos interlocutores Leonardo Garcia Marques e Guilherme Ayres Tolentino, pelas altas discussões sobre o tema desta tese e por ajudarem a organizar ideias e dar um pouco de ordem ao caos de meus pensamentos.

Aos companheiros de laboratório: Juliana Malagoli, Cássio Xavier Rocha, Walter Ragnev, Adelício Maximiano Sobrinho, pelas amizades e pela divisão de saberes e angústias. Aos amigos de UFU, Fábio Henrique "Corleone" Oliveira e Daniel Stefany, pelo suporte e incentivo nas horas mais inusitadas.

Aos técnicos da CAPES: Valdete Lopes e Mailson Gomes de Sousa, que com presteza me auxiliaram no processo de ida e volta da França.

Ao programa de pós-graduação da Faculdade de Engenharia Elétrica, em especial à Cinara Mattos, pela simpatia, presteza e solidariedade mesmo nas situações mais difíceis. Aos professores Alexandre Cardoso, Edgard Afonso Lamounier Júnior, Darizon Alves de Andrade, cada qual ao seu tempo, pela atenção necessária a fim de realizar esta pesquisa.

Aos professores Alfredo Júlio Fernandes Neto e Elmiro Santos Resende, cada qual em seu tempo de Reitor da Universidade Federal de Uberlândia, pela atenção necessária para a realização deste doutorado em cotutela.

Ao casal Fábio Leite e Silvia Maria Cintra da Silva e à Carmen Reis, pela grande amizade e pelo apoio familiar tão importante nesses tempos de doutorado.

Aos amigos da DGA na UNICAMP, pelo importante apoio no começo de minha jornada: Maria Estela Gomes, Pedro Emiliano Paro, Edna Coloma, Lúcia Mansur, Elsa Bifon, Soninha e Pedro Henrique Oliveira.

Aos amigos de Campinas: Carlos Augusto Fernandes Dagnone, Marco Antônio Zanon Prince Rodrigues, Bruno Mascia Daltrini, Daniel Granado, Alexandre Martins, Sérgio Pegado, Alexandre Loregian, João Marcos Dadico e Rodrigo Lício Ortolan. pela longa amizade e pela caminhada que trilhamos juntos. Estaremos sempre próximos, mesmo que distantes.

Remerciements

Le processus intensif de poursuivre la recherche et la rédaction d'une thèse n'est pas un processus individuel, mais on peut impliquer des nombreuses personnes, directement ou indirectement. Dans ces pages, je tiens à remercier toutes celles et ceux que j'ai rencontré durant ma vie de doctorant en France et qui, de près ou de loin, ont contribué à la réussite de mes travaux de recherche. Malheureusement, cette liste n'est pas exhaustive et ce n'est pas possible d'inclure tout le monde. C'est pourquoi je voudrais exprimer mes remerciements ...

Au Professeur Pierre COLLET, mon directeur de thèse en France, pour avoir établi une confiance en moi, les discussions sur le sujet de cette recherche et les indications afin de consulter divers experts. Aussi, pour le soutien continu au niveau académique et personnel. C'était une opportunité exceptionnelle de pouvoir travailler avec vous.

Au Professeur Paul BOURGINE, pour l'intérêt, le soutien, l'attention, mais surtout pour l'accueil et la générosité de partager des enseignements et des idées essentielles à cette recherche.

Au Professeur Jan DUSEK, à bien des discussions sur le sujet de cette thèse, et à Professeure Myriam MAUMY-BERTRAN, tous les deux pour faire partie du comité de suivi de thèse.

Au Professeur Thomas NOËL, en acceptant d'être le rapporteur français de cette thèse.

A mes compagnons « d'armes »: Andrés TROYA-GALVIS, Bruno BELARTE, Carlos CATANIA, Clément CHARNAY, Karim EL SOUFI, Manuela YAPOMO, Joseph PALLAMIDESSI, Wei YAN, Chowdhury FARHAN AHMED et tous les stagiaires qui ont été proche de moi pendant cette année en France. Merci pour l'accueil chaleureux, pour les enseignements et pour l'amitié qui restera, bien j'espère longtemps.

À Julie THOMPSON et au Olivier POCH, pour l'accueil sympathique et l'amitié manifestée.

A l'équipe BFO, les MCF Nicholas LACHICHE, Cecilia ZANNI-MERK, Stella MARC-ZWECKER, Pierre PARREND, François de Bertrand DE BEUVRON et Agnès BRAUD. Aussi, à les Professeurs Pierre GANÇARSKI et Christian MICHEL. C'était un honneur de travailler avec vous.

Aux anciens doctorants Frédéric KRÜGER et OGIER MAITRE, et l'ancienne post-doc Lidia YAMAMOTO, pour le guidage, même que nous ne nous rencontrâmes pas en personne.

À Professeure Evelyne LUTTON et au Alberto Paolo TONDA, pour l'intérêt manifesté sur le sujet de cette thèse.

À Laboratoire ICUBE, je voudrais spécialement remercieer Mme Christelle CHARLES, Mme Anne-Sophie PIMMEL, Mme Anne MULLER, Mme Fabienne VIDAL et le directeur Professeur Michel DE MATHELIN, pour l'accueil sympathique et l'attention particulière.

À École Doctorale MSII, en particulier à Mme Nathalie BOSSE, pour le soutien et l'attention toujours manifestés.

À UFR Mathématique-Informatique, je tiens en particulier à remercier Mme Marie Claire HANTSCH, pour l'accueil sympathique et chaleureux.

À Université de Strasbourg, je remercie Mme Isabelle LAPIERRE, pour le soutien et l'attention particulière.

Aux Professeurs Yves REMOND, directeur de l'École Doctorale, et Alain BERETZ, le président de l'Université de Strasbourg, pour l'attention nécessaire à la réalisation de ce doctorat en co-tutelle.

Aux amis en France: Laurent, Anna Lucas et famille KONDRATUK; Caroline et famille VIRIOT-GOELDEL; Ignacio, Sara et famille GOMEZ MIGUEL; Franck, Karine et famille HAGGIAG GLASSER, pour tous les moments passés ensemble et l'amitié éternelle.

Enfin, à Mme Marie Louise KOESSLER, en raison des cours de français donnés et des révisions éventuelles, mes très chaleureux remerciements.

Acknowledgements

This research was funded by the following Brazilian agencies:

- Conselho Nacional de Desenvolvimento Científico e Tecnológico (**CNPq**), full PhD scholarship category GD, while in Brazil;
- Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (**CAPES**), PDSE scholarship #18386-12-1, while in France.

A journey of a thousand miles
starts beneath one's feet.

Lao-Tzu in *Tao Te Ching*

Contents

List of Figures

List of Tables

List of Algorithms

List of Acronyms

CASM	Computer-Automated System Modelling
EA	Evolutionary Algorithm
EC	Evolutionary Computation
FEM	Finite Element Method
GP	Genetic Programming
GSE	Galerkin System of Equations
LDE	Linear Differential Equation
LODE	Linear Ordinary Differential Equation
LPDE	Linear Partial Differential Equation
ODE	Ordinary Differential Equation
PDE	Partial Differential Equation
SNR	Signal-to-Noise Ratio
TGE	Truncate Galerkin Expansion
WGN	White Gaussian Noise

Part I

Background

Chapter 1

Introduction

1.1 Overview

A system is defined by its interrelated parts, also known as entities, surrounded by an environment which is determined by an arbitrary boundary. More insights on the definition of a system could be achieved by accessing the work of [1]. This present work considers a system of interest as being concrete (in contrast to abstract) and possibly closed, even when it could be classified as open. The former classification means that the system can exist in this reality. The latter means that every entity has some relations with others, *i.e.*, if an entity is part of a system, that means it can affect and be affected by others, directly or indirectly, and is also responsible in some degree for the overall behaviour that the system presents.

A concrete system could be object of a simplified representation, known as a model, in order to be understood, to explain its behaviour with respect to its entities and to enable simulations and predictions of its behaviour according to an arbitrary initial state. In reality, it is usual to not totally represent a concrete system due to the great number of constituent entities involved together with a large set of complex interrelations. Normally, to build such representation (known as a model) is to optimize the compromise between simplification and accuracy. This work is interested about *in silico* models which refers to "simulations using mathematical models in computers, thus relying on silicon chips" [2]. The process of building such model to a system, approximately and adequately, needs to rely on its most relevant entities (independent variables) that have influence on the overall system behaviour (represented by one or more dependent variables). This process is widely known as System Modelling. Note that, as stated by [1], the number of significant entities and relations could change depending on the arbitrary determination of a boundary.

A representative model could be understood as some sort of underlying physical law [3, 4], or even a descriptor which could fulfil the variational principle of least action[1] [5, 6]. As stated by [7], "many physical processes in nature [...] are described by equations that involve physical quantities together with

[1] Also known as principle of stationary action.

their spatial and temporal rates of change". Actually, observations of natural phenomena were responsible to the early developments of the infinitesimal calculus discipline [8]. In other words, due to its properties of establishing connections and interactions between independent and dependent entities (*e.g.* physical, geometrical, relational), models to systems are expected to be one or a set of differential equations [9]. An ordinary differential equation (ODE), if only one entity is considered responsible for the behaviour of a system, or more commonly a partial differential equation (PDE) can describe how some observable quantities change with respect to others, tracking those changes throughout infinitesimal intervals.

Presenting as a simple example, the vertical trajectory of a cannonball when shot in an ideal scenario could be modelled by the ODE $g + \frac{d^2}{dt^2}y(t) = 0$. This equation presents the relation between the unknown function $y(t)$ — the instantaneous height of the cannonball relative to an inertial frame of reference with respect to a relative measure of time t — and the acceleration of gravity g. Initial state conditions such as $\frac{d}{dt}y(t)\big|_{t=0} = V_0$ and $y(0) = H_0$ effectively lead to the following well known solution: $y(t) = H_0 + V_0\, t - \frac{g\, t^2}{2}$. This solution to that differential model describes with ideal precision the cannonball vertical trajectory. If this system can be kept closed to outer entities (*e.g.*, air friction, strong winds), the mentioned differential model would still be the same, no matter the fact that different initial states could lead to different vertical trajectory solutions.

From the point of view of engineering, this work is interested in concrete systems whose entities enable some kind of quantitative measurements for related quantities[2]. If those measurements are taken from the main entities responsible for the behaviour of the system, then it is fair to suppose that an accurate enough model could be built.

Nowadays, the necessity for models is increasing, once science is dealing with concrete systems that could display a huge dataset of observations (Big Data researches) or even present chaotic behaviour (dynamic or complex systems). This work goes further into this idea and investigates how system modelling could be automated. This thesis is part of a research aimed to mitigate difficulties and propose methods to enable a computer-automated system modelling (CASM) tool to construct models from observed data.

1.2 Motivation

When defining a system of interest, researches intent to describe a great variety of phenomena, from Physics and Chemistry to Biology and Social sciences. Systems modelling have applications to problems of engineering, economics, population growth, the propagation of genes, the physiology of nerves, the regulation of heart-beats, chemical reactions, phase transitions, elastic buckling, the onset of turbulence, celestial mechanics, electronic circuits [10], extragalactic pulsation of quasars, fluctuations in sunspot activity on our sun,

[2] Qualitative measurements are not object of this thesis. More information on this subject could be found on "Fuzzy Modelling".

changing outdoor temperatures associated with the four seasons, daily temperature fluctuations in our bodies, incidence of infectious diseases, measles to the tumultuous trend of stock price [11], among many others examples. Models are essential to correctly understand, to predict and to control their respective systems. An inaccurate model will fail to do so.

The classic approach for system modelling is to apply regressions techniques of some kind on a set of measurements in order to retrieve a mathematical function that could explain that dataset.

Regression techniques involve developing causal relations (functions) of one or more entities (independent variables) to a sensible effect or behaviour (dependent variable of interest). Historically, those techniques have being used to system modelling starting from observed data. There are two main approaches to regression: classic (or conventional) regression and symbolic regression.

Conventional regression starts from a particular model form (a mathematical expression with a known structure) and follows by using some metrics to optimize parameters for a pre-specified model structure supposed to best fit the observed data. A clear disadvantage is that, after parametrized by using ill-behaved data, the chosen model could not be useful at all, or even work just within a limited region of the domain, failing in other regions. A specific difficult dataset example is shown in Figure 1.1. There, different conventional regression techniques fail to rediscover a known function from its randomly sampled sparse points. Note that, to achieve the full potential of those techniques, data must be well behaved (*e.g.*, equidistant points) and be available in a sufficient amount.

While conventional regression techniques seek to optimize the parameters for a pre-specified model structure, symbolic regression avoids imposing prior assumptions, and, instead, infers the model from the data.

Symbolic regression, in the other hand, searches for an appropriate model structure rather than imposing some prior assumptions. Genetic Programming (GP) is widely used for this purpose [12, 13]. GP is based on Genetic Algorithms (GA) and belongs to a class of Evolutionary Algorithms (EA) in which ideas from the Darwinian evolution and survival of the fittest are roughly translated into algorithms. Therefore, GP is known to evolve a model structure side-by-side with the respective necessary parameters. Also, there is the theoretical guarantee (in infinite time) that GP will converge to an optimum model[3] able to fit the observed data. As an example, if trigonometric functions are available as building blocks, Genetic Programming is capable of converging to the function

$$y(x) = 3\ \sin(\pi\, x)\, cos(16\, \pi\, x)$$

which is the correct function subjected to the sampling of points at random back in Figure 1.1.

To understand why this work does not simply use symbolic regression, take a close look at Figure 1.2. Both left and right plots show only two sampled points. Lets imagine this hypothetical situation where there are concrete systems, the "left" one and the "right" one, and from both there are only two

[3] In practice, researches expect a near-optimum solution only.

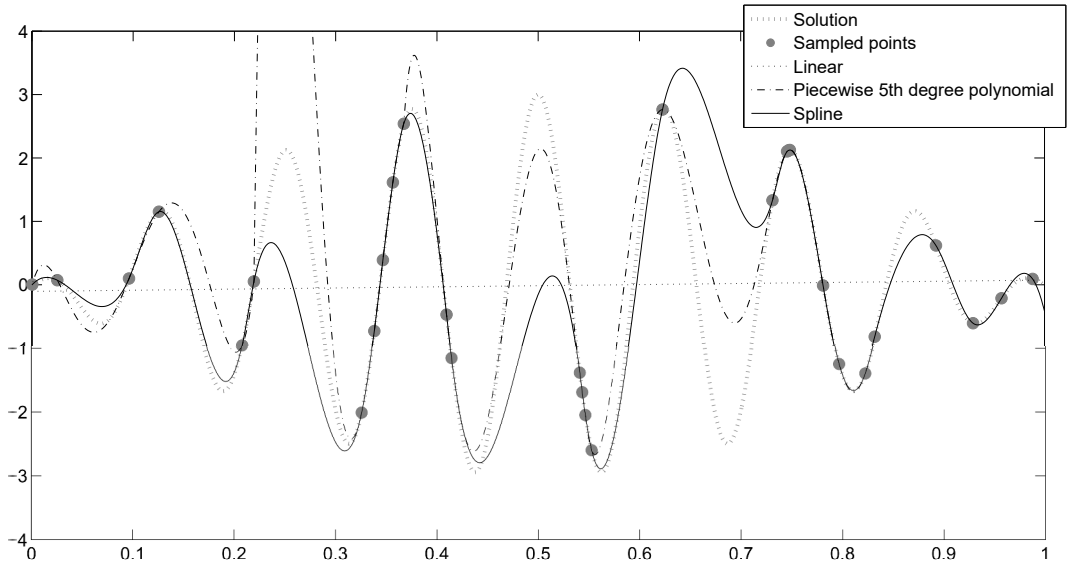

Figure 1.1: A situational example where different methods for conventional regression (linear, piece-wise 5th degree polynomial, spline) fail to find a known solution from a not so well behaved randomly sampled points.

measurements available for each one. The known behaviour of those systems are respectively described by

$$f(x) = 1 - e^{-x} \text{ and } g(x) = e^{-x} - 1.$$

The left plot also presents among other infinite possibilities the following functions that pass through the same two sampled points:

$$\begin{array}{ll} \text{(sine)} & f_s(x) = 0.6321\, \sin(\frac{\pi}{2}x) \\ \text{(polynomial)} & f_p(x) = 0.4773\, x^2 + 0.1548\, x \\ \text{(linear)} & f_l(x) = 0.6321\, x. \end{array}$$

The right plot also presents the functions:

$$\begin{array}{ll} \text{(sine)} & g_s(x) = -0.6321\, \sin(\frac{\pi}{2}x) \\ \text{(polynomial)} & g_p(x) = -0.4773\, x^2 - 0.1548\, x \\ \text{(linear)} & g_l(x) = -0.6321\, x. \end{array}$$

Note that they are one the mirror image of the other (related to the horizontal axis through $f(x) = g(x) = 0$), but lets move this information aside for a moment.

Actually, both plots refer to solutions for the same ODE:

$$\frac{d^2}{dx^2}y(x) + \frac{d}{dx}y(x) = 0$$

with different initial values, for the plot on the left:

$$\left.\frac{d}{dx}y(x)\right|_{x=0} = 1 \text{ and } y(0) = 0;$$

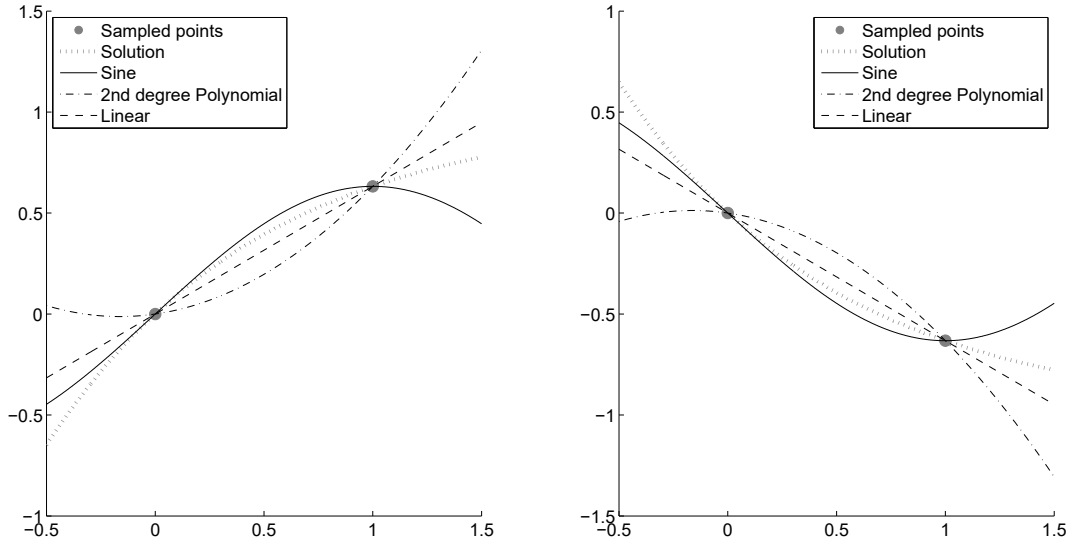

Figure 1.2: Hypothetical situation, two points sampled from each concrete system. (left) Known describing function is $f(x) = 1 - e^{-x}$; (right) Known describing function is $g(x) = e^{-x} - 1$.

and, for the plot on the right:

$$\left.\frac{d}{dx}y(x)\right|_{x=0} = -1 \text{ and } y(0) = 0.$$

Assuming the differential model for those systems is known, the solution of this ODE not only supplies a reliable *interpolation* function between those two points, but a reliable *extrapolation* function as well. The process of solving a differential model could benefit from measurements to infer initial states or boundaries and the solution would be valid as long as neither involved entities (tracked by independent variables) vanish nor others appear.

This hypothetical situation shows the possibility of the same model representing either two separate systems or the same system presented in two different states. As could be inferred, awareness of the initial state leads the model to present itself as having a unique solution. A purely symbolic regression approach would have two major difficulties when considering this very situation here[4]: (a) all enlisted functions — $f_s(x)$, $f_p(x)$, $f_l(x)$, $g_s(x)$, $g_p(x)$, $g_l(x)$ — would be considered valid solutions, as the same for any of the infinite possible functions that pass exactly through those two points; (b) each situation represented by both left and right systems have a high probability of having a different function model and, in this case, no relation between them would be uncovered. In other words, symbolic regression *per se* would not have enough information to even start to raise questions about similarities between those two systems. One could state that symbolic regression is directed to

[4]The intention of this elaborated example is just to exploit a line of thinking. Symbolic regression would have tools to support global-optimum solutions instead of local ones if more data is presented.

model only one "instantiation" of the system (a single possible initial state or adopted boundary) at a time.

Another argument, as known to those dealing with physics and calculus of variations, the action functional (a path integration) is an attribute of a system related to a path, i.e., a trajectory that a system presents between two boundary points in space-time. The principle of least action (also known as the principle of stationary action) states that such system will always present a path over which this action is stationary (an extreme, usually minimal and unique) [6]. This path of least action (the integrand of the action) is often described by a differential equation and describes the intrinsic relations of a system, the very type of differential model this work is aimed to look for.

Following this path, it is pretty straightforward to reach the conclusion that a CASM tool should search for differentials whose solutions could explain the observed data. Also, this tool should not keep the search within the domain of mathematical expressions, as done by classic symbolic regression. The domain of search becomes the space of differential equations. In that way, discussions about a possible unification for both left and right aforementioned systems would be possible. Such approach would be concerned about the model of the system itself, whichever "instantiation" (possible initial states or adopted boundary) it has been presented.

Given the domain of search for a model as the space of possible differential equations and concepts behind the principle of least action, this work starts from the idea that every observable concrete system from which some quantitative measurements could be taken is a valid candidate to construct a model. As stated in [14], "the idea of automating aspects of scientific activity dates back to the roots of computer science" and this research is no different. This work intends to investigate a possible way to enable CASM. Looking forward, as that work concluded, "human-machine partnering systems [...] can potentially increase the rate of scientific progress dramatically" [14].

1.3 Thesis Statement

One of the essential objectives of this work is to develop a computer-automated numerical solver for linear partial differential equations in order to assist a Genetic Programming application to evaluate fitness of model candidates. The provided input for the Genetic Programming application should be a dataset containing measurements taken from observations of the system of interest.

Research questions

Some questions have been guiding this research:

- Given a database which contains measurements from an observable concrete system, is there a more robust way to verify how fit is a theoretical model to this system, relying on those available data?
- As modelling presupposes observation, creativity and specific knowledge, is it feasible to achieve a CASM tool?

- Would such CASM tool be able to rediscover known models, propose modifications to them, or even reveal previously unknown models?

This thesis presents answers to the first two questions. The third one is partially answered, though. This is an open work in the sense that it points to several branches of possible research to be carried on.

Objectives

In this section, the general and specific objectives are presented.

General

Achieve a linear differential equation numerical solver to support a concrete system modelling tool which uses Genetic Programming to evolve sets of partial differential equations. A dataset of observations must be available.

Specific

- Develop a computer-automated numerical solver for linear partial differential equations with no restrictions besides linearity. The solver must assist the evolutionary search of the Genetic Programming application by enabling fitness evaluation of individuals constituted by linear partial differential equations.
- Develop a syntax tree representation for a candidate solution and a proper module for fitness evaluation in consonance with the proposed solver.
- Run some case studies where the observations dataset is generated through simulation of a known model; provide those simulated data as inputs to the Genetic Programming application with the intention of evolving the model to the known solution, turning this exercise into an inverse problem resolution.
- Evaluate the impact of adding noise to input data regarding the evolution of a previous known model. This should enable discussions about tolerance for measurements related to the system of interest.
- Identify and propose derived branches for future works.

1.4 Contributions

The present work brings the following contributions:

- A novel approach to the Ritz-Galerkin method to approximately solve linear differential equations: static choice of Jacobi-Legendre polynomials as basis functions; finite difference method inspired treatment of auxiliary conditions; use of linear algebra discipline to enable solution

of systems of linear equations (e.g. using metrics and procedures as rank, condition, pseudoinverse). The achieved solution is a polynomial approximation of the differential solution.

- A generic scheme to a computer-automated numerical solver for linear partial differential equations (ordinary ones included) using polynomial approximations for the differential solution. Also, the knowledge to expand this solver to some non-linear differential equations is already gathered and it is planned for the near future.
- A dynamic fitness evaluation scheme to be plugged into evolutionary algorithms to automatically solve linear differential equations and evaluate model candidates.

This work had to restrict itself to linear differential equations, though, but those models could present any structure inside the linearity restriction. Besides, the same method is used to both ODEs and PDEs. Indeed, the search for differential models has been tried before. Even so, authors have no knowledge of works which could deal with systems in general but the ones where further specifications on the form of the model is required.

1.5 Research tools

Numerical methods

As stated by [7], one of the "most general and efficient tool for the numerical solution of PDEs is the Finite element method (FEM)". Some limitations do not allow this work to follow this suggested path, though. FEM[15, 16, 17, 7] starts from solving a differential equation (or a set of) in order to present results over a mesh of points throughout the domain. The type of modelling this work is interest on implies in having the actual results of some system on some points over the domain and trying to recover the differential which could explain the behaviour of the system. This is an inverse problem and FEM could not help but to inspire some solutions here presented.

As could be imagined, the method of searching for differential equations must solve at some point those differentials in order to verify the quality of a model candidate. Moreover, integrals should also be useful. The classical and widely used numerical tools to do the job are: (a) using the technique of separating variables to partial differential equations and applying Runge-Kutta methods to approximate solutions for the achieved ordinary differential equations; and (b) Gauss Quadrature methods to perform numerical integrations for arbitrary functions [18, 19], multidimensional cases covered by tensor products or sparse grids [20, 21]. Numerical methods designed to directly solve partial differential equations are seldom explored in the literature, due to the success of the aforementioned methods, and the growing need for multidimensional integration (cubature) methods keeps it as an open research topic.

Diverging from the common sense, this work tries to generalize the process of modelling of multivariate systems. In order to do so, robust multivariate

operations are necessary, especially when dealing with partial differential equations. Parallelism is also desirable, once the entire process has the potential to be an eager customer of computational power. The possibility of transforming it into a Linear Algebra problem, as could be seen when dealing with FEM, is also very tempting. After a long period of experimentations and aiming for those purposes, this work has finally adopted the following numerical methods: (a) the Ritz-Galerkin method [22], specifically an own customization of the method, to build a system of equations from differential equations (ordinary or partial); (b) Monte Carlo integration [23] to perform multivariate integrals; and (c) matrix formulations with related operations to evaluate candidate models.

Evolving models

The GP technique is classified under the Evolutionary Computation (EC) research area in which, as suggested by its name, covers different algorithms that draw inspiration from the process of natural evolution [24]. GP is, at the most abstract level, a "systematic, domain-independent method for getting computers to solve problems automatically starting from a high-level statement of what needs to be done" [13]. That is an expected quality for evolving models by GP which is known to to find previous unthoughtful solutions for unsolved problems so far [25]. This feature could only be accessed if GP is allowed to build random individuals from a unconstrained search space.

Implementing CASM through GP have been proven the right choice in the literature, specially when modelling functions from data [26, 4, 27, 28]. For a system of interest with available measurements, this works instead aims to evolve a functional (partial differential equation) whose solution is a function that could explain the available data. Classic GP symbolic regression needs some adjustments to be able to do so.

Computer programming language

The chosen language for programming is C++. Besides high speed performances [29], C++ language has been listed on the top 5 programming languages rank [30], has support for several programming paradigms (*e.g.*, imperative, structured, procedural and object-oriented), has a large active community, could benefit from 300+ open source libraries [31] (including 100+ of `boost` set of libraries only) and several others freely distributed (*e.g.* BLAS and LAPACK[5] for linear algebra purposes; MPICH2, CUDA and OpenCL for parallel/concurrency programming), and allows the programmer to take control of every aspect of programming. In the other hand, C++ is strongly platform based (code has to be compiled in whatever operational system and/or hardware the executable is needed to run on) and the programmer has to be aware of every aspect of programming (depending on the aimed application, programmer also needs to know about the hardware involved). Those pros and cons were evaluated before this choice, including the need this project has for high performance computation.

[5] LAPACKE library for C++.

1.6 Outline of the text

This thesis is divided into three parts. The first one, Background, covers this introduction in Chapter 1. A non comprehensive list of related works that deal with system modelling through Genetic Programming is presented in Chapter 2. Related theory in Chapter 3 are addressed in order to understand the method proposed here: linear differential equations, Hilbert inner product and basis for function spaces, Ritz-Galerkin method, well-possessedness of a differential problem, Jacobi polynomials, linear mappings and change of variables, Monte Carlo integration and Genetic Programming.

The second part refers to the proposed method itself. It starts by explaining how the proposed method could be applied to linear ordinary differential equations in Chapter 4. The extension of those results when applying the method to linear partial differential equations is shown in Chapter 5.

The third and last part is about system modelling. A fitness scheme is proposed in Chapter 6 in order to evaluate differential model candidates. Chapter 7 brings a preliminary implementation of a Genetic Programming application to perform system modelling. Finally, some results, discussions and other extensions to this work as future research topics could be found in Chapter 8.

Appendices are presented addressing publications achieved during the time of this doctoral studies (Appendix A), as well as future topics in need to be addressed, as the massively parallel paradigm of GPGPUs (Appendix B) and a more robust parallel platform for GP known as EASEA (Appendix C).

Chapter 2

Related Works

2.1 A brief history of the field

Since decades ago, scientists have been trying to build models from observable data. Once datasets of interest starts to increase and underlying model structures became complicated to infer, scientists start thinking about automating the modelling process.

One of the first works that authors could find, the work of Crutchfield and McNamara [32] in 1987 shows the development of a numerical method based on statistics to reconstruct motion equations from dynamic/chaotic time-series data. In a subsequent work, Crutchfield joined Young [33] to address updates to that approach while introducing a metric of complexity for non-linear dynamic systems.

Still in the 1980's, some researchers had developed techniques capable of evolving computer programs, like the works of Cramer, Hicklin and Fujiko [34, 35, 36], respectively, as an attempt to inspire "creativity" into computer machines. These efforts culminate with the advent of Genetic Programming with the works of Koza [37, 12] to enable science in the 1990's to start experiencing computer-automated symbolic regression in the form of mathematical expressions constructed from data. In general, all family of Evolutionary Algorithms [38, 39, 40, 24] could be easily related with system modelling, but GP brought a lot of facilities and powerful tools into the subject [13].

Nevertheless, the work of Schmidt and Lipson [4] published in 2009 is often seen by the scientific community as a great landmark for computer-automated system modelling due to the broad impact it had on the media at the time it was published (*e.g.*, articles in [41, 42, 43]). Even considering that some relevant issues were raised by Hillar [44], Schmidt and Lipson provided observations from basic lab experiments to a computer and this computer was able, using GP-like techniques, to evolve some underlying physical laws in the form of mathematical expressions with respect to the phenomena addressed in the experiments, using 40 minutes to a few hours to do so, depending on the problem.

In the same issue of the journal Science that the paper of Schmidt and Lipson was published, Waltz and Buchanan [14] defended the need for an automation of science, without debunking the role of the researcher. They

pointed out that "computers with intelligence can design and run experiments, but learning from the results to generate subsequent experiments requires even more intelligence". This work has the perspective that computer-automated system modelling must be aimed to help scientists to understand, predict and control their object of study.

Therefore, this section is aimed to cover works that are relate to this thesis within the subject of computer-automated system modelling from observable data. Only works that also make use of GP or some other EA are addressed here. Note that the following list is not intent to be comprehensive, but should reflect the state of art in this field. The list is sorted from the early years to nowadays. When two or more works are from the same year, sort criteria turns to be lexicographic.

2.2 Early papers

before 2000

Gray *et al.* [26] uses GP to identify numerical parameters within parts of the non-linear differential equations that describes a dynamic system, starting from measured input-output response data. The proposed method is applied to model the fluid flow through pipes in a coupled water tank system.

2000 up to 2004

Cao *et al.* [45] describes an approach to the evolutionary modelling problem of ordinary differential equations including systems of ordinary differential equations and higher-order differential equations. They propose some hybrid evolutionary modelling algorithms (genetic algorithm embed in genetic programming) to implement the automatic modelling of one and multi-dimensional dynamic systems respectively. GP is employed to discover and optimize the structure of a model, while GA is employed to optimize its parameters.

Kumon *et al.* [46] present an evolutionary system identification method based on genetic algorithms for mechatronics systems which include various non-linearities. The proposed method can determine the structure of linear and non-linear elements of the system simultaneously, enabling combinatorial optimization of those variables.

Chen and Ely [47] compare the use of artificial neural networks (ANN), genetic programming, and mechanistic modelling of complex biological processes. They found these techniques to be effective means of simulation. They used Monte Carlo simulation to generate sufficient volumes of datasets. ANN and GP models provided predictions without prior knowledge of the underlying phenomenological physical properties of the system.

Banks [48] presents a prior approach to model Lyapunov functions. He has implemented a GP, in Mathematica®, which searches for a Lyapunov function of a given system. The project was successful in finding Lypunov functions for simple, two-dimensional systems.

Leung and Varadan [49] propose a variant to GP in order to demonstrate its ability to design complex systems that attempts to reconstruct the functional form of a non-linear dynamical system from its noisy time series measurements. They did different tests on chaotic systems and real-life radar sea scattered signals. Then they apply GP to the reverse problem of constructing optimal systems for generating specific sequences called spreading codes in CDMA communications. Based on computer simulations, they have shown improved performance of the GP-generated maps.

Hinchliffe and Willis [50] uses multi-objective GP to evolve dynamic process models. He uses GP ability to automatically discover the appropriate time history of model terms required to build an accurate model.

Xiong and Wang [51] propose both a new GP representation and algorithm that can be applied to both continuous and discontinuous functions regression applied to complex systems modelling. Their approach is able to identify both structure and discontinuity points of functions.

2005 up to 2009

Beligiannis *et al.* [52] adopts a GP-based technique to model the non-linear system identification problem of complex biomedical data. Simulation results show that the proposed algorithm identifies the true model and the true values of the unknown parameters for each different model structure, assisting the GP technique to converge more quickly to the (near) optimal model structure.

Bongard and Lipson [53], states that uncovering the underlying differential equations directly from observations poses a challenging task when dealing with complex non-linear dynamics. Aiming to symbolically model complex networked systems, they introduce a method that can automatically generate symbolic equations for a non-linear coupled dynamical system directly from time series data. They state that their method is applicable to any system that can be described using sets of ordinary non-linear differential equations and have an observable time series of all independent variables.

Iba [54] presents an evolutionary method for identifying models from time series data, adopting a model as a system of ordinary differential equations. Genetic programming and the least mean square were used to infer the systems of ODEs.

2.3 Contemporary papers, 2010+

McGough *et al.* [55] represent a line of research on GP-based generation of Lyapunov functions. As stated: "one of the fundamental questions that arises in nonlinear dynamical systems analysis is concerned with the stability properties of a rest point of the system". The theory of Lyapunov is used to understand the qualitative behaviour of the rest point. Their work uses a variant of GP to evolve Lyapunov functions for a given dynamic systems, aiming to explore their stability.

Gandomi and Alavi [56], propose a new multi-stage GP strategy for modelling non-linear systems. Based on both incorporation of each predictor variable individual effect and the interactions among them, their strategy was able to provide more accurate simulations.

Edited by Soto [27], a book about GP that has several chapters dedicated to examples of GP usage in system modelling.

Stanislawska *et al.*[28] use genetic programming to build interpretable models of global mean temperature as a function of natural and anthropogenic forcings. Each model defined is a multiple input, single output arithmetic expression built of a predefined set of elementary components.

Finally, Gaucel *et al.* [57] propose a new approach using symbolic regression to obtain a set of first-order Eulerian approximations of differential equations, and mathematical properties of the approximation are then exploited to reconstruct the original differential equations. Some highlighted advantages include the decoupling of systems of differential equations to be learned independently and the possibility of exploiting widely known techniques for standard symbolic regression.

2.4 Discussion

In general, a model is referred as a mathematical expression that translate abstract functions supposed to generate experimental observed data. Besides discussion in Section 1.2, this widely adopted point of view is of greater use in science. Nevertheless, this work aims to built "differential models" from observable data, *i.e.*, a differential equation with the potential of unveiling interrelations, physical quantities and energy transformations that could be obscure due to the complexity of available data.

In this section some related works are enlisted, related mainly to system modelling from data. From those, there are some who favoured the discussion similarly to this present thesis, *e.g.*, Gray [26], Cao [45], Bongard [53], Iba [54], and Gaucel [57], *i.e.*, they are also dealing with differential models within their works. While the work of Gray deals with structured non-linear differential equations, the others attacked the problem by assuming models as systems of ordinary equations. Both [53] and [57] stand out for given contributions. Bongard achieved symbolic equations as models, and Gaucel realizes some mathematical identities that are really relevant for the overall performance of CASM.

Even so, those works adopt different paradigms. This present work aims to evolve partial differential models from observable data. To accomplish this, an elaborated novel method is presented in order to be applied to any linear differential equation (ordinary or partial) to obtain unique projections for the solution. This proposed method acts the same, no matter the dimensionality of the problem. Authors have no knowledge about other works within CASM that uses something similar to the proposed approach present in this thesis.

Chapter 3

Theory

In this Section, some key subjects to understand contributions from this work are presented, as linear differential equations, Hilbert inner product space, Galerkin's method, well posed problems, Jacobi polynomials, linear mappings, change of variables, Monte Carlo integration, and Genetic Programming.

3.1 Linear differential equations

Linear differential equations (LDE) could be described basically by a linear operator $\mathcal{L}$ which operates a function $u(\vec{x})$ — the unknown or the solution — and results in a source function $s(\vec{x})$. LDEs are in the form $\mathcal{L}[u(\vec{x})] = s(\vec{x})$. A simple definition of a linear differential operator $\mathcal{L}$ of order Q with respect to each of D variables is shown in Equation (3.1).

$$\mathcal{L}[u(\vec{x})] = \sum_{q=0}^{Q^\star-1} k_q(\vec{x}) \left[\prod_{i=0}^{D-1} \frac{\partial^{\gamma_{q,i}}}{\partial x_i^{\gamma_{q,i}}} \right] u(\vec{x}). \tag{3.1}$$

where $\vec{x} = (\, x_0, x_1, \ldots x_{D-1}\,)^T$; $\gamma_{q,i}$ is the order of the partial derivative with respect to i^{th} variable designed by the q^{th} case from the $Q^\star$ possible combinatorial orders (see Chapter 5 for details); $k_q(\vec{x})$ refers to each term coefficient and could be a function itself, including constant, linear and even non-linear ones; and $u(\vec{x})$ is the multivariate function operand to the functional $\mathcal{L}$. Note that the definition $\frac{\partial^0}{\partial x_i^0} u(\vec{x}) \equiv u(\vec{x})$ has been adopted here.

Using definition of $\mathcal{L}$, multivariate LDEs could be written in the form of Equation (3.2):

$$\begin{gathered} \mathcal{L}[u\,(\vec{x})] = s\,(\vec{x}) \\ \sum_{q=0}^{Q^\star-1} k_q(\vec{x}) \left[\prod_{i=0}^{D-1} \frac{\partial^{\gamma_{q,i}}}{\partial x_i^{\gamma_{q,i}}} \right] u(\vec{x}) = s(\vec{x}) \end{gathered} \tag{3.2}$$

where $u\,(\vec{x})$ is the unknown function (dependent variable) which is the solution to the differential equation; and $s\,(\vec{x})$ is the source function, sometimes referred to as the source term. Note that both $k_q\,(\vec{x})$ and $s\,(\vec{x})$ could be constant, linear

or even non-linear functions with respect to independent variables addressed by $\vec{x}$.

Related to this definition, this work considers that: (a) $k_q(\vec{x})$ coefficients are real functions (constant, linear or non-linear), *i.e.*, $\forall x$, $k_q(\vec{x}) \in \mathbb{R}$; (b) the unknown function $u(\vec{x})$ refers to a scalar field; (c) the source function reflects either homogeneous — $s(\vec{x}) = 0$ — or inhomogeneous — $s(\vec{x}) \neq 0$ — differential equations.

An univariate $\mathcal{L}$, also known as a linear ordinary differential operator, could be defined as in Equation (3.3):

$$\mathcal{L}[u(x)] = \sum_{q=0}^{Q} k_q(x) \frac{d^q}{dx^q} f(x) \tag{3.3}$$

where Q is the order of the linear differential operator $\mathcal{L}$; $k_q(x)$ are the $Q+1$ coefficients from respective terms, with the restriction that $k_Q(x) \neq 0$; $u(x)$ is the operand for $\mathcal{L}$ and is assumed to be a function of the only independent variable x. Note that $\mathcal{L}$ contains a dependent variable $u(x)$ and its derivatives with respect to the independent x.

Using definition of $\mathcal{L}$, univariate LDEs could be written in the form of Equation (3.4):

$$\begin{gathered} \mathcal{L}[u(x)] = s(x) \\ \sum_{q=0}^{Q} k_q(x) \frac{d^q}{dx^q} u(x) = s(x) \end{gathered} \tag{3.4}$$

where $u(x)$ is the unknown function (dependent variable) which is the solution to the differential equation; and $s(x)$ is the source function, sometimes referred to as the source term. Note that both $k_q(x)$ and $s(x)$ could be constants, linear functions themselves or even non-linear functions with respect to the independent variable x.

Distinct from LDEs, non-linear differential equations have at least one term which is a power of the dependent variable and/or a product of its derivatives. An example for the former is the inviscid Burgers equation: $\frac{\partial}{\partial t} u(x,t) = -u(x,t)\, \frac{\partial}{\partial x} u(x,t)$. Other example for the latter could be formulate by any differential equation which has term with $\left(\frac{\partial}{\partial x} u(x,t)\right)^k$ or even $\left(\frac{\partial}{\partial x} u(x,t)\right) \cdot \left(\frac{\partial}{\partial t} u(x,t)\right)$. Note that terms as $\frac{\partial}{\partial x} \frac{\partial}{\partial t} u(x,t)$ are still linear. For now, non-linear differential equations are not object of this thesis.

3.2 Hilbert inner product and basis for function space

An inner product for functions can be defined as in Equation (3.5):

$$\langle\, f(x),\, g(x) \,\rangle = \int_a^b f(x) g(x) w(x)\, dx \tag{3.5}$$

where $f(x)$ and $g(x)$ are operands; a and b the domain interval for the independent variable x; and $w(x)$ is known as the weight function.

A Hilbert inner product space is then defined when choosing the interval $[\,a,\,b\,]$ and weight function $w(x)$, in order to satisfy the properties of conjugate symmetry, linearity in the first operand, and positive-definiteness [58, pp.203]. Note that, when in $\mathbb{R}$, the inner product is symmetric and also linear with respect to both operands.

Two functions $f_n(x)$ and $f_m(x)$ are then considered orthogonal to each other in respect to a Hilbert space by the definition present in Equation (3.6):

$$\langle\, f_n(x),\, f_m(x)\,\rangle = h_n\,\delta_{nm} = \begin{cases} 0 & \text{if } n \neq m \\ h_n & \text{if } n = m \end{cases} \tag{3.6}$$

where h_n is a constant dependent on $\langle\, f_n(x),\, f_n(x)\,\rangle$; and δ_{nm} is the Kronecker delta.

Following Equations(3.5) and (3.6), implication in Equation (3.7) is then valid:

$$\forall w(x),\ \langle w(x), f(x)\rangle = 0 \implies f(x) \equiv 0. \tag{3.7}$$

A complete basis for a function space $\mathcal{F}$ is a set of linear independent functions $\mathcal{B} = \{\phi_n(x)\}_{n=0}^{\infty}$, i.e., a set of orthogonal basis functions. An arbitrary function $f(x)$ could then be projected into this function space as a linear combination of those basis functions, as shown in Equation (3.8):

$$\forall f(x) \in \mathcal{F} \implies f(x) = \sum_{n=0}^{\infty} c_n\,\phi_n(x) \tag{3.8}$$

As an example, if $\mathcal{F}$ is defined as the set of all polynomials functions and power series, a complete basis should be $\mathcal{B} = \{x^i\}_{i=0}^{\infty}$, where it comes that $f(x) = \sum_{j=0}^{\infty} c_j\,x^j$.

Finally, from Equations (3.7) and (3.8), the implication in Equation (3.9) follows:

$$\forall \phi(x) \in \mathcal{B},\ \forall f(x) \in \mathcal{F},\ \langle \phi(x), f(x)\rangle = 0 \implies f(x) \equiv 0. \tag{3.9}$$

3.3 Galerkin method

The Ritz-Galerkin method, widely known as the Galerkin method [22], is one of the most fundamental tools of modern computing. Russian mathematician Boris G. Galerkin generalised the method whose authorship he assigned to Walther Ritz and showed that it could be used to approximate solve many interesting and difficult elliptic problems arising from applications [59]. The method is also a powerful tool in the solution of differential equations and function approximations when dealing with elliptic problems [7, 60].

Also, Galerkin method is considered to be a spectral method from the family of weighted residual methods. Traditionally, those methods are regarded as

the foundation of many numerical methods such as FEM, spectral methods, finite volume method, and boundary element method [61]. A non-exhaustive and interesting historical perspective for the development of the method can be found in [59].

As a class of spectral methods from the family of weighted residual methods, Galerkin method could be defined as a numerical scheme to approximate solve differential equations. Weighted residual methods in general are approximation techniques in which a functional named residual $R[u(x)]$, also known as the approximation error and defined in Equation (3.10), is supposed to be minimized [61].

$$R[u(x)] = \mathcal{L}[u(x)] - s(x) \approx 0 \tag{3.10}$$

Note that $R[u(x)]$ is also known as the residual form of the differential equation. The idea is to have a feasible approximation $\hat{u}(x)$ to the solution $u(x)$ in order to force $R[u(x)] \approx 0$. This approximation is built as a projection on the space defined by a proper chosen finite basis $\mathcal{B} = \{\phi_n(x)\}_{n=0}^{N}$ with a span of $N + 1$ functions. The approximation $\hat{u}(x)$ has the form present in Equation (3.11):

$$\hat{u}(x) = \sum_{n=0}^{N} \tilde{u}_n \phi_n(x), \tag{3.11}$$

where $\tilde{u}_n$ are the unknown coefficients of this weighted sum. The approximation $\hat{u}(x)$ is also known as the truncated Galerkin expansion (TGE) for a finite N. In the literature, the form $\hat{u}(x) = \tilde{u}_0 + \sum_{n=1}^{N} \tilde{u}_n \phi_n(x)$ is also found. However, this thesis adopts the requirement that $\phi_0(x) \equiv 1$ instead.

Galerkin's approach states that when the residual $R[u(x)]$ operates the approximation $\hat{u}(x)$ instead of the solution $u(x)$, this residual is required to be orthogonal to each one of the chosen basis functions in $\mathcal{B}$. This is accomplish by starting from both Equations (3.9) and (3.10) and can be seen in Equation (3.12):

$$\forall \phi(x) \in \mathcal{B},\ \langle\, \phi_n(x),\ R[\hat{u}(x)]\, \rangle = 0,\ n = 0 \dots N \tag{3.12}$$

Then, the method requires to solve those $N+1$ equations in order to find an unique approximate solution of the differential equation described by $R[u(x)]$ with respect to the chosen basis $\mathcal{B}$. Note that all basis functions $\phi(x) \in \mathcal{B}$ must satisfy some auxiliary conditions known *a priori* (usually linear homogeneous boundary conditions) to enable a well posed problem.

Finally, after plugging the approximation in Equation (3.11) to the residual in Equation (3.10) and following Equation (3.12), the Galerkin System of Equations (GSE) is then built, as shown in Equation (3.13):

$$
\begin{aligned}
&\langle\, \phi_n(x),\, R[\hat{u}(x)]\,\rangle\big|_{n=0}^{N} = 0 \\
&\Rightarrow \langle \phi_n(x),\, \mathcal{L}[\hat{u}(x)] - s(x)\rangle\big|_{n=0}^{N} = 0 \\
&\Rightarrow \left[\left\langle\, \phi_n(x),\, \mathcal{L}[\sum_{m=0}^{N} \tilde{u}_m \phi_m(x)]\, \right\rangle - \langle \phi_n(x),\, s(x)\rangle\right]_{n=0}^{N} = 0 \\
&\Rightarrow \left[\sum_{m=0}^{N} \tilde{u}_m\, \langle\, \phi_n(x),\, \mathcal{L}[\phi_m(x)]\,\rangle = \langle \phi_n(x),\, s(x)\rangle\right]_{n=0}^{N}
\end{aligned}
\tag{3.13}
$$

Solving the system of equations in Equation (3.13) for $N+1$ unknown coefficients $\tilde{u}_m$ and afterwards substituting them into Equation (3.11), an approximate solution to the differential equation is finally achieved.

According to [62], Galerkin's method "is not just a numerical scheme for approximating solutions to a differential or integral equations. By passing to the limit, we can even prove some existence results". More information on proofs to the bounded error and convergence of Galerkin method for elliptic problems could be found in [7, pg. 46–51]. Note the importance of choosing the right basis for the approximating finite dimensional subspaces. The work of [62] also emphasises the utilization of Galerkin methods with orthogonal or orthonormal basis functions, *i.e.*, a complete basis.

Note that using the identity in Equation (3.14), it is pretty straightforward to convert summations to a matrix form.

$$
\sum_{j=0}^{M} (a_j \cdot f_{i,j})\,\Big|_{i=0}^{N} = \begin{pmatrix} f_{0,0} & \cdots & f_{0,M} \\ \vdots & \ddots & \vdots \\ f_{N,0} & \cdots & f_{N,M} \end{pmatrix} \cdot \begin{pmatrix} a_0 \\ \vdots \\ a_M \end{pmatrix} \tag{3.14}
$$

Therefore, a GSE could be written in matrix formulation. From Equations (3.13) and (3.14), follows Equation (3.15) in the form:

$$
\begin{aligned}
&G \cdot \tilde{u} = s \Rightarrow \\
&\begin{pmatrix} \langle\, \phi_0(x),\, \mathcal{L}[\phi_0(x)]\,\rangle & \cdots & \langle\, \phi_0(x),\, \mathcal{L}[\phi_N(x)]\,\rangle \\ \vdots & \ddots & \vdots \\ \langle\, \phi_N(x),\, \mathcal{L}[\phi_0(x)]\,\rangle & \cdots & \langle\, \phi_N(x),\, \mathcal{L}[\phi_N(x)]\,\rangle \end{pmatrix} \cdot \begin{pmatrix} \tilde{u}_0 \\ \vdots \\ \tilde{u}_N \end{pmatrix} = \begin{pmatrix} \langle \phi_0(x),\, s(x)\rangle \\ \vdots \\ \langle \phi_N(x),\, s(x)\rangle \end{pmatrix}
\end{aligned}
\tag{3.15}
$$

where G is known as the coefficient (stiffness and mass) square matrix; $\tilde{u}$ is the unknown (displacements) column vector; and s is the source (forces) column vector. Names inside parenthesis are used by FEM.

3.4 Well-posed problems

French mathematician Jacques Salomon Hadamard, among other contributions, coined the widely used notion of well-posed problems for partial differential equations [63, 7]. Hadamard defined a problem to be well-posed only if:

1. A solution exists and is unique;
2. This solution depends continuously on the given data, *i.e.* solution is not unstable.

Therefore, if a problem does not meet all these criteria, it is said to be ill-posed. Note that, even if a problem is well-posed, it may still be ill-conditioned, which means that small numerical variations in elements from the coefficient matrix or the source vector implies in large differences between evaluations of unknowns. According to [7], from a point of view of numerical methods, there are several possible error sources when calculating PDE solutions (*e.g.*, computational domain, boundary and initial conditions, method related parameters, finite computer arithmetic). If a problem is ill-posed, or if it is ill-conditioned, no confidence in the numerical solution is then possible.

Using Equation (3.15) as reference of PDE problem, to test if a differential problem is well-posed, it is sufficient to verify if the coefficient matrix G has full rank. If it is rank deficient, an ill-posed problem is presented. Ill-conditioned problems could be tested if the coefficient matrix, even full rank, has a large condition number.

Both rank and condition number could be calculated from the singular value decomposition (SVD) of the matrix [19].

The rank of a matrix could then be defined as the number of singular values different from zero. Numerically speaking, a tolerance must be defined in order to test if a singular value is close to zero or not. It is common sense to use a tolerance with respect to the number of elements of the matrix and the machine epsilon [64] of the standard hardware floating-point data type that has been used.

The condition number could be defined as the ratio of the largest to smallest singular value of a matrix. A practical way to test conditioning of the coefficient matrix is to evaluate the unknowns using any method available and to try to recover the source vector. Large differences between the original source vector and the new evaluated one indicates an ill-conditioned coefficient matrix. Algorithm 1 presents this practical test if a more robust test is desired. Authors suggest to use the Moore-Penrose pseudo-inverse [65] instead of inverting the coefficient matrix.

Algorithm 1 Practical condition test; test if a coefficient matrix is well-conditioned or not.

Require: System of equations in the form $G \cdot \tilde{u} = s$
  $\tilde{u} \leftarrow (G^{-1} \cdot s)$
  $s^{\star} \leftarrow (G \cdot \tilde{u})$
  if $\|s^{\star} - s\| <$ tolerance **then**
    Matrix G is well-conditioned
  else
    Matrix G is ill-conditioned
  end if

3.5 Jacobi polynomials

The classical option for Galerkin basis functions in FEM are Lagrange interpolating polynomials. Besides the fact they are extremely useful when dealing with piecewise approximations, the restriction on using them in this work came from the fact that they are not orthogonal. The main idea of using polynomials for function approximations is that, given a subspace and an orthonormal[1] basis of n polynomial functions (up to $n \to \infty$), any function can described onto that subspace [66], as seen in Section 3.2.

In that sense, orthogonal polynomials have been widely used with spectral methods as an attractive framework [67]. Using orthogonal polynomials with Galerkin method ensures a Hilbert function space where any desirable smooth function could be projected, *i.e.*, an unique approximation could be built by TGEs. Jacobi polynomials is an interesting choice due to some of their properties, as to be shown.

Jacobi polynomials have the hypergeometric definition present in Equation (3.16), as shown by [68] and [67]:

$$P_n^{(\alpha,\beta)}(x) = \frac{\Gamma(n+\alpha+1)}{\Gamma(n+1)\Gamma(\alpha+1)}\, {}_2F_1\left(-n, n+\alpha+\beta+1; \alpha+1; \frac{1}{2}(1-x)\right) \tag{3.16}$$

where $\Gamma(\cdot)$ is the gamma function; ${}_2F_1(p, q; r; z)$ is the Gauss's hypergeometric function with respect to constants p, q, r, and the independent variable z; $\alpha \geq -1$; $\beta \geq -1$; and $n \geq 0$.

Note that this work uses definitions $\forall \alpha, \beta \begin{cases} n < 0 \implies P_n^{(\alpha,\beta)} \equiv 0; \\ P_0^{(\alpha,\beta)} \equiv 1 \end{cases}$

The use of Pochhammer symbol[2] and factorials allow the following definition to Jacobi polynomials:

$$P_n^{(\alpha,\beta)}(x) = \frac{\Gamma(n+\alpha+1)}{n!\,\Gamma(\alpha+1)} \sum_{k=0}^{n} \frac{(-n)_k\,(n+\alpha+\beta+1)_k}{(\alpha+1)_k} \left(\frac{1-x}{2}\right)^k \frac{1}{k!} \tag{3.17}$$

From [69], also shown by [67], an important identity for derivatives of Jacobi polynomials is presented in Equation (3.18):

$$\frac{d^k}{dx^k} P_n^{(\alpha,\beta)}(x) = \frac{\Gamma(n+\alpha+\beta+k+1)}{2^k \Gamma(n+\alpha+\beta+1)} P_{n-k}^{(\alpha+k,\beta+k)}(x) \tag{3.18}$$

Regarding Hilbert inner product space, Jacobi polynomials are orthogonal on the interval $[-1, +1]$ and have the weight function $w(x)$ presented in Equation (3.19):

$$w(x) = (1-x)^\alpha\,(1+x)^\beta \tag{3.19}$$

[1] Orthogonality restriction is usually enough.

[2] The Pochhammer symbol stands for $(x)_n = \frac{\Gamma(x+n)}{\Gamma(x)}$, besides some special cases.

Therefore, inner products for Jacobi polynomial are defined in Equation (3.20):

$$\left\langle P_n^{(\alpha,\beta)}(x),\, P_m^{(\alpha,\beta)}(x)\right\rangle = \int_{-1}^{1} P_n^{(\alpha,\beta)}(x) P_m^{(\alpha,\beta)}(x)\,(1-x)^{\alpha}(1+x)^{\beta}\,dx \tag{3.20}$$

where $\alpha \geq -1$ and $\beta \geq -1$ ensures the integrability of $w(x)$ [68].

The work of [68, pp.58] also presented mapped Jacobi polynomials, defined in Equation (3.21), which are orthogonal on the arbitrary interval $a \leq x \leq b$:

$$P_n^{(\alpha,\beta)}\left(\xi(x)\right) = \frac{\Gamma(n+\alpha+1)}{\Gamma(n+1)\Gamma(\alpha+1)}\,{}_2F_1\left(-n, n+\alpha+\beta+1; \alpha+1; \frac{b-x}{b-a}\right) \tag{3.21}$$

where $\xi(x) = 2\,\frac{x-a}{b-a} - 1$ and $[\,a,\, b\,]$ is the arbitrary interval on which the inner product is taken.

The associated error is asymptotically minimized in an $L^{p(\alpha)}$-norm, as stated by [67], given the appropriate choice of $\alpha = \beta$. Special cases of Jacobi polynomials could be found choosing appropriate α and β. Basis functions could be generated to be asymptotically similar to Chebyshev polynomials of the first kind, $T_n(z)$, choosing $\alpha = \beta = -1/2$; Chebyshev polynomials of the second kind, $U_n(z)$, choosing $\alpha = \beta = 1/2$; and Legendre polynomials, $P_n(z)$, choosing $\alpha = \beta = 0$. Also, the same work states that the asymptotic error between a given solution function for a differential equation and its TGE is minimized in the L^{∞}, L^1 and L^2-norms, respectively.

Therefore, with the support of mapped Jacobi polynomials and choosing for the sake of simplicity $\alpha = \beta = 0$ (Legendre polynomials), the univariate inner product operator could then be defined as in Equation (3.22):

$$\left\langle\, P_n^{(0,0)}(\xi(x)),\, P_m^{(0,0)}(\xi(x))\,\right\rangle = \int_{a}^{b} P_n^{(0,0)}(\xi(x))\, P_m^{(0,0)}(\xi(x))\, dx \tag{3.22}$$

By following the development of Equation (3.18), authors could achieve the derivative identity from Equation (3.23) which refers to mapped Jacobi polynomials on the finite interval $[\,a,\, b\,]$:

$$\frac{d^k}{dx^k} P_n^{(\alpha,\beta)}\left(\xi(x)\right) = \frac{\Gamma(n+\alpha+\beta+k+1)}{(b-a)^k\Gamma(n+\alpha+\beta+1)} P_{n-k}^{(\alpha+k,\beta+k)}\left(\xi(x)\right) \tag{3.23}$$

Note that identities in Equations (3.18) and (3.23) exchanges a derivative operation on a polynomial by another polynomial, a very useful treat.

3.6 Mappings and change of variables

In order to generalise the method of Galerkin, some mappings are required. Considering an arbitrary interval domain, the inner product implies the use of either the mapped Jacobi polynomials from Equation (3.21) or a mapped version of the differential equation under investigation to the interval $[\,-1,\,+1\,]$, the very interval where Jacobi polynomials are orthogonal. The underlying definite integral must be evaluated on the same interval for both multiplying functions. The change of variables for differentiation and integration then become very useful.

First, the interval of orthogonality for Jacobi polynomials should be hereby identified by the variable $\xi \in [\,-1,\,+1\,]$. In the other hand, arbitrary intervals should be represented by $x \in [\,a,\,b\,]$. Linear mappings in Equations (3.24) and (3.25) then become straightforward to understand.

$$x \mapsto \xi: \quad \xi(x) = 2\,\frac{x-a}{b-a} - 1 \tag{3.24}$$

$$\xi \mapsto x: \quad x(\xi) = \frac{b-a}{2}\,\xi + \frac{b+a}{2} \tag{3.25}$$

A change of variables for differentiations, as $\frac{d}{dx}\,f(x) \mapsto \frac{d}{d\xi}\,f(\xi)$, is presented in Equation (3.26).

$$\frac{d}{dx}\,f(x) = \frac{d\xi}{dx}\,\frac{d}{d\xi}\,f(\xi \mapsto x) = \frac{2}{b-a}\,\frac{d}{d\xi}\,f\left(x(\xi)\right) \tag{3.26}$$

Therefore, a change of variables for higher order derivatives is shown in Equation (3.27).

$$\frac{d^k}{dx^k}\,f(x) = \left(\frac{2}{b-a}\right)^k\,\frac{d^k}{d\xi^k}\,f\left(x(\xi)\right) \tag{3.27}$$

Finally, the change of variables for integrations, as $\int_a^b f(x)\,dx \mapsto \int_{-1}^1 f(\xi)\,d\xi$, known as integration by substitution in unidimensional cases, is presented in Equation (3.28).

$$\int_a^b f(x)\,dx = \int_{-1}^1 f(\xi \mapsto x)\,J\,d\xi = \int_{-1}^1 f\left(x(\xi)\right)\,\frac{dx}{d\xi}\,d\xi = \frac{b-a}{2}\int_{-1}^1 f\left(x(\xi)\right)\,d\xi \tag{3.28}$$

where J is the determinant of the Jacobian matrix. In this unidimensional case and because of linear mapping shown in Equation (3.25), this determinant $J = \frac{dx}{d\xi} = \frac{b-a}{2}$ is a constant.

3.7 Monte Carlo integration

In order to integrate an arbitrary function, Monte Carlo integration picks random points over a certain domain (a multidimensional volume) and calculate

Table 3.1: Monte Carlo integration applied to $f(x, y) = \exp(-x)\cos(y)$ defined in $\{\, 0 \leq x \leq 1;\ 0 \leq y \leq \frac{\pi}{2} \,\}$ (100 runs).

N points	$\int_0^1 \int_0^{\frac{\pi}{2}} \exp(-x)\cos(y)\, dx\, dy$				order of
	min	max	mean	standard deviation	execution time
$2^9(512)$	0.59363	0.66306	0.63085	0.014666	1× (ref)
2^{10}	0.60741	0.65514	0.63246	0.010593	1.75×
2^{12}	0.62093	0.64844	0.63104	0.005892	4.25×
2^{14}	0.62634	0.63793	0.63202	0.002552	17×
2^{16}	0.62918	0.63574	0.63231	0.001411	66×
2^{18}	0.63003	0.63454	0.63219	0.000752	249×
2^{20}	0.63117	0.63293	0.63216	0.000352	965×
2^{21}	0.63133	0.63279	0.63210	0.000292	1 905×
$2^{22}(4\,194\,304)$	0.63163	0.63263	0.63213	0.000175	3 786×

the mean value of the function taken on those random points [23], as presented in Equation (3.29).

$$\int_V f(\vec{x})\, d\vec{x} \approx \left(\prod_{i=0}^{V-1} b_i - a_i \right) \frac{1}{N} \sum_{j=0}^{N-1} f(\vec{X}_j) \tag{3.29}$$

where V is the hyper-volume which represents the number of dimensions; N is the number of random $\vec{X}_j$ points to be taken; a_i and b_i are the limits of integration for the i^{th} dimension.

For example, the function $f(x, y) = \exp(-x)\cos(y)$ has, for the domain $\{\, 0 \leq x \leq 1;\ 0 \leq y \leq \frac{\pi}{2} \,\}$, a multidimensional definite integral ≈ 0.6321 when analytically evaluated. Table 3.1 presents results from Monte Carlo integration with respect to the number of random points taken.

The great advantage of Monte Carlo integration is that multivariate integrals are straightforward to evaluate. This very scheme is flexible and adjustable on the fly. The disadvantage is that, to get reliable results, the number of random picked points must be large. Example in Table 3.1, for instance, needs something around 2^{20} ($\approx 10^6$, 1 million) points to drop standard deviation bellow 10^{-3}. For this quantity of random points, execution time is almost 1000× the time necessary when choosing 2^9 (≈ 500) points[3]. Execution time almost doubles each new increment on the power of 2 after that. Figure 3.1 shows how on average Monte Carlo integration performs very well, but individually its reliability is proportional to the number of picked random points. Also, depending on the function to be integrated, those numbers and analyses could change. A workaround is to fit the final modelling tool with some parallel paradigm to allow the use of a large set of points without compromising execution time.

[3] In this case, less than 10^{-3} sec.

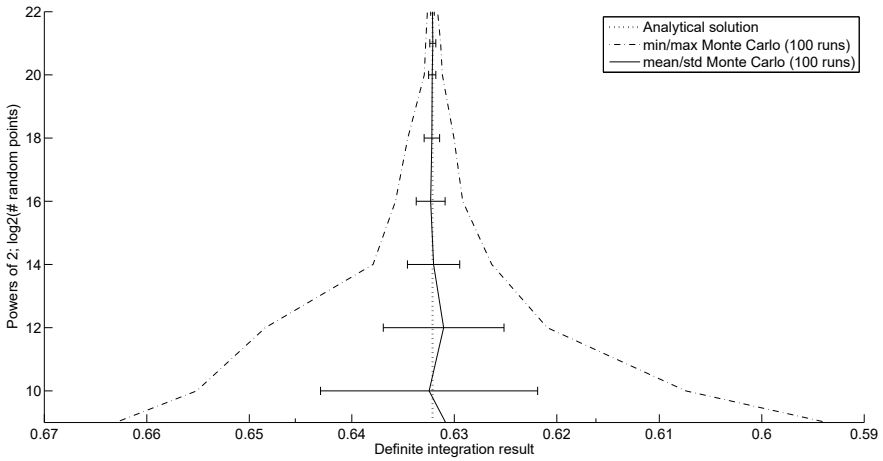

Figure 3.1: Monte Carlo integration performance on $f(x, y) = \exp(-x)\cos(y)$ defined in $\{\, 0 \leq x \leq 1;\ 0 \leq y \leq \frac{\pi}{2} \,\}$ (100 runs).

3.8 Genetic Programming

Koza [70] has defined GP as "an automatic technique for producing a computer program that solves, or approximately solves, a problem". He follows by stating that GP "addresses the challenge of getting a computer to solve a problem without explicitly programming it". GP is widely defined as an extension of the Genetic Algorithm from [71] which, in turn, is preceded by notions and concepts presented by [72].

John R. Koza is the reference when the subject is GP because of his milestone work [37]. He has written several books on this subject and helped to popularize GP across the world. Today GP is extensively described in books, in edited collections of papers, in conference proceedings, in journals (e.g. *Genetic Programming and Evolvable Machines journal*[4]), and at web sites such as `www.genetic-programming.org`. Interesting to mention, before Koza, other researches had built models capable of evolving computer programs also based on Genetic Algorithms, e.g. [34], [35], [36]. This could be seen as the reflex of humanity trying to inspire "creativity" into computer machines.

The main idea behind GP is that it is intended to work as "an automatic system whose input is a high-level statement of requirements" for a given problem and "whose output is a working program" that actually solves the problem [70]. As a recognized part of the EC field family, GP also artificially evolves individuals to fit a near-optimum solution for a predetermined problem. In the case of GP, those individuals are computer programs and/or instructions.

Since many problems can be easily recast as a search for a computer program, Koza states that GP "can potentially solve a wide range of types of problems, including problems of control, classification, system identification, and design" [70]. Design, specially, is "usually viewed as requiring creativity and human intelligence".

[4] `http://www.springer.com/computer/ai/journal/10710`

Glossary of Evolutionary Algorithm terms

Collet [73], based on Fogel [74], brought some historical details about EC: in early 1950s, when the first computers came out of research labs[5], EC had about ten independent beginnings in Australia, United States and Europe; however, Artificial Life and Artificial Evolution only came of age in the 1990s, when computers were finally powerful enough to find interesting results.

From EC field area, all EAs share the same evolutionary steps. Each one will progressively breed a population typically of thousands of randomly created candidate solutions over a series of generations. Using the Darwinian principle of natural selection, recombination (crossover), mutation, gene duplication, gene deletion, certain mechanisms of developmental biology, and firmly based on stochastic decisions, all EAs will eventually breed a most fit individual to be called the solution of the problem.

Because of their analogy with living beings, there are several terms in EC that are not proper of Computer Science. With the help of the work of [75], [13], among others, we can detach some important concepts for any EA, including GP, to end up with a mini glossary of terms borrowed from Biology and Genetics:

Individual – a candidate or potential solution to the problem being optimized.

Chromosome, genome, genotype – the representation or encoding for an individual within the search space of a solution, specific to the problem to be solved. Commonly, it is a vector which contains data that is supposed to be enough to understand the solution if you know the phenotype. Typically is a vector of numbers (binaries or real numbers), but its form is a key point in classifying historical EAs. All genetic operators will be performed onto this encoded form.

Gene – Each element that, when combined with others, arises the chromosome (aka genome).

Phenotype – The practical meaning of a chromosome. Used as a key to understand information from the genotype (chromosome). When a genotype is turned into a phenotype, the candidate solution acquires its full meaning.

Population – the set of all individuals. Typically, it does not change its size from one generations into another. Within the lifetime of a generation it could grow up, though, to be reduced in the end of the cycle.

Landscape or environment – the "location" where individuals survive; it represents the problem to be solved. Typically, it is the surface of some evaluation function. It is also described as the search space.

[5] IBM 650 in 1953.

Fitness – a characteristic of any individual that is a measure of how adapted each one is to survive in a predetermined environment. More specifically, it is the measure of quality of a given candidate solution regarding the problem to be solved, allowing comparisons between different individuals. Note that this is a crucial characteristic: the best fit individual within the population when generation cycles come to an end is the evolved near-optimum solution.

Evaluation or fitness function – this is a key part of an EA. It allows to rate an individual (determining its fitness) and is specific to the problem to be solved. The genotype representation of an individual must be turned into a phenotype representation in order to be evaluated. Typically, it determines the landscape (search space) of the problem.

Generation or evolutionary loop – a loop iteration executed by the EA. Figure 3.2 allows the visualization of a generation. It typically includes the following stages: selection of individuals to generate new ones (known as *parents*), performance of proper genetic operators to generate new individuals, evaluation of new individuals (aka *offspring* or *children*), and reduction of population to its original size.

Selection – typically based on fitness (there are some cases where it is random), it menas the mechanism to select individuals to generate new ones. Several strategies exist to select individuals to generate new ones. Those *parent* individuals in general are among the best fit individuals from its generation. Another use of a selection strategy is when it is time to reduce population; individuals that are departing from existence are commonly among the worst fit individuals from its generation.

Crossover or recombination – *genetic or variation operator.* If this random operator is enrolled to be performed, it will mix one or more genes from some selected *parent* individuals (typically two) to generate commonly up to two *offspring* individuals. There are several strategies to do so.

Mutation – *genetic or variation operator.* If this random operator is enrolled to be performed, it will change one or more genes (altering storage data) from an unique selected individual to generate an unique offspring. There are several strategies to do so.

Reproduction or elitism – simply the copy of an individual from a generation into another. In some cases, the individual is reproduced into the offspring, without warranty if it is going to survive the population reduction stage.

Reduction or replacement – intending for imitating the way of predators, natural disasters, diseases, and other catastrophes, population is reduced at the end of a generation lifetime. This reduction, however, is always bounded to keep population at its original size. This limitation arise from some issues on computational implementations for EAs and is never thought as a drawback. There are also several strategies to do so.

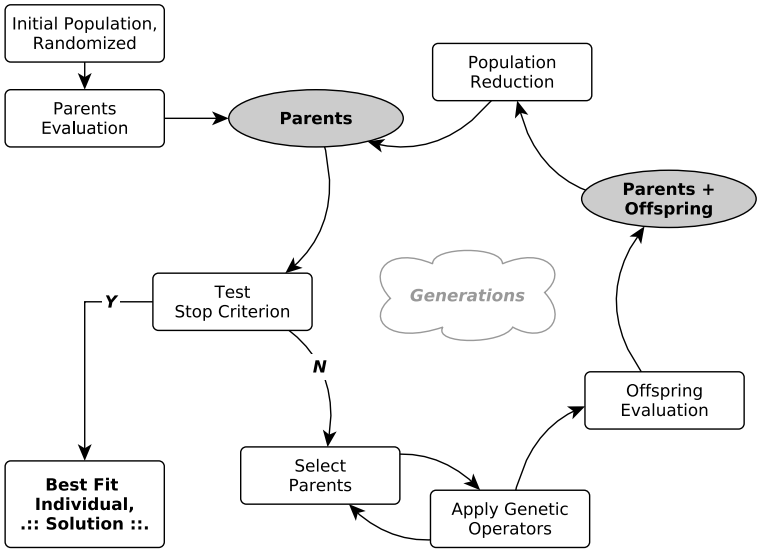

Figure 3.2: EA flowchart: each loop iteration is a generation; adapted from [75]

De Jong [40] shows in his work several discussions on how to parametrize EAs. We should not enter in details here, but there are lots of implications on each decision to be taken and on each regularization of a parameter to be done, e.g. size of a population, number of generations, termination criteria, probabilities for performing genetic operators, the chosen reduction strategy. De Jong presents some analysis that are worthy to understand when one is in the process of specialisation about EAs.

A brief field guide to Genetic Programming

Besides all potentialities from GP, a brief discussion is presented in this section about what distinguishes GP from other EAs and why is feasible to think about CASM through it. Perhaps, the most distinguish feature, and the one which gives more potential to GP, is the classic representation of an individual: the genome is represented by an abstract syntax tree (see Figure 3.3). Every time it is said that GP evolves computer programs, one must think of those trees instead of lines of code. This kind of malleable-arrangement and variable-length structure made GP a really versatile technique.

In order to clarify the necessary terminology when addressing GP, an arbitrary syntax tree has the structure of a special case of graph known simply as "trees". When representing algebraic expressions, trees are graphical representations of the prefix notation, also known as the Reverse Polish Notation (RPN). All entities in a tree is defined as a "node". The first entity of a tree is identified by being at the top — and, most important, without connections coming to it — is known as the "root". All terminations, *i.e.*, the ones without connections coming out of them, are named "leaves". Entities that have connections coming in and out are named "internal nodes".

Symbolic regression, also known as data modelling, is the main concern of this work. Based on [13], lets take a simple example about GP executions.

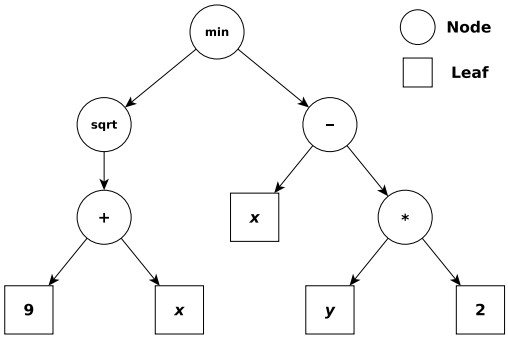

Figure 3.3: Example of an abstract syntax tree for the computation "$\min(\sqrt{9+x},\, x-2\,y)$", or, in RPN, `(min (sqrt (+ 9 x)) (- x (* 2 y)))`

Table 3.2: Preparation step for function approximation; adapted from [13].

Objective:	Find program whose output matches $x^2 + x + 1$ within $-1 \leq x \leq +1$ (data modelling)
Function Set (nodes):	$+$, $-$, % (protected division), $*$
Terminal Set (leafs):	x and random integer constants within $[-5, 5]$
Fitness:	sum of absolute errors for $x \in \{-1.0, -0.9, \ldots 0.9, 1.0\}$ ("area" between discretized curves)
Selection:	*Must define strategy before run, but it is regardless now*
Initial population strategy:	*Must define strategy before run, but it is regardless now*
Parameters:	Population size 4; no tree size limits; *probabilities for crossover, mutation, and reproduction must be defined before run, but they are regardless now*
Termination criterion:	*Must define strategy before run, but it is regardless now*

Table 3.2 is the preparatory step where is given to GP "a high-level statement of requirements" [70], including the *set of primitives* (function and terminal sets) that will constrain the search space. Figure 3.4 summarizes the evolutionary loop that took place, and Table 3.3 shows the fitness of every individual from each generation. Note that this is a minimization problem, so the lower is the fitness, the better is the individual as a candidate solution.

As one can note from this simple run, GP can take measured points from some observed phenomena (here the function $x^2 + x + 1$ was sampled at $x \in \{-1.0, -0.9, \ldots 0.9, 1.0\}$ to evaluate candidate solutions) and search for a function that could explain them within some sort of interval (here, the domain is limited to $-1 \leq x \leq +1$). From a random initial population and using genetic operators, GP was able to search for a solution inside that landscape, starting from some high-level directives (Table 3.2). To work effectively, GP must have function sets with the closure property [12], which means that functions must exhibit type consistency and evaluation safety (e.g. the operator %, protected division, typically forces $x \div 0 = 1$, for any x). Also, the primitive set (functions and terminals) must be sufficient, which means that "the set of all the possible recursive compositions of primitives includes at least one solution" [13].

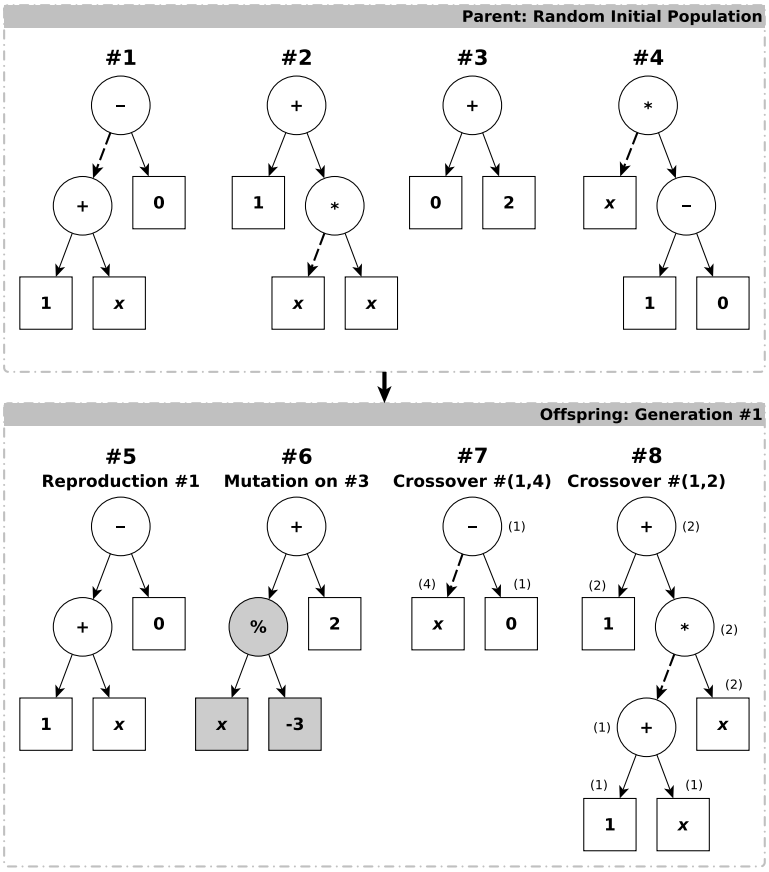

Figure 3.4: Summary of this simple run (see Table 3.2); darker elements were randomly generated; dashed arrows indicates cut points to mix genes in related crossovers; adapted from [13].

Table 3.3: Summary of this simple run (see Figure 3.4); note that there is a match (found solution) in generation 1; adapted from [13]

Generation	Individual	Function represented	Fitness	Note
0	1	$x + 1$	7.7	
0	2	$x^2 + 1$	11.0	
0	3	2	18.0	
0	4	x	28.7	
1	5	$x + 1$	7.7	
1	6	$-\frac{x}{3} + 2$	20.6	
1	7	x	28.7	
1	8	$x^2 + x + 1$	0.0	$\Leftarrow$ **match!**

There are several other issues that are not discussed here. The aim of this section is to present some potentialities of GP, even with the use of a mostly graphical example.

3.9 Precision on measurements

One of the main concerns to engineers is about the robustness of a method when the input measurements are taken with not ideal level of precision. This situation is very common in reality, as one could suppose, because measurements are taken with sensors and devices which are bounded to acceptable tolerances.

By the definition of a well-posed problem, a minor variation on input data will result in minor variations of output data. This reflects a natural robustness against low order errors. As a stochastic process, a measurement is subject up to several random independent variables with possible different distributions which will likely present a mean described by a normal distribution (see central limit theorem [76]). In order to simulate those lacks of absoluteness, *i.e.*, to presuppose data would contain tolerated noise, it is feasible to assume that noise would mostly present itself by being normally distributed (a Gaussian distribution).

The proper procedure to add noise to data which is intended to reflect reality is somehow tricky. Parametrization for noise distributions is both complicated and dependent of experiments, once researches could not tell without prior extended tests how fit are their measurement processes and that information can change from one experiment to another.

As a possible solution to the noise dilemma, signal processing are has the signal-to-noise ratio (SNR) metric which could be defined as the dimensionless ratio of the signal power to the noise power contained in a recording and is usually measured in decibels (dB). Engineers and scientists would use this ratio to parametrise performances of signal processing systems when the noise is normally distributed [77].

Therefore, this work adopts white Gaussian noise (WGN), defined by its SNR, to be added to all mathematical generated data when tolerance to errors is needed to be explored. White noise is an abstraction that is not feasible to exist, "but engineers frequently use it to describe noise that has a power spectrum that extends well beyond the signal's bandwidth" [77]. Here, WGN is used to simulate possible measurement errors. After both coordinates are taken randomly and the known function is used to generate the simulated data, noise is added to all of them in order to respect mathematical relationships and reflect measurement errors.

3.10 Discussion

The knowledge on subjects shown in this chapter is necessary to follow the developed method. The use of the Ritz-Galerkin method to project the solution of a linear differential equation onto a Jabobi polynomial orthogonal basis

is the core to the following chapters. With the exception of GP, the other aforementioned subjects are needed to complete mathematical derivations.

The proposed method takes into account that a desired GP application could evolve different types of linear differential equations. For that reason, the method itself should be obliged to handle generic linear differential forms.

Note that all mathematical developments to be presented do not consider existing tolerances on measurements. Discussions on such measurement errors are only presented in Section 8.3, part of the last chapter of this thesis.

Part II

Proposed Method

Chapter 4

Ordinary Differential Equations

4.1 Proposed method

In this chapter, the proposed method to solve linear ordinary differential equations (LODE) as defined in Equation (3.4) is presented. As could be inferred and for the sake of clarification, by using Jacobi polynomials as basis functions, the proposed method solves differential equations building a polynomial approximation for the differential solution. The option here is to use Jacobi polynomials parametrized with $\alpha = \beta = 0$ in order to be asymptotically similar to Legendre polynomials. This option is due to some properties presented by those polynomials, as (a) orthogonality to achieve a finite span of a complete basis for a function space — Section 3.2 —; (b) derivatives identity as other Jacobi polynomials — Equations (3.18) and (3.23) —; (c) and the fact that Jacobi-Legendre polynomials have weight function for inner products as $w(x) = (1-x)^0 (1+x)^0 = 1$ — Equation (3.19) —. Those properties are essential to favour envisioned computational implementations. As can be seen throughout this chapter, basis functions are not bounded to respect auxiliary conditions (initial or boundary) as in classical Galerkin-based methods. Instead, auxiliary conditions have a customized own treatment inspired by concepts of ordinary system of equations and the way finite difference methods [78] deals with them, *i.e.*, to include information about conditions into the matrix formulation of the problem.

An important subject to keep in mind is that, as stated by [17, pp.453], in order to implement a true Galerkin process, it is necessary to carry out integrations over domains. In other words, even if the problem refers to itself as an initial value problem (conditions known at one point only), it is necessary to define a full interval for the domain of calculation in order to use Galerkin method. This fact could lead to piecewise approximate solutions, as for FEM. Note that functions which result in improper integrals shall be avoided [17, p. 43]. However, depending on integration method, points of singularity could be avoided and a feasible approximation achieved.

4.2 The unidimensional case

Developments start from the unidimensional (univariate) case, a linear ordinary differential equation in the form of Equation (3.3). Assuming mapped Jacobi polynomials as functions to build an orthogonal basis $\mathcal{B}$ and using Equation (3.3), then Equation (3.13) is developed to achieve Equation (4.1).

$$\sum_{m=0}^{N} \tilde{u}_m \left\langle \phi_n(x),\, \mathcal{L}[\phi_m(x)] \right\rangle = \left\langle \phi_n(x),\, s(x) \right\rangle \quad \Leftrightarrow$$

$$\sum_{m=0}^{N} \tilde{u}_m \left\langle P_n^{(\alpha,\beta)}(\xi),\, \mathcal{L}[P_m^{(\alpha,\beta)}(\xi)] \right\rangle = \left\langle P_n^{(\alpha,\beta)}(\xi), s\,(x) \right\rangle \quad \Leftrightarrow$$

$$\sum_{m=0}^{N} \tilde{u}_m \left\langle P_n^{(\alpha,\beta)}(\xi),\, \mathcal{L}[P_m^{(\alpha,\beta)}(\xi)] \right\rangle = \left\langle P_n^{(\alpha,\beta)}(\xi),\, s(x) \right\rangle \quad \Leftrightarrow$$

$$\sum_{m=0}^{N} \tilde{u}_m \left\langle P_n^{(\alpha,\beta)}(\xi),\, \sum_{q=0}^{Q} k_q\,(x)\, \frac{d^q}{dx^q}\, P_m^{(\alpha,\beta)}\,(\xi) \right\rangle = \left\langle P_n^{(\alpha,\beta)}(\xi),\, s(x) \right\rangle \qquad (4.1)$$

where the index $n = 0 \ldots N$ identifies the n^{th} equation from this system of $N+1$ equations; and N is the polynomial degree intended for the approximation by the TGE.

4.3 Developments

In this section, two possibilities are derived here. The first one stands for inner products over the interval $a \leq x \leq b$, as could be followed in Equation (4.2).

$$\sum_{m=0}^{N} \tilde{u}_m \int_a^b P_n^{(0,0)}(\xi(x)) \cdot \sum_{q=0}^{Q} k_q(x)\, \frac{d^q}{dx^q} P_m^{(0,0)}(\xi(x))\, dx$$

$$= \int_a^b P_n^{(0,0)}(\xi(x)) \cdot s(x)\, dx \quad \Leftrightarrow$$

$$\sum_{m=0}^{N} \tilde{u}_m \int_a^b P_n^{(0,0)}(\xi(x)) \cdot \sum_{q=0}^{Q} k_q(x)\, \frac{\Gamma(m+q+1)}{(b-a)^q \Gamma(m+1)}\, P_{m-q}^{(q,q)}(\xi(x))\, dx$$

$$= \int_a^b P_n^{(0,0)}(\xi(x)) \cdot s(x)\, dx \quad \Leftrightarrow$$

$$\sum_{m=0}^{N} \tilde{u}_m \sum_{q=0}^{Q} \int_a^b P_n^{(0,0)}(\xi(x)) \cdot k_q(x) \, \frac{\Gamma(m+q+1)}{(b-a)^q \Gamma(m+1)} \, P_{m-q}^{(q,q)}(\xi(x)) \, dx$$

$$= \int_a^b P_n^{(0,0)}(\xi(x)) \cdot s(x) \, dx \quad (4.2)$$

The second possibility is a variation achieved by exploiting change of variables (Section 3.6) to enable inner products over the interval $-1 \leq \xi \leq +1$, as could be followed in Equation (4.3).

$$\sum_{m=0}^{N} \tilde{u}_m \int_{-1}^{+1} P_n^{(0,0)}(\xi) \cdot \sum_{q=0}^{Q} k_q(x(\xi)) \, \left(\frac{2}{b-a}\right)^q \frac{d^q}{d\xi^q} P_m^{(0,0)}(\xi) \, \frac{b-a}{2} \, d\xi$$

$$= \int_{-1}^{+1} P_n^{(0,0)}(\xi) \cdot s(x(\xi)) \, \frac{b-a}{2} \, d\xi \quad \Leftrightarrow$$

$$\frac{b-a}{2} \sum_{m=0}^{N} \tilde{u}_m \sum_{q=0}^{Q} \left(\frac{2}{b-a}\right)^q \int_{-1}^{+1} P_n^{(0,0)}(\xi) \cdot k_q(x(\xi)) \, \frac{\Gamma(m+q+1)}{2^q \Gamma(m+1)} \, P_{m-q}^{(q,q)}(\xi) \, d\xi$$

$$= \frac{b-a}{2} \int_{-1}^{+1} P_n^{(0,0)}(\xi) \cdot s(x(\xi)) \, d\xi \quad \Leftrightarrow$$

$$\frac{b-a}{2} \sum_{m=0}^{N} \tilde{u}_m \sum_{q=0}^{Q} \left(\frac{2}{b-a}\right)^q \frac{1}{2^q} \int_{-1}^{+1} P_n^{(0,0)}(\xi) \cdot k_q(x(\xi)) \, \frac{\Gamma(m+q+1)}{\Gamma(m+1)} \, P_{m-q}^{(q,q)}(\xi) \, d\xi$$

$$= \frac{b-a}{2} \int_{-1}^{+1} P_n^{(0,0)}(\xi) \cdot s(x(\xi)) \, d\xi \quad \Leftrightarrow$$

$$\sum_{m=0}^{N} \tilde{u}_m \sum_{q=0}^{Q} \left(\frac{1}{b-a}\right)^q \int_{-1}^{+1} P_n^{(0,0)}(\xi) \cdot k_q(x(\xi)) \, \frac{\Gamma(m+q+1)}{\Gamma(m+1)} \, P_{m-q}^{(q,q)}(\xi) \, d\xi$$

$$= \int_{-1}^{+1} P_n^{(0,0)}(\xi) \cdot s(x(\xi)) \, d\xi \quad (4.3)$$

From both options, Equations (4.2) and (4.3), the latter is preferred to the former. This decision has to do with strategies of implementation to improve final execution time performances. Note that Monte Carlo scheme is here adopted to handle those integrations, therefore integrands would have to be

evaluated for a large set of points. The former option means that for every domain inside every cycle of execution, the full integrand must be evaluated that large number of times. Instead, the latter option requires $P_n^{(0,0)}(\xi)\, P_{m-q}^{(q,q)}(\xi)$ to be evaluated a large number of times only once per execution, and every domain inside every cycle of execution has to handle just $k_q(x(\xi))$ coefficients.

Thence, Equation (4.3) could be converted to a "$A\,x = b$"-like matrix equation using identity in Equation (3.14), as seen in Equation (4.4):

$$\underbrace{\begin{pmatrix} 1\times Q^+ \cdot \int P_0 & Q^+ \times N^+ \\ \hline 1\times Q^+ \cdot \int P_1 & Q^+ \times N^+ \\ \hline \vdots & \\ \hline 1\times Q^+ \cdot \int P_N & Q^+ \times N^+ \end{pmatrix}}_{N^+\times N^+} \cdot \begin{pmatrix} \tilde{u}_0 \\ \tilde{u}_1 \\ \vdots \\ \tilde{u}_N \end{pmatrix} = \begin{pmatrix} \int_{-1}^{+1} P_0^{(0,0)}(\xi)\cdot s(x(\xi))\; d\xi \\ \int_{-1}^{+1} P_1^{(0,0)}(\xi)\cdot s(x(\xi))\; d\xi \\ \vdots \\ \int_{-1}^{+1} P_N^{(0,0)}(\xi)\cdot s(x(\xi))\; d\xi \end{pmatrix} \tag{4.4}$$

where $Q^+ = Q + 1$ and $N^+ = N + 1$; and the $N^+ \times N^+$ coefficient matrix is detailed by Equation (4.5).

$$\begin{pmatrix} \left(1\ \frac{1}{b-a}\ \cdots \frac{1}{(b-a)^Q}\right) \left[\int_{-1}^{+1} P_0^{(0,0)}(\xi)\cdot \begin{pmatrix} D_{0,0}(\xi) & \cdots & D_{0,N}(\xi) \\ \vdots & \ddots & \vdots \\ D_{Q,0}(\xi) & \cdots & D_{Q,N}(\xi) \end{pmatrix} d\xi\right] \\ \hline \left(1\ \frac{1}{b-a}\ \cdots \frac{1}{(b-a)^Q}\right) \left[\int_{-1}^{+1} P_1^{(0,0)}(\xi)\cdot \begin{pmatrix} D_{0,0}(\xi) & \cdots & D_{0,N}(\xi) \\ \vdots & \ddots & \vdots \\ D_{Q,0}(\xi) & \cdots & D_{Q,N}(\xi) \end{pmatrix} d\xi\right] \\ \hline \vdots \\ \hline \left(1\ \frac{1}{b-a}\ \cdots \frac{1}{(b-a)^Q}\right) \left[\int_{-1}^{+1} P_N^{(0,0)}(\xi)\cdot \begin{pmatrix} D_{0,0}(\xi) & \cdots & D_{0,N}(\xi) \\ \vdots & \ddots & \vdots \\ D_{Q,0}(\xi) & \cdots & D_{Q,N}(\xi) \end{pmatrix} d\xi\right] \end{pmatrix} \tag{4.5}$$

where $D_{q,m} = k_q\big(x(\xi)\big)\, \frac{\Gamma(m+q+1)}{\Gamma(m+1)}\, P_{m-q}^{(q,q)}(\xi)$, for the sake of readability.

Note that when using Monte Carlo integration to build the coefficient matrix, the identity $\int_{-1}^{+1} f(\xi)\; d\xi = \frac{2}{H} \sum_{h=0}^{H} f(\xi_h)$ is valid for a large set of ξ_h random points.

4.4 Solving ODEs

In order to solve differential problems, there is need for prior knowledge on the unknown solution in the form of complimentary equations, known generally as auxiliary conditions. In the literature, those conditions are known as initial value conditions or boundary conditions, depending on how they are presented for the respective problem.

The way to build or choose basis functions is implicit to Galerkin method. All functions in the basis span must satisfy some auxiliary conditions, usually linear homogeneous boundary conditions [67]. Using Jacobi polynomials as basis functions does not always respect this requirement. When using Jacobi polynomials as basis, some differential problems present themselves with their initial or boundary conditions neglected, leading to ill-posed problems (the coefficient matrix could be either rank deficient or ill-conditioned). Actually, this is the case for the vast majority of real world related problems.

Auxiliary conditions

Conditions refers to known values assumed by the unknown solution of a differential equation taken over a predetermined boundary or from the initial state related to the problem itself. Literature classifies them within some types of so called boundary conditions (BC): *Dirichlet BC* (1$^{\text{st}}$-type, function values on the boundary), *Neumann BC* (2$^{\text{nd}}$-type, derivative values for the function on the boundary), *Cauchy BC* (same as imposing both a Dirichlet and a Neumann boundary condition at the same point on boundary, sometimes called initial value conditions), *Robin BC* (3$^{\text{rd}}$-type, linear combination of function values and derivative values for the function on the boundary), *mixed BC* (different conditions on disjoint parts of the boundary).

Those are conditions that augment their respective differential equation and that the solution must satisfy on the boundary (ideally to ensure the existence of an unique solution). Also, they all could be described as a lesser order differential equations themselves, as in Equation (4.6):

$$\begin{aligned} \mathcal{H}_c\left[u(x)\right]_{x=X_c} &\approx \mathcal{H}_c\left[\hat{u}(x)\right]_{x=X_c} = V_c \\ \left\{\sum_{w=0}^{W} h_{w,c}(x)\,\frac{d^w}{dx^w}\hat{u}(x)\right\}_{x=X_c} &= V_c \end{aligned} \tag{4.6}$$

where c is a subscript that refers to the c^{th} known condition of the differential problem; $\mathcal{H}_c$ is a linear differential operator of lesser order than $\mathcal{L}$; W is defined here as the maximum order of all known conditions (normally, the order of the related differential problem minus one); X_c is the point within the domain of calculation at which we can determine the value of $\mathcal{H}_c[u(x)] = V_c$; and $h_{w,c}(x)$ is the w^{th} coefficient for the finite sum of $\mathcal{H}_c$ terms.

Equation (4.6) means that, at $x = X_c$, $u(x)$ subject to $\mathcal{H}_c$ has a known value V_c. Regarding boundary of the domain, note that if an order 0 is defined for a given condition, that could define a Dirichlet boundary condition. If the

order is a natural number greater than 0, then a Neumann boundary condition could be defined. More than one $h_{w,c}(x) \neq 0$ in each $\mathcal{H}_c$ can be used to define Robin boundary conditions. If all x_c are at the same point within the domain and all $\mathcal{H}_c$ have distinct orders (as in initial value problems), those conditions could be Cauchy-like conditions.

Expanding the left hand side of Equation (4.6) and substituting $u(x)$ by the trial function $\hat{u}(x)$ in Equation (3.11), then Equation (4.7) is derived:

$$
\begin{aligned}
&\mathcal{H}_c \left[\hat{u}(x)\right]_{x=X_c} = V_c \quad \Leftrightarrow \\
&\mathcal{H}_c \left[\sum_{m=0}^{N} \tilde{u}_m \, P_m^{(0,0)}\left(\xi(x)\right)\right]_{x=X_c} = V_c \quad \Leftrightarrow \\
&\left\{\sum_{w=0}^{W} h_{w,c}(x) \, \frac{d^w}{dx^w}\left(\sum_{m=0}^{N} \tilde{u}_m \, P_m^{(0,0)}\left(\xi(x)\right)\right)\right\}_{x=X_c} = V_c \quad \Leftrightarrow \\
&\left\{\sum_{w=0}^{W} h_{w,c}(x) \sum_{m=0}^{N} \tilde{u}_m \left(\frac{d^w}{dx^w} P_m^{(0,0)}\left(\xi(x)\right)\right)\right\}_{x=X_c} = V_c \quad \Leftrightarrow \\
&\sum_{w=0}^{W} h_{w,c}(X_c) \sum_{m=0}^{N} \tilde{u}_m \left(\frac{d^w}{dx^w} P_m^{(0,0)}\left(\xi(x)\right)\Big|_{x=X_c}\right) = V_c \quad \Leftrightarrow \\
&\sum_{w=0}^{W} h_{w,c}(X_c) \sum_{m=0}^{N} \tilde{u}_m \, \frac{\Gamma(m+w+1)}{(b-a)^w \Gamma(m+1)} \, P_{m-w}^{(w,w)}\left(\xi(X_c)\right) = V_c \;\Leftrightarrow \\
&\sum_{m=0}^{N} \tilde{u}_m \sum_{w=0}^{W} \frac{1}{(b-a)^w} \, h_{w,c}(X_c) \, \frac{\Gamma(m+w+1)}{\Gamma(m+1)} \, P_{m-w}^{(w,w)}\left(\xi(X_c)\right) = V_c
\end{aligned}
\tag{4.7}
$$

where $\xi(X_c) = 2\,\frac{X_c - a}{b-a} - 1$. Note that the identity in Equation (3.23) should be used to achieve this result.

The matrix formulation for the Equation (4.7) is given in Equation (4.8).

$$
\begin{pmatrix}
\left(1 \;\; \frac{1}{b-a} \;\; \ldots \left(\frac{1}{b-a}\right)^W\right) \begin{pmatrix} H_{0,0}^{(1)}(X_1) & \cdots & H_{W,0}^{(1)}(X_1) \\ \vdots & \ddots & \vdots \\ H_{0,N}^{(1)}(X_1) & \cdots & H_{W,N}^{(1)}(X_1) \end{pmatrix} \\
\hline
\vdots \\
\hline
\left(1 \;\; \frac{1}{b-a} \;\; \ldots \left(\frac{1}{b-a}\right)^W\right) \begin{pmatrix} H_{0,0}^{(C)}(X_C) & \cdots & H_{0,N}^{(C)}(X_C) \\ \vdots & \ddots & \vdots \\ H_{W,0}^{(C)}(X_C) & \cdots & H_{W,N}^{(C)}(X_C) \end{pmatrix}
\end{pmatrix}
\cdot
\begin{pmatrix} \tilde{u}_0 \\ \tilde{u}_1 \\ \vdots \\ \tilde{u}_N \end{pmatrix}
= \begin{pmatrix} V_1 \\ V_2 \\ \vdots \\ V_C \end{pmatrix}
\tag{4.8}
$$

where $H^{(c)}_{w,m}(X_c) = h_{w,c}(X_c)\,\frac{\Gamma(m+w+1)}{\Gamma(m+1)}\,P^{(w,w)}_{m-w}\,(\xi(X_c))$, for the sake of readability; and the coefficient matrix here is of order $C \times N^+$ (registering C conditions).

Using the fact that when well-conditioned, a system of equations is formed by coupled equations which refer to the same solution, this work proposes an approach to deal with auxiliary conditions: to substitute equations from the GSE (bottom rows in its matrix formulation, in Equations (4.4) and (4.5)), by equations derived from those conditions, those rows built as presented in Equation (4.8).

The final Galerkin-like system is presented in Equation (4.9).

$$\underbrace{\begin{pmatrix} 1\times Q^+ \cdot \int P_0 & Q^+ \times N^+ \\ \hline 1\times Q^+ \cdot \int P_1 & Q^+ \times N^+ \\ \hline \vdots & \\ \hline 1\times Q^+ \cdot \int P_{N-C} & Q^+ \times N^+ \\ \hline 1 \times W^+ \cdot & W^+ \times N^+ \quad (\#1) \\ \hline \vdots & \\ \hline 1 \times W^+ \cdot & W^+ \times N^+ \quad (\#C) \end{pmatrix}}_{N^+ \times N^+} \cdot \begin{pmatrix} \tilde{u}_0 \\ \tilde{u}_1 \\ \vdots \\ \tilde{u}_N \end{pmatrix} = \begin{pmatrix} \int_{-1}^{+1} P_0^{(0,0)}(\xi)\cdot s(x(\xi))\,d\xi \\ \int_{-1}^{+1} P_1^{(0,0)}(\xi)\cdot s(x(\xi))\,d\xi \\ \vdots \\ \int_{-1}^{+1} P_{N-C}^{(0,0)}(\xi)\cdot s(x(\xi))\,d\xi \\ \hline V_1 \\ \vdots \\ V_C \end{pmatrix} \tag{4.9}$$

Example

Two examples are presented in this section, one from an under-damped oscillator problem and the other from a Poisson equation for electrostatics.

Under-damped oscillator An oscillator problem is identified by the differential equation for a mass-spring-damper system:

$$m\,\frac{d^2}{dt^2}u(t) + b\,\frac{d}{dt}u(t) + k\,u(t) = 0$$

where $u(t)$ is the displacement of the mass in function of time; m is the mass; b is the damping coefficient; and k is the spring constant. Quantities known as the undamped angular frequency $\omega_0 = \sqrt{\frac{k}{m}}$ and the damping ratio $\zeta = \frac{b}{2\sqrt{m\,k}}$ are also useful to classify this system. The oscillator becomes under-damped

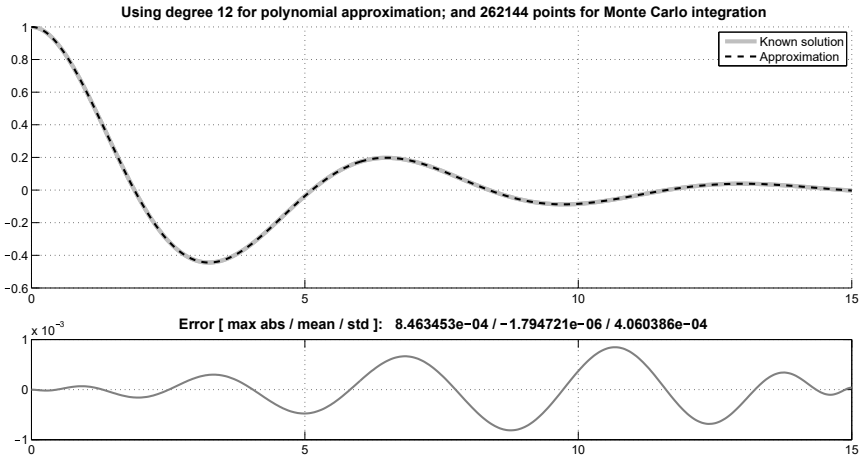

Figure 4.1: Solution to an under-damped oscillator problem, polynomial approximation of degree 12.

when $0 \leq \zeta < 1$. In this very case, the differential equation has the following known solution:

$$u(t) = e^{-\zeta\,\omega_0\,t}\left[u(0)\,\cos\left(t\,\omega_0\,\sqrt{1-\zeta^2}\right) + \frac{\zeta\,\omega_0\,u(0) + u_t(0)}{\omega_0\,\sqrt{1-\zeta^2}}\,\sin\left(t\,\omega_0\,\sqrt{1-\zeta^2}\right)\right],$$

where $u_t(0)$ stands for $\frac{d}{dt}u(t)\big|_{t=0}$.

As a numerical example, consider $m = 2\,kg$, $b = 1\,kg\,s^{-1}$ and $k = 2\,kg\,s^{-2}$. Also consider initial values $u(0) = 1$ and $u_t(0) = 0$. The known solution to this numerical example is shown in Equation (4.10):

$$u(t) = \left[\cos\left(x\,\frac{\sqrt{15}}{4}\right) + \frac{\sqrt{15}}{15}\,\sin\left(x\,\frac{\sqrt{15}}{4}\right)\right]\,\exp\left(-\frac{x}{4}\right). \tag{4.10}$$

Even that this is considered an initial value problem, the proposed method requires the definition of a domain. In this case, $0 \leq t \leq 15$ was adopted. The system of equations built by the proposed method choosing degree 12 for the polynomial approximation is shown in Equation (4.11). Figure 4.1 shows on top a joint plot of both the TGE polynomial approximation and the known solution, and on bottom the error in function of t is found by subtracting one from the other.

$$
\begin{pmatrix} \tilde{u}_0 \\ \tilde{u}_1 \\ \vdots \\ \tilde{u}_{12} \end{pmatrix} =
$$

$$
\begin{pmatrix}
4.00 & 2.67\,10^{-1} & 2.13\,10^{-1} & 2.67\,10^{-1} & 7.11\,10^{-1} & 2.67\,10^{-1} & 1.49 \\
1.46\,10^{-12} & 1.33 & 2.67\,10^{-1} & 3.56\,10^{-1} & 2.67\,10^{-1} & 9.96\,10^{-1} & 2.67\,10^{-1} \\
1.53\,10^{-5} & 1.02\,10^{-6} & 8.00\,10^{-1} & 2.67\,10^{-1} & 4.98\,10^{-1} & 2.67\,10^{-1} & 1.28 \\
-1.53\,10^{-15} & 1.53\,10^{-5} & 3.05\,10^{-6} & 5.71\,10^{-1} & 2.67\,10^{-1} & 6.40\,10^{-1} & 2.67\,10^{-1} \\
1.53\,10^{-5} & 1.02\,10^{-6} & 1.61\,10^{-5} & 6.10\,10^{-6} & 4.44\,10^{-1} & 2.67\,10^{-1} & 7.82\,10^{-1} \\
0.00 & 1.53\,10^{-5} & 3.05\,10^{-6} & 1.93\,10^{-5} & 1.02\,10^{-5} & 3.64\,10^{-1} & 2.67\,10^{-1} \\
1.53\,10^{-5} & 1.02\,10^{-6} & 1.61\,10^{-5} & 6.10\,10^{-6} & 2.75\,10^{-5} & 1.53\,10^{-5} & 3.08\,10^{-1} \\
3.14\,10^{-15} & 1.53\,10^{-5} & 3.05\,10^{-6} & 1.93\,10^{-5} & 1.02\,10^{-5} & 4.37\,10^{-5} & 2.14\,10^{-5} \\
1.53\,10^{-5} & 1.02\,10^{-6} & 1.61\,10^{-5} & 6.10\,10^{-6} & 2.75\,10^{-5} & 1.53\,10^{-5} & 7.22\,10^{-5} \\
1.28\,10^{-13} & 1.53\,10^{-5} & 3.05\,10^{-6} & 1.93\,10^{-5} & 1.02\,10^{-5} & 4.37\,10^{-5} & 2.14\,10^{-5} \\
1.53\,10^{-5} & 1.02\,10^{-6} & 1.61\,10^{-5} & 6.10\,10^{-6} & 2.75\,10^{-5} & 1.53\,10^{-5} & 7.22\,10^{-5} \\
1.00 & -1.00 & 1.00 & -1.00 & 1.00 & -1.00 & 1.00 \\
0.00 & 1.33\,10^{-1} & -4.00\,10^{-1} & 8.00\,10^{-1} & -1.33 & 2.00 & -2.80
\end{pmatrix}^{-1}
$$

$$
\begin{pmatrix}
2.67\,10^{-1} & 2.56 & 2.67\,10^{-1} & 3.91 & 2.67\,10^{-1} & 5.55 \\
1.92 & 2.67\,10^{-1} & 3.13 & 2.67\,10^{-1} & 4.62 & 2.67\,10^{-1} \\
2.67\,10^{-1} & 2.35 & 2.67\,10^{-1} & 3.70 & 2.67\,10^{-1} & 5.33 \\
1.56 & 2.67\,10^{-1} & 2.77 & 2.67\,10^{-1} & 4.27 & 2.67\,10^{-1} \\
2.67\,10^{-1} & 1.85 & 2.67\,10^{-1} & 3.20 & 2.67\,10^{-1} & 4.84 \\
9.25\,10^{-1} & 2.67\,10^{-1} & 2.13 & 2.67\,10^{-1} & 3.63 & 2.67\,10^{-1} \\
2.67\,10^{-1} & 1.07 & 2.67\,10^{-1} & 2.42 & 2.67\,10^{-1} & 4.05 \\
2.67\,10^{-1} & 2.67\,10^{-1} & 1.21 & 2.67\,10^{-1} & 2.70 & 2.67\,10^{-1} \\
2.85\,10^{-5} & 2.35\,10^{-1} & 2.67\,10^{-1} & 1.35 & 2.67\,10^{-1} & 2.99 \\
1.18\,10^{-4} & 3.66\,10^{-5} & 2.11\,10^{-1} & 2.67\,10^{-1} & 1.49 & 2.67\,10^{-1} \\
2.85\,10^{-5} & 1.86\,10^{-4} & 4.58\,10^{-5} & 1.91\,10^{-1} & 2.67\,10^{-1} & 1.64 \\
-1.00 & 1.00 & -1.00 & 1.00 & -1.00 & 1.00 \\
3.73 & -4.80 & 6.00 & -7.33 & 8.80 & -10.40
\end{pmatrix}
\cdot
\begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}
\tag{4.11}
$$

Finally, Equation (4.12) presents the coefficients of the TGE which is the approximate solution to this under-damped oscillator problem. Sub-indices relate each coefficient to a Jacobi polynomial, as $P_0^{(0,0)}(\xi(x))$, $P_1^{(0,0)}(\xi(x))$, and so on, up to $P_{12}^{(0,0)}(\xi(x))$.

$$
\begin{pmatrix} u_0 & u_1 & \cdots & u_{12} \end{pmatrix}^T =
$$

$$
\begin{pmatrix}
3.49\,10^{-2} & -1.15\,10^{-1} & 2.60\,10^{-1} & -3.87\,10^{-1} & 4.63\,10^{-1} & -2.68\,10^{-1} & -2.23\,10^{-1} \\
3.42\,10^{-1} & -6.98\,10^{-2} & -7.91\,10^{-2} & 3.85\,10^{-2} & 5.23\,10^{-3} & -5.34\,10^{-3}
\end{pmatrix}^T
\tag{4.12}
$$

Simplifying the expanded equation with the aid of Maxima CAS software [79], the polynomial approximation solution is shown in Equation (4.13):

$$
\begin{aligned}
\hat{u}(t) = {} & -1.11\,10^{-10}\,x^{12} + 1.05\,10^{-8}\,x^{11} - 4.18\,10^{-7}\,x^{10} + 9.22\,10^{-6}\,x^{9} \\
& - 1.19\,10^{-4}\,x^{8} + 8.80\,10^{-4}\,x^{7} - 2.95\,10^{-3}\,x^{6} - 1.49\,10^{-3}\,x^{5} \\
& + 2.35\,10^{-2}\,x^{4} + 8.84\,10^{-2}\,x^{3} - 5.01\,10^{-1}\,x^{2} + 3.47\,10^{-9}\,x + 1
\end{aligned}
\tag{4.13}
$$

Poisson electrostatic Adapted from [80, pp.210], consider the following spherical problem in Equation (4.14):

$$\nabla^2\varphi(r) = -4\pi\,\rho(r), \tag{4.14}$$

where φ is the electrostatic potential in function of radius r; and

$$\rho(r) = Q\sqrt{\left(\frac{\eta}{\pi}\right)^3}\,\exp\left(-\eta\,r^2\right)$$

is a static spherically symmetric Gaussian charge density, centred at the origin in real space.

Note that the differential equation now is ordinary with respect to radius, once both the Laplace operator in spherical coordinates and the spherical symmetry redefines the problem to be solved, as shown in Equation (4.15):

$$\frac{d^2}{dr^2}\varphi(r) + \frac{2}{r}\frac{d}{dr}\varphi(r) = -4\pi\,Q\sqrt{\left(\frac{\eta}{\pi}\right)^3}\,\exp\left(-\eta\,r^2\right). \tag{4.15}$$

When considering the initial value $\varphi(0) = 2\,Q\sqrt{\frac{\eta}{\pi}}$, the known solution [80] for this problem is

$$\varphi(r) = \frac{1}{r}\,Q\,\operatorname{erf}\left(\sqrt{\eta}\,r\right).$$

As a numerical example, consider the electric charge $Q = 1$ and the Gaussian parameter $\eta = 0.5$. The known solution is shown in Equation (4.16):

$$\varphi(r) = \frac{1}{r}\,\operatorname{erf}\left(\frac{\sqrt{2}}{2}\,r\right) \tag{4.16}$$

Also, this example is considered an initial value problem. The proposed method requires the definition of a domain, therefore $0 \leq r \leq 10$ was adopted. The system of equations built by the proposed method choosing degree 12 for the polynomial approximation. Figure 4.2 shows on top a joint plot of both the TGE polynomial approximation and the known solution, and on bottom the error in function of r is found by subtracting one from the other.

Equation (4.17) presents the coefficients of the TGE which is the approximate solution to this Poisson equation for electrostatics. Sub-indices relate each coefficient to a Jacobi polynomial, as $P_0^{(0,0)}(\xi(x))$, $P_1^{(0,0)}(\xi(x))$, and so on, up to $P_{12}^{(0,0)}(\xi(x))$.

$$\begin{pmatrix} \varphi_0 & \varphi_1 & \cdots & \varphi_{12} \end{pmatrix}^T = \\ \begin{pmatrix} 2.88\,10^{-1} - 3.21\,10^{-1} & 1.90\,10^{-1} - 6.58\,10^{-2} - 6.21\,10^{-3} & 2.92\,10^{-1} - 2.43\,10^{-2} \\ 1.14\,10^{-2} - 1.80\,10^{-3} - 2.08\,10^{-3} & 2.01\,10^{-3} - 7.56\,10^{-4} & 8.93\,10^{-5} \end{pmatrix}^T \tag{4.17}$$

Simplifying the expanded equation, the polynomial approximation solution is shown in Equation (4.18):

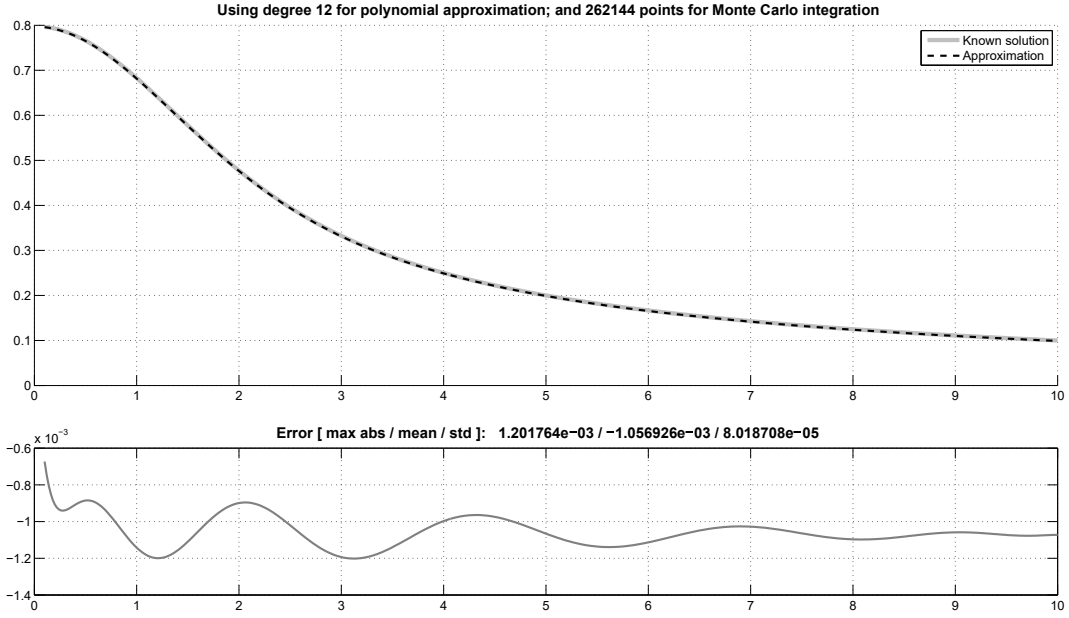

Figure 4.2: Solution to a Poisson equation for electrostatic subject to a static spherically symmetric Gaussian charge density, polynomial approximation of degree 12.

$$\begin{aligned}\hat{\varphi}(r) = {} & 0.059\,r^{12} - 0.260\,r^{11} + 0.194\,r^{10} + 0.485\,r^{9} - 0.768\,r^{8} \\ & + 0.077\,r^{7} + 0.437\,r^{6} - 0.291\,r^{5} + 0.137\,r^{4} - 0.165\,r^{3} + 0.192\,r^{2} \\ & - 0.195\,r + 0.197 \end{aligned} \quad (4.18)$$

Extra bits

Lets remember the example in Figure 1.2 from Chapter 1. This section is aimed to present results from the proposed method applied to that same specific ODE:

$$\frac{d^2}{dx^2}y(x) + \frac{d}{dx}y(x) = 0$$

with respect to different set of initial conditions, one for the "left" system:

$$\left.\frac{d}{dx}y(x)\right|_{x=0} = 1 \text{ and } y(0) = 0;$$

and other for the "right" system:

$$\left.\frac{d}{dx}y(x)\right|_{x=0} = -1 \text{ and } y(0) = 0.$$

From the same differential model, two different conditions achieve two different solutions, one related to the "left" system (see Figure 4.3):

$$\hat{y}(x) = 5.15\,10^{-3}\,x^5 - 3.86\,10^{-2}\,x^4 + 1.65\,10^{-1}\,x^3 - 5.00\,10^{-1}\,x^2 + 1.00\,x$$

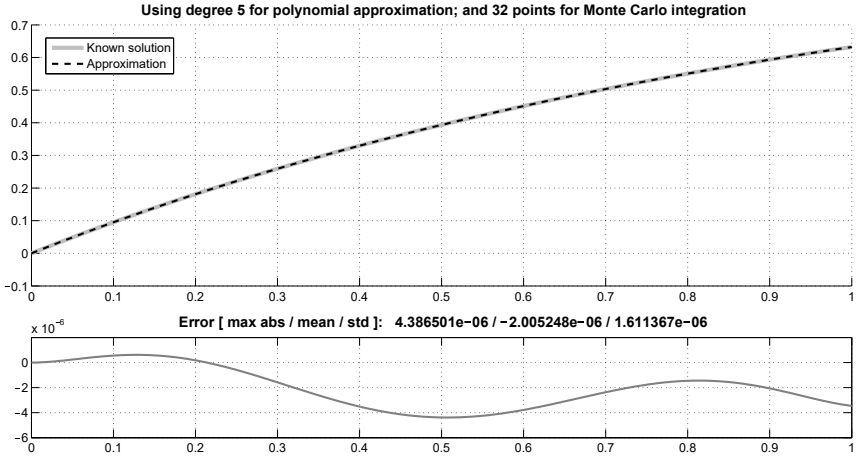

Figure 4.3: Approximation by the proposed method to the ODE that generated Figure 1.2, left plot; same differential as in Figure 4.4, different boundary conditions. Solution $y(x) = 1 - \exp(-x)$ approximated to: $\hat{y}(x) = 5.15\,10^{-3}\,x^5 - 3.86\,10^{-2}\,x^4 + 1.65\,10^{-1}\,x^3 - 5.00\,10^{-1}\,x^2 + 1.00\,x$.

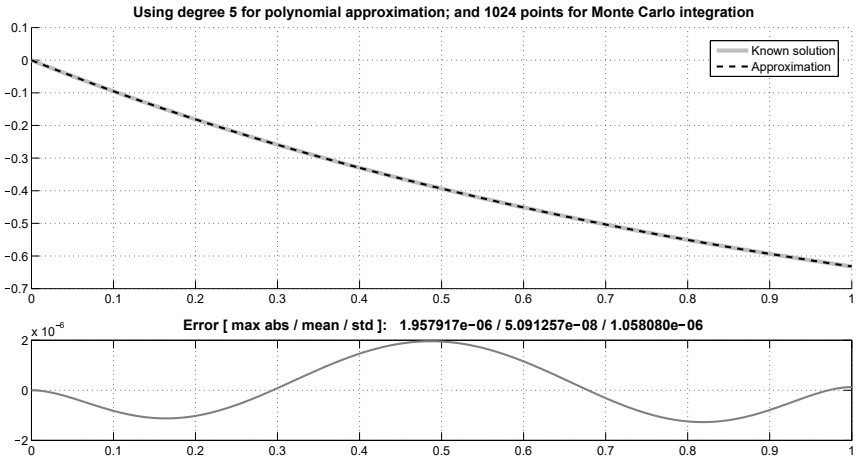

Figure 4.4: Approximation by the proposed method to the ODE that generated Figure 1.2, right plot; same differential as in Figure 4.3, different boundary conditions. Solution $y(x) = \exp(-x) - 1$ approximated to: $\hat{y}(x) = -5.15\,10^{-3}\,x^5 + 3.86\,10^{-2}\,x^4 - 1.65\,10^{-1}\,x^3 + 5.00\,10^{-1}\,x^2 - 1.00\,x$.

and the other to the "right" system (see Figure 4.4):

$$\hat{y}(x) = -5.15\,10^{-3}\,x^5 + 3.86\,10^{-2}\,x^4 - 1.65\,10^{-1}\,x^3 + 5.00\,10^{-1}\,x^2 - 1.00\,x.$$

In this simple example, as both data has the same differential model, a possible discussion to be attended would be if they represent the same system in different states or not.

4.5 Discussion

This method is a customization of Ritz-Galerkin method adopting Jacobi polynomials as orthogonal basis functions. The customization lies on the use of extra equations which reflect boundary conditions to cover the absence of those when dealing with the adopted polynomial basis.

Decisions taken during mathematical derivations are due to enable a low computational effort for the method, *e.g.*, a truncate polynomial basis, Legendre-like polynomials, change of variables to source function.

As seen by results, the method is robust and can present fair approximations to differential solutions. Note that different function basis could result in different approximations. In the next chapter, the expansion of this method to multivariate domains is shown.

Chapter 5

Partial Differential Equations

5.1 Proposed method

This chapter presents the proposed method to solve linear partial differential equations (LPDE) defined in Equation (3.2). Same assumptions from Section 4.1 are taken. Basically, developments addressed here are due to extensions for the method proposed from LODEs to LPDEs. By using combinations of Jacobi polynomials as basis functions, the achieved solution is a multivariate polynomial approximation.

5.2 Classification of PDEs

When addressing to PDEs of the form in Equation (5.1), as well as PDEs in more than two variables, there are three main types of PDE depending on the discriminant $AC - B^2$ [81], as shown in Table 5.1.

$$A\,\frac{\partial^2}{\partial x^2}u(x,y) + 2\,B\,\frac{\partial^2}{\partial x\,\partial y}u(xy) + C\,\frac{\partial^2}{\partial y^2}u(x,y) = F\left(x, y, u(x,y), \frac{\partial}{\partial x}u(x,y), \frac{\partial}{\partial x}u(x,y)\right) \quad (5.1)$$

where A, B, and C are the coefficients of the second order derivative terms.

From the point of view of engineering, this classification is important because if a type of PDE is identified for a problem that could lead to known solutions. For the proposed method, the solver scheme used does not make differences between any of those types.

5.3 Powers matrix

The proposed method is supposed to be generic. Also, the number of independent variables, the desired polynomial degree for the approximate solution and the differential order, those all are known only at execution time. Hence, a supporting scheme needed to be developed to handle combinatorial degrees

Table 5.1: Types of PDE, adapted from [81].

Type	Defining Condition	Example
Hyperbolic	$AC - B^2 < 0$	Wave equation
Parabolic	$AC - B^2 = 0$	Diffusion or heat equation
Elliptic	$AC - B^2 > 0$	Laplace equation

and orders to be used in function of the number of variables. As an example, an arbitrary three-variate differential equation such as:

$$\frac{\partial^3}{\partial x^3}u(x,y,z) - \frac{\partial\,\partial}{\partial x\,\partial y}u(x,y,z) + u(x,y,z) = 0$$

has the first derivative term is of order 3 (3 with respect to x, and order 0 for y and z), the second term of order 2 (1 with respect to x and y, and order 0 for z), and a third term of order 0 (for x, y and z).

This generic scheme was developed using the idea of integer partition, a form of representing positive integer numbers as a sum of positive integer numbers. For example, a full partitioning of the number 5 results in the set:

$$\begin{aligned}
&5,\\
&4+1,\\
&3+2,\\
&3+1+1,\\
&2+2+1,\\
&2+1+1+1,\\
&1+1+1+1+1
\end{aligned}$$

Algorithm 2 (adapted from integer partition algorithm ZS1 [82]) presents a partitioning scheme capable of limiting the number of summands (parts) an integer could be represented by.

Algorithm 3 uses Algorithm 2 to prepare a list with permutation sequences from all zeros up to a predetermined degree, respecting the number of considered variables. Each sequence from that list represents the distribution of either degrees to polynomials or orders to differentials.

Therefore, multivariate cases could benefit from this limited partitioning scheme in order to keep track of both degrees in multivariate polynomials and orders in partial derivatives. Table 5.2 presents an example where there is need to enlist degrees up to the 3^{rd} with respect to a bivariate domain. The output of Algorithm 3, those powers (degrees or orders) listed, should be known as the powers matrix. Note that hereby the powers matrix when identified by $\delta_{n,i}$ keeps track of polynomial degrees and when identified by $\gamma_{q,i}$ keeps track of differential orders.

The formula to learn about the number of all possibilities (lines of the powers matrix) is given in Equation (5.2).

Algorithm 2 Integer Partition; enlisted in *out* are all unique possibilities of v summands for the integer n, regardless order; adapted from [82]

Require: $n > 0 \wedge v > 0$
  $x \Leftarrow \{\, n,\, 1,\, \ldots\, 1 \,\}$ (ensure length: n)
  $m \Leftarrow 1$
  $h \Leftarrow 1$
  $q \Leftarrow 0$
  $aux \Leftarrow \{\, x[0 \ldots m-1],\, 0,\, \ldots\, 0 \,\}$ (ensure length: v)
  $out[q][\ldots] \Leftarrow aux$
  while $x[0] \neq 1$ **do**
    if $x[h-1] = 2$ **then**
      $m \Leftarrow m + 1$
      $x[h-1] \Leftarrow 1$
      $h \Leftarrow h - 1$
    else
      $r \Leftarrow x[h-1] - 1$
      $t \Leftarrow m - h + 1$
      $x[h-1] \Leftarrow r$
      while $t \geq r$ **do**
        $h \Leftarrow h + 1$
        $x[h-1] \Leftarrow r$
        $t \Leftarrow t - r$
      end while
      if $t = 0$ **then**
        $m \Leftarrow h$
      else
        $m \Leftarrow h + 1$
        if $t > 1$ **then**
          $h \Leftarrow h + 1$
          $x[h-1] \Leftarrow t$
        end if
      end if
    end if
    $aux \Leftarrow \{\, x[0 \ldots m-1],\, 0,\, \ldots\, 0 \,\}$ (ensure length: v)
    $q \Leftarrow q + 1$
    $out[q][\ldots] \Leftarrow aux$
  end while

Algorithm 3 Powers Matrix, enlisted in *pows* are the multivariate (v variables) polynomial n-degrees or differential n-orders.

Require: $n > 0 \wedge v > 0$
  $q \Leftarrow 0$
  $ipow \Leftarrow \{ 0, 0, \ldots 0 \}$ (ensure length: v)
  $pows[q][\ldots] \Leftarrow ipow$
  $q \Leftarrow q + 1$
  for $p \Leftarrow 1$ **to** n **do**
    $ipow \Leftarrow$ integer_partition(p, v) /* Algorithm 2 */
    $r \Leftarrow 0$
    for $i \Leftarrow 0$ **to** $ipow.size() - 1$ **do**
      $aux \Leftarrow ipow[i][\ldots]$
      $pwall[r][\ldots] \Leftarrow aux$
      $r \Leftarrow r + 1$
      while there is still a valid permutation of aux **do**
        $pwall[r][\ldots] \Leftarrow$ next permutation(aux)
        $r \Leftarrow r + 1$
      end while
    end for
    sort(*pwall*, reverse_lexicographic)
    for $i \Leftarrow 0$ **to** $r - 1$ **do**
      $pows[q + i][\ldots] \Leftarrow pwall[i][\ldots]$
    end for
    $q \Leftarrow q + r$
  end for

$$N^{\star} = \frac{(n+v)!}{n!\,v!} \tag{5.2}$$

where n is the maximum degree or order desired; and v is the number of variables.

Regarding Table 5.2, this formula could predict the example powers matrix number of rows: $\frac{(3+2)!}{3!\,2!} = 10$.

5.4 Multivariate adjustments

In order to expand this work from ordinary to partial LDEs, some concepts must be revisited. First, a multivariate set of basis functions and a definition for a proper inner product must be built. Equation (5.3) refers to the inner product definition which is extended to support multiple integrations. Note that the use of Legendre polynomials ($\alpha = \beta = 0$) simplifies this effort, once $w(x) = 1$ whatever the adopted number of variables:

$$\langle\, f(\vec{x}),\, g(\vec{x}) \,\rangle = \underbrace{\int_{a_0}^{b_0} \int_{a_1}^{b_1} \ldots \int_{a_{D-1}}^{b_{D-1}}}_{D \text{ times}} f(\vec{x})\, g(\vec{x})\, d\vec{x} \tag{5.3}$$

Table 5.2: Examples of integer partition of numbers 0 upto 3 with 2 parts maximum (Algorithm 2) and 3^{rd} degree polynomials or 3^{rd} order derivatives with 2 variables (Algorithm 3); this should be called the powers matrix.

Integer Partition		Powers of	
$i[0]$	$i[1]$	x_0	x_1
0	0	0	0
1	0	1	0
2	0	0	1
1	1	2	0
3	0	1	1
2	1	0	2
		3	0
		2	1
		1	2
		0	3

where D is the number of independent variables (dimension) from the domain of the problem.

In order to be solved for the unknown coefficients of the expansion, the GSE is built as shown in Equation (5.4):

$$\begin{gathered} \langle\, \phi_n(\vec{x}),\, R\left[\hat{u}(\vec{x})\right] \rangle\big|_{n=0}^{N^\star-1} = \\ \int_{a_0}^{b_0}\int_{a_1}^{b_1}\cdots\int_{a_{D-1}}^{b_{D-1}} \phi_n(\vec{x}) \cdot \left\{\mathcal{L}\left[\hat{u}(\vec{x})\right] - s(\vec{x})\right\}\, dx_0\, dx_1 \ldots dx_{D-1}\Big|_{n=0}^{N^\star-1} = 0 \end{gathered} \tag{5.4}$$

where this system has $N^\star$ equations (see Equation (5.2), for $n = N$ and $v = D$); $\vec{x}$ is the vector of D variables $(x_0, x_1, \ldots x_{D-1})^T$; and a_i and b_i are the i-th lower and upper limits of integration, respectively, for $i = 0, \ldots D-1$.

The TGE must be adjusted as in (5.5):

$$\hat{u}(\vec{x}) = \sum_{i=0}^{N^\star-1} \tilde{u}_i\, \phi_i(\vec{x}) \tag{5.5}$$

where $\phi_i(\vec{x})$ is a multivariate basis function from a finite basis set with a span of $N^\star$ functions.

Multivariate mappings follow Equations (3.24) and (3.25) for each dimension. In other words, linear maps like $x_i \mapsto \xi_i$ and $\xi_i \mapsto x_i$ are defined in the same way for each dimension i with respect to their respective inferior and superior limits a_i and b_i.

A proper basis set is built based on those linear mappings and on combinations of univariate polynomials, supported by the powers matrix from Algorithm 3. Each one of those mappings and Jacobi-Legendre polynomials are

with respect to each variable that constitutes the domain. This multivariate basis set is shown in Equation (5.6):

$$\mathcal{B} = \{\phi_n(\vec{x})\}|_{n=0}^{N^\star-1} = \left\{ \prod_{i=0}^{D-1} P_{\delta_{n,i}}^{(0,0)} \left(\xi_i(x_i)\right) \right\} \Bigg|_{n=0}^{N^\star-1} \tag{5.6}$$

where $\delta_{n,i}$ is the element δ located at the n^{th} row and the i^{th} column in the powers matrix (like the one presented in Table 5.2) evaluated case by case.

As an example, if $N = 3$ and $D = 2$, the 8^{th} basis function with reference to Table 5.2 would be:

$$\phi_7(x_0, x_1) = P_2^{(0,0)} \left(\xi_0(x_0)\right) \cdot P_1^{(0,0)} \left(\xi_1(x_1)\right).$$

Each basis function is essentially the product of D univariate polynomials with degrees that sum up to N. This is very handful when analysing multivariate derivatives with the product rule. In this way, the multivariate context could help to define derivatives for those basis functions, as in Equation (5.7):

$$\begin{aligned} \left[\prod_{j=0}^{D-1} \frac{\partial^{\gamma_{q,j}}}{\partial x_j^{\gamma_{q,j}}} \right] \prod_{i=0}^{D-1} P_{\delta_{n,i}}^{(0,0)} \left(\xi_i(x_i)\right) = \prod_{i=0}^{D-1} \frac{\partial^{\gamma_{q,i}}}{\partial x_i^{\gamma_{q,i}}} P_{\delta_{n,i}}^{(0,0)} \left(\xi_i(x_i)\right) = \\ \prod_{i=0}^{D-1} \frac{\Gamma(\delta_{n,i} + \gamma_{q,i} + 1)}{(b_i - a_i)^{\gamma_{q,i}} \Gamma(\delta_{n,i} + 1)} P_{\delta_{n,i}-\gamma_{q,i}}^{(\gamma_{q,i},\gamma_{q,i})} \left(\xi_i(x_i)\right) \end{aligned} \tag{5.7}$$

where $\delta_{n,i}$ is the element δ located at the n^{th} row and the i^{th} column in the respective powers matrix; and $\gamma_{q,i}$ is the element γ located at the q^{th} row and the i^{th} column in the respective powers matrix resulted from Algorithm 3 with $n = Q$ and $v = D$. Note that identity in Equation (3.23) is still useful here.

As another example, if $N = 3$, $Q = 3$ and $D = 2$ are adopted, the 7^{th} multivariate derivative taken over the 8^{th} derivative of the 8^{th} basis function would be:

$$\frac{\partial^3}{\partial x_0^2 \, \partial x_1} \phi_7(x_0, x_1) = \frac{\partial^2}{\partial x_0^2} P_2^{(0,0)} \left(\xi_0(x_0)\right) \cdot \frac{\partial}{\partial x_1} P_1^{(0,0)} \left(\xi_1(x_1)\right).$$

Multidimensional change of variables should also be considered when expanding the findings from ODEs to PDEs. The multidimensional derivative is then shown in Equation (5.8):

$$\begin{aligned} \left[\prod_{i=0}^{D-1} \frac{\partial^{\gamma_{q,i}}}{\partial x_j^{\gamma_{q,i}}} \right] f(x_0, x_1, \ldots x_{D-1}) = \\ \left[\prod_{i=0}^{D-1} \left(\frac{2}{b_i - a_i} \right)^{\gamma_{q,i}} \frac{\partial^{\gamma_{q,i}}}{\partial \xi^{\gamma_{q,i}}} \right] f\left(x_0(\xi_0), x_1(\xi_1), \ldots x_{D-1}(\xi_{D-1})\right) \end{aligned} \tag{5.8}$$

as well as the multidimensional integration is presented in Equation (5.9). Note that both of those have a role in the new multivariate approach.

$$\int_{a_0}^{b_0}\int_{a_1}^{b_1}\cdots\int_{a_{D-1}}^{b_{D-1}} f\left(x_0,\, x_1,\, \ldots x_{D-1}\right)\, dx_0\, dx_1 \cdots dx_{D-1} =$$

$$\prod_{i=0}^{D-1}\frac{b_i - a_i}{2}\int_{-1}^{1}\int_{-1}^{1}\cdots\int_{-1}^{1} f\left(x_0(\xi_0),\, x_1(\xi_1),\, \ldots x_{D-1}(\xi_{D-1})\right)\, d\xi_0\, d\xi_1 \cdots d\xi_{D-1} \tag{5.9}$$

where the determinant of the Jacobian matrix $J = \prod_{i=0}^{D-1}\frac{\partial x_i}{\partial \xi_i} = \prod_{i=0}^{D-1}\frac{b_i - a_i}{2}$ is both constant and the result of the product on the main diagonal of the Jacobian matrix, due to the linear nature of each respective dimensional mapping $\xi_i \mapsto x_i$.

Updated coefficient matrix

To update the coefficient matrix is the same as to augment its univariate version. Starting point is the following unidimensional expression excerpt from Equation 4.3:

$$\sum_{m=0}^{N}\tilde{u}_m \sum_{q=0}^{Q}\frac{1}{(b-a)^q}\int_{-1}^{1} k_q(x(\xi))\cdot P_n^{(0,0)}(\xi)\cdot\frac{\Gamma(m+q+1)}{\Gamma(m+1)}\, P_{m-q}^{(q,q)}(\xi)\, d\xi$$

where each line with respect to $n = 0 \ldots N$ contributes to the final matrix representation.

The new expression to build the multivariate coefficient matrix is presented in Equation (5.10), each row with respect to $n = 0 \ldots N^\star - 1$.

$$\sum_{m=0}^{N^\star-1}\tilde{u}_m \sum_{q=0}^{Q^\star-1}\left[\prod_{i=0}^{D-1}\frac{1}{(b_i-a_i)^{\gamma_{q,i}}}\right]\int_{-1}^{1}\int_{-1}^{1}\cdots\int_{-1}^{1} k_q(x_0(\xi_0),\ldots x_{D-1}(\xi_{D-1}))\cdot$$
$$\left[\prod_{i=0}^{D-1} P_{\delta_{n,i}}^{(0,0)}(\xi_i)\right]\cdot\left[\prod_{i=0}^{D-1}\frac{\Gamma(\delta_{m,i}+\gamma_{q,i}+1)}{\Gamma(\delta_{m,i}+1)} P_{\delta_{m,i}-\gamma_{q,i}}^{(\gamma_{q,i},\gamma_{q,i})}(\xi_i)\right]\, d\xi_0 \ldots d\xi_{D-1} \tag{5.10}$$

The multivariate coefficient matrix is then built as shown by Equation (5.11).

$$G_{1\times Q^\star} = \left(1\ \prod_{i=0}^{D-1}\frac{1}{(b_i-a_i)^{\gamma_{1,i}}}\ \cdots \prod_{i=0}^{D-1}\frac{1}{(b_i-a_i)^{\gamma_{Q^\star-1,i}}}\right)$$

$$D_{q,m}(\vec{\xi}) = k_q(x_0(\xi_0),\ldots x_{D-1}(\xi_{D-1}))\cdot\left[\prod_{i=0}^{D-1}\frac{\Gamma(\delta_{m,i}+\gamma_{q,i}+1)}{\Gamma(\delta_{m,i}+1)} P_{\delta_{m,i}-\gamma_{q,i}}^{(\gamma_{q,i},\gamma_{q,i})}(\xi_i)\right]$$

$$D_{Q^\star \times N^\star} = \begin{pmatrix} D_{0,0}(\vec{\xi}) & \cdots & D_{0,N^\star-1}(\vec{\xi}) \\ \vdots & \ddots & \vdots \\ D_{Q^\star-1,0}(\vec{\xi}) & \cdots & D_{Q^\star-1,N^\star-1}(\vec{\xi}) \end{pmatrix}$$

$$G^{(n)}_{Q^\star \times N^\star} = \left[\int_{-1}^{1} \int_{-1}^{1} \cdots \int_{-1}^{1} \left[\prod_{i=0}^{D-1} P^{(0,0)}_{\delta_{n,i}}(\xi_i) \right] \cdot \left(D_{Q^\star \times N^\star} \right) d\xi_0 \, d\xi_1 \ldots dx_{D-1} \right]$$

$$S^{(n)}_{1\times 1} = \int_{-1}^{1} \int_{-1}^{1} \cdots \int_{-1}^{1} \prod_{i=0}^{D-1} P^{(0,0)}_{\delta_{n,i}}(\xi_i) \cdot s(x_0(\xi_0), \ldots x_{D-1}(\xi_{D-1})) \, d\xi_0 \, d\xi_1 \ldots dx_{D-1}$$

$$\underbrace{\begin{pmatrix} G_{1\times Q^\star} \cdot G^{(0)}_{Q^\star \times N^\star} \\ \hline G_{1\times Q^\star} \cdot G^{(1)}_{Q^\star \times N^\star} \\ \hline \vdots \\ \hline G_{1\times Q^\star} \cdot G^{(N^\star-1)}_{Q^\star \times N^\star} \end{pmatrix}}_{N^\star \times N^\star} \cdot \begin{pmatrix} \tilde{u}_0 \\ \tilde{u}_1 \\ \vdots \\ \tilde{u}_{N^\star-1} \end{pmatrix} = \begin{pmatrix} S^{(0)}_{1\times 1} \\ S^{(1)}_{1\times 1} \\ \vdots \\ S^{(N^\star-1)}_{1\times 1} \end{pmatrix} \tag{5.11}$$

5.5 Solving PDEs

As discussed before, when solving differential equations, the use of Jacobi polynomials as basis functions almost ever fails to respect auxiliary conditions. It is very unusual to the coefficient matrix of the GSE not being rank deficient or even ill-conditioned. In the rare cases where the existence of a solution is verified without further data, specially when dealing with homogeneous differentials, such solution is trivial (all coefficients of the TGE are zero), hence not of interest. The desired solution must be non-trivial and unique, both by the necessity of a model and by the fact that, if a real system is modelled by a differential equation, the solution is observable in reality.

As normally found in the literature about PDEs, auxiliary conditions are functions by themselves. Even based on Galerkin method, the here proposed method cannot deal with the existence of undefined independent variables within boundary conditions respective equations. The proposed method requires scalars evaluated on some points over the domain. The implication is that all auxiliary conditions need to be discretized over variables they are functions of.

So, instead of using function-like auxiliary conditions, the proposed method requires a set of known values from those conditions over their respective original domains. The here proposed workaround is to define representative points on the boundary and to evaluate the original conditions there, as if those conditions were sampled. The required exact number of points is strongly dependent on the coefficient matrix either rank or condition number. As an initial guess, an empirical recommendation is to choose 2^D points per condition, where D is the total number of variables, and after start to iteratively increase that number until the coefficient matrix become full rank. Note that, if the coefficient matrix is already full ranked and well-conditioned, a minimal number of points must be chosen anyway to avoid raising a trivial solution.

Keep in mind that this method is supposed to support computer-automated system modelling for real systems, so the solution is supposed to be observable in reality. The feasible expectation is that at some step of the iterative aggregation of those points as conditions, solution becomes possible to be found.

Updated auxiliary conditions

Starting from the following excerpt from Equation (4.7):

$$\sum_{m=0}^{N} \tilde{u}_m \sum_{w=0}^{W} \frac{1}{(b-a)^w} h_{w,c}(X_c) \frac{\Gamma(m+w+1)}{\Gamma(m+1)} P_{m-w}^{(w,w)}\left(\xi(X_c)\right),$$

the multivariate version could be derived, as shown in Equation (5.12):

$$\sum_{m=0}^{N^\star-1} \tilde{u}_m \sum_{w=0}^{W^\star-1} \left[\prod_{i=0}^{D-1} \frac{1}{(b_i-a_i)^{\gamma_{w,i}}}\right] h_{w,c}\left(X_0^{(c)}, \dots X_{D-1}^{(c)}\right) \cdot \left[\prod_{i=0}^{D-1} \frac{\Gamma(\delta_{m,i}+\gamma_{w,i}+1)}{\Gamma(\delta_{m,i}+1)} P_{\delta_{m,i}-\gamma_{w,i}}^{(\gamma_{w,i},\gamma_{w,i})}\left(\xi_i\left(X_i^{(c)}\right)\right)\right] \quad (5.12)$$

Finally, from Equations (5.11) and (5.12), the proposed method achieves the final multidimensional Galerkin-like system as presented in Equation (5.13):

$$B_{1\times W^\star} = \left(1 \prod_{i=0}^{D-1} \frac{1}{(b_i-a_i)^{\gamma_{1,i}}} \cdots \prod_{i=0}^{D-1} \frac{1}{(b_i-a_i)^{\gamma_{W^\star-1,i}}}\right)$$

$$B_{w,m}^{(c)}\left(\vec{X}^{(c)}\right) = h_{w,c}\left(\vec{X}^{(c)}\right) \cdot \prod_{i=0}^{D-1} \frac{\Gamma(\delta_{m,i}+\gamma_{w,i}+1)}{\Gamma(\delta_{m,i}+1)} P_{\delta_{m,i}-\gamma_{w,i}}^{(\gamma_{w,i},\gamma_{w,i})}\left(\xi_i\left(X_i^{(c)}\right)\right)$$

$$B_{W^\star\times N^\star}^{(c)} = \begin{pmatrix} B_{0,0}\left(\vec{X}^{(c)}\right) & \cdots & B_{0,N^\star-1}\left(\vec{X}^{(c)}\right) \\ \vdots & \ddots & \vdots \\ B_{W^\star-1,0}\left(\vec{X}^{(c)}\right) & \cdots & B_{W^\star-1,N^\star-1}\left(\vec{X}^{(c)}\right) \end{pmatrix}$$

$$\underbrace{\left(\begin{array}{c|c} G_{1\times Q^\star} \cdot G^{(0)}_{Q^\star\times N^\star} \\ \hline \vdots \\ \hline G_{1\times Q^\star} \cdot G^{(N^\star-1-C)}_{Q^\star\times N^\star} \\ \hline B_{1\times W^\star} \cdot B^{(1)}_{W^\star\times N^\star} \\ \hline \vdots \\ \hline B_{1\times W^\star} \cdot B^{(C)}_{W^\star\times N^\star} \end{array}\right)}_{N^\star\times N^\star} \cdot \begin{pmatrix} \tilde{u}_0 \\ \tilde{u}_1 \\ \vdots \\ \tilde{u}_{N^\star-1} \end{pmatrix} = \left(\begin{array}{c} S^{(0)}_{1\times 1} \\ \vdots \\ S^{(N^\star-1-C)}_{1\times 1} \\ \hline V_1 \\ \vdots \\ V_C \end{array}\right) \tag{5.13}$$

Examples

In this section, three examples are shown, each one from a different PDE type as classified in Table 5.1. As can be seen, the same proposed method is applied to all of those problems.

Hyperbolic equation Starting with a simple dynamic one-dimensional wave PDE problem:

$$\frac{\partial^2}{\partial x^2}u(x,t) - \frac{\partial^2}{\partial t^2}u(x,t) = 0$$

where $u(x,t)$ is the height of the wave in function of length x and time t. The domain of calculation is $0 < x < 1,\ t > 0$. Boundary conditions that augment this PDE are:

$$\begin{array}{c|l} \text{[BC]} & \begin{cases} u(0,t) = 0 \\ u(1,t) = 0 \end{cases} \\ \hline \text{[initial value]} & \begin{cases} u(x,0) = \sin(\pi\, x) \\ u_t(x,0) = 0 \end{cases} \end{array}$$

where $u_t(x,t)$ is another way of representing $\frac{\partial}{\partial t}u(x,t)$, $u_{tt}(x,t)$ would represent $\frac{\partial^2}{\partial t^2}u(x,t)$, and so on.

The known solution to this example is:

$$u(x,t) = \sin(\pi\, x)\, \cos(\pi\, t)$$

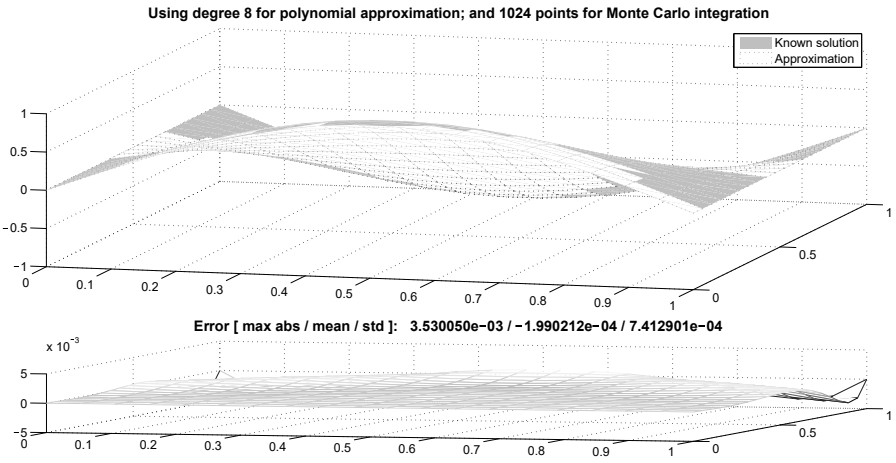

Figure 5.1: Solution to a dynamic one-dimensional wave problem; approximate solution adopts a degree 8 bivariate polynomial.

As aforementioned, the proposed method cannot deal with the existence of undefined independent variables within boundary conditions. Also, due to characteristics of the proposed method, the initial value problem needs to be confined in time. To achieve that, the new adopted domain for this example is $0 < x < 1,\ 0 < t < 1$. Also, following the proposed workaround, the boundary conditions to be considered in this numerical example are:

$$
\begin{array}{l}
\text{[BC]} \quad \left\{ \begin{array}{llll}
u(0,\,0) = & 0; & u(0,\,\frac{1}{3}) = & 0 \\
u(0,\,\frac{2}{3}) = & 0; & u(0,\,1) = & 0 \\
\hline
u(1,\,0) = & 0; & u(1,\,\frac{1}{3}) = & 0 \\
u(1,\,\frac{2}{3}) = & 0; & u(1,\,1) = & 0
\end{array} \right. \\
\hline
\text{[IC]} \quad \left\{ \begin{array}{llll}
u(0.2,\,0) = & \sin(0.2\,\pi); & u(0.4,\,0) = & \sin(0.4\,\pi) \\
u(0.5,\,0) = & \sin(0.5\,\pi); & u(0.6,\,0) = & \sin(0.6\,\pi) \\
u(0.8,\,0) = & \sin(0.8\,\pi) & & \\
u_t(0,\,0) = & 0; & u_t(\frac{1}{3},\,0) = & 0 \\
u_t(\frac{2}{3},\,0) = & 0; & u_t(1,\,0) = & 0
\end{array} \right.
\end{array}
$$

Note that the option for equidistant points from domain limits was taken, with variation of t in boundary conditions (where $x = 0$) and of x in initial conditions (where $t = 0$). Also that to evaluate values at boundary, functions from original conditions were used. Observe that two points that would appear derived from latter initial conditions — $(0, 0)$ and $(1, 0)$ — could not be used due to previous appearance in former boundary conditions.

Figure 5.1 shows on top a joint plot of the TGE polynomial approximation and the known solution, and on bottom the error as the difference from the approximation to the known solution. Note that error has a mean value of $-2\,10^{-4}$ and a standard deviation of $7\,10^{-4}$.

Finally, Equation (5.14) presents the coefficients of the TGE which are responsible for the approximate solution to this dynamic one-dimensional wave

problem. Sub-indices relates each coefficient to a product of Jacobi polynomials whose degrees are tracked by the proper row in the respective powers matrix, as in $P_0^{(0,0)}(x \mapsto \xi_0) \cdot P_0^{(0,0)}(t \mapsto \xi_1)$, $P_1^{(0,0)}(x \mapsto \xi_0) \cdot P_0^{(0,0)}(t \mapsto \xi_1)$, and so on, up to $P_0^{(0,0)}(x \mapsto \xi_0) \cdot P_8^{(0,0)}(t \mapsto \xi_1)$.

$$\begin{pmatrix} u_0 & u_1 & \cdots & u_{44} \end{pmatrix}^T = \big(-1.615\,10^{-4} \quad 0 \quad -0.774 \quad -4.018\,10^{-4} \quad 0 \quad -4.018\,10^{-4} \quad 0 \quad 0.834 \quad 0 \quad 0.143 \quad -1.489\,10^{-4} \quad 0 \quad -1.640\,10^{-3} \quad 0 \quad -1.489\,10^{-4} \quad 0 \quad -6.315\,10^{-2} \quad 0 \quad -0.156 \quad 0 \quad -5.750\,10^{-3} \quad 1.212\,10^{-4} \quad 0 \quad 2.115\,10^{-4} \quad 0 \quad 2.115\,10^{-4} \quad 0 \quad 1.212\,10^{-4} \quad 0 \quad 1.929\,10^{-3} \quad 0 \quad 1.273\,10^{-2} \quad 0 \quad 7.072\,10^{-3} \quad 0 \quad 1.484\,10^{-4} \quad 7.083\,10^{-6} \quad 0 \quad 4.604\,10^{-4} \quad 0 \quad 1.302\,10^{-3} \quad 0 \quad 4.604\,10^{-4} \quad 0 \quad 7.083\,10^{-6}\big) \tag{5.14}$$

Simplifying the expanded equation also with the aid of Maxima CAS software [79], the polynomial approximation solution is shown in Equation (5.15).

$$\begin{aligned} \hat{u}(x,t) = {} & 9.116\,10^{-2}\,t^8 + 0.145\,t^7 + 2.552\,x^2\,t^6 - 2.552\,x\,t^6 - 0.649\,t^6 + 3.035\,x^2\,t^5 \\ & - 3.035\,x\,t^5 + 0.678\,t^5 + 6.381\,x^4\,t^4 - 12.762\,x^3\,t^4 - 9.737\,x^2\,t^4 + 16.118\,x\,t^4 \\ & - 0.373\,t^4 + 5.059\,x^4\,t^3 - 10.118\,x^3\,t^3 + 6.779\,x^2\,t^3 - 1.720\,x\,t^3 + 0.128\,t^3 \\ & + 2.552\,x^6\,t^2 - 7.657\,x^5\,t^2 - 9.737\,x^4\,t^2 + 32.236\,x^3\,t^2 - 2.239\,x^2\,t^2 - 15.155\,x\,t^2 \\ & - 1.888\,10^{-2}\,t^2 + 1.012\,x^6\,t - 3.035\,x^5\,t + 3.389\,x^4\,t - 1.720\,x^3\,t + 0.383\,x^2\,t \\ & - 2.872\,10^{-2}\,x\,t + 9.116\,10^{-2}\,x^8 - 0.365\,x^7 - 0.649\,x^6 + 3.224\,x^5 - 0.373\,x^4 - 5.052\,x^3 \\ & - 1.888\,10^{-2}\,x^2 + 3.143\,x \end{aligned} \tag{5.15}$$

Parabolic equation Here is a diffusion problem, the homogeneous heat conduction equation with insulated boundary conditions:

$$\alpha^2 \frac{\partial^2}{\partial x^2} u(x,t) - \frac{\partial}{\partial t} u(x,t) = 0$$

where $u(x,t)$ is the temperature distribution function of a wire (with length L); and the positive constant α^2 is the thermo diffusivity constant of the wire. The domain of calculation is $0 < x < L,\ t > 0$. Boundary conditions that augment this partial differential equation are:

$$\begin{array}{ll} \text{[BC]} & \begin{cases} u_x(0,t) = 0 \\ u_x(1,t) = 0 \end{cases} \\ \hline \text{[IC]} & u(x,0) = f(x) \end{array}$$

A numerical example is presented, with $\alpha = 1$; $L = 1$; and $f(x) = 1 + \cos(\pi\,x) + 0.5\,\cos(3\,\pi\,x)$. The known solution to this example is:

$$u(x,t) = 1 + e^{-\pi^2\,t}\,\cos(\pi\,x) + 0.5\,e^{-(3\,\pi)^2\,t}\,\cos(3\,\pi\,x)$$

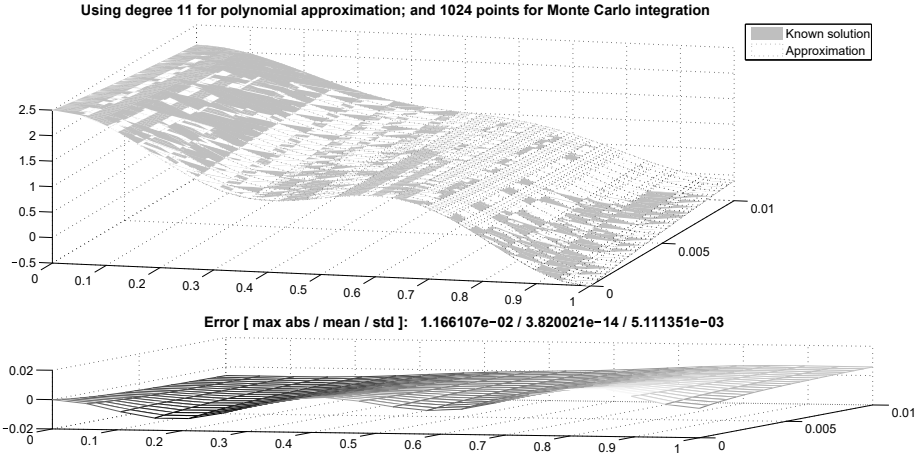

Figure 5.2: Solution to a homogeneous heat conduction equation with insulated boundary; approximate solution adopts a degree 11 bivariate polynomial.

Therefore, following the proposed workaround, the adopted domain for this example is $0 < x < 1,\ 0 < t < 0.01$ and boundary conditions to be considered in this numerical example are:

$$
\begin{array}{l}
\text{[BC]} \quad \left\{ \begin{array}{rrrr}
u_x(0,\,0) = & 0; & u_x(0,\,\frac{1}{300}) = & 0 \\
u_x(0,\,\frac{2}{300}) = & 0; & u_x(0,\,0.01) = & 0 \\
\hline
u_x(1,\,0) = & 0; & u_x(1,\,\frac{1}{300}) = & 0 \\
u_x(1,\,\frac{2}{300}) = & 0; & u_x(1,\,0.01) = & 0
\end{array} \right. \\
\hline
\text{[IC]} \quad \left\{ \begin{array}{l}
u(0,\,0) = \quad 1 + \cos(\pi\,0) + 0.5\,\cos(3\,\pi\,0) = 2.5 \\
u(\frac{1}{3},\,0) = \quad 1 + \cos(\pi\,\frac{1}{3}) + 0.5\,\cos(3\,\pi\,\frac{1}{3}) = 1 \\
u(\frac{2}{3},\,0) = \quad 1 + \cos(\pi\,\frac{2}{3}) + 0.5\,\cos(3\,\pi\,\frac{2}{3}) = 1 \\
u(1,\,0) = \quad 1 + \cos(\pi\,1) + 0.5\,\cos(3\,\pi\,1) = -0.5
\end{array} \right.
\end{array}
$$

Note that the option for equidistant points from domain limits was taken, with variation of t in insulated boundary conditions and of x in the initial condition. Also, to evaluate values at boundary, functions from original conditions were used.

Figure 5.2 shows on top a joint plot of the TGE polynomial approximation and the known solution, and on bottom the error as the difference from the approximation to the known solution. Note that error has a mean value of ≈ 0 and a standard deviation of $5\,10^{-3}$.

Finally, Equation (5.16) presents the coefficients of the TGE which are responsible for the approximate solution to this homogeneous heat conduction equation with insulated boundary. Sub-indices relates each coefficient to a product of Jacobi polynomials whose degrees are tracked by the proper row in the respective powers matrix, as $P_0^{(0,0)}(x \mapsto \xi_0) \cdot P_0^{(0,0)}(t \mapsto \xi_1)$, $P_1^{(0,0)}(x \mapsto \xi_0) \cdot P_0^{(0,0)}(t \mapsto \xi_1)$, and so on, up to $P_0^{(0,0)}(x \mapsto \xi_0) \cdot P_1 1^{(0,0)}(t \mapsto \xi_1)$

$$\begin{pmatrix} u_0 & u_1 & \cdots & u_{77} \end{pmatrix}^T = \begin{pmatrix} 1 & -1.195 & 0 & 7.649\,10^{-2} & 0 & -3.449\,10^{-1} \\ 0 & -3.875\,10^{-3} & 0 & 0 & 2.351\,10^{-1} & 0 & 3.015\,10^{-4} & 0 & 3.153\,10^{-1} \\ 0 & -3.644\,10^{-2} & 0 & -1.993\,10^{-5} & 0 & 0 & -1.390\,10^{-1} & 0 \\ 3.405\,10^{-3} & 0 & -8.293\,10^{-6} & 0 & -6.001\,10^{-2} & 0 & 1.942\,10^{-2} \\ 0 & -2.488\,10^{-4} & 0 & 0 & 0 & 0 & 2.365\,10^{-2} & 0 & -1.382\,10^{-3} & 0 \\ 0 & 0 & 0 & 0 & 5.036\,10^{-3} & 0 & -2.416\,10^{-3} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1.421\,10^{-3} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1.781\,10^{-4} \\ & & & & & & & & & & & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \quad (5.16)$$

Simplifying the expanded equation, the polynomial approximation solution is shown in Equation (5.17).

$$\begin{aligned} \hat{u}(x,t) = {} & -4.180\,10^7\,x\,t^5 + 2.090\,10^7\,t^5 - 3.483\,10^7\,x^3\,t^4 + 5.225\,10^7\,x^2\,t^4 \\ & - 2.013\,10^7\,x\,t^4 + 1358587.9\,t^4 - 6966366.3\,x^5\,t^3 + 1.742\,10^7\,x^4\,t^3 \\ & - 1.342\,10^7\,x^3\,t^3 + 2717175.8\,x^2\,t^3 + 414272.4\,x\,t^3 - 79468.3\,t^3 \\ & - 497597.6\,x^7\,t^2 + 1741591.6\,x^6\,t^2 - 2013309.2\,x^5\,t^2 + 679294.0\,x^4\,t^2 \\ & + 207136.2\,x^3\,t^2 - 119202.5\,x^2\,t^2 - 2553.6\,x\,t^2 + 2320.6\,t^2 \\ & - 13822.2\,x^9\,t + 62199.7\,x^8\,t - 95871.9\,x^7\,t + 45286.3\,x^6\,t + 20713.6\,x^5\,t \\ & - 19867.1\,x^4\,t - 851.2\,x^3\,t + 2320.6\,x^2\,t + 4.660\,x\,t - 56.245\,t \\ & - 125.656\,x^{11} + 691.11\,x^{10} - 1331.6\,x^9 + 808.68\,x^8 + 493.18\,x^7 \\ & - 662.24\,x^6 - 42.560\,x^5 + 193.38\,x^4 + 0.777\,x^3 - 28.122\,x^2 + 2.5 \end{aligned} \quad (5.17)$$

Elliptic equation Here is a problem regarding the steady-state temperature distribution of a thin plate in the form of Laplace equation:

$$\nabla^2 u(x,y) = \frac{\partial^2}{\partial x^2}u(x,y) + \frac{\partial^2}{\partial y^2}u(x,y) = 0$$

where $u(x,y)$ is the temperature distribution function over the surface of a thin plate (with width L and height H); The domain of calculation is $0 < x < L$ and $0 < y < H$. Dirichlet boundary conditions that augment this partial differential equation are:

$$\text{[BC]} \quad \begin{cases} u(0,y) = 0 \\ u(L,y) = 0 \\ u(x,0) = 0 \\ u(x,H) = f(x) \end{cases}$$

The known solution to this example is:

$$u(x,y) = \sum_{n=1}^{\infty} B_n \, \sin(\mu_n \, x) \, \sinh(\mu_n \, y)$$

where $B_n = \frac{2}{L \, \sinh\left(\frac{n\pi H}{L}\right)} \int_0^L f(x) \, \sin(\frac{n\pi x}{L}) \, dx$; and $\mu_n = \frac{n\pi}{L}$.

As a numerical example, an instance of this problem is achieved when assuming $L = H = 1$, turning the domain into $0 < x < 1, \ 0 < y < 1$, and assuming $u(x,1) = f(x) = \sin(\pi x)$. Now this very problem has a known solution, and it is:

$$u(x,y) = \frac{\sin(\pi x) \, \sinh(\pi y)}{\sinh(\pi)}$$

Following the proposed workaround, the adopted domain for this example is $0 < x < 1, \ 0 < t < 1$ and the boundary conditions to be considered in this numerical example are:

$$[\text{BC}] \quad \begin{cases} u(0,\,0) = 0; & u(0,\,0.2) = 0; & u(0,\,0.4) = 0 \\ u(0,\,0.6) = 0; & u(0,\,0.8) = 0; & u(0,\,1) = 0 \\ u(0.2,\,0) = 0; & u(0.4,\,0) = 0; & u(0.6,\,0) = 0 \\ u(0.8,\,0) = 0; & u(1,\,0) = 0; & u(1,\,0.2) = 0 \\ u(1,\,0.4) = 0; & u(1,\,0.6) = 0; & u(1,\,0.8) = 0 \\ u(1,\,1) = 0; & u(0.2,\,1) = \sin(0.2\pi); & u(0.4,\,1) = \sin(0.4\pi) \\ u(0.6,\,1) = \sin(0.6\pi); & u(0.8,\,1) = \sin(0.8\pi) & \end{cases}$$

Note that due to the rank deficiency of the original coefficient matrix, the number of points to represent Dirichlet conditions needed to be raised in order to properly define the boundary.

Figure 5.3 shows on top a joint plot of the TGE polynomial approximation and the known solution, and on bottom the error as the difference from the approximation to the known solution. Note that error has a mean value of $-8\,10^{-4}$ and a standard deviation of $3\,10^{-4}$.

Finally, Equation (5.18) presents the coefficients of the TGE which are responsible for the approximate solution to this steady-state temperature in a thin plate problem. Sub-indices relates each coefficient to a product of Jacobi polynomials whose degrees are tracked by the proper row in the respective powers matrix, as $P_0^{(0,0)}(x \mapsto \xi_0) \cdot P_0^{(0,0)}(t \mapsto \xi_1)$, $P_1^{(0,0)}(x \mapsto \xi_0) \cdot P_0^{(0,0)}(t \mapsto \xi_1)$, and so on, up to $P_0^{(0,0)}(x \mapsto \xi_0) \cdot P_9^{(0,0)}(t \mapsto \xi_1)$

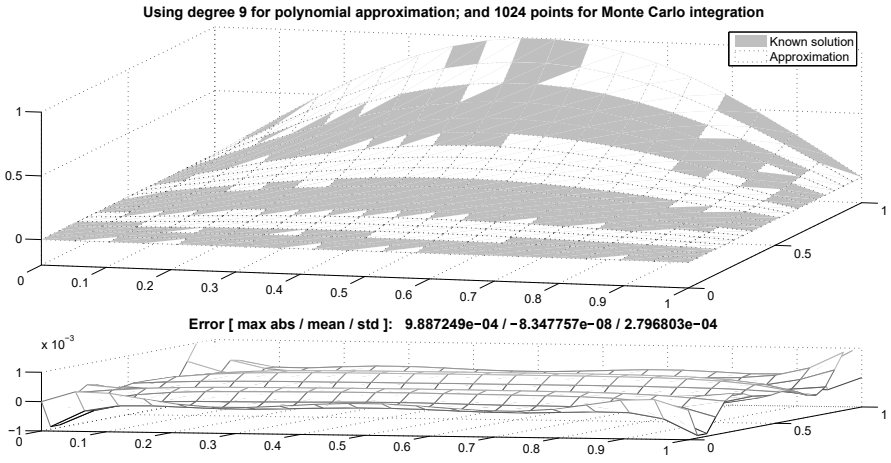

Figure 5.3: Solution to a steady-state temperature in a thin plate (Laplace equation); approximate solution adopts a degree 9 bivariate polynomial.

$$\begin{pmatrix} u_0 & u_1 & \cdots & u_{54} \end{pmatrix}^T = (0.186 \quad 0 \quad 0.276 \quad -0.201 \quad 0 \quad 0.124 \quad 0 \\ -0.298 \quad 0 \quad 4.095\,10^{-2} \quad 1.512\,10^{-2} \quad 0 \quad -0.134 \quad 0 \quad 8.218\,10^{-3} \\ 0 \quad 2.259\,10^{-2} \quad 0 \quad -4.405\,10^{-2} \quad 0 \quad 1.855\,10^{-3} \quad -3.821\,10^{-4} \\ 0 \quad 1.010\,10^{-2} \quad 0 \quad -8.882\,10^{-3} \quad 0 \quad 1.930\,10^{-4} \quad 0 \quad -3.950\,10^{-4} \\ 0 \quad 3.498\,10^{-3} \quad 0 \quad -2.320\,10^{-3} \quad 0 \quad -7.386\,10^{-4} \quad 3.636\,10^{-6} \\ 0 \quad -2.363\,10^{-4} \quad 0 \quad 6.685\,10^{-4} \quad 0 \quad -2.363\,10^{-4} \quad 0 \quad 3.636\,10^{-6} \\ 0 \quad -1.224\,10^{-5} \quad 0 \quad 1.591\,10^{-4} \quad 0 \quad -2.500\,10^{-4} \quad 0 \quad 6.120\,10^{-5} \\ 0 \quad 2.868\,10^{-4}) \quad (5.18)$$

Simplifying the expanded equation, the polynomial approximation solution is shown in Equation (5.19).

$$\begin{aligned} \hat{u}(x,y) = {} & 4.679\,10^{-2}\,y^8 - 2.722\,y^7 - 1.310\,x^2\,y^6 + 1.310\,x\,y^6 + 9.138\,y^6 \\ & + 0.423\,x^2\,y^5 - 0.423\,x\,y^5 - 12.544\,y^5 + 3.276\,x^4\,y^4 - 6.551\,x^3\,y^4 \\ & + 4.785\,x^2\,y^4 - 1.509\,x\,y^4 + 8.814\,y^4 - 1.654\,x^4\,y^3 + 3.308\,x^3\,y^3 \\ & - 5.365\,x^2\,y^3 + 3.711\,x\,y^3 - 3.316\,y^3 - 1.310\,x^6\,y^2 + 3.931\,x^5\,y^2 \\ & - 3.360\,x^4\,y^2 + 0.169\,x^3\,y^2 + 1.468\,x^2\,y^2 - 0.897\,x\,y^2 + 0.631\,y^2 \\ & + 0.580\,x^6\,y - 1.741\,x^5\,y + 2.903\,x^4\,y - 2.904\,x^3\,y + 0.238\,x^2\,y \\ & + 0.924\,x\,y - 0.0481\,y + 4.679\,10^{-2}\,x^8 - 0.187\,x^7 + 9.926\,10^{-2}\,x^6 \\ & + 0.357\,x^5 - 0.592\,x^4 + 0.371\,x^3 - 0.106\,x^2 + 1.136\,10^{-2}\,x \\ & + 2.449\,10^{-4} \quad (5.19) \end{aligned}$$

5.6 Discussion

As a final discussion in this chapter, there are some issues of the proposed method to point out. One important issue is the workaround made for conditions that are functions themselves. Polynomials are bounded to inflections which, in turn, are strongly dependent on their polynomial degrees. Because of that, when sampling boundary conditions, some "ripples" on boundary are perceived, specially when analysing error plots. This is the case that could invalidate dealing with piecewise sub-domains, where values over borders could be of low confidence. One workaround could be to do overlap sub-domains, but this is an entire discussion by itself. Though, if the purpose of such polynomial approximation is to interpolate the solution within boundary, the proposed method is extremely useful.

Because the use of polynomial approximations for differential solutions, the considered domain of calculation must be relatively small in order to minimize the polynomial degree required for a feasible approximation. When dealing with periodic or trigonometric solutions, the problem worsens. Each point of inflection of such solutions reflects as an increment to the required polynomial degree. High polynomial degrees lead to possible floating-point calculation errors and that should be avoid. A workaround is to consider piecewise sub-domains. Unfortunately, there are a lot of situations where this is not possible. In the case of system modelling from data, however, this is a characteristic that must be exploit, as can be seen in next chapters.

Finally, the rank of coefficient matrix also depends on the chosen polynomial degree for the approximation. The higher it is, the more likely that the matrix starts to have increased its rank deficiency. The number of sampled points from boundary conditions needs to compensate this effect. Note that, specially on homogeneous differential equations, conditions that have non-zero values are the most useful ones.

Part III

System Modelling

Chapter 6

Evaluating model candidates

6.1 A brief introduction

The main objective of this work is to take the proposed method and exploit it to support CASM. The decision of using GP to model systems from data has guided the following developments. To accomplish that intention, the proposed method needs to be part of a fitness evaluation scheme to support evolution of models, by solving any LPDE that could be randomly generated by GP and evaluating how fit its respective approximate solution is to the system observed data.

In Section 5.6, some issues for the proposed method are addressed. In special, the ones stating that both small domains of calculation are required to decrease the required polynomial degree for approximations and the limitation about the small number of known conditions the PDE has to enable performing piecewise calculations throughout the desired domain. Those are not concerns hereby and were exploited to bring up a way to evaluate differential models.

It is a requirement that measurements from the system taken over some points are saved in a data file containing the observed quantities values (up to this far, this work considers scalar fields only) and the coordinates where the quantity to be modelled assumes those values. It is pretty straightforward to assume that every point in the database is a possible Dirichlet boundary condition.

By taking groups of closer points within the database, the proposed method could perform piecewise approximations with a great degree of confidence with a relative low polynomial degree. If other points are also taken just to enable comparisons between the approximate solution and their respective database values, then a metric can be developed to evaluate how close a model is from the observed data. From these ideas, the derived fitness evaluation scheme is presented in this chapter.

6.2 A brief description

A brief description presented here about the numerical scheme to compare a LPDE candidate with observed measurements from a dataset.

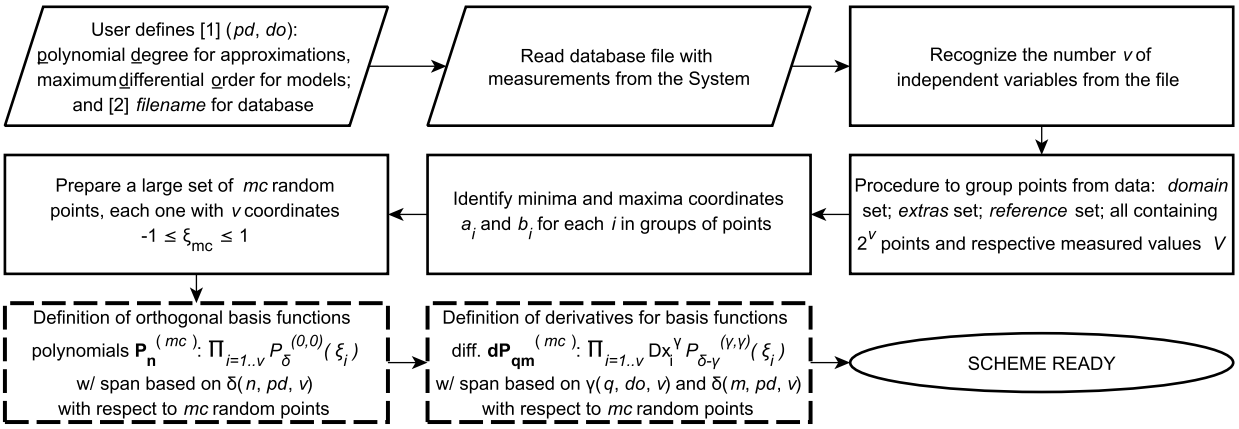

Figure 6.1: Preparation steps for the proposed fitness evaluation method; dashed border nodes can benefit from parallelism.

i. Dataset constituted of coordinates and respective measurements from the quantity of interest is then divided into groups of points. Each group contains a number of points which depends on the dimension of the problem (3×2^D points, where D is the number of independent variables).

ii. One third of those points are used to define a domain of calculation (domain set) in order to retrieve the TGE coefficients (as a "training set"). Other third, named as extras set, is responsible to augment auxiliary conditions whenever the rank of the individual GSE coefficient matrix requires to. The final third, addressed as the reference set, are used as "targets" to enable numerical comparison between values found from the approximate solution to those in the dataset.

iii. For each group, a local polynomial approximation is built using the method shown in Chapter 5 with respect to points in the domain set, sometimes augmented by the ones in the extras set.

iv. The numerical difference between the polynomial approximation subject to the reference points and their respective values retrieved from data is the base for the fitness evaluation of a model candidate. In the end, an overall average is performed over all groups and this metric is used as the model candidate fitness.

6.3 Method, step by step

Figure 6.1 presents the preparation steps and figure 6.2 presents the flowchart for the proposed method of fitness evaluation. Note that dashed border nodes are eager to benefit from parallelism.

Following sections explain some details about the flowchart.

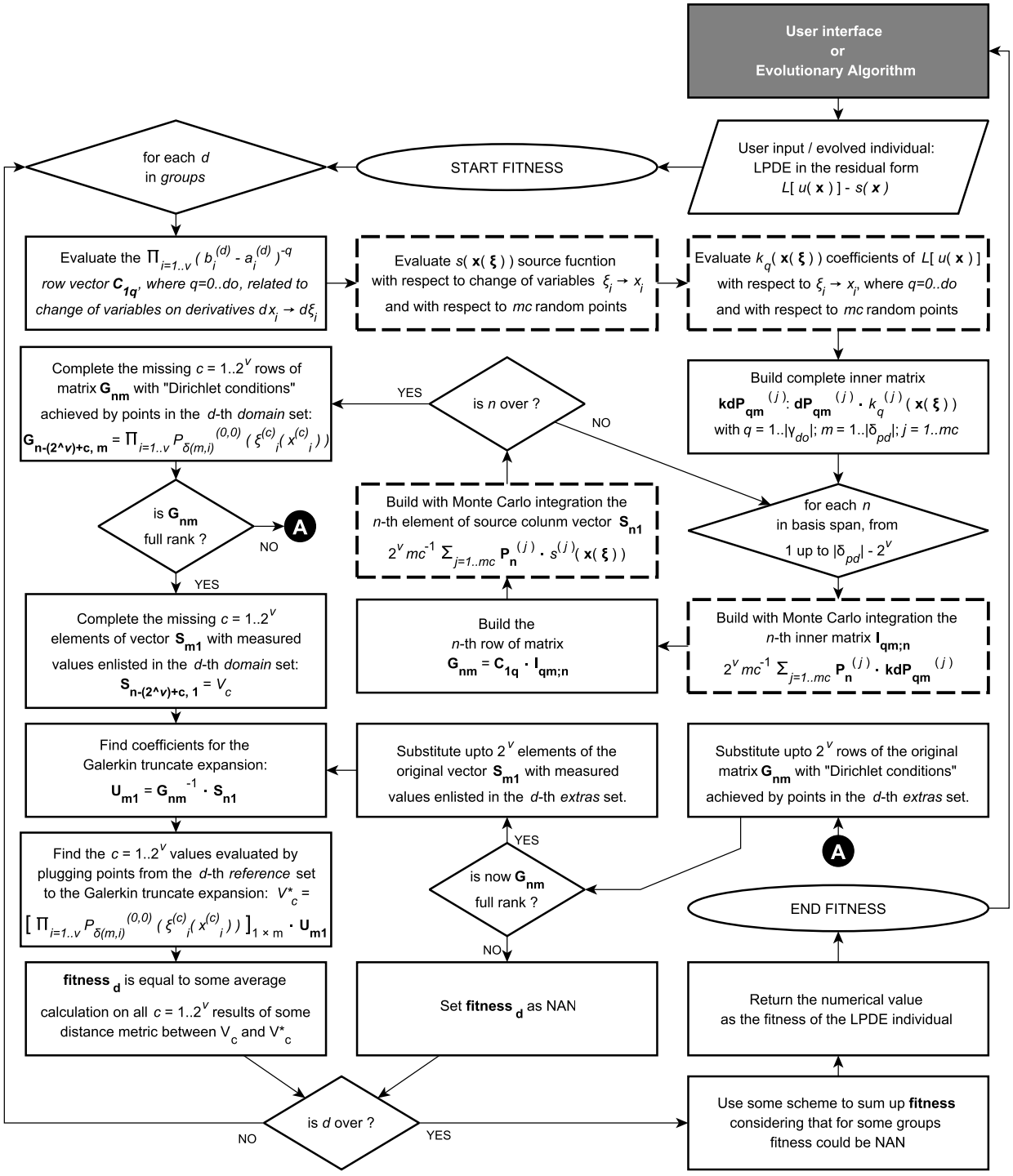

Figure 6.2: Flowchart for the proposed fitness evaluation method; dashed border nodes can benefit from parallelism.

Table 6.1: Example of a data file with 3 independent variables and 1 dependent variable (quantity of interest representing a scalar field).

0.0	0.0	0.2	3.12
0.1	0.3	0.5	-0.13
0.2	0.1	0.1	3.11
⋮	⋮	⋮	⋮
2.0	1.8	3.1	5.2

User definition

User needs to inform: (a) the polynomial degree to be used in polynomial approximations; (b) the maximum order for derivatives, based on his feelings about the system, or a high enough order to uncover something unexpected; (c) and the name of the data file. If the maximum order informed is greater than the polynomial degree, the order will be automatically decreased (high order derivatives applied to low degree polynomials are useless within this situation). Also, the polynomial degree is responsible for the order of the coefficient matrix, once it defines the expansion of the TGE. Depending on the number of variables, this degree is increased to ensure some rows of the original GSE matrix formulation to be in the final coefficient matrix.

Read data file

A file containing measured data is expected to feed the application. Up to this far, only scalar fields are supported, so there is no need for a header line with complementary information. The file could be something as shown in Table 6.1.

Also, the number of independent variables (D) to be used hereby is automatically retrieved from the data file.

Grouping points from dataset

Before grouping of points, some ideal cluster centroids are calculated based on the full domain of the data. The number of those centroids is defined as the total number of points in the dataset divided by the number of points inside a group. Those centroids are equidistantly spread throughout the domain in order to ensure that every corner is represented. Then, based on Euclidean distances, from the original dataset are taken the closest points to each of those centroids.

After, the scheme performs the grouping of points (coordinates and quantity respective measurements) by finding the closest points to those already kept in order to define a group. The number of points in each group is based on the identified number of independent variables (3×2^D). The achieved number of groups in this stage is the number of centroids. There could be same points in different groups.

Then, each chosen group has its boundary calculated (for each dimension, the minimum and the maximum values are found) and its points are divided into three sets: domain, extras and reference; each one of these contains the same number of points (2^D). The first one, the domain set, has to contain the closest points to the group limits.

This stage ends with groups containing information about each dimension boundary, and their respectively points divided into those three sets.

Random chosen points

In order to carry the method out, some discussion about random points must be addressed. If every time a large set of random points is needed they would effectively be generated, the final application would suffer with respect to execution time, perhaps invalidating the whole process. A workaround was delineated though. Once per execution, a large set of random points is generated from the interval $[-1, 1]^D$ and saved for future uses.

This may seems odd, once randomness is the core idea for some of the techniques shown in this work. Addressing specifically to Monte Carlo integration, when using always the same random chosen points (in this case, only random coordinates are generated), a bias is introduced to the results. In the other hand, if the set is large enough, the quality of results would be trustful. Also, if the GSE coefficient matrix is well-conditioned, small variations on integration results would not be sensed when finding TGE coefficients.

Other argument to reuse the pre-generated large set is that some functions in need to perform Galerkin method could be evaluated once per run — as orthogonal base functions and respective derivatives —, if some change of variables are introduced here. Other functions, as the ones introduced by the process of changing variables, could be performed once per group of points, regardless the model under evaluation. The rest of functions are always dependent on the differential equation to be evaluated.

Following those ideas, this scheme proposes to generate such large set and reuses those random points only when a Monte Carlo integration is performed. Other needs for randomness, as the ones related to the evolutionary algorithm itself, should be fulfilled by a proper pseudo random number generator. Note that the quantity for those random points could also be a parameter to the user.

Definition of orthogonal basis functions

Using the user defined polynomial degree and Equation (5.6) as a guide, with $\alpha = \beta = 0$ (Legendre polynomials), orthogonal base functions are then automatically defined respecting the powers matrix from Algorithm 3.

Those base functions should be evaluated with respect to the large set of random points previously generated.

Definition of derivatives for basis functions

Using the user defined differential order and Equation (5.7) as a guide, derivatives for base functions are then automatically defined respecting the powers matrix from Algorithm 3.

Those derivatives should be evaluated with respect to the large set of random points previously generated.

User interface or evolutionary algorithm

This stage is due to capture the differential model to be evaluated.

Start/end of group-based loops

The proposed method is then applied, as seen in Chapter 5. A loop on each group is then performed. Some notes about the process:

- Matrices from GSE are built using Monte Carlo integration when needed. Note that basis functions and derivatives are already evaluated with respect to each of the large set of random points.
- Auxiliary conditions considered here are Dirichlet-like conditions built from points in the domain set and always inserted in the GSE original matrix, transforming it in the G_{nm} matrix from flowchart.
- After building it, the rank of G_{nm} is verified. If it is rank deficient, one by one Dirichlet-like conditions built from points in the extra set substitutes a different row of G_{nm} until it becomes full rank or the extra set runs out of points. In this latter case, the partial fitness for this group is considered invalid.
- When testing for the full rank requirement, a test for the condition of matrix G_{nm} is also performed, as shown in Algorithm 1. The effect for an ill-conditioned matrix is the same as if the matrix was rank deficient, *i.e.*, the partial fitness for this group is considered invalid as well.
- The partial fitness of each group is then evaluated by averaging the absolute error between values in dataset and from local approximation, both with respect to points in the reference set.

When all partial fitness are evaluated for all groups, a global metric is finally applied ignoring non-valid partial fitness values. This metric penalizes LPDE individuals that have groups with those non-valid values.

6.4 Examples

Two examples of fitness evaluation are addressed here, one from an underdamped oscillator problem and the other from a Poisson equation for electrostatics.

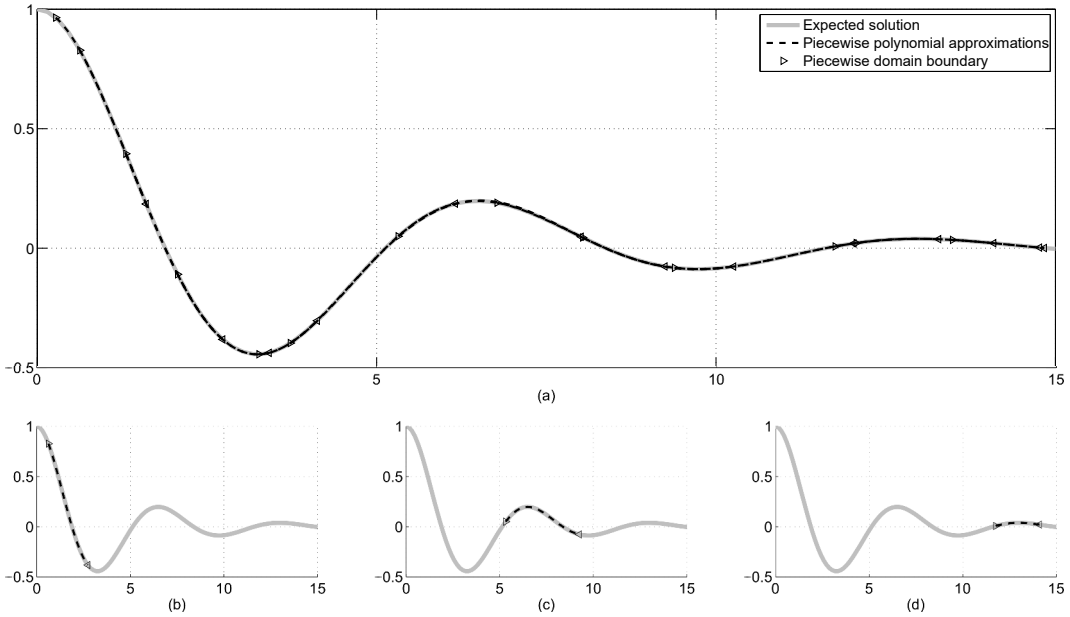

Figure 6.3: Overlap of solution plot and piecewise approximations for the under-damped oscillator problem; overall fitness evaluated as $4\,10^{-4}$. (top) All 13 piecewise domains and approximations. (low left) Approximation over the 2nd considered domain. (low center) Approximation over the 7th considered domain. (low right) Approximation over the 11th considered domain.

Under-damped oscillator The same under-damped oscillator problem in Section 4.4 is used here as a numerical example. For this example, Equation (4.10) has been simulated in order to sample some points (respective displacements in function of time) to build from scratch a dataset file. A total of 30 points were sampled. The achieved data file was presented to the implemented fitness scheme. After the preparation stage, a total of 13 groups of points were evaluated.

Results could be seen in Figure 6.3. Table 6.2 presents the TGE coefficients for 13 piecewise polynomial approximations of 5th degree. Fitness was evaluated as $3.61\,10^{-4}$ when adopting $2^9 = 512$ points for Monte Carlo integration.

Poisson electrostatic A second example is the same Poisson equation for electrostatics problem in Section 4.4. For this example, Equation (4.16) has been simulated in order to sample some points (respective electrostatic potential in function of radius) to build from scratch a dataset file. A total of 30 points were sampled. The achieved data file was presented to the implemented fitness scheme. After the preparation stage, a total of 13 groups of points were evaluated.

Results could be seen in Figure 6.4. Table 6.3 presents the TGE coefficients for 13 piecewise polynomial approximations of 5th degree. Fitness was evaluated as $7.44\,10^{-6}$ when adopting $2^9 = 512$ points for Monte Carlo integration.

Table 6.2: TGE coefficients retrieved for the under-damped oscillator example, each set related to one of 13 groups of points.

Domain #	Interval	$\tilde{u}_0$	$\tilde{u}_1$	$\tilde{u}_2$	$\tilde{u}_3$	$\tilde{u}_4$	$\tilde{u}_5$
1	$[0.2727, 1.6041]$	0.6232	−0.4054	−0.0484	0.0155	$-1.19\,10^{-4}$	$-1.05\,10^{-4}$
2	$[0.6273, 2.7321]$	0.1661	−0.6477	0.0620	0.0431	−0.0052	$-4.66\,10^{-4}$
3	$[1.3062, 3.4157]$	−0.1737	−0.4324	0.1584	0.0158	−0.0062	$7.77\,10^{-5}$
4	$[2.0680, 4.1260]$	−0.3594	−0.0867	0.1568	−0.0107	−0.0040	$4.05\,10^{-4}$
5	$[3.2670, 6.1594]$	−0.1299	0.3693	−0.0050	−0.0555	0.0059	0.0014
6	$[3.7294, 8.0121]$	0.0241	0.2541	−0.2342	−0.0299	0.0354	−0.0019
7	$[5.3168, 9.2467]$	0.0908	−0.1232	−0.1077	0.0644	0.0035	−0.0044
8	$[6.7746, 10.2580]$	0.0118	−0.1587	0.0523	0.0258	−0.0077	$-5.36\,10^{-4}$
9	$[8.0350, 12.0310]$	−0.0417	0.0086	0.0801	−0.0230	−0.0061	0.0021
10	$[9.3891, 13.2800]$	−0.0208	0.0820	−0.0054	−0.0232	0.0037	0.0010
11	$[11.7560, 14.0910]$	0.0303	0.0051	−0.0168	0.0016	$5.25\,10^{-4}$	$-6.88\,10^{-5}$
12	$[12.0670, 14.7650]$	0.0274	−0.0135	−0.0160	0.0042	$4.44\,10^{-4}$	$-1.54\,10^{-4}$
13	$[13.4850, 14.8370]$	0.0184	−0.0175	$-8.33\,10^{-4}$	$6.02\,10^{-4}$	$-1.80\,10^{-5}$	$-3.72\,10^{-6}$

Table 6.3: TGE coefficients retrieved for the Poisson electrostatics example, each set related to one of 13 groups of points.

Domain #	Interval	$\tilde{\varphi}_0$	$\tilde{\varphi}_1$	$\tilde{\varphi}_2$	$\tilde{\varphi}_3$	$\tilde{\varphi}_4$	$\tilde{\varphi}_5$
1	[0.2178, 1.3861]	0.7122	−0.0981	−0.0155	0.0033	$8.06\,10^{-5}$	$-4.88\,10^{-5}$
2	[0.8510, 1.7056]	0.6243	−0.0894	−0.0013	0.0011	$-6.05\,10^{-5}$	$-5.14\,10^{-6}$
3	[0.9298, 2.3669]	0.5499	−0.1440	0.0064	0.0032	$-5.76\,10^{-4}$	$-8.09\,10^{-7}$
4	[1.6085, 3.1810]	0.4188	−0.1203	0.0160	$-1.66\,10^{-4}$	$-3.94\,10^{-4}$	$6.97\,10^{-5}$
5	[1.7399, 4.4825]	0.3393	−0.1473	0.0367	−0.0054	$-5.69\,10^{-4}$	$4.59\,10^{-4}$
6	[2.7034, 4.7470]	0.2752	−0.0764	0.0137	−0.0020	$1.81\,10^{-4}$	$5.62\,10^{-6}$
7	[3.4375, 5.3549]	0.2312	−0.0510	0.0075	$-9.65\,10^{-4}$	$1.11\,10^{-4}$	$-9.64\,10^{-6}$
8	[4.5604, 5.7314]	0.1952	−0.0223	0.0017	$-1.16\,10^{-4}$	$7.58\,10^{-6}$	$-4.66\,10^{-7}$
9	[5.3325, 6.5754]	0.1686	−0.0176	0.0012	$-7.73\,10^{-5}$	$4.66\,10^{-6}$	$-2.71\,10^{-7}$
10	[5.4296, 7.5067]	0.1560	−0.0252	0.0027	$-2.63\,10^{-4}$	$2.48\,10^{-5}$	$-2.23\,10^{-6}$
11	[6.0596, 7.9634]	0.1435	−0.0196	0.0018	$-1.46\,10^{-4}$	$1.15\,10^{-5}$	$-8.74\,10^{-7}$
12	[6.9232, 9.1420]	0.1253	−0.0174	0.0016	$-1.34\,10^{-4}$	$1.08\,10^{-5}$	$-8.33\,10^{-7}$
13	[7.8138, 9.9617]	0.1131	−0.0137	0.0011	$-8.07\,10^{-5}$	$5.66\,10^{-6}$	$-3.82\,10^{-7}$

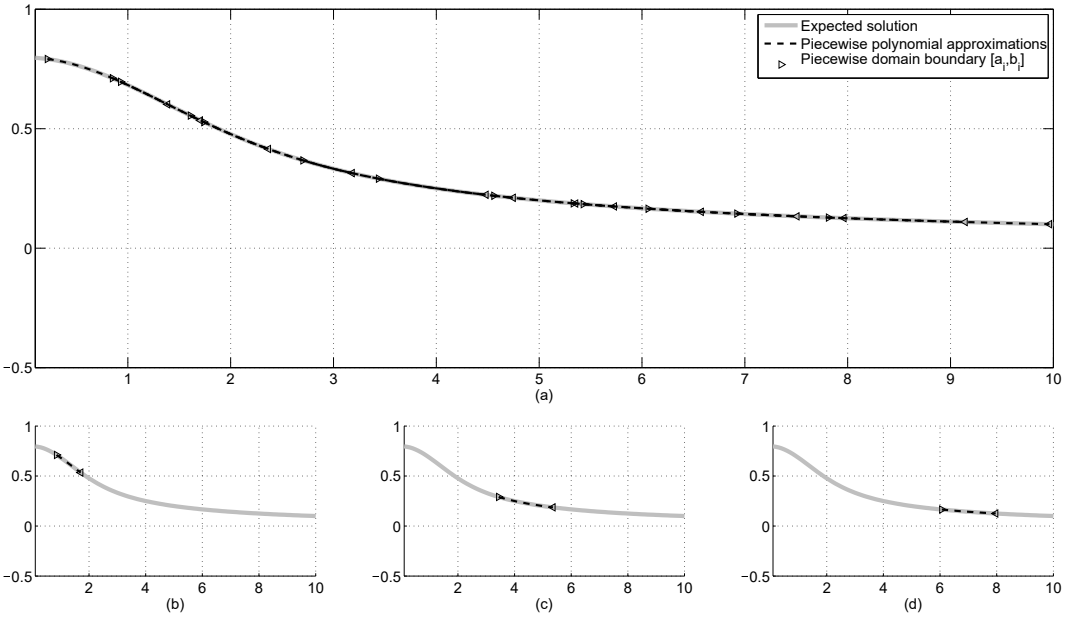

Figure 6.4: Overlap of solution plot and piecewise approximations for the Poisson electrostatics problem, overall fitness evaluated as $7.44\,10^{-6}$. (top) All 13 piecewise domains and approximations. (low left) Approximation over the 2nd considered domain. (low center) Approximation over the 7th considered domain. (low right) Approximation over the 11th considered domain.

6.5 Discussion

In this section, a fitness evaluation scheme is presented. As shown, some issues addressed by discussions in the latter chapter are exploited here as advantages to the proposed scheme.

Domains of calculation tend to be small because of grouping strategy adopted (closer points to be taken as needed groups), as long as there are enough points in the database to represent the phenomenon to be modelled. Being so allows to use small degrees in order to perform a piecewise polynomial approximation. The existence of the "extra" group allows to deal with deficient matrices with a higher probability of transforming them into well-posed problems.

Examples shown in this chapter indicate the feasibility of the proposed scheme. Although presented examples are LODEs, in the next chapters a first time automate solved LPDE example constitutes an important result to this thesis.

Chapter 7

System Modelling program

7.1 Background

In this section, some key subjects for the GP implementation are discussed. This is a preliminary implementation to investigate how the proposed method behaves when guiding evolution of LPDE models.

Representing a LPDE

To this project, an individual means a differential model candidate, *i.e.*, a LPDE in its residual form. In order to evolve models for a system of interest, a proper representation should be adopted in order to enable an evolutionary algorithm to operate on them. Also, as starting from linear differential equations paradigm, models under analysis here need to be linear. The way of ensure this requirement is to fix a macro structure to protect an individual randomly built from being a non-linear PDE (in which case the proposed method would fail to evaluate). To fit this requirement, a fix-sized vector dependent on the number of independent variables and the maximum order for the differential of randomly built coefficients is here designed.

Figure 7.1 presents an example of representation that means the differential equation: $[5\,\cos(\pi x)] \cdot \frac{\partial^2}{\partial y^2} u(x,y) + [-2.5] \cdot \frac{\partial}{\partial x}\frac{\partial}{\partial y} u(x,y) + \left[\exp\left(-\frac{y}{2}\right) + x\right] \cdot \frac{\partial}{\partial y} u(x,y) + [1] \cdot u(x,y) = \sin(\pi x)\,\cos(\pi y)$. Note that each coefficient is an algebraic abstract syntax tree (AAST), either a constant, a linear or a non-linear function, and is supposed to be built at random. Those coefficients populate a fix-sized vector. This vector has the length of $N^\star + 1$ — from Equation (5.2) plus 1 element to get the source function $s(\vec{x})$ itself — and holds elements that are mainly coefficients for a set of derivatives whose individual orders are based on powers matrix up to user's definition. In this present example, user has defined models of order 2 with respect to a system whose measurements covers 2 independent variables (x, y). Those decisions will affect the powers matrix presented in Algorithm 3 and reflect to the size of the vector.

Therefore, by using this developed representation, allows the proposed scheme for evaluation of models is allowed to work with any instance of evolutionary algorithms. Some modifications should be applied if the EA in question does not support variable sized individuals. As an example, if one wants to

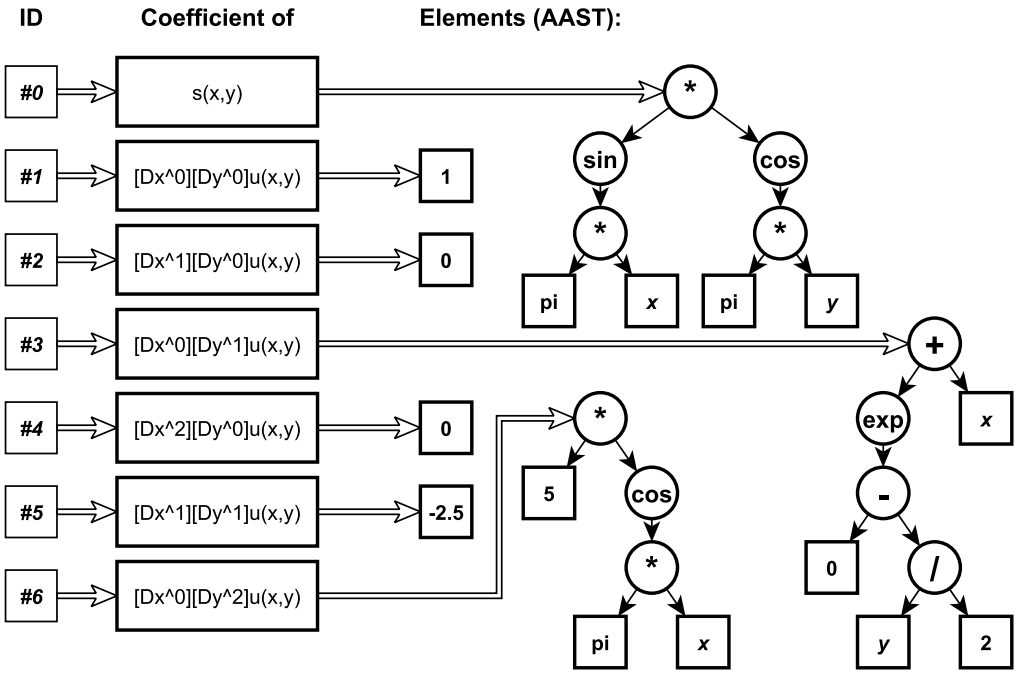

Figure 7.1: An example of model candidate representation, a vector of AAST's. This example represents the LPDE $[5\cos(\pi x)] \cdot \frac{\partial^2}{\partial y^2}u(x,y) + [-2.5] \cdot \frac{\partial}{\partial x}\frac{\partial}{\partial y}u(x,y) + \left[\exp\left(-\frac{y}{2}\right) + x\right] \cdot \frac{\partial}{\partial y}u(x,y) + [1] \cdot u(x,y) = \sin(\pi x)\cos(\pi y)$ and each coefficient (an AAST) is supposed to be built at random for a deterministic vector which length and element meanings are based on user's definition of order 2 for differentials regarding a system whose measurements covers 2 independent variables (x, y).

use this with classic Genetic Algorithm, it is enough to fix all coefficients as being constants. This work, however, uses Genetic Programming to show the full potential of the proposed method.

Genetic Operators

Based on individuals representation, some genetic operators had to be developed in order to perform individual structural changes at random and generate a new individual. There are basically two types of operators within an evolutionary cycle, those ones who operate on a single individual (unary) and those ones who operate multiple individuals. For this project, the latter type was defined to operate just on two individuals (binary).

Starting from binary operations, this implementation has three equiprobable variants here named as type-I recombination, type-II recombination and type-III recombination. Those are presented in Figure 7.2.

When binary operations is randomly chosen to be applied, two individuals to be known as "parents" are selected by a proper method. During the operation, a third individual, named as "offspring", is then created. Note that if the

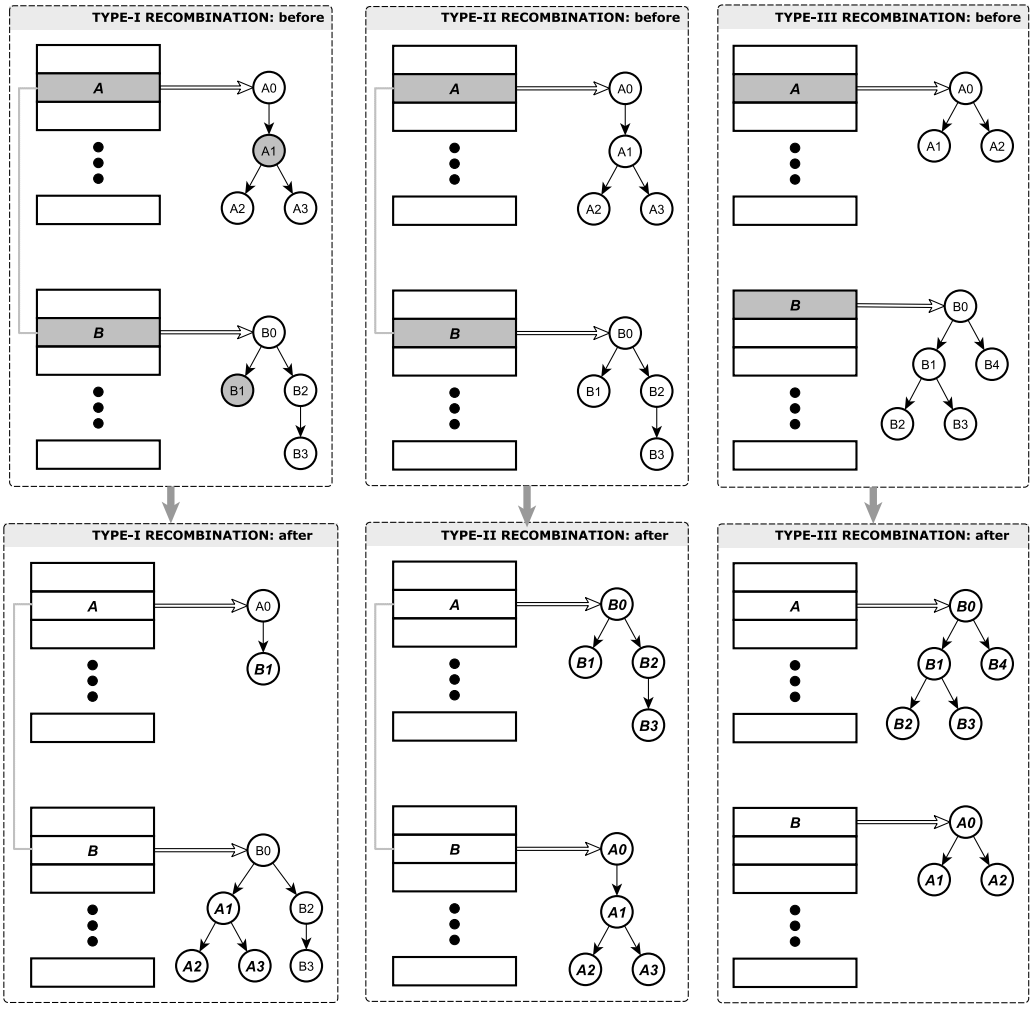

Figure 7.2: Types of binary operators. Shaded elements represent random chosen points for operations. (left) Type-I recombination. (center) Type-II recombination. (right) Type-III recombination.

application of binary operators fail to occur, one of the selected individuals is copied as is to the next generation. This copy operation is called reproduction.

Type-I recombination is based on the classic recombination from GP, here adapted to the described individual representation. Each allele (element from the vector of AASTs) is then checked against a proper probability[1]. If it is allowed, both AASTs placed on that allele position, one from the first parent and other from the second one, have to exchange sub-trees. The nodes to be exchanged are randomly chosen once per parent, with 90% probability of selection an internal node.

Type-II recombination is based on the classic one from Genetic Algorithms. Each allele is checked against a proper probability. If it is allowed, both parents exchange their entire AASTs in the same indicated allele position. Note that the crossover operator could be mimicked by recombination, if both parents had their randomly chosen nodes being their respective AAST roots.

[1]The scheme adopted here has a long term average of one allele chosen per individual.

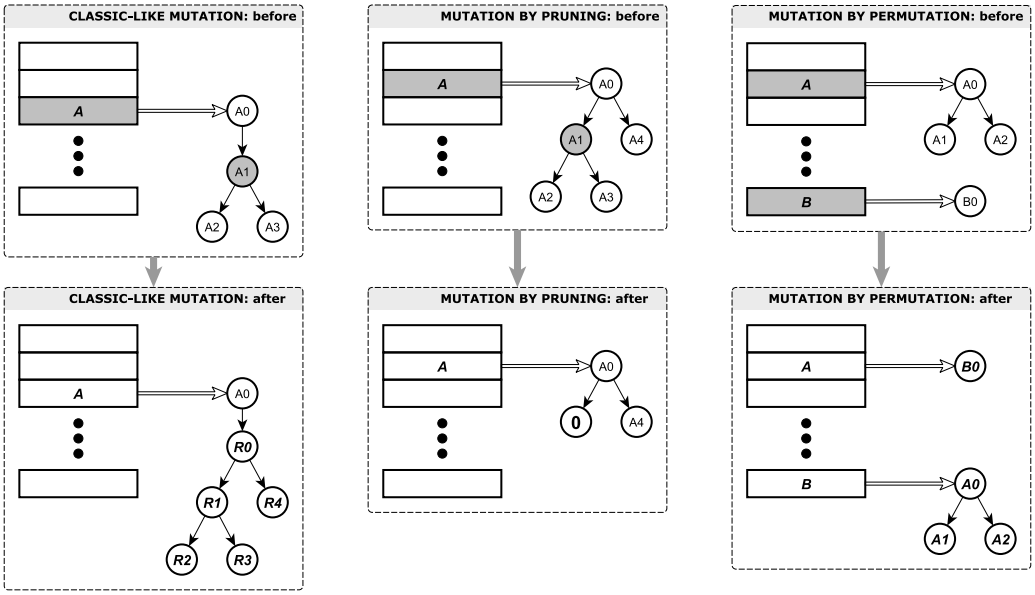

Figure 7.3: Types of unary operations. Shaded elements represent random chosen points for operations. (left) Classic-like mutation. (center) Mutation by pruning. (right) Mutation by permutation.

Type-III recombination is a disruptive operator at some level, used here to raise diversity. Its behaviour is the same as the type-II, but the alleles from both parents could be different. Note that the type-II recombination operator could also be mimicked by bartering, once the separated choice of alleles for both parents ends up to be of the same position.

Note that all binary variants generate two offspring. This project assumes to discard one of them at random to ensure that every genetic operator, no matter if binary or unary, generates only one new individual.

As unary operations, this implementation has also three equiprobable variants: the classic-like mutation, mutation by pruning and mutation by permutation. Those are presented in Figure 7.3.

Classic-like mutation is based on classic mutation from GP. A node from the AAST in the elected allele is randomly chosen to be replaced by a random built AAST. The choice of which node is selected to be modified follows the classic GP probability: 90% of being an internal node and 10% of being a leaf. Tendency is to the growth of the tree.

Mutation by pruning is similar to mutation, but essentially different. Instead of substituting a random chosen node by a new AAST, this operator substitutes it by the zero terminal (constant). A special case needs to be delineated when the elected allele is but the two first ones (the ones representing the source function and the coefficient of the unknown solution without derivatives). This operator counts how many of the alleles containing derivative terms are already zero. If all of those alleles are zero, pruning makes the substitution by the constant 1 instead of 0, to preserve the LPDE nature of the model.

Mutation by permutation is similar to type-III recombination, but the two alleles to exchange AASTs belong to the same individual. The elected allele is exchanged by another randomly chosen one.

7.2 GP preparation step

This project assumes as the terminal set some useful constants to support a wide range of problems: $\{-1, 0, 1, 2, 3, \pi\}$. User is also invited to augment the terminal set with known constants from the system of interest, *e.g.*, acceleration of gravity, body mass, Rayleigh factor, spring constant. Note that this work is limited to real numbers.

The function set has all basic operators: $\{+, -, \times, \div\}$. Some other common functions are included, some in their unary versions, others encapsulating more than one function:

- $\texttt{NEG}(x) = -1\,x$
- $\texttt{INV}(x) = \frac{1}{x}$
- $\texttt{POW2}(x) = x^2$
- $\texttt{POW3}(x) = x^3$
- $\texttt{EXPn}(x) = \exp(-x)$
- $\texttt{SQRT}(x) = \sqrt{x}$
- $\texttt{CBRT}(x) = \sqrt[3]{x}$

The adoption of such set was due to both the need of those functions and the the numerical instability they bring to automated modelling. For example, if $\exp(x)$ is adopted, a tree as $e^{e^{e^2}} \to \infty$ becomes undesirably possible. By user's discretion, other functions could be joint to this set. Note that all functions with singular points in the domain has been protected. This means that every time a function returns an invalid value, the protected function returns the value 1. No corruption of the final individual is sensible, however, due to the fact that evolved solutions are simplified before presentation to the user and the simplification technique was built to incorporate the protected versions of those functions.

Selection of parents is performing using the tournament method, *i.e.*, some arbitrary number of individuals (three by default here) are chosen at random and the one with the best fitness wins, turning himself into the first parent. This procedure is repeated to retrieve a second parent. From a point of view of evolution, this method is very useful for tuning evolutionary pressure.

Reduction is performed by two possible methods, chosen at random. The first method is to discard the entire population of parents and keep the offspring as the next generation. The best individual is always kept, though. This method inflicts a very low evolutionary pressure. The second method is to go

through the list of parents side-by-side with the list of offspring and, for each pair, the best of the two is kept for the next generation. This inflicts a mild evolutionary pressure.

Initial population strategy for randomly build initial candidates, as classic GP, is 50% grow and 50% full per each allele.

Parameters for the GP: (a) number of individuals in the population; (b) probability for binary genetic operations; (c) probability for unary genetic operations; (d) number of individuals to participate a tournament; (e) maximum depth for initial AASTs; (f) maximum number of generations.

Parameters for CASM: (a) number of Monte Carlo random points for integration purposes; (b) the desired polynomial degree for approximations; (c) expected differential order for the solution.

The termination criteria adopted here is a predetermined maximum number of generations or when the standard deviation for the entire population fitness keeps unchanged for a significant number of generations.

7.3 GP run

A GP run of the proposed CASM would be:

1. Perform the stages of preparation to fitness (see Chapter 6), *i.e.*, inform the polynomial degree for approximations, the maximum order for differentials and the file name of the dataset; the load the datafile; retrieve the number of independent variables; separate points from dataset into groups; identify minima and maxima coordinates for each group; generate a large set of pseudo random numbers; define and evaluate basis functions and respective derivatives with respect to the random points.
2. Initialize population at random, generating differential model candidates.
3. Perform the fitness evaluation for the initial population.
4. Initiate the evolutionary cycle.
5. Generate offspring population by applying binary and unary genetic operators to individuals in the parent population.
6. Perform the fitness evaluation for the offspring.
7. Reduce population to its initial number of individuals.
8. Gather control data from the new population.
9. Verify if there is a match for termination criteria; if there is not, return to item 4.
10. Return the best fitness individual as the proposed differential model to explain dataset.

7.4 C++ supporting classes

In order to implement the CASM desired application in C++, some supporting classes were developed:

- `cPRNGXS.h`, a class to produce pseudo random number generators, based on xorshift*64 and xorshift*1024 [83].
- `cAAST.h`($\star$) and `cexpAAST.h`($\diamond$), classes to implement Algebraic Abstract Syntax Trees (AAST), from scratch. Possible nodes[2] are:
  - Constants $\star$: arbitrary real numbers;
  - Variables $\star$: identified by sub-indices of the x symbol (*e.g.*, x_0, x_1);
  - Operators $\star$: the four arithmetic operators – plus, minus, division, and product;
  - Functions $\star$: trigonometric, mathematical or user defined ones;
  - Unknown functions $\diamond$: this special node represents the multivariate function (with respect to all related variables) that is the solution of the desired partial differential model;
  - Derivatives $\diamond$: this node represents a derivative operator of any order;
  - Unknown constant $\diamond$: represents the coefficients of the TGE, the unknowns of the custom system of equations.
  - Polynomials $\diamond$: to represent Jacobi polynomials in a nutshell.
- `cMatrix.h`, a class to implement matrices and linear algebra operations and operators, based on BLAS and LAPACKE libraries for C++.
- `prepareData.h`, a class designed to read ASCII data file and prepare points and measurements to the fitness evaluation procedure.
- `cFitness.h`, a class to encapsulate the proposed fitness scheme by itself (see Chapter 6).
- `GenProg.h`, a simple class to implement dedicated procedures to support a simple Genetic Programming application based on AASTs.
- `myfun.h`, a general class to host several supporting functions, as the pochhammer symbol, integer partition, *etc.*

All those classes were designed to keep the programmer in control of the full process. Some of them depends on the others, others are independent for arbitrary uses. Their hierarchical order is: `myfun` → `prepareData` → `cAAST/cexpAAST` → `cMatrix` → `cPRNGXS` → `cFitness` → `GenProg`. So, to develop a Genetic Programming application to be used as a CASM tool, is enough to just include `GenProg`. Note that classes `myfun`, `cAAST`, `cMatrix` and `cPRNGXS` could be used independently for purposes other than the original ones. Also, those classes should be understood as prototypes provided "as is".

[2] After changing the calculation strategy to matrices, some of those nodes became useless.

7.5 Discussion

Results and discussion for the implemented GP subject of this chapter are presented in the next chapter.

Chapter 8

Results and Discussion

8.1 Outline

In this chapter, are presented: (a) preliminary results from the GP CASM tool is presented; (b) electronic address from the repository where one can find source codes used in this thesis; (c) discussions about contributions; (d) indications for future works.

8.2 GP for system modelling

In this section, some details on implementation and GP execution are shown, as well as some preliminary results which indicates the potential of CASM for LPDE models through GP.

Computational environment

All examples in this thesis were executed in a `Intel Core I7-3770 CPU @ 3.40GHz` machine, `8Gb` RAM memory, running Ubuntu `14.04LTS 64bits` with `3.16.0-46-generic` kernel.

Preliminary results

A simple concentration problem is presented, adapted from [84]:

$$\text{PDE} \quad u_x + u_t = 0 \tag{8.1}$$

$$\text{IC} \quad u(x,0) = \cos(x) \tag{8.2}$$

The known solution for this problem is shown in Equation (8.3).

$$u(x,t) = cos(x-t) \tag{8.3}$$

For this example, Equation (8.3) has been simulated in order to sample some points at random to build a dataset file from scratch. A total of 30 points were sampled from the domain $0 \leq x \leq 3,\ 0 \leq t \leq 3$. The achieved data file

was presented to the implemented fitness scheme. After the preparation stage, a total of 13 groups of points were evaluated.

Chosen parameters for the GP run:

- number of individuals in the population: 100
- probability for pairwise genetic operations: 90% (10% reproduction)
- probability for oneself genetic operations: 20%
- number of individuals to participate a tournament: 3
- maximum depth for initial AASTs: 7
- maximum number of generations: 30

This simple example has converged to the correct differential model `( D_x0 ( UX ) + D_x1 ( UX ) )` in approximately 18 minutes. The proposed method to fitness evaluation had to perform 3000 times with little effort because of the low polynomial degree (3rd degree), a low differential order problem (1st order) and a relative small set of random points for Monte Carlo integrations ($2^9 \approx 500 points$).

Figure 8.1 shows the register from the best individual fitness for every generation, *i.e.*, the path for evolution to find the solution. Plot for mean population fitness is not included due to the high variability presented. Results could be seen in Figure 8.2. Table 8.1 presents the TGE coefficients for 9 piecewise polynomial approximations of 3rd degree. Fitness was evaluated as $3.36\,10^{-3}$ when adopting $2^9 = 512$ points for Monte Carlo integration.

A non-convergent example

There was an attempt to model the system described in Equation (4.15), the Poisson equation for the spherical symmetric electrostatic potential. However, the Genetic Programming was not successful to converge to a fine solution. Results achieved a fitness between $5\,10^{-4}$ and $2\,10^{-3}$. The correct solution has a fitness of $7\,10^{-6}$ (see Figure 6.4).

A test was then performed in order to verify if the proposed method was compromised. The PDE known to be the solution was inserted into the initial population to check if it could be either dumped or degenerated over generations. The result, however, was very promising. The inserted individual reached the termination criteria as the proposed solution to this model. This fact points out that the convergence problem lays down on the missing proper tuning for the Genetic Programming application. Migration to a more robust Genetic Programming framework, as EASEA (see Appendix C), is predicted for the near future.

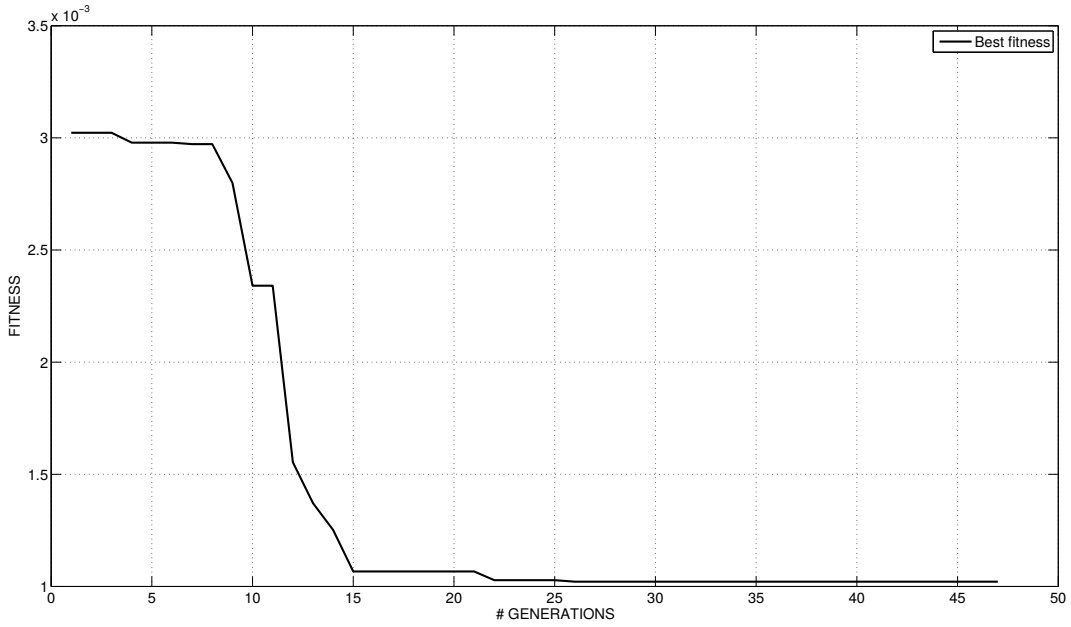

Figure 8.1: Plot for the best individual fitness through generations. Note that, in this very example, the convergence to the solution is already stabilized by the 25th generation.

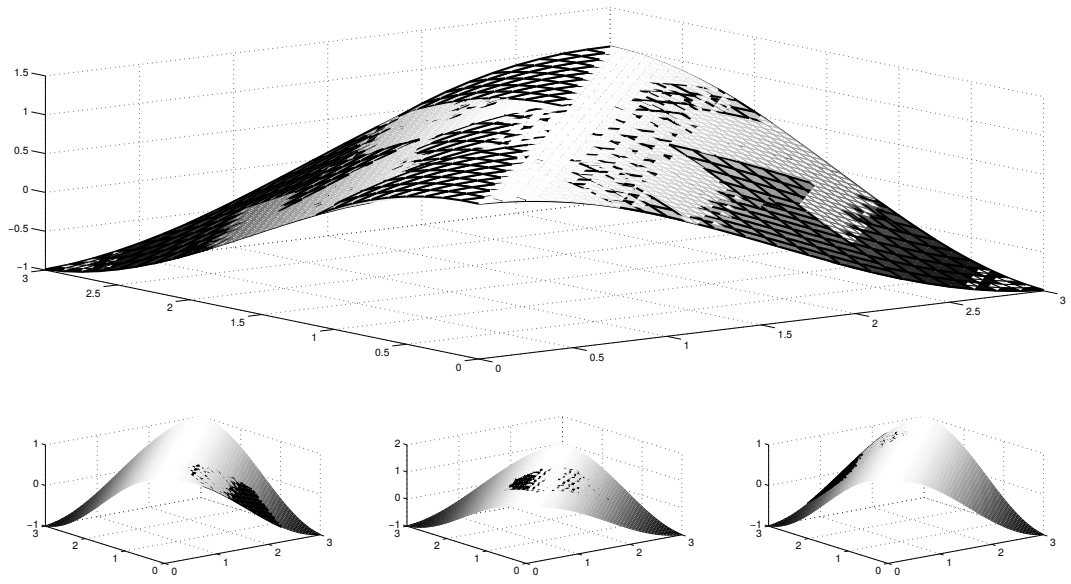

Figure 8.2: Overlap of solution plot and piecewise approximations for the concentration problem. (top) All 9 piecewise domains and approximations. (low left) Approximation over the 2nd considered domain. (low center) Approximation over the 5th considered domain. (low right) Approximation over the 8th considered domain.

Table 8.1: TGE coefficients retrieved for the concentration bi-dimensional example, each set related to one of 9 groups of points.

Interval x		Interval t		$\tilde{u}_0$	$\tilde{u}_1$	$\tilde{u}_2$	$\tilde{u}_3$	$\tilde{u}_4$	$\tilde{u}_5$	$\tilde{u}_6$	$\tilde{u}_7$	$\tilde{u}_8$	$\tilde{u}_9$
0	1.125	0	1.125	0.905	-0.013	0.013	-0.095	0.284	-0.095	0.002	-0.010	0.010	-0.002
0.75	2.25	0	0.75	0.384	-0.621	0.302	-0.075	0.113	-0.019	0.024	-0.060	0.030	-0.003
1.875	3	0	1.125	-0.267	-0.497	0.497	0.033	-0.098	0.033	0.009	-0.045	0.045	-0.009
0	0.75	0.75	2.25	0.381	0.303	-0.625	-0.019	0.113	-0.075	-0.003	0.032	-0.064	0.026
0.75	2.25	0.75	1.875	0.845	-0.132	0.098	-0.176	0.395	-0.099	0.003	-0.010	0.007	-0.001
2.25	3	0.75	2.25	0.381	-0.303	0.625	-0.019	0.113	-0.075	0.003	-0.032	0.064	-0.026
0	1.125	1.875	3	-0.267	0.497	-0.497	0.033	-0.098	0.033	-0.009	0.045	-0.045	0.009
0.75	2.25	0 2.25	3	0.384	0.621	-0.302	-0.075	0.113	-0.019	-0.024	0.060	-0.030	0.003
1.875	3	1.875	3	0.905	-0.013	0.013	-0.095	0.284	-0.095	0.002	-0.010	0.010	-0.002

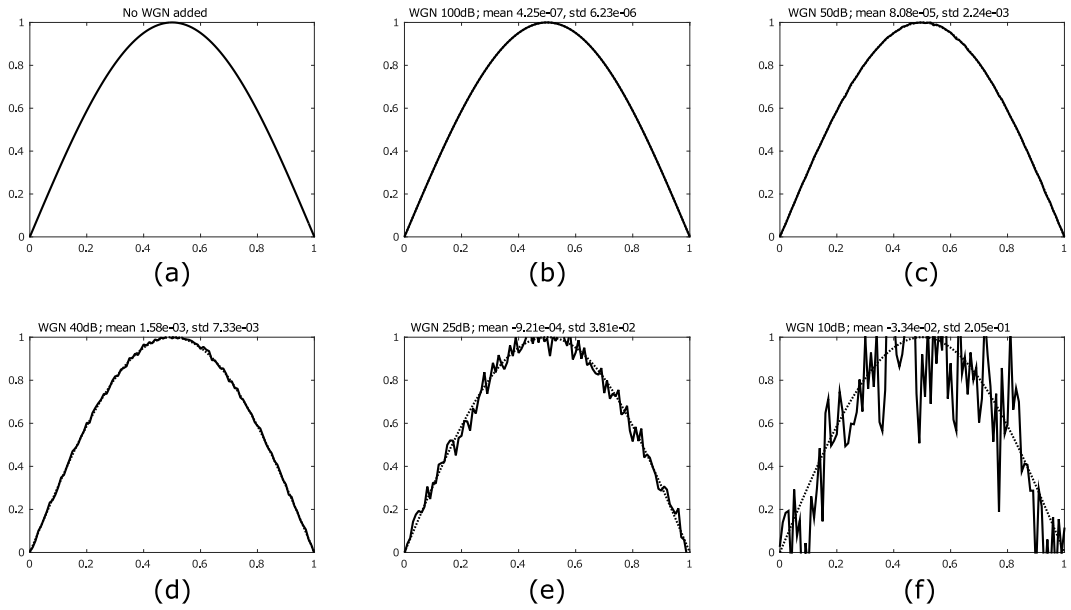

Figure 8.3: White Gaussian noise (WGN) added to signal. (a) Half-period sine signal, no noise added. (b) WGN 100dB added to signal; error distribution with mean $\bar{e} = 4.25\,10^{-7}$ and standard deviation $s = 4.25\,10^{-7}$. (c) WGN 50dB; $\bar{e} = 8.08\,10^{-5}$, $s = 2.24\,10^{-3}$. (d) WGN 40dB; $\bar{e} = 1.58\,10^{-3}$, $s = 7.33\,10^{-3}$. (e) WGN 25dB; $\bar{e} = -9.21\,10^{-4}$, $s = 3.81\,10^{-2}$. (f) WGN 10dB; $\bar{e} = -3.34\,10^{-2}$, $s = 2.05\,10^{-1}$.

8.3 Noise added data

This work adopts white Gaussian noise (WGN), defined by its signal-to-noise ratio (SNR), to be added to all mathematical generated data in order to simulate tolerated measurement errors. The MATLAB®[1] software has implemented the `awgn` command, here used by the author with the argument `'measured'` to generate WGN added data.

Some advantages on this approach relies on the fact that SNR could be understood by its order of magnitude rather than by its parameters or absolute values. Also, a normal distribution is ensured. Figure 8.3 presents some plots where white Gaussian noises with different SNR's were added to a half-period sine signal. The noise both mean and the standard deviation are highlighted for each plot. This approach tends to be very disruptive when dealing with a small number of points as in the present example, but it could give a fair idea on how measurement errors could affect the proposed method.

In Table 8.2 the impact of noise added to data can be unveiled. Column "WGN [dB]" indicates the level of noise in decibels that was added to the simulated data. Column "Fitness for the known model" presents the evaluated fitness for the known model `( D_x0 ( UX ) + D_x1 ( UX ) )` when evaluated using both the noise added data and $2^{22} = 4\,194\,304$ random points to Monte Carlo integration. Column "Possible convergence of GP" indicates if the GP successfully achieved the known model during performed test runs. Note

[1] See `http://www.mathworks.com/products/matlab/` for more information.

Table 8.2: Preliminary results for noise added data.

WGN [dB]	Fitness for the known model	Possible GP convergence	Known model evolved [%]
100	$1.40\,10^{-3}$	yes	60.0%
50	$6.64\,10^{-3}$	yes	46.7%
40	$4.46\,10^{-2}$	yes	40.0%
25	0.958	no	0.0%
10	0.700	no	0.0%

that this column also reflects the fact that even when GP did not converge to the known model, it also did not achieve another model with a better fitness than the one evaluated in the second column. Also, note that GP is supposed to evolve the residual form of the LPDE, so it is feasible that an evolved model could present multiples of the known coefficients and actually be the known model before normalization. For example:

$$2\,x_0\,\frac{d}{dx_0}u(x_0,x_1) + 2\,x_0\,\frac{d}{dx_1}u(x_0,x_1) = 0$$

is equal to the known model if normalized by dividing all coefficients by a factor of $2\,x_0$. With this in mind, column "Known model evolved" presents the percentage in 15 runs that GP actually evolved the known model, even that the result had to be normalized.

When using 100dB, 50dB or 40dB WGN to add to the original simulated data, GP is successful in achieving the known model. The 3rd degree polynomial approximation adopted here is part of the explanation, as well as the scheme to ensure well-conditioned matrices. Both WGN with 25dB and 10dB examples guided the evolution of expressions with non-zero coefficients for terms `D_x0 ( UX )`, `D_x1 ( UX )`, `UX` as well as a non-zero source function. This could be explained as GP trying to model the error as if the error is part of the phenomenon.

Returning to Figure 8.3, it is fair to understand both mean and standard deviations as percentages, once the maximum value for the clean signal is the unity. Regarding preliminary results from Table 8.2, an empirical lower limit of 40dB is there indicated for the data SNR to enable GP to evolve consistent models. Analysing both informations together for the case study, it is feasible here to assume that a tolerance of 0.1% with respect to the maximum value of measurements (order 10^{-3} to the unity) would not invalidate the possibility of evolving a proper model by the implemented GP.

8.4 Repository

All source codes related to this work are available in the following Github repository: `https://github.com/iperetta/system-modelling/`. The following resources are available:

- A MATLAB function to solve LPDEs with a simple text interface to data input;
- Source files for the Genetic Programming application implemented in C++;
- Database files (`.TXT`) used in this thesis, as examples.

To compile the C++ code, download all related files, go to the proper folder and use the command:

```
g++ -std=c++0x -o EXECNAME FILENAME.CPP -llapacke -lblas
```

Note that LAPACK and BLAS must be installed in the host system. Run the executable with the indicated parameters.

8.5 Discussion on contributions

Recovering contributions in Section 1.4, this work could address to those through the course of its pages.

About a novel approach to the Ritz-Galerkin method to approximate solve linear differential equations, Chapters 4 and 5 present the proposed method first applied to LODEs, then generalized to LPDEs. Some issues are also addressed and the method is shown to solve linear differential problems of different types.

About a generic scheme to a computer-automated numerical solver for linear partial differential equations (ordinary ones included) using polynomial approximations for the differential solution, the same Chapters 4 and 5 cover the proposed method and how it can be used as a numerical solver for PDEs of any order.

Regarding some non-linear differentials, the same proposed method here could be extended to handle some types of them, in special the ones that turn out to have polynomial-like unknowns. To explain this idea, note that terms which have a product of derivatives or a power of the solution define a non-linear PDE. Examining terms like these:

$$[\, u(\vec{x}) \,]^2 \quad \text{or} \quad u(\vec{x}) \, \frac{\partial}{\partial x} u(\vec{x}),$$

it could be demonstrated that the matrix formulation of the proposed method reflects such terms as multivariate polynomial unknowns

$$\left(\tilde{u}_0^n, \quad \tilde{u}_0^{n-1} \, \tilde{u}_1, \quad \tilde{u}_0^{n-2} \, \tilde{u}_1^2, \quad \ldots \quad \tilde{u}_1^n \right).$$

There are some widely known strategies and numerical methods to solve systems of non-linear equations. However, the work of [85] brings up linear algebra to solve systems of polynomial equations, and that is the case here. Note that even powers matrix could be useful to control powers of unknowns. More investigation is needed though.

Finally, a dynamic fitness evaluation scheme to be plugged into evolutionary algorithms to automatically solve generic linear differential equations and evaluate model candidates, Chapter 6 shows the fitness evaluation scheme proposed here. This scheme exploits some characteristics of the proposed method. Chapter 7 then presents a preliminary application based on GP which uses the proposed scheme. Promising results show indeed that the proposed method supports GP to perform CASM.

Throughout this process, all necessary mathematical developments were done in order to enable the proposed method to benefit from parallelism.

Finally, a list of published works could be found in Appendix A.

8.6 Indications for future works

Those are subjects related to this research still to be explored:

a.) Exploit parallel paradigms in order to benefit from high performance computing (see Appendix B).

b.) Integrate the proposed method to a more robust massively GP platform, as EASEA-CLOUD (see Appendix C), to benefit from high performance computing parallelism.

c.) Investigate the impact of exchanging Monte Carlo integration by a similar one like quasirandom Monte Carlo.

d.) Include in this work and future ones the support for vector fields.

e.) Include support for complex numbers and understand their impact in some systems.

f.) Regarding the fact that some types of systems requires more than one PDE (a set of) as models, implement support for those.

g.) Understand how to use Linear Algebra to solve polynomial system of equations [85] in order to expand the present work to achieve solutions for certain class of non-linear differential equations, *e.g.*, Navier-Stokes.

h.) Investigate other orthogonal functions to work as complete basis for the proposed method. For example, Fourier-like series as $\sin(x)$ and $\cos(x)$ form a complete orthogonal system over $[-\pi, \pi]$ and could be defined as:

$$f(x) = \frac{a_0}{2} + \sum_{n=1}^{\infty} a_n \cos(nx) + \sum_{n=1}^{\infty} b_n \sin(nx)$$

$$\text{where} \begin{cases} a_0 = \frac{1}{\pi} \int_{-\pi}^{\pi} f(x)\, dx \\ a_n = \frac{1}{\pi} \int_{-\pi}^{\pi} f(x)\, cos(nx)\, dx \\ b_n = \frac{1}{\pi} \int_{-\pi}^{\pi} f(x)\, sin(nx)\, dx \end{cases}$$

Besides the fact that trigonometric functions are numerically expansive, maybe some properties could be found in order to enhance the quality of approximations.

Appendix A

Publications

Here is a list of published works related to this thesis:

- Peretta, I. S.; Yamanaka, K.; Bourgine, P. and Collet, P. Proposal and Preliminary Investigation of a Fitness Function for Partial Differential Models. In : *Genetic Programming, Lecture Notes in Computer Science*, Springer International Publishing, 2015, 9025, 179-191. DOI : `10.1007/978-3-319-16501-1_15`

- Peretta, I. S.; Yamanaka, K. and Collet, P. From Measure Data to Evaluation of Models: System Modeling through Custom Galerkin-Jacobi, *IEEE Latin America Transactions*, 2015, 13, 1556-1561. DOI : `10.1109/TLA.2015.7112015`

And a list of other works published during doctoral studies:

- Pais, M. S.; Peretta, I. S.; Yamanaka, K. and Pinto, E. R. Factorial design analysis applied to the performance of parallel evolutionary algorithms, *Journal of the Brazilian Computer Society*, 2014, 20:6

- Mendes Lima, G.; Lamounier, E. A.; Barcelos, S.; Cardoso, A.; Peretta, I. S.; Rigon, E. and Sadaiti Muramoto, W. A TEO-Based Algorithm to Detect Events Over OTDR Measurements in FTTH PON Networks, *IEEE Latin America Transactions*, 2013, 11, 886-891

- Tavares, J. A.; Peretta, I. S.; de Lima, G. F. M.; Yamanaka, K. and Pais, M. S. SLPTEO e SCORC: abordagens para segmentação de linhas, palavras e caracteres em textos impressos; *Avanços em Visão Computacional*, Omnipax, 2012, 239-264

Appendix B

Massively Parallel Programming

B.1 GPGPU

Historically[1], the first GPUs were designed to act as graphics accelerators, supporting only specific fixed-function pipelines. In the late 1990s, the hardware became increasingly programmable, culminating in 1999 with NVIDIA®'s first GPU. Less than a year after, the GPGPU movement had dawned: researchers were exploring the technology in their works, specially due to its floating point performance. As stated by [86], "modern GPUs now include fully programmable processing units that support vectorized floating-point operations on values stored at full IEEE single precision". Today, there are also GPUs that have included full IEEE *double* precision.

Modern NVIDIA® GPGPUs typically contain up to several hundred cores. Newer versions released up to 2015 can contain more than five thousands cores (*e.g.*, *Tesla K80* has 4992 CUDA cores, while *GTX Titan Z* has 5760 cores). Due to the fact that those cores have a size and transistor thinness comparable to conventional multi-cores processors, the increasing number of cores is possible through the simplification of the whole processor, e.g. by the differences on the structuring of cores. Owens *et al.* [86], in their survey, explain the reason for graphics hardware performance to increase faster (and getting faster quickly) than that of CPUs as semiconductor capability increases at the same rate for both platforms:

> The disparity can be attributed to fundamental architectural differences: CPUs are optimized for high performance on sequential code, with many transistors dedicated to extracting instruction-level parallelism with techniques such as branch prediction and out-of-order execution. On the other hand, the highly data-parallel nature of graphics computations enables GPUs to use additional transistors more directly for computation, achieving higher arithmetic intensity with the same transistor count.

Simply describing, NVIDIA® GPGPUs use the parallelism principle of vector computation, where units are grouped around a single instruction unit

[1] "History of GPU Computing" at `http://www.nvidia.com/object/cuda_home_new.html`

```
[style=example]
void saxpy_serial(int n,
  float a,float *x,float *y)
{
    for (int i = 0; i < n; ++i)

        y[i] += a*x[i];
}

// Perform SAXPY on 1M elements
saxpy_serial(4096*256, 2.0, x, y);
```

```
[style=example]
__global__ void saxpy_serial(int n,
  float a,float *x,float *y)
{
    int i = blockIdx.x*blockDim.x +
      threadIdx.x;
    if (i < n) y[i] += a*x[i];
}

// Perform SAXPY on 1M elements
saxpy_serial<<<4096,256>>>(n, 2.0, x, y);
```

Figure B.1: Example on CUDA C. (left) Standard C Code; (right) Parallel C Code; adapted from website `http://www.nvidia.com`

which decodes instructions that the group of cores will execute. This implies that at most one instruction is decoded at each cycle and is applied to all cores together, commonly on different data, configuring a SIMD[2] architecture in Flynn's taxonomy. This is the parallel paradigm when programming GPGPUs.

Inventive uses by developers of an increasing GPGPU's flexibility scenario have enabled diverse applications (others than the original ones for which GPUs were designed) to experience parallelism. Massively parallel programming aims to deal with huge amounts of calculations within a feasible execution time. GPGPUs have a great performance per cost ratio and a high theoretical peak computing power, turning them into a very likely choice when going massively parallel.

B.2 CUDA platform

CUDA is the development framework created by NVIDIA® in 2006 to integrate their GPGPU hardware to a programming model that extends C/C++ with data-parallel constructs. According to the official website[3] "The CUDA parallel computing platform provides a few simple C and C++ extensions that enable expressing fine-grained and coarse-grained data and task parallelism." Figure B.1 shows an example on programming using CUDA extesions to C.

[2] Acronym for Single Instruction, Multiple Data.

[3] `http://www.nvidia.com/cuda`

Appendix C

EASEA Platform

When dealing with EAs, one issue permeates all discussions: how fast is this technique to be applied to a certain type of problem. The fact that EAs are essentially parallelizable needs to mean something. Obviously, this is not a simple answer. However, technology has provided some gadgets which allow the improvement of the speed-up for such heuristics. CUDA framework and GPGPU card by NVIDIA® are the technologies to be investigated in the near future.

EAsy Specification of Evolutionary Algorithms (EASEA) is a platform designed to help with the creation of evolutionary algorithms. It has been developed since 1998 [87]. EASEA platform is currently supported by the SONIC (Stochastic Optimisation and Nature Inspired Computing) group of the BFO team at *Université de Strasbourg*. The GP framework of EASEA platform is our current choice to manage EA.

According to its official website[1],

> EASEA is an Artificial Evolution platform that allows scientists with only basic skills in computer science to exploit the massive parallelism of many-core architectures in order to optimize virtually any real-world problems (continous, discrete, combinatorial, mixed and more (with Genetic Programming)), typically allowing for speed-ups up to ×1,000 on a $5,000 machine, depending on the complexity of the evaluation function of the inverse problem to be solved.

More details on the scope of the EASEA project can be found in the works of [88, 89].

The modern version of EASEA was developed for the PhD thesis of Maitre [75], using CUDA C/C++ programming language. It aims to enable the efficient use of massively parallel machines equipped with one or several GPGPU cards for the execution of parallel EAs. Also, the EASEA-CLOUD version[89] is something to look for.

This work could strongly benefit from EASEA abilities by integrating a PDE solver that could be used to model complex systems. The GP framework

[1] `http://easea.unistra.fr/easea/index.php/EASEA_platform`

```
[style=CPP]
\GenomeClass::evaluator :
 float Score= 0.0;
 Score= Weierstrass(Genome.x, SIZE);
 return Score;
\end
```

Figure C.1: Example of EASEA syntax for specification of a Genome Evaluator

will be handled by EASEA, but a proper individual representation is also object of studying. Another important subject, the fitness evaluator module will need to be supported by the solver. The programming language adopted is a dedicated one, mainly based on C++ templates. The solver must be a C++ function, though. To work in "modules" allows EASEA to handle them together. One can find more details about this in [75], or in the official web site[1]. Figure C.1 shows an example of a Genome Evaluator specified in EASEA syntax.

Bibliography

[1] A. Backlund, "The definition of system," *Kybernetes*, vol. 29, no. 4, pp. 444–451, 2000.

[2] E. Griffiths, "What is a model?" *Sheffield University*, 2010. [Online]. Available: https://sites.google.com/a/ncsu.edu/emily-griffiths/whatisamodel.pdf

[3] R. P. Feynman, *The character of physical law*, 1st ed. MIT Pr, 1985, 12th printing.

[4] M. Schmidt and H. Lipson, "Distilling Free-Form Natural Laws from Experimental Data," *Science*, vol. 324, no. 5923, pp. 81–85, 2009.

[5] R. P. Feynman, "The Principle of Least Action in Quantum Mechanics," Thesis (Ph.D.), Department of Physics, Princeton University, Princeton, NJ, USA, 1942.

[6] C. Lanczos, *The Variational Principles of Mechanics*, ser. Dover Books On Physics. Dover Publications, 1970.

[7] P. Šolín, *Partial Differential Equations and the Finite Element Method*. Wiley-Interscience, 2006.

[8] C. Boyer, *The History of the Calculus and Its Conceptual Development: (The Concepts of the Calculus)*, ser. Dover Books on Advanced Mathematics. Dover Publications, 1959.

[9] N. Boccara, *Modeling Complex Systems*, ser. Graduate Texts in Contemporary Physics. Springer Verlag, 2004.

[10] P. G. Drazin, *Nonlinear Systems*. Cambridge Texts in Applied Mathematics, 1997, reprint.

[11] D. Kaplan and L. Glass, *Understanding Nonlinear Dynamics*. Springer, 1995.

[12] J. R. Koza, *Genetic programming: on the programming of computers by means of natural selection*. Cambridge, MA, USA: MIT Press, 1992.

[13] R. Poli, W. B. Langdon, and N. F. McPhee, *A Field Guide to Genetic Programming*. Lulu Enterprises, UK Ltd, 2008.

[14] David Waltz and Bruce G. Buchanan, "Automating Science," *Science*, vol. 324, no. 5923, pp. 43–44, 3 Apr. 2009, perspective.

[15] T. J. R. Hughes, *The finite element method: linear static and dynamic finite element analysis*. Prentice Hall, 1987.

[16] K.-J. Bathe, *Finite Element Procedures*. Prentice Hall, 1996.

[17] O. C. Zienkiewicz and R. L. Taylor, *The Finite Element Method: The Basis*, 5th ed. Butterworth-Heinemann, 2000, vol. 1.

[18] D. V. Griffiths and I. M. Smith, *Numerical MMethod for Engineers: A Programming Approach*. CRC Press, 1991.

[19] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical Recipes 3rd Edition: The Art of Scientific Computing*, 3rd ed. New York, NY, USA: Cambridge University Press, 2007.

[20] R. Cools, "Advances in multidimensional integration," *Journal of Computational and Applied Mathematics*, vol. 149, no. 1, pp. 1 – 12, 2002, scientific and Engineering Computations for the 21st Century - Me thodologies and Applications Proceedings of the 15th Toyota Conference. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0377042702005174

[21] B. S. Skrainka and K. L. Judd. (2011) High Performance Quadrature Rules: How Numerical Integration Affects a Popular Model of Product Differentiation. Cemmap working paper CWP03/11. [Online]. Available: http://www.cemmap.ac.uk/wps/cwp0311.pdf

[22] B. G. Galerkin, "Series occuring in various questions concerning the elastic equilibrium of rods and plates," *Engineers Bulletin (Vestnik Inzhenerov)*, vol. 19, pp. 897–908, 1915, (in Russian).

[23] P. R. C. Kent, "Techniques and Applications of Quantum Monte Carlo," Ph.D. dissertation, University of Cambridge, 1999.

[24] A. E. Eiben and J. E. Smith, *Introduction to Evolutionary Computing*. Springer, 2007, corrected 2nd printing.

[25] G. S. Hornby, J. D. Lohn, and D. S. Linden, "Computer-automated Evolution of an X-band Antenna for Nasa's Space Technology 5 Mission," *Evol. Comput.*, vol. 19, no. 1, pp. 1–23, Mar. 2011.

[26] G. Gray, D. J. Murray Smith, Y. Li, and K. C. Sharman, "Nonlinear Model Structure Identification Using Genetic Programming," *Control Engineering Practice*, vol. 6, no. 11, pp. 1341–1352, 1998.

[27] S. V. Soto, Ed., *Genetic Programming: New Approaches and Successful Applications*. InTech, 2012.

[28] K. Stanislawska, K. Krawiec, and Z. W. Kundzewicz, "Modeling Global Temperature Changes with Genetic Programming," *Comput. Math. Appl.*, vol. 64, no. 12, pp. 3717–3728, Dec. 2012.

[29] J. Ong, "Stop Comparing Programming Languages With Benchmarks," /dev/something, online blog, August 29 2013. [Online]. Available: http://www.jeremyong.com/blog/2013/08/29/stop-comparing-programming-languages-with-benchmarks/

[30] N. Diakopoulos and S. Cass, "Interactive: The Top Programming Languages 2015," IEEE Spectrum, online, July 20 2015. [Online]. Available: http://spectrum.ieee.org/static/interactive-the-top-programming-languages-2015

[31] "A list of open source C++ libraries," Online, 2015, retrieved on Aug 16th, 2015. [Online]. Available: http://en.cppreference.com/w/cpp/links/libs

[32] J. P. Crutchfield and B. S. McNamara, "Equations of Motion from a Data Series," *Complex Systems*, vol. 1, no. 3, pp. 417–452, 1987. [Online]. Available: http://www.complex-systems.com/pdf/01-3-3.pdf

[33] J. P. Crutchfield and K. Young, "Inferring Statistical Complexity," *Physical Review Letters*, vol. 63, no. 2, pp. 105–108, 1989.

[34] N. L. Cramer, "A Representation for the Adaptive Generation of Simple Sequential Programs," in *Proceedings of the 1st International Conference on Genetic Algorithms*. Hillsdale, NJ, USA: L. Erlbaum Associates Inc., 1985, pp. 183–187.

[35] J. F. Hicklin, "Application of the genetic algorithm to automatic program generation," Master's thesis, University of Idaho, 1986.

[36] C. Fujiko and J. Dickinson, "Using the genetic algorithm to generate LISP source code to solve the prisoner's dilemma," in *Proceedings of the Second International Conference on Genetic Algorithms on Genetic algorithms and their application*. Hillsdale, NJ, USA: L. Erlbaum Associates Inc., 1987, pp. 236–240.

[37] J. R. Koza, "Genetic programming: a paradigm for genetically breeding populations of computer programs to solve problems," Stanford University, TR 90-1314, 1990.

[38] D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*, 1st ed. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1989.

[39] S. Forrest, "Genetic algorithms: principles of natural selection applied to computation," *Science*, vol. 261, no. 5123, pp. 872–878, 1993.

[40] K. A. De Jong, *Evolutionary Computation: A Unified Approach*. Mit Press, 2006.

[41] B. Steele, "Move over, Newton: Scientifically ignorant computer derives natural laws from raw data," Cornell Chronicles, online, Apr 2 2009, cornell University. [Online]. Available: http://www.news.cornell.edu/stories/2009/04/computer-derives-natural-laws-observation

[42] E. Cartlidge, "Algorithm discovers physical laws," IOP Physics World, online, Apr 2 2009. [Online]. Available: http://physicsworld.com/cws/article/news/2009/apr/02/algorithm-discovers-physical-laws

[43] I. Sample, "'Eureka machine' puts scientists in the shade by working out laws of nature," The Guardian, online, Apr 3 2009. [Online]. Available: http://www.theguardian.com/science/2009/apr/02/eureka-laws-nature-artificial-intelligence-ai

[44] C. Hillar and F. T. Sommer, "Comment on the article "Distilling free-form natural laws from experimental data"," *ArXiv e-prints*, Oct 26 2012. [Online]. Available: http://arxiv.org/abs/1210.7273

[45] H. Cao, L. Kang, Y. Chen, and J. Yu, "Evolutionary Modeling of Systems of Ordinary Differential Equations with Genetic Programming," *Genetic Programming and Evolvable Machines*, vol. 1, no. 4, pp. 309–337, 2000. [Online]. Available: http://dx.doi.org/10.1023/A%3A1010013106294

[46] T. Kumon, M. Iwasaki, T. Suzuki, T. Hashiyama, N. Matsui, and S. Okuma, "Nonlinear system identification using genetic algorithm," in *26th Annual Conference of the IEEE Industrial Electronics Society (IECON)*, vol. 4, 2000, pp. 2485–2491.

[47] J. Chen and R. L. Ely, "Comparison of Artificial Neural Network, Genetic Programming, and Mechanistic Modeling of Complex Biological Processes," *Environmental engineering science*, vol. 18, no. 5, pp. 267–278, 2001.

[48] C. Banks, "Searching for Lyapunov Functions using Genetic Programming," 2002. [Online]. Available: http://www.aerojockey.com/files/lyapunovgp.pdf

[49] H. Leung and V. Varadan, "System modeling and design using genetic programming," in *First IEEE International Conference on Cognitive Informatics, Proceedings*, 2002, pp. 88–97.

[50] M. Hinchliffe, "Dynamic systems modelling using genetic programming," *Computers & Chemical Engineering*, 2003.

[51] S. Xiong and W. Wang, "A new hybrid structure genetic programming in symbolic regression," in *The 2003 Congress on Evolutionary Computation (CEC'03)*, vol. 3, 2003, pp. 1500–1506.

[52] G. Beligiannis, L. Skarlas, S. Likothanassis, and K. Perdikouri, "Nonlinear model structure identification of complex biomedical data using a genetic-programming-based technique," *Instrumentation and Measurement, IEEE Transactions on*, vol. 54, no. 6, pp. 2184–2190, 2005.

[53] J. Bongard and H. Lipson, "Automated reverse engineering of nonlinear dynamical systems," *Proceedings of the National Academy of Sciences*, vol. 104, no. 24, pp. 9943–9948, 2007. [Online]. Available: http://www.pnas.org/content/104/24/9943.abstract

[54] H. Iba and E. Sakamoto, "Inference of differential equation models by genetic programming," *Information Sciences*, vol. 178, no. 23, pp. 4453 – 4468, 2008, including Special Section: Genetic and Evolutionary Computing. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0020025508002909

[55] J. S. McGough, A. W. Christianson, and R. C. Hoover, "Symbolic Computation of Lyapunov Functions using Evolutionary Algorithms," in *IASTED ISMM International Proceedings*, 2010, pp. 508–515.

[56] Amir Hossein Gandomi and Amir Hossein Alavi, "Multi-stage genetic programming: A new strategy to nonlinear system modeling," *Information Sciences*, vol. 181, no. 23, pp. 5227 – 5239, 2011. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0020025511003586

[57] S. Gaucel, M. Keijzer, E. Lutton, and A. Tonda, *Genetic Programming: Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2014, ch. Learning Dynamical Systems Using Standard Symbolic Regression, pp. 25–36.

[58] P. Jain, K. Ahmad, and O. Ahuja, *Functional Analysis*. New Age International, 1995.

[59] M. J. Gander and G. Wanner, "From Euler, Ritz, and Galerkin to Modern Computing," *SIAM Review*, vol. 54, no. 4, pp. 627–666, 2012.

[60] P. W. Livermore, "Galerkin Orthogonal Polynomials," *Journal of Computational Physics*, vol. 229, pp. 2046–2060, 2010.

[61] J. Shen, T. Tang, and L.-L. Wang, *Spectral Methods: Algorithms, Analysis and Applications*. Springer-Verlag, 2011.

[62] R. Venkatraman. (2011, December) Lecture Notes: The Galerkin Method. Portable Document Format. Simon Fraser University, Burnaby BC. [Online]. Available: http://arxiv.org/pdf/1112.1176.pdf

[63] J. S. Hadamard, "Sur les problèmes aux Dérivées partielles et leur signification physique," *Princeton University Bulletin*, vol. 13, pp. 49–52, 1902.

[64] *IEEE Standard for Floating-Point Arithmetic*, IEEE Std Std., Aug 2008.

[65] R. Penrose, "A generalized inverse for matrices," in *Proc. Cambridge Philos. Soc.*, vol. 51, 1955, pp. 406–413.

[66] S. Johnson, "Gram-Schimdt for functions: Legendre polynomials," Portable Document Format, 2009, supplement to textbook, MIT Spring course 18.06.

[67] P. W. Livermore and G. R. Ierley, "Quasi-L^p norm orthogonal Galerkin expansions in sums of Jacobi polynomials," *Numerical Algorithms*, vol. 54, no. 4, pp. 533–569, 2010.

[68] G. Szegö, *Orthogonal Polynomials*, ser. American Mathematical Society colloquium publications. American Mathematical Society, 1959, no. v. 23, revised edition.

[69] Y. L. Luke, *The Special Functions and their Approximations*. Academic Press, 1969.

[70] J. R. Koza, *Advances in Evolutionary Computing: Theory and Applications*. Springer, 2003, ch. Human-Competitive Applications of Genetic Programming, pp. 663–682.

[71] J. Holland, *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. University of Michigan Press, 1975.

[72] A. M. Turing, "Computing Machinery and Intelligence," *Mind*, vol. 59, no. 236, pp. 433–460, 1950.

[73] P. Collet. (2009) Artificial Life/Artificial Evolution. Portable Document Format. JFFoS: Japanese-Fench Frontiers of Science. Abstract, Session Chair (Symposium).

[74] D. Fogel, *Evolutionary computation: the fossil record*. IEEE Press, 1998.

[75] O. Maitre, "GPGPU for Evolutionary Algorithms," Ph.D. dissertation, Université de Strasbourg, 2011.

[76] D. Montgomery and G. Runger, *Applied Statistics and Probability for Engineers*. John Wiley & Sons, 2010.

[77] D. H. Johnson, "Signal-to-noise ratio," vol. 1, no. 12, p. 2088, 2006, revision #91770.

[78] O. Zienkiewicz and K. Morgan, *Finite Elements and Approximation*, ser. Dover books on engineering. Dover Publications, 2006. [Online]. Available: https://books.google.com.br/books?id=wGE7BzNk3W4C

[79] Maxima. (2015) Maxima, a Computer Algebra System. Version 5.36.1. http://maxima.sourceforge.net/. [Online]. Available: http://maxima.sourceforge.net/

[80] G. Grosso and G. P. Parravicini, *Solid State Physics*. Elsevier Science, 2000.

[81] E. Kreyszig, *Advanced Engineering Mathematics*, 10th ed. Wiley, 2011.

[82] I. Stojmenovic and A. Zoghbi, "Fast algorithms for genegrating integer partitions," *International Journal of Computer Mathematics*, vol. 70, no. 2, pp. 319–332, 1998.

[83] S. Vigna, "An experimental exploration of Marsaglia's xorshift generators, scrambled," *CoRR*, vol. abs/1402.6246, 2014. [Online]. Available: http://arxiv.org/abs/1402.6246

[84] S. J. Farlow, *Partial Differential Equations for Scientists and Engineers*. Dover Publications, 1993.

[85] P. Dreesen, "Back to the Roots - Polynomial System Solving Using Linear Algebra," Ph.D. dissertation, Faculty of Engineering, KU Leuven, Sep. 2013.

[86] J. D. Owens, D. Luebke, N. Govindaraju, M. Harris, J. Krüger, A. Lefohn, and T. J. Purcell, "A Survey of General-Purpose Computation on Graphics Hardware," *Computer Graphics Forum*, vol. 26, no. 1, pp. 80–113, 2007. [Online]. Available: http://www.blackwell-synergy.com/doi/pdf/10.1111/j.1467-8659.2007.01012.x

[87] P. Collet, E. Lutton, M. Schoenauer, and J. Louchet, "Take It EASEA," in *Parallel Problem Solving from Nature PPSN VI*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2000, vol. 1917, pp. 891–901.

[88] P. Collet, M. Schoenauer, E. Lutton, and J. Louchet, "EASEA: un langage de spécification pour les algorithmes évolutionnaires," INRIA, RR 4218, June 2001.

[89] P. Collet, F. Krüger, and O. Maitre, *Massively Parallel Evolutionary Computation on GPGPUs*, ser. Natural Computing Series. Springer, 2013, ch. Automatic Parallelization of EC on GPGPUs and Clusters of GPGPU Machines with EASEA and EASEA-CLOUD, pp. 35–61.