THE MULTIVARIATE THEORY OF FUNCTIONAL CONNECTIONS: AN N-DIMENSIONAL CONSTRAINT EMBEDDING TECHNIQUE APPLIED TO PARTIAL DIFFERENTIAL EQUATIONS

A Dissertation Proposal

by

CARL DAKOTA LEAKE

Submitted to the **Graduate and Professional School** of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Chair of Committee,	Daniele Mortari
Committee Members,	John Hurtado
	Junuthula Reddy
	Moble Benedict
Head of Department,	Srinivas Rao Vadali

August 2021

Major Subject: Aerospace Engineering

ABSTRACT

The Theory of Functional Connections (TFC) is a functional interpolation framework founded upon the so-called constrained expression: a functional that expresses the family of all possible functions that satisfy some user-specified, linear constraints. These constrained expressions can be utilized to transform constrained problems into unconstrained ones. The benefits of doing so include faster solution times, more accurate solutions, and more robust convergence. This dissertation contains a comprehensive, self-contained presentation of the TFC theory beginning with simple univariate point constraints and ending with general linear constraints in n-dimensions; relevant mathematical theorems and clarifying examples are included throughout the presentation to expand and solidify the reader's understanding. Furthermore, this dissertation describes how TFC can be applied to estimate differential equations' solutions, its primary application to date. In addition, comparisons with other state-of-the-art algorithms that estimate differential equations' solutions are included to showcase the advantages and disadvantages of the TFC approach. Lastly, the aforementioned concepts are leveraged to estimate solutions of differential equations from the field of flexible body dynamics.

DEDICATION

This dissertation is dedicated to the ocean of human knowledge:

what follows is a molecule of H_2O.

ACKNOWLEDGMENTS

It is a nearly impossible task to list all of those who have positively nudged one's trajectory through life. Even the smallest push can have a massive effect, as I conjecture life is a chaotic system. Moreover, I am fortunate enough to have been incredibly blessed with supportive and encouraging family, friends, teachers, and mentors throughout my education. I would love to list them all, but I fear I will miss one or more, and the list would be much too long to expect any reader to trudge through. Hence, I have elected to name just six people who I felt played the largest part. To everyone else who has encouraged me, shaped me, helped me, pushed me, loved me, and challenged me throughout this exciting journey, please know that even if you are not called out by name I am still eternally grateful and I love and appreciate you: this would not be possible without you.
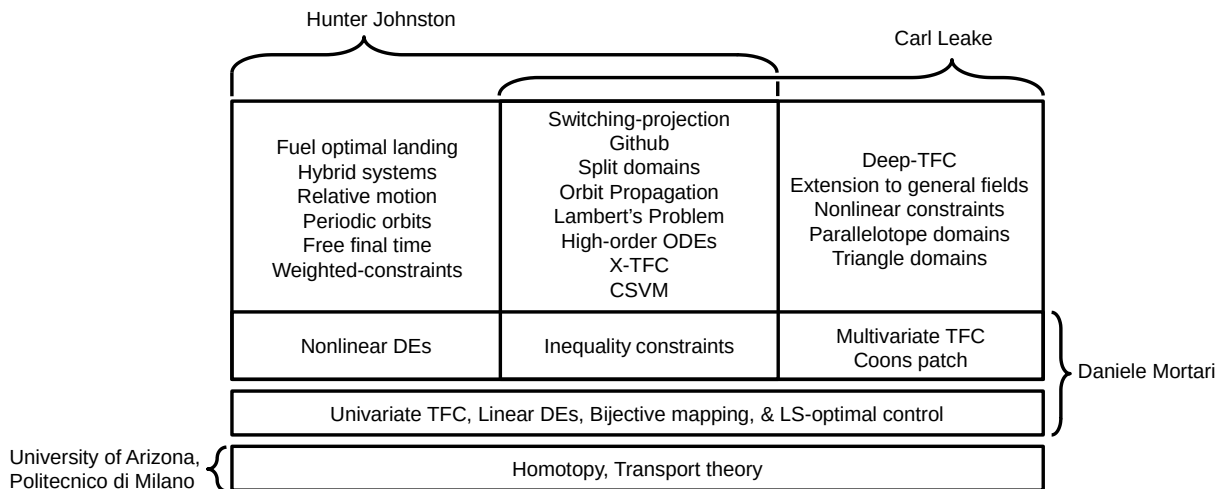
I believe a good parent is hard to find, and an excellent parent is exceedingly rare. Yet, I find myself with two excellent parents: my mother, Donna Leake, and father, Robert Leake. Without your guidance, love, and support I would not be where I am today. To Nina Rogerson, the woman I love, thank you for your companionship and support. It is has been an extraordinary six years, and I look forward to many, many more together. To Hunter Johnson and Daniele Mortari, without your friendship and intellectual support it would surely have been a dull and markedly less productive four years. To Aaron Schutte, thank you for your mentorship and support throughout two internships and an NSTRF fellowship.

CONTRIBUTORS AND FUNDING SOURCES

Contributors

This work was supported by a dissertation committee consisting of Professor Daniele Mortari (advisor) and Professors John Hurtado and Moble Benedict of the Department of Aerospace Engineering and Professor Junuthula Reddy of the Department of Mechanical Engineering.

The Theory of Functional Connections was collaboratively developed by Daniele Mortari (advisor), Hunter Johnston (PhD candidate), and Carl Leake (author/PhD candidate). To clarify the major contributions of each, the following figure is included.

Funding Sources

NOMENCLATURE

API	Application Programming Interface
BFGS	Broyden-Fletcher-Goldfarb-Shanno
CSVM	Constrained Support Vector Machine
DE	Differential equation
Deep-TFC	Deep Theory of Functional Connections
ELM	Extreme Learning Machine
FEM	Finite Element Method
i.i.d.	Independently and identically distributed
JIT	Just-in-time (compiler)
JPL	Jet Propulsion Laboratory
LS	Least-squares
LS-SVM	Least-squares support vector machine
NN	Neural network
NSTRF	NASA Space Technology Research Fellowship
ODE	Ordinary differential equation
PDE	Partial differential equation
SVM	Support vector machine
TFC	Theory of Function Connections
XLA	Accelerated Linear Algebra
X-TFC	Extreme Theory of Functional Connections
c_k	Slope in the linear map for the k-th independent variable that maps the basis function domain to the problem domain
$^{(k)}\mathfrak{C}_i$	Constraint operator for the i-th constraint of the k-th independent variable

$g(\boldsymbol{x})$	Free function $\mathbb{R}^n \mapsto \mathbb{R}$. Note that a superscript may be used to denote the free function for a specific dependent variable, e.g., $g^u(\boldsymbol{x})$ is the free function for the dependent variable u.
$\mathcal{J}$	Jacobian matrix
$\mathbb{L}$	Loss function $\mathbb{R}^m \mapsto \mathbb{R}^n$
L^1	Space of Lebesgue-integrable functions, i.e., $\int_\Omega |f| \, \mathrm{d}\mu < \infty$
L^2	Space of square-Lebesgue-integrable functions, i.e., $\int_\Omega |f|^2 \, \mathrm{d}\mu < \infty$
$\mathbb{R}$	Set of real numbers
$\mathbb{S}_{ij}$	Support matrix
$\boldsymbol{x}$	A vector of the independent variables, i.e., $\boldsymbol{x} = \{x_1, x_2, \cdots, x_n\}$, where n is the number of independent variables.
x_k	The k-th independent variable
$\mathbb{Z}$	Set of integers
$\mathbb{Z}^+$	Set of positive integers
$\mathbb{Z}/n\mathbb{Z}$	Set of integers modulo n
z_k	Basis function domain variable for the k-th independent variable
δ_{ij}	Kronecker delta
$^{(k)}\kappa_i(\boldsymbol{x})$	Portion of the i-th constraint on the k-th independent variable that does not contain the dependent variable. Note the pre-superscript is dropped in the univariate formulation as there is only one independent variable.
$\mu(z)$	Measure function $\mathbb{R} \mapsto \mathbb{R}$
$^{(k)}\rho_i(\boldsymbol{x}, g(\boldsymbol{x}))$	Projection functional for the i-th constraint of the k-th independent variable. Note that in the univariate formulation, the pre-superscript is dropped as there is only one independent variable.
$^{(k)}\phi_i(x_k)$	Switching function for the i-th constraint on the k-th independent variable. Note that in the univariate formulation, the pre-superscript is dropped as there is only one independent variable.
Ω	Domain
$\mathbb{1}(x, x_1)$	Heaviside function, $\mathbb{R} \mapsto \mathbb{R}$
$\mathbb{1}_0(x)$	Heaviside function where $x_1 = 0$
$\lceil x \rceil$	Rounds x to the next largest integer

$\lfloor x \rceil$ Rounds x to the nearest integer

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF EXAMPLES

1. INTRODUCTION AND MOTIVATION

Interpolation is a concept that many use every day—for example, to extract an estimated value between experimental data points or between data points of tabulated values for a computationally expensive function—but pay little attention to. This is not surprising given that most imagine a discrete set of points when considering interpolation, a fairly simplistic problem, and the age of the algorithms used to solve them dates as far back as Waring polynomial interpolation, which was published in 1779 [4]. Yet, if the interpolation complexity is increased, it quickly becomes difficult or impossible to point to an algorithm that can easily perform the interpolation.

Consider the advantages of satisfying properties at points other than their value, for example, their derivatives or a linear combination of derivatives and values between points. Rather than thinking of properties at points, it may be easier to imagine them as constraints, for example, $u_x(x_0) + \pi u(x_1) = v(x_2)$, where x is an independent variable, x_0, x_1, and x_2 are some specific values in the domain, u and v are dependent variables, and u_x denotes a derivative of u with respect to x. Dream bigger. What if one could do this in n-dimensional domains or write *all* possible functions that satisfy the constraints rather than just one function that satisfies the constraints? At this point, one is describing something much more complicated than simple point-wise interpolation; rather, they are describing a sort of function-based interpolation.

A rich framework for function-based interpolation could transform problems with linear constraints into unconstrained problems. In terms of optimization-type problems, this would mean one could use simpler optimizers, as an optimizer that handles constraints would no longer be needed, and/or the function to be minimized would not need to be augmented to include the constraints. Indeed, such a framework would enhance one's ability to solve such problems and is the driving motivation behind the Theory of Functional Connections (TFC): a general framework for function-based interpolation.

The concept of function-based interpolation itself is not new, and numerous methods exist [5, 6, 7, 8, 9, 10]; however, these previous techniques only work for a class or sub-class of functions and cannot be used to describe all functions that satisfy a set of constraints. Therefore, their scope of applications is limited; TFC does not have this restriction.

1.1 Original Idea

The idea that sparked the Theory of Functional Connections (TFC) was conceived by Daniele Mortari while teaching the Waring, better known as Lagrange, polynomial interpolation method [4], which is used to generate an interpolating function that passes through a set of points. For example, the Lagrange polynomial, $y(x)$, for a set of n points, $(x_1, y_1), \ldots, (x_k, y_k), \ldots, (x_n, y_n)$, can be written as,

$$y(x) = \sum_{k=1}^{n} y_k \prod_{i \neq k} \frac{x - x_i}{x_k - x_i}.$$

The Lagrange polynomial represents one function that passes through the n points. In other words, the Lagrange polynomial is an interpolating function for these points: a function that satisfies the constraints $y(x_k) = y_k$.

Mortari's original insight was that by replacing x with an arbitrary function $g(x)$, x_k with $g(x_k)$, and so on, one could write the the family of *all* possible functions that passes through the set of points. That is,

$$y(x) = \sum_{k=1}^{n} y_k \prod_{i \neq k} \frac{x - x_i}{x_k - x_i} \qquad \rightarrow \qquad y(x, g(x)) = \sum_{k=1}^{n} y_k \prod_{i \neq k} \frac{g(x) - g(x_i)}{g(x_k) - g(x_i)}.$$

From this seed of an idea sprouted Mortari's 2016 seminal article [11] that demonstrated how to embed univariate value and derivative constraints, and constraints consisting of linear combinations of values and derivatives at points, into the TFC framework. Since then, the TFC framework has grown to encompass a larger variety of increasingly exotic constraints, including integral [12], component [13], and inequality [14] constraints. Moreover,

the original univariate framework has been extended to multiple variables [1, 15] and some non-rectangular domains [16, 17].

1.2 Overview of the Remaining Chapters

In lieu of a large literature review concentrated at the beginning of the dissertation, most chapters contain their own smaller literature review whose contents pertain specifically to that chapter. The rest of this dissertation is structured as follows.

Chapter 2. Theory of Functional Connections

This chapter describes the theory behind the TFC functional interpolation framework. It is split into two major sections: the univariate theory and the multivariate theory. The univariate theory is introduced first and describes how to construct constrained expressions for value, derivative, integral, and component constraints, and linear combinations thereof. In addition, it includes mathematical theorems that pertain to the univariate constrained expression. The multivariate theory section generalizes this to n-dimensions. Examples are included throughout to help solidify the reader's understanding.

Chapter 3. Applications in Differential Equations

This chapter utilizes the TFC framework introduced in the previous chapter to solve differential equations by embedding the differential equation constraints into the constrained expression and using the free function to minimize the differential equation's residual at a discrete set of points. In addition, the chapter discusses useful free function choices and optimization methods and includes a summary of the TFC numerical implementation. To strengthen the reader's understanding, a simple PDE is numerically estimated using each of the common free function choices; additional examples are provided that highlight the strengths and weaknesses of each free function choice.

Chapter 4. Applications in Flexible Body Problems

Building on the foundations of the previous chapters, this chapter utilizes the TFC framework introduced in Chapter 2 and its application to differential equations introduced in Chapter 3 to apply the method to differential equations that appear in flexible body problems. In other words, this chapter contains flexible-body-related ODEs and PDEs that are solved via TFC.

Chapter 5. Summary and Conclusions

This chapter summarizes the major ideas covered in the dissertation and draws conclusions based on the content discussed throughout. In addition, this chapter presents ideas for future study.

Appendices

The appendices include more detailed explanations of some of the topics covered in the main body of the text. In addition, they also include the following extensions of the TFC framework: nonlinear constraints, inequality constraints, parallelotope domains, lower-dimensional constraints in n-dimensions, and an extension to general fields, i.e., beyond the field of real numbers.

2. THEORY OF FUNCTIONAL CONNECTIONS

The seminal article on the Theory of Functional Connections[1] (TFC) introduced the notion of a *constrained expression* [11]: a mathematical expression that utilizes a function that can be chosen by the user, the so-called free function, that can describe *all* possible functions satisfying a given set of linear constraints. At the time the article was written, this statement was merely a conjecture but has since been proven mathematically. Since their conception, the process for deriving constrained expressions and the language used to discuss them has changed, but what they are in mathematical terms has remained constant: constrained expressions are functionals.[2] Therefore, it is useful to define a functional and investigate some of its properties before delving further into TFC constrained expressions.

2.1 Functionals

To begin, consider the following definition of a functional.

Definition 1

A functional, e.g., $f(x, g(x))$, has independent variable(s) and function(s) as inputs and produces a function as an output.

Note that a functional as defined here coincides with the computer science definition of a functional. One can think of a functional as a map for functions. That is, the functional takes a variable or variables and a function or functions as inputs and produces a function as its output, e.g., $f^*(x) = f(x, g(x))$. This dissertation is focused on constraint embedding, or in other words, functional interpolation; hence, for now, there is no need to concern oneself with the domains and ranges of the input and output functions. Rather, functionals can be

---

[1]This theory was originally published under the name "Theory of Connections." However, this name conflicted with a specific theory in differential geometry and was not the most accurate description of the functional interpolation method. Therefore, in 2019, this name was changed to the "Theory of Functional Connections" to highlight the tie to functional interpolation and the fact that it provides *all* functions satisfying a set of linear constraints in n-dimensional space.

[2]In other literature, functionals are also referred to as "functions of functions" or "higher-order functions."

discussed in the context of their potential input functions, hereon referred to as the domain of the functional, and potential output functions, hereon referred to as the codomain of the functional.

Next, the definitions of injective, surjective, and bijective are extended from functions to functionals.

Definition 2

A functional is injective if every function in its codomain is the image of at most one function in its domain.

Definition 3

A functional, $f(x, g(x))$, is surjective if for every function in the codomain, $f^*(x)$, there exists at least one function, $g(x)$, in the domain such that $f^*(x) = f(x, g(x))$.

Definition 4

A functional is bijective if it is both injective and surjective.

To elaborate, Figure 2.1 gives a graphical representation of each of these functionals, and examples of each of these functionals follow. Note that the phrase "smooth functions" is used here to denote continuous, infinitely differentiable, real-valued functions.

Consider the functional $f(x, g(x)) = e^{-g(x)}$ whose domain is all smooth functions and whose codomain is all smooth functions. The functional is injective because for every $f^*(x)$ in the codomain there is at most one $g(x)$ that maps $f(x, g(x))$ to $f^*(x)$. However, the functional is not surjective, because the functional does not span the codomain. For example, consider the desired output function $f^*(x) = -2$: there is no $g(x)$ that produces this output.

Next, consider the functional $f(x, g(x)) = g(x) - g(0)$ whose domain is all smooth functions and whose codomain is all smooth functions $f^*(x)$ such that $f^*(0) = 0$. This functional is surjective, because it spans the set of all smooth functions that are 0 when $x = 0$, but it is not injective. For example, the functions $g(x) = x$ and $g(x) = x + 3$ produce the same
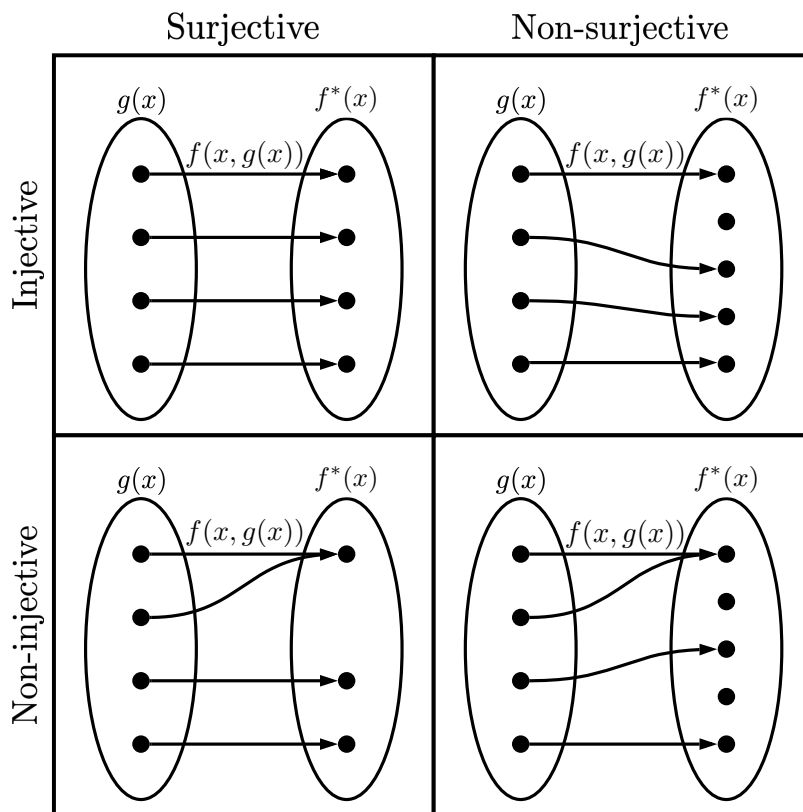
Figure 2.1: Graphical representation of injective and surjective functionals.

result, i.e., $f(x,x) = f(x, x+3) = x$.

Finally, consider the functional $f(x, g(x)) = g(x)$ whose domain is all smooth functions and whose codomain is all smooth functions. This functional is bijective, because it is both injective and surjective.

In addition, the notion of projection is extended to functionals. Consider an analogy to vector projection wherein a projection matrix, i.e., an idempotent matrix $P^n = P \ \forall n \in \mathbb{Z}^+$, projects a vector from one vector space to another. In other words, the properties of P are (1) it transports vectors from one vector space to another, and (2) when it operates on itself (the operator being matrix multiplication), it produces itself ($P^n = P$). A projection property for functionals can be defined similarly. Functionals already have the first property: they transport functions from one set, their domain, to another set, their codomain, e.g., the constrained expression transports functions from the set of all real-valued functions

defined at the constraints to the set of functions that satisfy the constraints. Following the analogy, if a functional produces itself when operating on itself, where the operator is using the functional's output as its input function, then that functional is said to be a projection functional.

Definition 5

A functional is said to be a projection functional if it produces itself when operating on itself.

For example, consider a functional operating on itself, $f(x, f(x, g(x)))$. If $f(x, f(x, g(x))) = f(x, g(x))$, then the functional is a projection functional. Note that proving $f(x, f(x, g(x))) = f(x, g(x))$ automatically extends to a functional operating on itself n times where $n \in \mathbb{Z}^+$: for example, $f(x, f(x, f(x, g(x)))) = f(x, f(x, g(x))) = f(x, g(x))$, and so on.

2.2 Univariate Theory

The majority of this dissertation focuses on multivariate TFC; however, the multivariate TFC framework is built by recursively applying univariate TFC. Hence, it is paramount the reader understands univariate TFC before moving to the multivariate case. First, the original form of the univariate constrained expression from Reference [11] will be presented via Example 2.1. Then, the constrained expression will be manipulated to expose an underlying structure made up of so-called projection functionals and switching functions [1]. Throughout the remainder of this section, that structure will be utilized to create constrained expressions for various types of linear constraints and prove mathematical theorems related to univariate constrained expressions.

Example 2.1: Constraints at a point

Constraints at a point consist of constraints on the value and derivatives at the point. Consider the follow constraints,

$$y(0) = 1, \quad y_x(1) = 2, \quad \text{and} \quad y(2) = 3.$$

Given a set of k point constraints, the univariate constrained expression takes the following form [11],

$$y(x, g(x)) = g(x) + \sum_{j=1}^{k} s_j(x)\, \eta_j(x, g(x)), \tag{2.1}$$

where $g(x)$ is a free function, $s_j(x)$ are k linearly independent functions called support functions, and $\eta_j(x, g(x))$ are k coefficient functionals that are solved by imposing the constraints. The free function $g(x)$ can be chosen to be any function provided that it is defined at the constraints' locations.

For this example, the support functions are chosen to be $s_1(x) = 1$, $s_2(x) = x^2$, and $s_3(x) = x^3$. Following Equation (2.1) and imposing the three constraints leads to the simultaneous set of equations

$$y(0) = 1 = g(0) + \eta_1(x, g(x))$$

$$y_x(1) = 2 = g_x(1) + 2\eta_2(x, g(x)) + 3\eta_3(x, g(x))$$

$$y(2) = 3 = g(2) + \eta_1(x, g(x)) + 4\eta_2(x, g(x)) + 8\eta_3(x, g(x)).$$

Solving this set of equations for the unknowns $\eta_j(x, g(x))$ leads to the solution,

$$\eta_1(x, g(x)) = 1 - g(0)$$

$$\eta_2(x, g(x)) = \frac{10 - 3g(0) + 3g(2) - 8g_x(1)}{4}$$

$$\eta_3(x, g(x)) = \frac{g(0) - g(2) + 2g_x(1)}{2}.$$

Substituting the coefficient functionals back into Equation (2.1) and simplifying yields,

$$y(x, g(x)) = g(x) + \frac{-2x^3 + 3x^2 + 4}{4}\left(1 - g(0)\right) + \left(-x^3 + 2x^2\right)\left(2 - g_x(1)\right)$$
$$+ \frac{2x^3 - 3x^2}{4}\left(3 - g(2)\right). \tag{2.2}$$

It is simple to verify that regardless of how $g(x)$ is chosen, provided $g(x)$ exists at the constraint points, Equation (2.2) always satisfies the given constraints.

The support functions in the previous example were selected as $s_1(x) = 1$, $s_2(x) = x^2$, and $s_3(x) = x^3$. However, these support functions could have been any linearly independent set of functions that permits a solution for the coefficient functionals $\eta_j(x, g(x))$; to clarify the latter of these requirements, consider the same constraints with support functions $s_1(x) = 1$, $s_2(x) = x$, and $s_3(x) = x^2$. Then, the set of equations with unknowns $\eta_j(x, g(x))$ is,

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 2 \\ 1 & 2 & 4 \end{bmatrix} \begin{Bmatrix} \eta_1(x, g(x)) \\ \eta_2(x, g(x)) \\ \eta_3(x, g(x)) \end{Bmatrix} = \begin{Bmatrix} 1 - g(0) \\ 2 - g_x(1) \\ 3 - g(2) \end{Bmatrix}.$$

Notice that when using these support functions, the matrix that multiplies the coefficient functionals is singular. Thus, no solution exists, and therefore, the support functions $s_1(x) = 1$, $s_2(x) = x$, and $s_3(x) = x^2$ are an invalid set for these constraints. Note that the matrix singularity does not depend on the free function. This means that the singularity arises when a linear combination of the selected support functions cannot be used to interpolate the constraints. Therefore, the support function matrix's singularity depends on both the support functions chosen and the specific constraints

to be embedded. This raises another important restriction on the expression of the support functions: not only must they be linearly independent, but they must constitute an interpolation model that is consistent with the specified constraints.

Notice that each term, except the term containing only the free function, in the constrained expression is associated with a specific constraint and has a particular structure. To illustrate, examine the first constraint term from Equation (2.2),

$$\underbrace{\frac{-2x^3 + 3x^2 + 4}{4}}_{\phi_1(x)} \underbrace{(1 - g(0))}_{\rho_1(x, g(x))}.$$

The first term in the product, $\phi_1(x)$, is called a *switching function*[a] and is a function that is equal to 1 when evaluated at the constraint it is referencing and equal to 0 when evaluated at all the other constraints. For example, when evaluating the switching function $\phi_1(x)$ at the constraint it is referencing it is equal to 1, i.e., $\phi_1(0) = 1$, and when it is evaluated at the other constraints it is equal to 0, i.e., $\frac{\partial \phi_1}{\partial x}(1) = 0$ and $\phi_1(2) = 0$. The second term of the product, $\rho_1(x, g(x))$, is called a *projection functional*, and is derived by setting the constraint function equal to zero and replacing $y(x)$ with $g(x)$. In the case of constraints at a point, this is simply the difference between the constraint value and the free function evaluated at the constraint point. It is called the projection functional because it projects the free function to the set of functions that vanish at the constraint.

The switching-projection structure is important because it shows up in other constraint types too. Based on this structure, an alternate way to define the constrained expression can be derived,

$$y(x, g(x)) = g(x) + \sum_{j=1}^{k} \phi_j(x) \, \rho_j(x, g(x)). \tag{2.3}$$

11

For this case, the projection functionals are simple to derive, but the switching functions require some attention. From their definition, these functions must go to 1 at their associated constraint and 0 at all other constraints. Hence, the following algorithm for deriving the switching functions is proposed:

1. Choose k support functions, $s_k(x)$.

2. Write each switching function as a linear combination of the support functions with unknown coefficients.

3. Based on the switching function definition, write a system of equations to solve for the unknown coefficients.

To validate that this algorithm works, consider the same constraints and support functions and rederive the constrained expression shown in Equation (2.2). Hence, $\phi_1(x) = s_i(x)\,\alpha_{i1}$, $\phi_2(x) = s_i(x)\,\alpha_{i2}$, and $\phi_3(x) = s_i(x)\,\alpha_{i3}$, for some as yet unknown coefficients α_{ij}. Note that in the previous mathematical expressions and throughout the remainder of the dissertation, the Einstein summation convention is used to improve readability. Now, the definition of the switching function is used to come up with a set of equations. For example, the first switching function has the three equations,

$$\phi_1(0) = 1, \quad \frac{\partial \phi_1}{\partial x}(1) = 0, \quad \text{and} \quad \phi_1(2) = 0.$$

These equations are expanded in terms of the support functions,

$$\phi_1(0) = (1) \cdot \alpha_{11} + (0) \cdot \alpha_{21} + (0) \cdot \alpha_{31} = 1$$
$$\frac{\partial \phi_1}{\partial x}(1) = (0) \cdot \alpha_{11} + (2) \cdot \alpha_{21} + (3) \cdot \alpha_{31} = 0$$
$$\phi_1(2) = (1) \cdot \alpha_{11} + (4) \cdot \alpha_{21} + (8) \cdot \alpha_{31} = 0,$$

which can be compactly written as,

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 2 & 3 \\ 1 & 4 & 8 \end{bmatrix} \begin{Bmatrix} \alpha_{11} \\ \alpha_{21} \\ \alpha_{31} \end{Bmatrix} = \begin{Bmatrix} 1 \\ 0 \\ 0 \end{Bmatrix}.$$

The same is done for the other two switching functions to produce a set of equations that can be solved by matrix inversion.

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 2 & 3 \\ 1 & 4 & 8 \end{bmatrix} \begin{bmatrix} \alpha_{11} & \alpha_{12} & \alpha_{13} \\ \alpha_{21} & \alpha_{22} & \alpha_{23} \\ \alpha_{31} & \alpha_{32} & \alpha_{33} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} \alpha_{11} & \alpha_{12} & \alpha_{13} \\ \alpha_{21} & \alpha_{22} & \alpha_{23} \\ \alpha_{31} & \alpha_{32} & \alpha_{33} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 2 & 3 \\ 1 & 4 & 8 \end{bmatrix}^{-1} = \begin{bmatrix} 1 & 0 & 0 \\ \frac{3}{4} & 2 & -\frac{3}{4} \\ -\frac{1}{2} & -1 & \frac{1}{2} \end{bmatrix}.$$

Substituting the constants back into the switching functions and simplifying yields,

$$\phi_1(x) = \frac{-2x^3 + 3x^2 + 4}{4}, \quad \phi_2(x) = -x^3 + 2x^2, \quad \text{and} \quad \phi_3(x) = \frac{2x^3 - 3x^2}{4}.$$

Substituting the projection functionals and switching functions back into the constrained expression shown in Equation (2.3) yields,

$$y(x, g(x)) = g(x) + \frac{-2x^3 + 3x^2 + 4}{4}\Big(1 - g(0)\Big) + \Big(-x^3 + 2x^2\Big)\Big(2 - g_x(1)\Big)$$
$$+ \frac{2x^3 - 3x^2}{4}\Big(3 - g(2)\Big),$$

which is identical to Equation (2.2).

[a]Reference [11] introduced these switching functions as "coefficient" functions, $\boldsymbol{\beta}_k$, but they were not used in the same way the switching-projection form uses them.

As demonstrated in Example 2.1, the switching-projection approach, Equation (2.3), is a valid method for deriving constrained expressions; although it was only demonstrated for one set of constraints here, this constrained expression derivation technique will be proven mathematically in Section 2.2.4. Similar to the original approach, Equation (2.1), there is a risk of obtaining a singular matrix when solving for α_{ij} if the support functions selected are not able to interpolate the constraints. However, as will be demonstrated in the sections that follow, the switching-projection approach can be used for many constraint types, easily extended to multivariate domains via recursive applications of the univariate theory, and lends itself nicely to mathematical proofs. Before moving to these other topics, it is useful to first examine the anatomy of a constraint and define the so-called constraint operator. This analysis will prove invaluable as it provides a method to unify the way linear constraints are written. Consequently, mathematical analyses can be done on this unified form and thereby applied to all linear constraints: without this method, one would need to conduct the same mathematical analysis for each constraint type separately.

2.2.1 Anatomy of a Linear Constraint

Linear constraints can be conveniently dissected into two portions: (1) an operator that operates on a dependent variable and (2) the remaining constants and functions of the constraint. Let the former be called the constraint operator and denoted by the symbol $\mathfrak{C}$ and the latter denoted by the symbol κ. Using this nomenclature, a constraint on the dependent variable y would typically be written in the form,

$$\kappa = \mathfrak{C}[y].$$

For example, the constraint $3 = 2y(2) - \pi y_{xx}(0)$ consists of $\kappa = 3$ and $\mathfrak{C}[y] = 2y(x) - \pi y_{xx}(0)$. Definition 6 defines the constraint operator more rigorously.

Definition 6

The constraint operator, $\mathfrak{C}_i$, is a linear operator that operates on a function and returns the function evaluated at the i-th specified constraint.

The word evaluation in the previous definition requires some elaboration; evaluation means to evaluate the operand function in the same way as the dependent variable in the constraint. Notice that this means the constraint operator is not affected by terms in the constraint that do not contain the dependent variable. As an example, again consider the constraint $3 = 2y(2) + \pi y_{xx}(0)$, and suppose it is the first constraint in the set ($i = 1$). For this constraint, the constraint operator operates as follows,

$$\mathfrak{C}_1[f(x)] = 2f(2) + \pi f_{xx}(0).$$

In addition, notice that the constraint operator satisfies the two properties of a linear operator:

1. $\mathfrak{C}_i[f(x) + g(x)] = \mathfrak{C}_i[f(x)] + \mathfrak{C}_i[g(x)]$

2. $\mathfrak{C}_i[ag(x)] = a\mathfrak{C}_i[g(x)]$

For example, again consider the linear constraint $3 = 2y(2) + \pi y_{xx}(0)$,

$$\mathfrak{C}_1[f(x) + g(x)] = \mathfrak{C}_1[f(x)] + \mathfrak{C}_1[g(x)] = 2f(2) + \pi f_{xx}(0) + 2g(2) + \pi g_{xx}(0)$$
$$\mathfrak{C}_1[af(x)] = a\mathfrak{C}_1[f(x)] = a\left(2f(2) + \pi f_{xx}(0)\right).$$

Naturally, the constraint operator has specific properties when operating on the support functions, switching functions, and projection functionals.

Property 1

The constraint operator acting on the support functions $s_j(x)$ produces the support matrix

$$\mathbb{S}_{ij} = \mathfrak{C}_i[s_j(x)].$$

Consider the example given in Example 2.1 where the support functions were $s_1(x) = 1$, $s_2(x) = x^2$, and $s_3(x) = x^3$. By applying the constraint operator,

$$\mathbb{S}_{ij} = \mathfrak{C}_i[s_j(x)] = \begin{bmatrix} \mathfrak{C}_1[s_1(x)] & \mathfrak{C}_1[s_2(x)] & \mathfrak{C}_1[s_3(x)] \\ \mathfrak{C}_2[s_1(x)] & \mathfrak{C}_2[s_2(x)] & \mathfrak{C}_2[s_3(x)] \\ \mathfrak{C}_3[s_1(x)] & \mathfrak{C}_3[s_2(x)] & \mathfrak{C}_3[s_3(x)] \end{bmatrix}$$

$$= \begin{bmatrix} s_1(0) & s_2(0) & s_3(0) \\ \frac{\partial s_1}{\partial x}(1) & \frac{\partial s_2}{\partial x}(1) & \frac{\partial s_3}{\partial x}(1) \\ s_1(2) & s_2(2) & s_3(2) \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 2 & 3 \\ 1 & 4 & 8 \end{bmatrix},$$

which is identical to the support matrix from Example 2.1. It follows that $\mathbb{S}_{ij}\,\alpha_{jk} = \alpha_{ij}\,\mathbb{S}_{jk} = \delta_{ik}$, where δ_{ik} is the Kroneker delta, and the solution of the α_{ij} coefficients can be determined by simply inverting the support matrix.

Property 2

The constraint operator acting on the switching functions $\phi_j(x)$ produces the Kronecker delta.

$$\mathfrak{C}_i[\phi_j(x)] = \mathfrak{C}_i[s_k(x)\alpha_{kj}] = \mathfrak{C}_i[s_k(x)]\alpha_{kj} = \mathbb{S}_{ik}\alpha_{kj} = \delta_{ij}$$

This property is just a mathematical restatement of the linguistic definition of the switching function given earlier. One can intuit this property from the switching function definition, since they evaluate to 1 at their specified constraint condition, i.e., $i = j$, and to 0 at all other constraint conditions, i.e., $i \neq j$.

Using the constraint operator definition, one can define the projection functional in a compact and precise manner.

Definition 7

The projection functional is the difference between the numerical portion of the constraint and the constraint operator acting on the free function. Mathematically,

$$\rho_i(x, g(x)) = \kappa_i - \mathfrak{C}_i[g(x)].$$

Definition 8

The univariate free function is any function $g(x) \colon \mathbb{R} \mapsto \mathbb{R}$ such that $\mathfrak{C}_i[g]$ is defined.

Again, consider the constraint $3 = 2y(2) + \pi y_{xx}(0)$,

$$\rho_1(x, g(x)) = \kappa_1 - \mathfrak{C}_1[g(x)] = 3 - 2g(2) - \pi g_{xx}(0).$$

Note that in the univariate case, κ_i is a scalar value, i.e., $\kappa_i \in \mathbb{R}$, but in the multivariate case, κ_i can be a function. In addition, notice what happens if $g(x)$ is a function that already satisfies the constraints.

Property 3

If $g(x)$ is a function that satisfies the constraints, then the projection functional is equal to zero.

Property 3 follows from the definition of the projection functional; if $g(x)$ satisfies the constraints, then,

$$\rho_i(x, g(x)) = \kappa_i - \mathfrak{C}_i[g(x)]$$
$$= \kappa_i - \kappa_i$$
$$= 0.$$

Now that the constraint operator has been defined, and consequently, rigorous definitions for the projection functionals and switching functions have been provided, other constraint types become easy to embed into univariate constrained expressions: For example, integral constraints.

Example 2.2: Integral constraints

Consider the following set of constraints,

$$\int_{-2}^{3} y(x) \, \mathrm{d}x = 5 \quad \text{and} \quad \int_{0}^{2} 3y(x) \, \mathrm{d}x = 2.$$

Based on Definition 7, the projection functionals for these constraints can be written as,

$$\rho_1(x, g(x)) = 5 - \int_{-2}^{3} g(\tau) \, \mathrm{d}\tau$$

$$\rho_2(x, g(x)) = 2 - \int_{0}^{2} 3g(\tau) \, \mathrm{d}\tau.$$

Notice that the integrals in the projection functionals use a dummy variable, τ, rather than x. Furthermore, based on Property 2, the switching function equations can be written as,

$$\int_{-2}^{3} \phi_1(x) \, \mathrm{d}x = 1, \qquad \qquad \int_{0}^{2} 3\phi_1(x) \, \mathrm{d}x = 0,$$

$$\int_{-2}^{3} \phi_2(x) \, \mathrm{d}x = 0, \qquad \qquad \int_{0}^{2} 3\phi_2(x) \, \mathrm{d}x = 1.$$

Setting $\phi_1(x)$ and $\phi_2(x)$ to be a linear combination of the support functions $s_1(x) = 1$ and $s_2(x) = x$ with unknown coefficients α_{ij} yields,

$$\begin{bmatrix} 5 & \frac{5}{2} \\ 6 & 6 \end{bmatrix} \begin{bmatrix} \alpha_{11} & \alpha_{12} \\ \alpha_{21} & \alpha_{22} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

18

$$\begin{bmatrix} \alpha_{11} & \alpha_{12} \\ \alpha_{21} & \alpha_{22} \end{bmatrix} = \begin{bmatrix} \frac{2}{5} & -\frac{1}{2} \\ -\frac{1}{6} & \frac{1}{3} \end{bmatrix}.$$

Hence,

$$\phi_1(x) = \frac{2 - 2x}{5} \quad \text{and} \quad \phi_2(x) = \frac{2x - 1}{6}.$$

Thus, following Equation (2.3), the constrained expression for these constraints is,

$$y(x, g(x)) = g(x) + \frac{2 - 2x}{5}\left(5 - \int_{-2}^{3} g(\tau)\,\mathrm{d}\tau\right) + \frac{2x - 1}{6}\left(2 - \int_{0}^{2} 3g(\tau)\,\mathrm{d}\tau\right).$$

The previous constrained expression will always satisfy the constraints regardless of how the free function, $g(x)$, is chosen.

2.2.2 Component Constraints

When handling component constraints, one must decide which dependent variable's constrained expression the component constraint will be embedded into. This dependent variable will define the constraint operator, and all other dependent variables will become part of the constraint's κ term. Regardless of which dependent variable is chosen, a valid constrained expression will be produced.

Example 2.3: Component constraints

Consider the following set of constraints,

$$u(0) + v(0) = 5 \quad \text{and} \quad u_x(2) + v(3) = 4.$$

Two different sets of constrained expressions will be produced: one where the component constraints are embedded into the constrained expression for u, and the second where the component constraints are embedded into the constrained expression for v.

If the constraints are embedded into u, then the projection functionals are,

$$\rho_1(x, g^u(x), g^v(x)) = 5 - g^u(0) - v(0, g^v(x))$$

$$\rho_2(x, g^u(x), g^v(x)) = 4 - g^u_x(2) - v(3, g^v(x)),$$

where $g^u(x)$ is the free function used in the u constrained expression; similarly, $g^v(x)$ will be the free function used in the v constrained expression. The equations for the switching function are,

$$\phi_1(0) = 1, \qquad \frac{\partial \phi_1}{\partial x}(2) = 0$$

$$\phi_2(0) = 0, \qquad \frac{\partial \phi_2}{\partial x}(2) = 1.$$

Let the support functions be $s_1(x) = 1$ and $s_2(x) = x$, then

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \alpha_{11} & \alpha_{12} \\ \alpha_{21} & \alpha_{22} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} \alpha_{11} & \alpha_{12} \\ \alpha_{21} & \alpha_{22} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}.$$

Thus, the switching functions are,

$$\phi_1(x) = 1 \quad \text{and} \quad \phi_2(x) = x,$$

and the first set of constrained expressions, where the component constrains are embedded into u is,

$$u(x, g^u(x), g^v(x)) = g^u(x) + 5 - g^u(0) - v(0, g^v(x))$$
$$+ x\left(4 - g_x^u(2) - v(3, g^v(x))\right) \tag{2.4}$$
$$v(x, g^v(x)) = g^v(x).$$

A similar derivation yields the second set of constrained expressions, where the component constraints are embedded into v,

$$u(x, g^u(x)) = g^u(x)$$
$$v(x, g^v(x), g^u(x)) = g^v(x) + \frac{3 - x}{3}\left(5 - u(0, g^u(x)) - g^v(0)\right) \tag{2.5}$$
$$+ \frac{x}{3}\left(4 - u_x(2, g^u(x)) - g^v(3)\right).$$

Notice that regardless of how $g^u(x)$ and $g^v(x)$ are chosen, Equations (2.4) and (2.5) will always satisfy the constraints.

Example 2.3 shows that component constraints can be placed on either dependent variable. However, notice that in the previous example, one could not put one component constraint on one dependent variable and the other component constraint on the other; doing so would result in an infinite recursion whenever trying to evaluate either constrained expression because each constrained expression would require an evaluation of the other. For example, suppose one tried to embed the first component constraint in u and the second in v, then, the constrained expressions would be,

$$u(x, g^u(x)) = g^u(x) + 5 - g^u(0) - v(0, g^v(x))$$
$$v(x, g^v(x), g^u(x)) = g^v(x) + 4 - g^v(3) - u_x(2, g^u(x)).$$

21

Notice that evaluating either constrained expression requires an evaluation of the other; hence, an infinite recursion is encountered.

The aforementioned infinite recursions can be avoided, in general, by choosing to embed as many component constraints as possible into one dependent variable, then embed as many component constraints that remain as possible into the second dependent variable, and so on, until all component constraints are accounted for. However, there may be instances when one is interested in all ways in which a set of component constraints can be embedded. Fortunately, graph theory provides a succinct method to do just that. For readers unfamiliar with the basics of graph theory, see Appendix A.

For a given set of constraints, consider a directed graph whose nodes are composed of all dependent variables that contain component constraints and whose edges connect nodes if there is a constraint between them. The direction of the edges will denote dependency in the processing order, i.e., for every edge, the target must be processed before its source is processed. Thus, to determine the order in which to create the constrained expressions, one need only trace the graph backwards, starting at the leaf node(s) and working towards the root node(s). Infinite recursions can be avoided by checking that the resultant graph is acyclic. As mentioned in Appendix A, if a directed graph's adjacency matrix is nilpotent, then the graph is acyclic [18]. Hence, one can create all possible graphs for a given set of constraints by considering all permutations of all source/target pairs—2^n possibilities where n denotes the number of source/target pairs—and then reduce the set to those that do not contain infinite recursions by using the adjacency matrix.

Example 2.4: Component constraint graphs

Consider the follow set of component constraints,

$$u(0) + v(0) + w(0) = 5, \quad u_x(1) + v(2) = \pi$$

$$u_x(3) + v_x(4) = e, \quad \text{and} \quad v(1) + w(2) = 1.$$
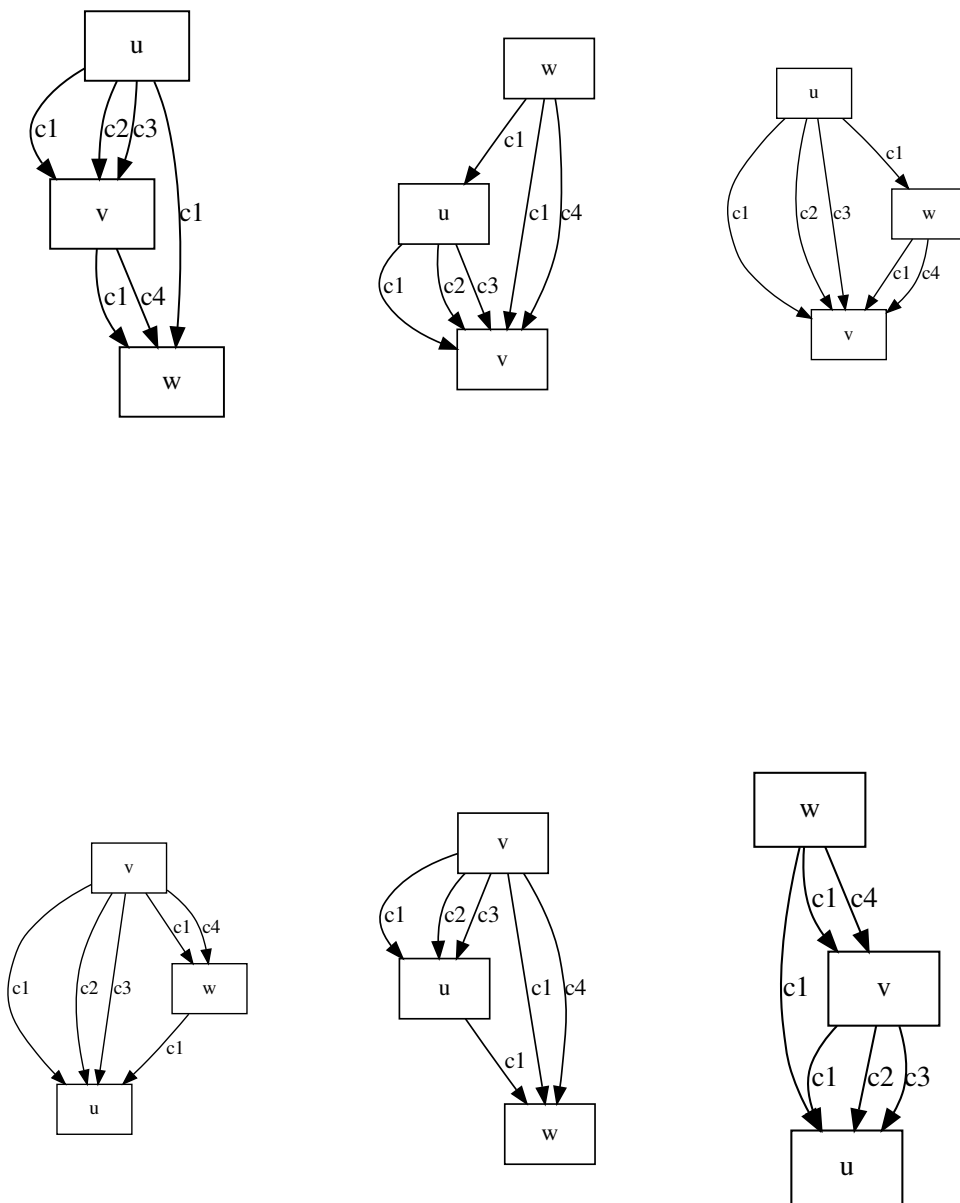
Figure 2.2: Valid component constraint graphs.

Let c_1 denote the first component constraint, c_2 the second component constraint, and so on. Using the graph theory method just described, a set of directed, acyclic

graphs can be created that show all possible ways in which the component constraints can be embedded; these graphs are shown in Figure 2.2. For example, the first graph in Figure 2.2 is associated with embedding the first, second, and third component constraints into the constrained expression for u, and the fourth component constraint into the constrained expression for v. Moreover, based on the same graph, when constructing the constrained expressions, one must build the constrained expression for w first, the constrained expression for v second, and the constrained expression for u last: this order was determined by traversing the graph backwards from leaf node to root node.

2.2.3 Linear Constraints

The term linear constraints refers to constraints that consist of linear combinations of the constraint types shown earlier. These constraints can be embedded by simply applying the techniques introduced previously.

Example 2.5: Linear constraints

Consider the following set of constraints,

$$u(0) + u_x(0) = \pi, \quad u(1) + v(1) = 5,$$
$$\int_{-1}^{1} v(x)\, \mathrm{d}x + v(1) = 6, \quad \text{and} \quad v(2) = e,$$

and suppose the choice is made to embed the component constraint inside the u constrained expression: although in this case, it would be equally valid to choose to embed it inside the constrained expression for v. Thus, for u, the projection functionals are,

$$\rho_1(x, g^u(x)) = \pi - g^u(0) - g_x^u(0) \quad \text{and} \quad \rho_2(x, g^u(x), g^v(x)) = 5 - g^u(1) - v(1, g^v(x)),$$

and for v they are,

$$\rho_1(x, g^v(x)) = 6 - \int_{-1}^{1} g^v(\tau) \, \mathrm{d}\tau - g^v(1) \quad \text{and} \quad \rho_2(x, g^v(x)) = e - g^v(2).$$

For u, the switching function equations are,

$$\phi_1(0) + \frac{\partial \phi_1}{\partial x}(0) = 1, \qquad \phi_1(1) = 0,$$

$$\phi_2(0) + \frac{\partial \phi_2}{\partial x}(0) = 0, \qquad \phi_2(1) = 1,$$

and for v they are,

$$\int_{-1}^{1} \phi_1(x) \, \mathrm{d}x + \phi_1(1) = 1, \qquad \phi_1(2) = 0,$$

$$\int_{-1}^{1} \phi_2(x) \, \mathrm{d}x + \phi_2(1) = 0, \qquad \phi_2(2) = 1.$$

For u, let the support functions be $s_1(x) = x$ and $s_2(x) = x^2$. Then,

$$\begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} \alpha_{11} & \alpha_{12} \\ \alpha_{21} & \alpha_{22} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} \alpha_{11} & \alpha_{12} \\ \alpha_{21} & \alpha_{22} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ -1 & 1 \end{bmatrix},$$

and the switching functions are,

$$\phi_1(x) = x - x^2 \quad \text{and} \quad \phi_2(x) = x^2.$$

For v, let the support functions be $s_1(x) = 1$ and $s_2(x) = x$. Then,

$$
\begin{bmatrix} 3 & 1 \\ 1 & 2 \end{bmatrix} \begin{bmatrix} \alpha_{11} & \alpha_{12} \\ \alpha_{21} & \alpha_{22} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}
$$

$$
\begin{bmatrix} \alpha_{11} & \alpha_{12} \\ \alpha_{21} & \alpha_{22} \end{bmatrix} = \begin{bmatrix} \frac{2}{5} & -\frac{1}{5} \\ -\frac{1}{5} & \frac{3}{5} \end{bmatrix},
$$

and the switching functions are,

$$
\phi_1(x) = \frac{2 - x}{5} \quad \text{and} \quad \phi_2(x) = \frac{3x - 1}{5}.
$$

Putting the projection functionals and switching functions together yields the constrained expressions,

$$
u(x, g^u(x), g^v(x)) = g^u(x) + (x - x^2)\Big(\pi - g^u(0) - g_x^u(0)\Big)
$$
$$
+ x^2\Big(5 - g^u(1) - v(1, g^v(x))\Big)
$$
$$
v(x, g^v(x)) = g^v(x) + \frac{2 - x}{5}\Big(6 - \int_{-1}^{1} g^v(\tau)\, \mathrm{d}\tau - g^v(1)\Big) + \frac{3x - 1}{5}\Big(e - g^v(2)\Big).
$$

As before, regardless of how $g^u(x)$ and $g^v(x)$ are chosen, these two constrained expressions will always satisfy the constraints.

In addition to actual linear constraints, this technique can be applied to constraints that can be rewritten as linear constraints. For example, consider the nonlinear constraints shown in Appendix B; these nonlinear constraints can be rewritten as a set of linear constraints, which ultimately means they can be embedded into constrained expressions.

2.2.4 Univariate Constrained Expression Theorems

This section presents important theorems related to univariate TFC constrained expressions. Theorem 1 shows that the constrained expression form given in the previous section satisfies the constraints regardless of how the free function is chosen. This theorem is critical, as this is the objective of constrained expressions.

Theorem 1

The switching-projection form of the univariate constrained expression,

$$y(x, g(x)) = g(x) + \phi_j(x)\rho_j(x, g(x)),$$

satisfies the user-specified constraints for any free function.

Proof: One must show that $\mathfrak{C}_i[y(x, g(x))] = \kappa_i$. Apply $\mathfrak{C}_i$ to $y(x, g(x))$ and drop the x and $g(x)$ arguments for clarity.

$$\mathfrak{C}_i[y] = \mathfrak{C}_i[g] + \mathfrak{C}_i[\phi_j \rho_j]$$

Expand ρ_j and simplify,

$$\mathfrak{C}_i[y] = \mathfrak{C}_i[g] + \mathfrak{C}_i\left[\phi_j(\kappa_j - \mathfrak{C}_j[g])\right]$$
$$\mathfrak{C}_i[y] = \mathfrak{C}_i[g] + \mathfrak{C}_i[\phi_j](\kappa_j - \mathfrak{C}_j[g])$$
$$\mathfrak{C}_i[y] = \mathfrak{C}_i[g] + \delta_{ij}(\kappa_j - \mathfrak{C}_j[g])$$
$$\mathfrak{C}_i[y] = \mathfrak{C}_i[g] + \kappa_i - \mathfrak{C}_i[g]$$
$$\mathfrak{C}_i[y] = \kappa_i.$$

Therefore, $\mathfrak{C}_i[y(x, g(x))] = \kappa_i$ for an any free function $g(x)$. ∎

The natural question that arises after learning that the constrained expression satisfies the constraints for any free function is, can the constrained expression represent any function that satisfies the constraints? In other words, does the constrained expression represent the family of all possible functions that satisfy the constraints? Theorem 2 shows that indeed it does.

Theorem 2

For any function satisfying the constraints, $f(x)\colon \mathbb{R} \mapsto \mathbb{R}$, there exists at least one free function, $g(x)$, such that the constrained expression $y(x, g(x)) = f(x)$. In other words, constrained expressions are surjective functionals whose domain is all free functions and whose codomain is all functions that satisfy the constraints.

Proof: As highlighted in Property 3, the projection functionals are equal to zero whenever $g(x)$ satisfies the constraints. Thus, if $g(x)$ is a function that satisfies the constraints, then the constrained expression becomes,

$$y(x, g(x)) = g(x) + \rho_i(x, g(x))\phi_i(x)$$
$$= g(x) + 0$$
$$= g(x).$$

Hence, by choosing $g(x) = f(x)$, the constrained expression becomes $y(x, f(x)) = f(x)$. Therefore, for any function satisfying the constraints, $f(x)$, there exists at least one free function, $g(x) = f(x)$, such that the constrained expression is equal to the function satisfying the constraints, i.e., $y(x, f(x)) = f(x)$. ∎

Given that the codomain of the constrained expression functional is the set of all functions satisfying the constraints, but the domain is the set of all functions, one might hypothesize that there may be multiple free function choices that produce the same output: Theorem 3

shows that there are.

Theorem 3

For a given function satisfying the constraints, $f(x)\colon \mathbb{R} \mapsto \mathbb{R}$, the free function, $g(x)$, such that the constrained expression $y(x, g(x)) = f(x)$ is not unique. In other words, constrained expressions are not injective functionals over the domain of all free functions and codomain of all functions that satisfy the constraints.

Proof: Consider the free function $g(x) = f(x) + \beta_j\, s_j(x)$ where β_j are scalar values on $\mathbb{R}$ and $s_j(x)$ are the support functions used to construct the switching functions. Substituting this free function into the constrained expression,

$$y(x, g(x)) = g(x) + \phi_i(x)\, \rho_i(x, g(x)),$$

and dropping the x and $g(x)$ arguments for clarity yields,

$$y = f + \beta_j s_j + \phi_i \rho_i.$$

Now, expand the projection functionals and simplify,

$$y = f + \beta_j\, s_j + \phi_i\Big(\kappa_i - \mathfrak{C}_i[f + \beta_j\, s_j]\Big)$$
$$y = f + \beta_j s_j + \phi_i\Big(\kappa_i - \mathfrak{C}_i[f] - \mathfrak{C}_i[s_j]\beta_j\Big)$$
$$y = f + \beta_j s_j - \phi_i\mathfrak{C}_i[s_j]\beta_j.$$

Next, decompose the switching functions and simplify,

$$y = f + \beta_j s_j - \alpha_{ki} s_k \mathbb{S}_{ij}\beta_j$$
$$y = f + \beta_j\Big(\delta_{jk} - \alpha_{ki}\mathbb{S}_{ij}\Big)s_k$$

29

$$y = f + \beta_j \Big(\delta_{jk} - \delta_{jk} \Big) s_k$$

$$y = f.$$

The result obtained is independent of the $\beta_j s_j(x)$ terms in the free function. Therefore, for any function, $f(x)$, satisfying the constraints, the free function, $g(x)$, that produces $f(x)$ via the constrained expression, i.e., $y(x, g(x)) = f(x)$, is not unique. ∎

Notice that the non-uniqueness of $g(x)$ depends on the support functions used in the constrained expression, which has an immediate consequence when using constrained expressions in optimization. If any terms in $g(x)$ are linearly dependent to the support functions used to construct the constrained expression, their contribution is negated and thus arbitrary. For some optimization techniques, it is critical that the linearly dependent terms that do not contribute to the final solution be removed; else, the optimization technique becomes impaired. For example, when solving differential equations using a linear combination of basis functions as the free function and least-squares as the optimization process [1, 19, 20], the basis functions that are linearly dependent to the support functions have to be omitted from the free function to maintain full rank matrices in the least-squares.

Based on the previous results, one convenient way to think of the constrained expression is a functional that projects the free function to the set of functions that satisfy the constraints. As Theorem 4 shows, thinking of the constrained expression as a projection functional is a valid perspective.

Theorem 4

The constrained expression is a projection functional.

Proof: One must show that $y(x, y(x, g(x))) = y(x, g(x))$. Theorem 1 states that the constrained expression returns a function that satisfies the constraints. In other

words, for any $g(x)$ that is defined at the constraints, $y(x, g(x))$ is a function that satisfies the constraints. From Theorem 2, if the free function used in the constrained expression satisfies the constraints, then the constrained expression returns that free function exactly. Hence, if the constrained expression functional is given itself as the free function, it will simply return itself. ∎

The previous proofs coupled with the functional-related definitions given earlier provide a more rigorous definition for the univariate constrained expression: the univariate constrained expression is a surjective, projection functional whose domain is the set of all free functions and whose codomain is the set of all functions that satisfy the constraints. It is surjective because it spans the set of all functions that satisfy the constraints, its codomain, based on Theorem 2, but it is not injective because Theorem 3 shows that functions in the codomain are the image of more than one function in the domain; constrained expressions are thus not bijective either because they are not injective. Moreover, the constrained expression is a projection functional as shown in Theorem 4.

2.3 Multivariate Theory

This section utilizes the univariate theory introduced in the previous section to extend TFC to the multivariate case. As such, one should ensure they have a firm grasp of the concepts introduced in the univariate section before moving on. The section begins by introducing the recursive method: a method for generating multivariate constrained expressions by using the univariate constrained expression for one independent variable as the free function in the univariate constrained expression for a different independent variable. Afterward, the mathematical theorems presented for univariate constrained expressions are also extended to the multivariate case. Finally, a compact tensor form of the multivariate constrained expression is presented.

31

2.3.1 Recursive Method

Oftentimes, the constraints of a problem do not include integral constraints. In these cases, one independent variable's constraints will not interfere with another independent variable's constraints. Consequently, using the univariate constrained expression for one independent variable as the free function in the univariate constrained expression of another independent variable produces a function that satisfies both independent variables' constraints. To prove this, one must first understand how the constraint operator of one independent variable affects the constrained expression of another independent variable; the following discussion and properties will help achieve this understanding. A pre-superscript will be used to distinguish the operators, functions, and functionals of one independent variable from another. For example, $^{(k)}\mathfrak{C}_j$ denotes the constraint operator for the j-th constraint of the k-th independent variable.

Property 4

For non-integral constraints, the constraint operator for the k-th independent variable operating on a product of functions wherein one function is not a function of the k-th independent variable and the other is leads to,

$$^{(k)}\mathfrak{C}_j[f(x_1,\ldots,x_{k-1},x_{k+1},\ldots,x_n)h(x_1,\ldots,x_k,\ldots,x_n)] =$$
$$f(x_1,\ldots,x_{k-1},x_{k+1},\ldots,x_n)^{(k)}\mathfrak{C}_j[h(x_1,\ldots,x_k,\ldots,x_n)],$$

where f is not a function of the k-th independent variable, x_k, but h is.

Property 4 holds for non-integral constraints because $^{(k)}\mathfrak{C}_j$ operates on the k-th independent variable only, and f is not a function of the k-th independent variable, i.e., it is effectively a constant. In particular, this property is useful in multivariate expressions, which oftentimes contain such products. For example, $^{(k)}\mathfrak{C}_i[^{(k)}\phi_j\,^{(k)}\kappa_j] = {}^{(k)}\mathfrak{C}_i[^{(k)}\phi_j]\,^{(k)}\kappa_i$.

Property 5

A set of non-integral constraints is consistent if and only if ${}^{(k)}\mathfrak{C}_j[{}^{(l)}\kappa_i] = {}^{(l)}\mathfrak{C}_i[{}^{(k)}\kappa_j]$.

Property 5 is easiest to understand via an example of inconsistent constraints:

$$z(x,0) = 5 \quad \text{and} \quad z(0,y) = 4.$$

Clearly, these constraints cannot simultaneously be satisfied at the intersection point $z(0,0)$.

In addition, for multivariate constraints, the free function must be locally C^m in the neighborhood of the geometric intersection of constraints, where m is the sum of the orders of derivatives of the intersecting constraints. This restriction on the free function is necessary for generating constrained expressions using recursive applications of univariate expressions, as it ensures that Clairaut's theorem holds for the free function, and thus, ${}^{(l)}\mathfrak{C}_i\left[{}^{(k)}\mathfrak{C}_j[g]\right] = {}^{(k)}\mathfrak{C}_j\left[{}^{(l)}\mathfrak{C}_i[g]\right]$.

Definition 9

The multivariate free function is any function $g(x)\colon \mathbb{R}^n \mapsto \mathbb{R}$ such that ${}^{(i_j)}\mathfrak{C}_k[g]$ is defined and ${}^{(i_j)}\mathfrak{C}_m\left[\cdots\left[{}^{(i_k)}\mathfrak{C}_n[g]\right]\cdots\right]$ is defined, where the latter consists of at most one constraint operator from each dimension; the latter must be freely permutable, e.g., ${}^{(i_j)}\mathfrak{C}_m\left[\cdots\left[{}^{(i_k)}\mathfrak{C}_n[g]\right]\cdots\right] = {}^{(i_k)}\mathfrak{C}_n\left[\cdots\left[{}^{(i_j)}\mathfrak{C}_m[g]\right]\cdots\right]$, for any non-integral constraints.

These properties and restriction on the free function are utilized in Theorem 5 to show that the recursive method produces a valid multivariate constrained expression for non-integral constraints.

Theorem 5

For non-integral constraints, a valid multivariate constrained expression can be constructed by recursively applying the univariate constrained expression from one in-

dependent variable as the free function in the constrained expression for another independent variable. In this recursion, all univariate constrained expressions must be used once and only once, and the first univariate constrained expression is built using a regular free function.

Proof: First, show that $^{(k)}u(\boldsymbol{x}, {}^{(l)}u(\boldsymbol{x}, g(\boldsymbol{x})))$ is a valid bivariate constrained expression that satisfies both sets of constraints, where $\boldsymbol{x}$ represents a vector of the independent variables, i.e., $\boldsymbol{x} = \{x_1, x_2, \cdots, x_n\}$. Then, apply it n times recursively to produce an n-dimensional, multivariate constrained expression that satisfies the constraints on all n dimensions. Consider two univariate constrained expressions:

$$^{(k)}u(\boldsymbol{x}, g(\boldsymbol{x})) = g(\boldsymbol{x}) + {}^{(k)}\phi_j(x_k)\, {}^{(k)}\rho_j(\boldsymbol{x}, g(\boldsymbol{x})),$$

$$^{(l)}u(\boldsymbol{x}, g(\boldsymbol{x})) = g(\boldsymbol{x}) + {}^{(l)}\phi_i(x_l)\, {}^{(l)}\rho_i(\boldsymbol{x}, g(\boldsymbol{x})).$$

Substitute the univariate constrained expression for the l-th independent variable as the free function in the univariate constrained expression for the k-th independent variable,

$$^{(k)}u(\boldsymbol{x}, {}^{(l)}u(\boldsymbol{x}, g(\boldsymbol{x}))) = {}^{(l)}u(\boldsymbol{x}, g(\boldsymbol{x})) + {}^{(k)}\phi_j(x_k)\, {}^{(k)}\rho_j(\boldsymbol{x}, {}^{(l)}u(\boldsymbol{x}, g(\boldsymbol{x}))).$$

Clearly, from Theorem 1, which shows that a univariate constrained expression satisfies the constraints for any free function that is defined at the constraints, the constraints of the k-th independent variable must be satisfied, as $^{(l)}u(\boldsymbol{x}, g(\boldsymbol{x}))$ is a valid free function. Next, expand $^{(l)}u(\boldsymbol{x}, g(\boldsymbol{x}))$ and $^{(k)}\rho_j(\boldsymbol{x}, g(\boldsymbol{x}))$ and drop the $\boldsymbol{x}$ and $g(\boldsymbol{x})$ arguments for

34

clarity.

$$
\begin{aligned}
{}^{(k)}u &= g + {}^{(l)}\phi_i \, {}^{(l)}\rho_i + {}^{(k)}\phi_j \left({}^{(k)}\kappa_j - {}^{(k)}\mathfrak{C}_j[g] - {}^{(k)}\mathfrak{C}_j \left[{}^{(l)}\phi_i \, {}^{(l)}\rho_i \right] \right) \\
&= g + {}^{(l)}\phi_i \left({}^{(l)}\kappa_i - {}^{(l)}\mathfrak{C}_i[g] \right) + {}^{(k)}\phi_j \left({}^{(k)}\kappa_j - {}^{(k)}\mathfrak{C}_j[g] \right. \\
&\quad \left. - {}^{(l)}\phi_i \left({}^{(k)}\mathfrak{C}_j[{}^{(l)}\kappa_i] - {}^{(k)}\mathfrak{C}_j \left[{}^{(l)}\mathfrak{C}_i[g] \right] \right) \right),
\end{aligned}
$$

where Property 4 has been used to simplify the expression. Now, evaluate ${}^{(k)}u$ at the m-th constraint for the l-th independent variable,

$$
\begin{aligned}
{}^{(l)}\mathfrak{C}_m[{}^{(k)}u] &= {}^{(l)}\mathfrak{C}_m[g] + \delta_{mi} \left({}^{(l)}\kappa_i - {}^{(l)}\mathfrak{C}_i[g] \right) + {}^{(k)}\phi_j \left({}^{(l)}\mathfrak{C}_m[{}^{(k)}\kappa_j] - {}^{(l)}\mathfrak{C}_m \left[{}^{(k)}\mathfrak{C}_j[g] \right] \right. \\
&\quad \left. - \delta_{mi} \left({}^{(k)}\mathfrak{C}_j[{}^{(l)}\kappa_i] - {}^{(k)}\mathfrak{C}_j \left[{}^{(l)}\mathfrak{C}_i[g] \right] \right) \right) \\
&= {}^{(l)}\kappa_m \\
&\quad + {}^{(k)}\phi_j \left({}^{(l)}\mathfrak{C}_m[{}^{(k)}\kappa_j] - {}^{(k)}\mathfrak{C}_j[{}^{(l)}\kappa_m] - {}^{(l)}\mathfrak{C}_m \left[{}^{(k)}\mathfrak{C}_j[g] \right] + {}^{(k)}\mathfrak{C}_j \left[{}^{(l)}\mathfrak{C}_m[g] \right] \right) \\
&= {}^{(l)}\kappa_m.
\end{aligned}
$$

Therefore, ${}^{(l)}\mathfrak{C}_m[{}^{(k)}u] = {}^{(l)}\kappa_m$ as required, and the expression ${}^{(k)}u$ satisfies both sets of original univariate constraints. ∎

Example 2.6 demonstrates Theorem 5.

Example 2.6: Multivariate non-integral constraints

Consider the following set of constraints,

$$
u(0, y) = y^2 \sin(\pi y), \quad u(1, y) + u(2, y) = y \sin(\pi y),
$$

$$
u_y(x, 0) = 0, \quad \text{and} \quad u(x, 0) = u(x, 1).
$$

The univariate constrained expressions for the constraints on x and y are,

$$^{(1)}u(x,y,g(x,y)) = g(x,y) + \frac{3-2x}{3}\left(y^2\sin(\pi y) - g(0,y)\right)$$
$$+ \frac{x}{3}\left(\cos(\pi y) - g(2,y) - g(1,y)\right)$$
$$^{(2)}u(x,y,g(x,y)) = g(x,y) - (y-y^2)g_y(x,0) - y^2\left(g(x,1) - g(x,0)\right).$$

Then, $^{(1)}u$ is used as the free function in $^{(2)}u$,

$$^{(2)}u(x,y,{}^{(1)}u(x,y,g(x,y))) = {}^{(1)}u(x,y,g(x,y)) - (y-y^2)\,{}^{(1)}u_y(x,0,g(x,y))$$
$$- y^2\left({}^{(1)}u(x,1,g(x,y)) - {}^{(1)}u(x,0,g(x,y))\right).$$

Substituting in $^{(1)}u$ and simplifying yields,

$$u(x,y,g(x,y)) = g(x,y) + \left(y-y^2\right)\left(\frac{3-2x}{3}g_y(0,0) - \frac{x}{3}(-g_y(1,0) - g_y(2,0))\right.$$
$$\left. - g_y(x,0)\right) - y^2\left(\frac{3-2x}{3}g(0,0) - \frac{3-2x}{3}g(0,1) - \frac{x}{3}(-g(1,0) - g(2,0))\right.$$
$$\left. + \frac{x}{3}(-g(1,1) - g(2,1)) - g(x,0) + g(x,1)\right) + \frac{3-2x}{3}\left(y^2\sin(\pi y) - g(0,y)\right)$$
$$+ \frac{x}{3}\left(-g(1,y) - g(2,y) + y\sin(\pi y)\right).$$
$$(2.6)$$

Note that substituting $^{(2)}u$ as the free function in $^{(1)}u$, after simplifying, yields the same result given in Equation (2.6). Equation (2.6) satisfies the constraints for any $g(x,y)$ satisfying Definition 9. Figure 2.3 shows the constrained expression when $g(x,y) = x^2\cos y + \sin(2x)$, where the blue line signifies the constraint on $u(0,y)$, the black lines signify the derivative constraint on $u_y(x,0)$, and the magenta lines signify the relative constraint $u(x,0) = u(x,1)$. The linear constraint $u(1,y) + u(2,y) = y\sin(\pi y)$ is not easily visualized but is nonetheless satisfied by the constrained expression.
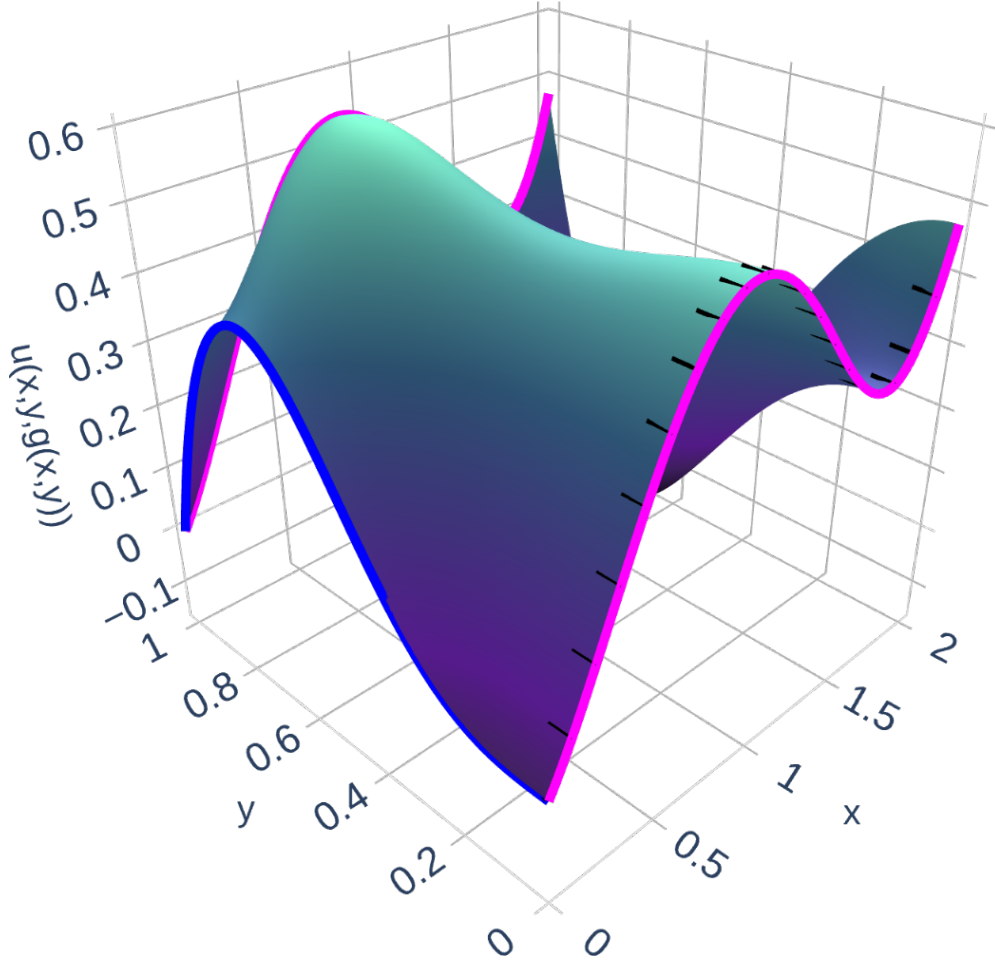
Figure 2.3: Constrained expression evaluated using $g(x,y) = x^2 \cos y + \sin(2x)$. The blue line signifies the constraint on $u(0,y)$, the black lines signify the derivative constraint on $u_y(x,0)$, and the magenta lines signify the relative constraint $u(x,0) = u(x,1)$. The linear constraint $u(1,y) + u(2,y) = y\sin(\pi y)$ is not easily visualized but is nonetheless satisfied by the constrained expression.

2.3.1.1 Integral Constraints

Theorem 5 is proven using Property 4 applied to expressions such as ${}^{(l)}\mathfrak{C}_i\left[{}^{(k)}\phi_j\,{}^{(k)}\mathfrak{C}_j[g]\right] = {}^{(k)}\phi_j\,{}^{(l)}\mathfrak{C}_i\left[{}^{(k)}\mathfrak{C}_j[g]\right]$. These expressions are true so long as the constraint operator of the l-th independent variable does not affect functions that do not contain l. This is true for all constraint types introduced thus far, except integral constraints. Integral constraints may

have constraint operators like,

$$^{(l)}\mathfrak{C}_i[f(x_1, \ldots, x_l, \ldots, x_n)] = \int_a^b f(x_1, \ldots, c, \ldots, x_n) \, \mathrm{d}x_k,$$

where $a, b, c \in \mathbb{R}$, which affect functions of the k-th independent variable, even though the constraint operator is for the l-th independent variable. Although this type of constraint is rare in PDEs, in the interest of introducing a general function interpolation technique, the next section presents a method to embed integral constraints into multivariate constrained expressions.

The interference between independent variables introduced by integral constraints can be avoided by modifying the switching functions and processing order of the univariate constrained expressions.

Theorem 6

Processing the independent variables that appear as integration variables in integral constraints after the independent variables associated with the integral constraints and modifying the switching functions of all constraints of the variables of integration such that they yield zero when operated on by the constraint operators of said integral constraints is a valid method for embedding integral constraints into multivariate constrained expressions.

Proof: Let the constraints of the l-th independent variable contain an integral constraint whose variable of integration is the k-th independent variable. Moreover, as per the theorem statement, let the switching functions of the k-th independent variable be defined such that $^{(l)}\mathfrak{C}_m[^{(k)}\phi_j] = 0$ if the m-th constraint is the integral constraint. Now, following the recursive method, the bivariate constrained expression for the k-th

and l-th independent variables is,

$$^{(k)}u(\boldsymbol{x}, {}^{(l)}u(\boldsymbol{x}, g(\boldsymbol{x}))) = {}^{(l)}u(\boldsymbol{x}, g(\boldsymbol{x})) + {}^{(k)}\phi_j(x_k) \, {}^{(k)}\rho_j(\boldsymbol{x}, {}^{(l)}u(\boldsymbol{x}, g(\boldsymbol{x}))).$$

From Theorem 5, the constraints of the k-th independent variable and non-integral constraints of the l-th independent variable are satisfied. As in Theorem 5, expand the expression for $^{(k)}u$ and drop the $\boldsymbol{x}$ and $g(\boldsymbol{x})$ arguments for clarity,

$$
\begin{aligned}
^{(k)}u = {}&g + {}^{(l)}\phi_i\left({}^{(l)}\kappa_i - {}^{(l)}\mathfrak{C}_i[g] \right) \\
&+ {}^{(k)}\phi_j\left({}^{(k)}\kappa_j - {}^{(k)}\mathfrak{C}_j[g] - {}^{(l)}\phi_i\left({}^{(k)}\mathfrak{C}_j[{}^{(l)}\kappa_i] - {}^{(k)}\mathfrak{C}_j\left[{}^{(l)}\mathfrak{C}_i[g] \right] \right) \right).
\end{aligned}
$$

Apply the constraint operator for the integral constraint,

$$
\begin{aligned}
^{(l)}\mathfrak{C}_m[{}^{(k)}u] = {}&{}^{(l)}\mathfrak{C}_m[g] + \delta_{mi}\left({}^{(l)}\kappa_i - {}^{(l)}\mathfrak{C}_i[g] \right) \\
&+ {}^{(l)}\mathfrak{C}_m\left[{}^{(k)}\phi_j \underbrace{\left({}^{(k)}\kappa_j - {}^{(k)}\mathfrak{C}_j[g] - {}^{(l)}\phi_i\left({}^{(k)}\mathfrak{C}_j[{}^{(l)}\kappa_i] - {}^{(k)}\mathfrak{C}_j\left[{}^{(l)}\mathfrak{C}_i[g] \right] \right) \right)}_{\text{Not a function of the } k\text{-th independent variable}} \right].
\end{aligned}
$$

As noted in the above expression, the function highlighted by the underbrace is not a function of the k-independent variable; hence, the integration portion of the $^{(l)}\mathfrak{C}_m$ constraint operator only acts on $^{(k)}\phi_j$. Moreover, recall that this method redefines the switching functions such that $^{(l)}\mathfrak{C}_m[{}^{(k)}\phi_j] = 0$. Thus,

$$
\begin{aligned}
^{(l)}\mathfrak{C}_m[{}^{(k)}u] &= {}^{(l)}\mathfrak{C}_m[g] + \delta_{mi}\left({}^{(l)}\kappa_i - {}^{(l)}\mathfrak{C}_i[g] \right) \\
&= {}^{(l)}\mathfrak{C}_m[g] + {}^{(l)}\kappa_m - {}^{(l)}\mathfrak{C}_m[g] \\
&= \kappa_m,
\end{aligned}
$$

as desired. Therefore, all constraints, integral and non-integral, on both the k-th and

l-th independent variables are satisfied. Applying this proof recursively shows that this is a valid method for constructing multivariate constrained expressions that contain integral constraints. ∎

Example 2.7 provides a concrete demonstration of Theorem 6.

Example 2.7: Multivariate integral constraints

Consider the following set of constraints,

$$u(x,0) = 2u_y(x,1), \quad u(x,2) = \sin(x), \quad \text{and} \quad \int_{-1}^{1} u(2,y)\, \mathrm{d}y = 5.$$

Based on the previous discussion, the x independent variable will be processed first, because it has an integral constraint with integration variable y, and the switching functions for the constraints in y must be created such that they are equal to zero when evaluated with the constraint operator for the integral constraint. That is, the equations for the y switching functions are,

$$^{(2)}\phi_1(0) - 2\frac{\partial\,^{(2)}\phi_1}{\partial y}(1) = 1, \qquad ^{(2)}\phi_1(2) = 0, \qquad \int_{-1}^{1} {}^{(2)}\phi_1(y)\, \mathrm{d}y = 0,$$

$$^{(2)}\phi_2(0) - 2\frac{\partial\,^{(2)}\phi_2}{\partial y}(1) = 0, \qquad ^{(2)}\phi_2(2) = 1, \qquad \int_{-1}^{1} {}^{(2)}\phi_2(y)\, \mathrm{d}y = 0.$$

Even though there are only two constraints in the y-dimension, each switching function must satisfy three sets of equations. Therefore, each switching function should be a linear combination of three linearly independent support functions with unknown coefficients, $^{(2)}\phi_i(y) = \alpha_{ij}s_j(y)$ where $i \in \{1,2\}$ and $j \in \{1,2,3\}$. As before, the

equations can be written in a compact matrix form and solved via matrix inversion.

$$
\begin{bmatrix} 1 & -2 & -4 \\ 1 & 2 & 4 \\ 2 & 0 & \frac{2}{3} \end{bmatrix} \begin{bmatrix} \alpha_{11} & \alpha_{12} \\ \alpha_{21} & \alpha_{22} \\ \alpha_{31} & \alpha_{32} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix}
$$

$$
\begin{bmatrix} \alpha_{11} & \alpha_{12} \\ \alpha_{21} & \alpha_{22} \\ \alpha_{31} & \alpha_{32} \end{bmatrix} = \begin{bmatrix} \frac{1}{2} & \frac{1}{2} \\ \frac{11}{4} & \frac{13}{4} \\ -\frac{3}{2} & -\frac{3}{2} \end{bmatrix}
$$

Next, the univariate constrained expressions for each of the independent variables can be written as,

$$
{}^{(1)}u(x, y, g(x, y)) = g(x, y) + \frac{1}{2}\left(5 - \int_{-1}^{1} g(2, \tau)\, d\tau\right),
$$

$$
{}^{(2)}u(x, y, g(x, y)) = g(x, y) + \frac{2 + 11y - 6y^2}{4}\left(2g_y(x, 1) - g(x, 0)\right)
$$
$$
+ \frac{2 + 13y - 6y^2}{4}\left(\sin(x) - g(x, 2)\right),
$$

and following the method outlined earlier, the full multivariate constrained expression can be written as,

$$
\begin{aligned}
u(x, y, g(x, y)) &= {}^{(2)}u(x, y, {}^{(1)}u(x, y, g(x, y))) \\
&= g(x, y) + \frac{1}{4}\Big[2(2 - y)\left((6y + 1)g_y(x, 1) + 3y\left(\int_{-1}^{1} g(2, \tau)\, d\tau - 5\right)\right) \\
&\quad + (y - 2)(6y + 1)g(x, 0) + \left(y(6y - 13) - 2\right)g(x, 2) \\
&\quad + (y(13 - 6y) + 2)\sin(x)\Big].
\end{aligned} \tag{2.7}
$$

As expected, Equation (2.7) satisfies the constraints for any valid free function $g(x, y)$.

It is important to note that this method cannot embed sets of integral constraints whose

41

independent variables refer to one another, such as,

$$\int_0^1 u(x,0) \, \mathrm{d}x = 1 \quad \text{and} \quad \int_0^1 u(0,y) \, \mathrm{d}y = 1.$$

The reason is that the first integral constraint requires that the y independent variable be processed before x, but the second integral constraint requires that the x independent variable be processed before y: obviously, these two requirements cannot be satisfied simultaneously. Therefore, this method cannot embed such constraints.

2.3.1.2 Component Constraints

As in the univariate case, one must choose which dependent variable a component constraint will be assigned to. Again, graph theory can be used in the same manner as before to determine all possible ways in which a set of component constraints can be embedded, see Example 2.4. However, when moving to the multivariate case, one must be cautious of the intersections between component constraints and other constraints: Example 2.8 highlights this nuance.

Example 2.8: Multivariate component constraints

Consider the following set of constraints,

$$u(x,0) = 5 \quad \text{and} \quad u(0,y) + v(0,y) = 3.$$

If one chose to embed the component constraint into u and process the constraints on x first, then the constrained expressions would be,

$$u(x,y,g^u(x,y),g^v(x,y)) = g^u(x,y) + 5 - g^u(x,0) - g^u(0,y) - v(0,y,g^v(x,y))$$
$$+ v(0,0,g^v(x,y)) + g^u(0,0)$$
$$v(x,y,g^v(x,y)) = g^v(x,y).$$

Clearly, the constrained expression for u does not satisfy the two constraints for any valid free function, e.g., choosing $g^u(x, y) = 3$ and $g^v(x, y) = 2$ yields $u(0, y, 3) + v(0, y, 2) = 7 \neq 3$; the reason stems from the intersection between the two constraints.[a] At the intersection, u must be equal to 5, but simultaneously be equal to $3 - v(0, y, g^v(0, y))$. If $g^v(x, y)$ was chosen in such a way that $g^v(0, 0) = -2$, then all constraints would be satisfied, but of course the objective of constrained expressions is to provide a functional that satisfies the constraints wherein the free function can be chosen without restriction. In other words, since at the intersection of the constraints, $(x, y) = (0, 0)$, $u = 5$ as specified by the first constraint, the only way to simultaneously satisfy the component constraint is to change v; hence, the component constraint must be placed on v. Doing so results in the constrained expressions,

$$u(x, y, g^u(x, y)) = g^u(x, y) + 5 - g^u(x, 0)$$

$$v(x, y, g^v(x, y), g^u(x, y)) = g^v(x, y) + 3 - g^v(0, y) - u(0, y, g^u(x, y)).$$

which satisfy the constraints for any valid $g^u(x, y)$ and $g^v(x, y)$.

---

[a]Although this example only shows the intersection issue when processing the constraints on x first, the same issue arises even if the constraints on y are processed first.

As demonstrated in Example 2.8, component constraints must only be placed on dependent variables that do not have other constraints that intersect with the component constraint. Of course, if each variable in the component constraint has a constraint at the intersection point, then the component constraint can be placed on either variable. For example, if the constraints in Example 2.8 were,

$$u(x, 0) = 5, \quad v(x, 0) = -2, \quad \text{and} \quad u(0, y) + v(0, y) = 3,$$

then the component constraint could have been embedded into either u or v. Therefore,

43

while the graph theory introduced in the univariate section can be used to determine component constraint embeddings that avoid infinite recursions when evaluating the constrained expressions, it is up to the user to further reduce this set of graphs to those that avoid the intersection issues described above.

After considering the previous restrictions on component constraints, one may contrive a set of equations where it is impossible to meet the aforementioned conditions. For example, consider the following constraints,

$$u(0, y) = 5, \quad v(1, y) = 2, \quad \text{and} \quad u(x, 0) + v(x, 0) = 3.$$

The component constraint cannot be placed on u because of the intersection at $(0, 0)$, but it also cannot be placed on v because of the intersection at $(1, 0)$. The only option here is to split the domain along the x-axis for some $x_{\text{split}} \in (0, 1)$. Then, in the left sub-domain, $x < x_{\text{split}}$, the component constraint will be embedded into v, and in the right sub-domain, $x > x_{\text{split}}$, the component constraint will be embedded into u. At the intersection of these two sub-domains, $x = x_{\text{split}}$, one can enforce C^n continuity—n is chosen by the user or dictated by the problem—by adding constraints at the intersection that can ultimately be embedded into the constrained expressions. For more information and an example on splitting the domain, see Appendix D.

2.3.1.3 Linear Constraints

Multivariate linear constraints consist of linear combinations of the previously introduced constraint types. Thus, one must be conscientious of the nuances of both integral and component constraints if they appear in the linear constraints. The following step-by-step procedure can be used to construct multivariate constrained expressions:

1. Generate the directed, acyclic graphs that show all the valid ways that the component constraints can be embedded. Of these graphs, either choose one that satisfies the intersection restriction discussed in the previous section or choose one and split the

domain as needed. The chosen graph will dictate the order in which the dependent variables' constrained expressions are created.

2. For each dependent variable, choose the order in which the univariate constrained expressions will be processed. This order is dictated in part by the presence of integral constraints.

3. Build the multivariate constrained expressions.

This step-by-step procedure can be used for any embeddable set of constraints. Of course, steps in this procedure may be omitted depending on the types of constraints present. For example, if the set does not contain any component constraints, then there is no need to perform step 1, and the dependent variables' constrained expressions can be created in any order. Example 2.9 demonstrates this process.

Example 2.9: Multivariate linear constraints

Consider the following set of constraints,

$$u(0, y) = \cos(\pi y), \quad \int_{-1}^{2} u(1, y) \, \mathrm{d}y = e, \quad u(x, 1) - u(x, 2) = -2,$$

$$u(x, 0) + v(x, 0) = 5, \quad \text{and} \quad v(0, y) = 5 - \cos(\pi y).$$

As per the procedure outlined above, step 1 is to generate the directed graphs that dictate the valid ways in which the component constraint can be embedded. In this example, those graphs are trivial: the single component constraint can be embedded into either u or v without producing a set of constrained expressions that require an infinite recursion upon evaluation. However, u contains an integral constraint along the $x = 1$ line, and there is no corresponding constraint at $x = 1$ in v. Hence, the component constraint is embedded into the v constrained expression, and the intersection issue is avoided. It follows that the u constrained expression will be

45

created before the v constrained expression.

Next, the processing order for the independent variables must be decided. For v, the order does not matter as no integral constraints are present. In contrast, for u, the constraints on x must be processed before those on y, as the constraints on x contain an integral constraint wherein y is an integration variable.

Now the constrained expressions can be created. First the u constrained expression is created. The univariate constrained expressions for the constraints on x and y are,

$$^{(1)}u(x, y, g^u(x, y)) = g^u(x, y)) + (1 - x)\left(\cos(\pi y) - g^u(0, y)\right) + \frac{x}{3}\left(e - \int_{-1}^{2} g^u(1, \tau)\, \mathrm{d}\tau\right)$$

$$^{(2)}u(x, y, g^u(x, y)) = g^u(x, y) + \frac{1 - 2y}{2}\left(g^u(x, 2) - g^u(x, 1) - 2\right),$$

where monomials have been used as the support functions. These two constrained expressions are used in the processing order defined above to produce the multivariate constrained expression for u,

$$u(x, y, g^u(x, y)) = g^u(x, y) + \frac{1}{3}x\left(e - \int_{-1}^{2} g^u(1, \tau)\, \mathrm{d}\tau\right) + \frac{1 - 2y}{2}\big((1 - x)(1 + g^u(0, 1))$$

$$+ (1 - x)(1 - g^u(0, 2)) - g^u(x, 1) + g^u(x, 2) - 2\big) + (1 - x)(\cos(\pi y) - g^u(0, y)).$$

For v, the univariate constrained expressions for the constraints on x and y are,

$$^{(1)}v(x, y, g^v(x, y)) = g^v(x, y) + 5 - \cos(\pi y) - g^v(0, y)$$

$$^{(2)}v(x, y, g^v(x, y), g^u(x, y)) = g^v(x, y) + 5 - g^v(x, 0) - u(x, 0, g^u(x, y)),$$

where monomials have been used as the support functions. The full multivariate

constrained expression for v is,

$$v(x, y, g^v(x, y), g^u(x, y)) = g^v(x, y) - g^v(x, 0) - g^v(0, y) + g^v(0, 0) - u(x, 0, g^u(x, y))$$

$$+ u(0, 0, g^u(x, y)) - \cos(\pi y) + 5.$$

The constrained expressions satisfy the constraints for any valid $g^u(x, y)$ and $g^v(x, y)$. Figure 2.4 shows the constrained expressions for u and v evaluated using $g^u(x, y) = xy + \sin(x) + y^2$ and $g^v(x, y) = x^2 y \cos(y) e^x$. The value constraints that are easily visualized, the constraints on $u(x, 0)$ and $v(x, 0)$, are shown as black lines. The remaining constraints are not easily visualized and are therefore not shown, but they are satisfied nonetheless.

(a) $u(x, y, g^u(x, y))$ (b) $v(x, y, g^v(x, y), g^u(x, y))$
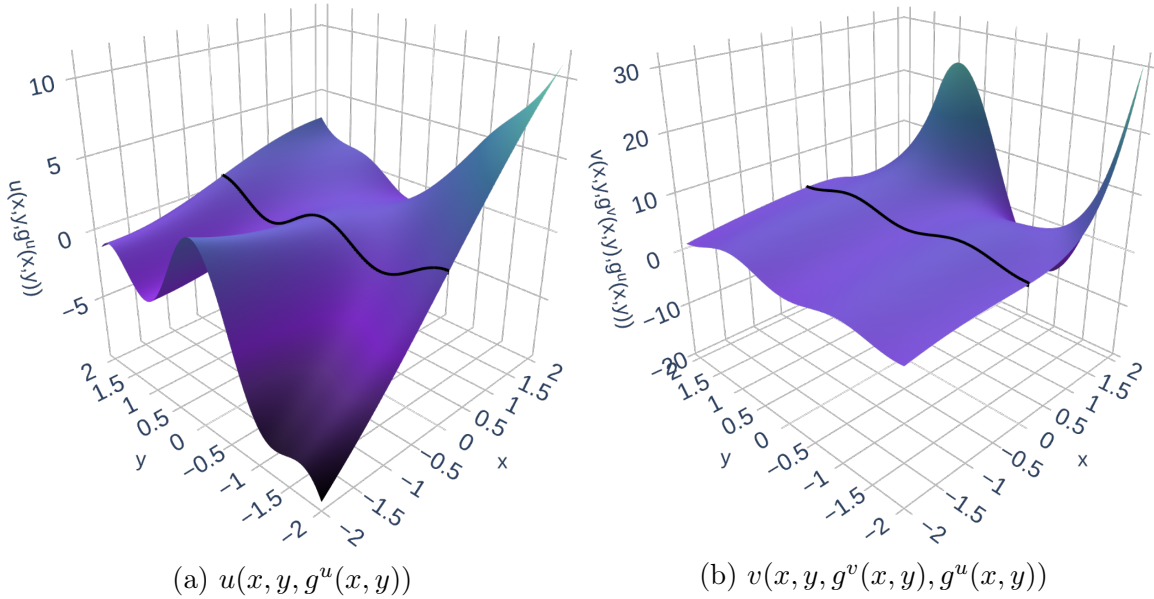
Figure 2.4: Constrained expressions evaluated using $g^u(x, y) = xy + \sin(x) + y^2$ and $g^v(x, y) = x^2 y \cos(y) e^x$. The value constraints on $u(x, 0)$ and $v(x, 0)$ are shown as black lines. The remaining constraints are not easily visualized and are therefore not shown but are satisfied nonetheless.

2.3.2 Multivariate Constrained Expression Theorems

This section introduces theorems for multivariate constrained expressions that parallel the theorems for univariate constrained expressions given in Section 2.2.4. Note that Theorems 5 and 6 for multivariate constrained expressions given earlier parallel Theorem 1 for univariate constrained expressions, and will not be repeated here.

Theorem 7 shows that the constrained expression functional represents the family of all possible functions that satisfy the constraints.

Theorem 7

For any function satisfying the constraints, $f(\boldsymbol{x})\colon \mathbb{R}^n \mapsto \mathbb{R}$, there exists at least one free function, $g(\boldsymbol{x})$, such that the constrained expression $u(\boldsymbol{x}, g(\boldsymbol{x})) = f(\boldsymbol{x})$. In other words, constrained expressions are surjective functionals whose domain is all free functions and whose codomain is all functions that satisfy the constraints.

Proof: Note that the processing order used in this proof is chosen arbitrarily, and could be modified to use the processing order required for any set of constraints and still produce the same final result. Based on Theorem 2, the univariate constrained expression will return the free function if the free function satisfies the constraints. Let $^{(1)}u(\boldsymbol{x}, g(\boldsymbol{x}))$ represent the univariate constrained expression for the independent variable x_1 that uses the free function $g(\boldsymbol{x})$, $^{(2)}u(\boldsymbol{x}, {}^{(1)}u(\boldsymbol{x}, g(\boldsymbol{x})))$ represent the univariate constrained expression for the independent variable x_2 that uses the free function $^{(1)}u(\boldsymbol{x}, g(\boldsymbol{x}))$, and so on up to $^{(n)}u(\boldsymbol{x}, {}^{(n-1)}u(\boldsymbol{x}, g(\boldsymbol{x})))$, which is simply the constrained expression $u(\boldsymbol{x}, g(\boldsymbol{x}))$. If one chooses $g(\boldsymbol{x}) = f(\boldsymbol{x})$, then based on Theorem 2 $^{(1)}u(\boldsymbol{x}, f(\boldsymbol{x})) = f(\boldsymbol{x})$. Applying Theorem 2 recursively leads to $^{(2)}u(\boldsymbol{x}, {}^{(1)}u(\boldsymbol{x}, g(\boldsymbol{x}))) = f(\boldsymbol{x})$ and so on until $u(\boldsymbol{x}, f(\boldsymbol{x})) = f(\boldsymbol{x})$. Hence, for any function satisfying the constraints, $f(\boldsymbol{x})$, there exists a free function, $g(\boldsymbol{x}) = f(\boldsymbol{x})$, such that the multivariate constrained expression is equal to the function satisfying the

constraints, i.e., $u(\boldsymbol{x}, f(\boldsymbol{x})) = f(\boldsymbol{x})$. ∎

Based on the univariate constrained expression theorems, one is inclined to guess that the free function obtained in the previous theorem is not unique. As Theorem 8 shows, this inclination is correct.

Theorem 8

For a given function satisfying the constraints, $f(\boldsymbol{x})\colon \mathbb{R}^n \mapsto \mathbb{R}$, the free function, $g(\boldsymbol{x})$, in the constrained expression $u(\boldsymbol{x}, g(\boldsymbol{x})) = f(\boldsymbol{x})$ is not unique. In other words, constrained expressions are not injective functionals over the domain of all free functions and codomain of all functions that satisfy the constraints.

Proof: Since each expression $^{(i)}u(\boldsymbol{x}, g(\boldsymbol{x}))$ used in deriving the multivariate constrained expression is derived through the univariate formulation, the results of the proof of Theorem 3 apply for each each $^{(i)}u(\boldsymbol{x}, g(\boldsymbol{x}))$, and therefore, the free function $g(\boldsymbol{x})$ is not unique. ∎

Like in the univariate case, this proof has immediate implications when using the constrained expression for optimization. Through the recursive application of the univariate TFC approach, for cases with no integral constraints, any terms in $g(\boldsymbol{x})$ that are linearly dependent to the the support functions, $s_i(x_1)$, $s_j(x_2)$, ... , $s_k(x_n)$, will not contribute to the solution. In the multivariate case, this also includes products of the support functions that include one and exactly one support function from each independent variable, e.g., $s_i(x_1)s_j(x_2)...s_k(x_n)$. For example, suppose the support functions $s_i(x) = \{1, x, x^2\}$ and $s_j(y) = \{y, y^2\}$ were used when deriving a multivariate constrained expression. Then, any terms in the free function linearly dependent to any of the following functions $\{1, x, x^2, y, y^2, xy, x^2y, xy^2, x^2y^2\}$ can be removed, as they do not affect the output of the constrained expression.

49

Integral constraints may add a slight complication, as they change the number of support functions used. In general, when using integral constraints in the multivariate case, the expression,

$$\alpha_{ki}S_{ij} = \delta_{jk},$$

is no longer true. Thus, some functions that are linearly dependent to the support functions may still be significant when included in the free function. Example 2.10 shows how to determine which functions linearly dependent to the support functions still have a significant contribution when included in the free function using the constraints from Example 2.7.

Example 2.10: Integral constraint linear dependence

Consider the constraints from Example 2.7, copied below for the reader's convenience:

$$u(x,0) = 2u_y(x,1), \quad u(x,2) = \sin(x), \quad \text{and} \quad \int_{-1}^{1} u(2,y) \, \mathrm{d}y = 5.$$

In Example 2.7, the two constrained expressions were derived as,

$$^{(1)}u(x,y,g(x,y)) = g(x,y) + \frac{2+11y-6y^2}{4}\Big(2g_y(x,1) - g(x,0)\Big)$$
$$+ \frac{2+13y-6y^2}{4}\Big(\sin(x) - g(x,2)\Big)$$
$$^{(2)}u(x,y,g(x,y)) = g(x,y) + \frac{1}{2}\Big(5 - \int_{-1}^{1} g(2,\tau) \, \mathrm{d}\tau\Big),$$

using the support functions $s_1(x) = 1$ for x and $s_1(y) = 1$, $s_2(y) = y$, and $s_3(y) = y^2$ for y. Theorem 3 applies without modification to $^{(2)}u$, as $^{(2)}u$ was created using the regular univariate theory. In contrast, the switching functions of $^{(1)}u$ were modified to include $\int_{-1}^{1} {}^{(y)}\phi_k(y) \, \mathrm{d}y = 0$ for $k \in \{1,2\}$. As mentioned earlier, this means that Theorem 3 must be modified slightly as $\alpha_{ki}S_{ij} \neq \delta_{jk}$ in this case.

For the reader's convenience, the last few lines of Theorem 3 have been copied

below:

$$y(x) = f(x) + \beta_j \left(\delta_{jk} - \alpha_{ki} \, \mathbb{S}_{ij} \right) s_k(x)$$

$$y(x) = f(x) + \beta_j \left(\delta_{jk} - \delta_{jk} \right) s_k(x)$$

$$y(x) = f(x).$$

For the constraints given in this example, the last few lines are re-derived. Let B_{jk} be defined by

$$B_{jk} = \alpha_{ki} \mathbb{S}_{ij} = \begin{bmatrix} 1 & 1 \\ -2 & 2 \\ -4 & 4 \end{bmatrix} \begin{bmatrix} \frac{1}{2} & \frac{11}{4} & -\frac{3}{2} \\ \frac{1}{2} & \frac{13}{4} & -\frac{3}{2} \end{bmatrix} = \begin{bmatrix} 1 & 6 & -3 \\ 0 & 1 & 0 \\ 0 & 2 & 0 \end{bmatrix}.$$

Suppose that $\beta_j = \left\{ a, \ b, \ c \right\}$, then,

$$\beta_j \left(\delta_{jk} - \alpha_{ki} \mathbb{S}_{ij} \right) s_k(x) = \beta_j \left(\delta_{jk} - B_{jk} \right) s_k(x)$$

$$= \left\{ a, \ b, \ c \right\} \left(\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} - \begin{bmatrix} 1 & 6 & -3 \\ 0 & 1 & 0 \\ 0 & 2 & 0 \end{bmatrix} \right) \left\{ \begin{matrix} 1 \\ y \\ y^2 \end{matrix} \right\}$$

$$= \left\{ a, \ b, \ c \right\} \left(\begin{bmatrix} 0 & -6 & 3 \\ 0 & 0 & 0 \\ 0 & -2 & 1 \end{bmatrix} \right) \left\{ \begin{matrix} 1 \\ y \\ y^2 \end{matrix} \right\}$$

$$= 3ay^2 - 6ay + cy^2 - 2cy.$$

Hence, only the constants a and c affect the final results. However, notice that the first and third row of $\delta_{jk} - B_{jk}$ are linearly dependent. Consequently, the effect of a

and c on the final solution differs only by a constant. Therefore, one concludes that in this case, any functions linearly dependent to y do not affect the final solution, and the effect of any functions linearly dependent to 1 on the final solution will be linearly dependent with the effect of any functions linearly dependent to y^2 on the final solution.

Consequently, one can remove any functions linearly dependent to two different functions, either 1 and y, or y and y^2, from the free function $g(x, y)$ without changing the final result. This can be interpreted intuitively, as the number of functions to be removed from $g(x, y)$ due to constraints on y matches the number of constraints on y: two. Moreover, because this is a multivariate case, one can also remove any products of the support functions that include one and exactly one support function from each independent variable. However, in this case, the only support function used for the x constrained expression is 1. Therefore, this does not contribute to the terms to be removed from $g(x, y)$.

As Example 2.10 shows, one must examine the matrix $\delta_{jk} - \alpha_{ki}\mathbb{S}_{ij}$ to calculate which terms linearly dependent to the support functions contribute to the non-uniqueness of the free function. In this matrix, a row of all zeros indicates that the corresponding support function does not contribute to the final result; for the optimization process, this corresponds to removing terms linearly dependent to that support function from the free function. Linear dependence between rows of the matrix indicates that the effect of the corresponding support functions on the final result differs only by a constant; for the optimization process, this corresponds to removing the terms in the free function linearly dependent to one of the support functions associated with the linearly dependent rows in the matrix.

As in the univariate case, the multivariate constrained expressions can be shown to be projection functionals: this is done in Theorem 9.

Theorem 9

The multivariate constrained expression is a projection functional.

Proof: To prove Theorem 9, one must show that $u(\boldsymbol{x}, u(\boldsymbol{x}, g(\boldsymbol{x}))) = u(\boldsymbol{x}, g(\boldsymbol{x}))$. Theorems 5 and 6 show that constrained expression returns a function that satisfies the constraints. In other words, for any $g(\boldsymbol{x})$, $u(\boldsymbol{x}, g(\boldsymbol{x}))$ is a function that satisfies the constraints. From Theorem 7, if the free function used in the constrained expression satisfies the constraints, then the constrained expression returns that free function exactly. Hence, if the constrained expression function is given itself as the free function, it will simply return itself. ∎

In addition, just as in the univariate case, Theorems 7, 8, and 9 allow for a more rigorous definition of the multivariate constrained expression. The multivariate constrained expression is a surjective, projection functional whose domain is the set of all free functions and whose codomain is the set of all functions that satisfy the constraints.

2.3.3 Tensor Form

Recursive applications of univariate TFC lead to constrained expressions that lend themselves nicely to mathematical proofs, such as those in the previous section. However, at times it may be more convenient to express the constrained expression in a more compact form. Conveniently, multivariate constrained expressions that are formed from recursive applications of univariate TFC can be succinctly expressed in the following tensor form,

$$u(\boldsymbol{x}) = g(\boldsymbol{x}) + \mathcal{M}(\rho(\boldsymbol{x}, g(\boldsymbol{x}))_{i_1 i_2 \ldots i_n} \Phi_{i_1}(x_1) \Phi_{i_2}(x_2) \ldots \Phi_{i_n}(x_n)$$

where $i_1, i_2, \ldots, i_n$ are n indices associated with the n-dimensions that have constraints, $\mathcal{M}$ is an n-dimensional tensor whose elements are based on the projection functionals, $\rho(\boldsymbol{x}, g(\boldsymbol{x}))$, and the n vectors Φ_{i_k} are vectors whose elements are based on the switching functions for

the associated dimension.

The $\mathcal{M}$ tensor can be constructed using a simple two-step process. Note that the arguments of functionals are dropped in this explanation for clarity.

1. The elements of the first order sub-tensors of $\mathcal{M}$ acquired by setting all but one index equal to one are a zero followed by the projection functionals for the dimension associated with that index. Mathematically,

$$\mathcal{M}_{1...i_k...1} = \left\{ 0, \quad {}^{(k)}\rho_1, \quad \cdots, \quad {}^{(k)}\rho_{\ell_k} \right\},$$

where ${}^{(k)}\rho_j$ indicates the j-th projection functional of the k-independent variable and ℓ_k is the number of constraints associated with the k-th independent variable.

2. The remaining elements of the $\mathcal{M}$ tensor, those that have more than one index not equal to one, are the geometric intersection of the associated projection functionals multiplied by a sign ($-$ or $+$). Mathematically, this can be written as,

$$\mathcal{M}_{i_1 i_2 ... i_n} = {}^{(j)}\mathfrak{C}_{i_j-1} \left[{}^{(k)}\mathfrak{C}_{i_k-1} \left[\cdots \left[{}^{(h)}\rho_{i_h-1} \right] \cdots \right] \right] (-1)^{m+1}, \tag{2.8}$$

where i_j, i_k, $\ldots$, i_h are the indices of $\mathcal{M}_{i_1 i_2 ... i_n}$ that are not equal to one and m is equal to the number of non-one indices. If no integral constraints are present, i.e., the processing order of the independent variables does not matter, then by multiple applications of Clairaut's Theorem the variables associated with the constraint operators and projection functional in Equation (2.8) can be freely permuted [1, 15]. For example, if no integral constraints are present, then Equation (2.8) could be re-written as,

$$\mathcal{M}_{i_1 i_2 ... i_n} = {}^{(h)}\mathfrak{C}_{i_h-1} \left[{}^{(j)}\mathfrak{C}_{i_j-1} \left[\cdots \left[{}^{(k)}\rho_{i_k-1} \right] \cdots \right] \right] (-1)^{m+1}.$$

If integral constraints are present, then the processing order of the associated elements of the $\mathcal{M}$ tensor must match the processing order used in the recursive formulation.

The elements of the vectors Φ_{i_k} are composed of a 1 followed by the switching functions associated with the k-th independent variable. Mathematically,

$$\Phi_{i_k} = \left\{ 1, \quad {}^{(k)}\phi_1, \quad \cdots, \quad {}^{(k)}\phi_{\ell_k} \right\},$$

where ${}^{(k)}\phi_j$ denotes the j-th switching function of the k-th independent variable.

To solidify the reader's understanding of the tensor form explained above, some of the previous examples' constrained expressions are re-derived below in Examples 2.11, 2.12, and 2.13.

Example 2.11: Non-integral constraints in tensor form

Consider the constraints from Example 2.6

$$u(0, y) = y^2 \sin(\pi y), \quad u(1, y) + u(2, y) = y \sin(\pi y),$$

$$u_y(x, 0) = 0, \quad \text{and} \quad u(x, 0) = u(x, 1).$$

The first step of the two-step process yields the first order sub-tensors of $\mathcal{M}$.

$$\mathcal{M}_{ij}(x, y, g(x, y)) = \begin{bmatrix} 0 & -g_y(x, 0) & g(x, 1) - g(x, 0) \\ y^2 \sin(\pi y) - g(0, y) & - & - \\ y \sin(\pi y) - g(2, y) - g(1, y) & - & - \end{bmatrix}$$

Then, the elements of $\mathcal{M}$ associated with more than one, non-one index can be found using step two. For example,

$$M_{22} = (-1)^{3\,(1)}\mathfrak{C}_1[{}^{(2)}\rho_1] = -\left[-g_y(x, 0)\right]\Big|_{x=0} = g_y(0, 0)$$

$$= (-1)^{3\,(2)}\mathfrak{C}_1[{}^{(1)}\rho_1] = -\frac{\partial[y^2 \sin(\pi y) - g(0, y)]}{\partial y}\bigg|_{y=0} = g_y(0, 0).$$

55

Hence, the full $\mathcal{M}$ tensor can be written as,

$$\mathcal{M}_{ij}(x,y,g(x,y)) =$$

$$\begin{bmatrix} 0 & -g_y(x,0) & g(x,1)-g(x,0) \\ y^2\sin(\pi y)-g(0,y) & g_y(0,0) & g(0,0)-g(0,1) \\ y\sin(\pi y)-g(2,y)-g(1,y) & g_y(2,0)+g_y(1,0) & g(2,0)+g(1,0)-g(2,1)-g(1,1) \end{bmatrix}.$$

The Φ vectors are built using the switching functions from the univariate cases,

$$\Phi_i(x) = \left\{ 1, \ \tfrac{3-2x}{3}, \ \tfrac{x}{3} \right\} \quad \text{and} \quad \Phi_j(y) = \left\{ 1, \ y-y^2, \ -y^2 \right\}.$$

Using the $\mathcal{M}$ tensor and the Φ vectors, the full constrained expression is,

$$u(x,y,g(x,y)) = g(x,y) + \mathcal{M}_{ij}(x,y,g(x,y))\Phi_i(x)\Phi_j(y).$$

Expanding this expression and simplifying yields,

$$\begin{aligned}
u(x,y,g(x,y)) &= g(x,y) + \left(y-y^2\right)\left(\frac{3-2x}{3}g_y(0,0) - \frac{x}{3}\left(-g_y(1,0)-g_y(2,0)\right)\right.\\
&\left. - g_y(x,0)\right) - y^2\left(\frac{3-2x}{3}g(0,0) - \frac{3-2x}{3}g(0,1) - \frac{x}{3}(-g(1,0)-g(2,0))\right.\\
&\left. + \frac{x}{3}(-g(1,1)-g(2,1)) - g(x,0) + g(x,1)\right) + \frac{3-2x}{3}\left(y^2\sin(\pi y)-g(0,y)\right)\\
&+ \frac{x}{3}\left(-g(1,y)-g(2,y)+y\sin(\pi y)\right),
\end{aligned}$$

the same result as in Example 2.6.

Example 2.12: Integral constraints in tensor form

Consider the constraints from Example 2.7

$$u(x,0) = 2u_y(x,1), \quad u(x,2) = \sin(x), \quad \text{and} \quad \int_{-1}^{1} u(2,y)\,\mathrm{d}y = 5.$$

Using the same two step process the $\mathcal{M}$ tensor is constructed,

$$\mathcal{M}_{ij}(x,y,g(x,y)) = \begin{bmatrix} 0 & 2g_y(x,1) - g(x,0) & \sin(x) - g(x,2) \\ 5 - \int_{-1}^{1} g(2,\tau)\,d\tau & \int_{-1}^{1} g(2,\tau)\,d\tau - 5 & \int_{-1}^{1} g(2,\tau)\,d\tau - 5 \end{bmatrix}.$$

Since all the elements of the $\mathcal{M}$ tensor with more than one, non-one index contain intersections including integral constraints, they must be processed in a specific order. For example,

$$M_{22} = (-1)^{3\,(2)}\mathfrak{C}_1[^{(1)}\rho_1] = 2\frac{\partial\left(5 - \int_{-1}^{1} g(2,\tau)\,d\tau\right)}{\partial y}\Big|_{y=1} - \left(5 - \int_{-1}^{1} g(2,\tau)\,d\tau\right)\Big|_{y=0}$$

$$= \int_{-1}^{1} g(2,\tau)\,d\tau - 5,$$

produces the correct result that leads to a valid constrained expression, whereas,

$$(-1)^{3\,(1)}\mathfrak{C}_1[^{(2)}\rho_1] = \int_{-1}^{1}\left(g(2,0) - 2g_y(x,1)\right)d\tau,$$

does not.

The Φ vectors are built using the switching functions from the univariate cases,

$$\Phi_i(x) = \left\{1, \ \tfrac{1}{2}\right\} \quad \text{and} \quad \Phi_j(y) = \left\{1, \ \tfrac{2+11y-6y^2}{4}, \ \tfrac{2+13y-6y^2}{4}\right\}.$$

Using the $\mathcal{M}$ tensor and the Φ vectors, the full constrained expression is,

$$u(x,y,g(x,y)) = g(x,y) + \mathcal{M}_{ij}(x,y,g(x,y))\Phi_i(x)\Phi_j(y).$$

Expanding this expression and simplifying yields,

$$u(x, y, g(x, y)) = g(x, y) + \frac{1}{4}\left(2(2 - y)\left((6y + 1)g_y(x, 1) + 3y\left(\int_{-1}^{1} g(2, \tau)\, d\tau - 5\right)\right)\right.$$

$$\left. + (y - 2)(6y + 1)g(x, 0) + (y(6y - 13) - 2)g(x, 2) + (y(13 - 6y) + 2)\sin(x)\right),$$

the same result as in Example 2.7.

Example 2.13: Linear constraints in tensor form

Consider the constraints from Example 2.9,

$$u(0, y) = \cos(\pi y), \quad \int_{-1}^{2} u(1, y)\, dy = e, \quad u(x, 1) - u(x, 2) = -2,$$

$$u(x, 0) + v(x, 0) = 5, \quad \text{and} \quad v(0, y) = 5 - \cos(\pi y).$$

Using the two step process the $\mathcal{M}$ tensors for u and v are constructed,

$$\mathcal{M}_{ij}^{u}(x, y, g^{u}(x, y), g^{v}(x, y)) = \begin{bmatrix} 0 & -2 - g^{u}(x, 1) + g^{u}(x, 2) \\ \cos(\pi y) - g^{u}(0, y) & 2 + g^{u}(0, 1) - g^{u}(0, 2) \\ e - \int_{-1}^{2} g^{u}(1, \tau)\, d\tau & 0 \end{bmatrix},$$

$$\mathcal{M}_{ij}^{v}(x, y, g^{u}(x, y), g^{v}(x, y)) =$$

$$\begin{bmatrix} 0 & 5 - g^{v}(x, 0) - u(x, 0, g^{u}(x, y)) \\ 5 - \cos(\pi y) - g^{v}(0, y) & -5 + g^{v}(0, 0) + u(0, 0, g^{u}(x, y)) \end{bmatrix}.$$

The Φ vectors are built using the switching functions from the univariate cases,

$$\Phi_i^u(x) = \left\{1, \quad 1 - x, \quad \tfrac{x}{3}\right\}, \quad \Phi_j^u(y) = \left\{1, \quad \tfrac{1 - 2y}{2}\right\},$$

$$\Phi_i^v(x) = \left\{1, \quad 1\right\}, \quad \text{and} \quad \Phi_j^v(y) = \left\{1, \quad 1\right\}.$$

Using the $\mathcal{M}$ tensors and the Φ vectors, the full constrained expressions are,

$$u(x, y, g^u(x, y)) = g^u(x, y) + \mathcal{M}_{ij}(x, y, g^u(x, y))\Phi_i^u(x)\Phi_j^u(y),$$

$$v(x, y, g^v(x, y), g^u(x, y)) = g^v(x, y) + \mathcal{M}_{ij}(x, y, g^v(x, y), g^u(x, y))\Phi_i^v(x)\Phi_j^v(y).$$

Expanding these expressions and simplifying yields,

$$u(x, y, g^u(x, y)) = g^u(x, y) + \frac{1}{3}x\left(e - \int_{-1}^{2} g^u(1, \tau)\, d\tau\right) + \frac{1 - 2y}{2}\left((1 - x)(1 + g^u(0, 1))\right)$$

$$+ (1 - x)(1 - g^u(0, 2)) - g^u(x, 1) + g^u(x, 2) - 2\Big) + (1 - x)(\cos(\pi y) - g^u(0, y)),$$

$$v(x, y, g^v(x, y), g^u(x, y)) = g^v(x, y) - g^v(x, 0) - g^v(0, y) + g^v(0, 0) - u(x, 0, g^u(x, y))$$

$$+ u(0, 0, g^u(x, y)) - \cos(\pi y) + 5,$$

the same result as in Example 2.9.

3. APPLICATIONS IN DIFFERENTIAL EQUATIONS

The constrained expressions introduced in the previous chapter provide a way to analytically embed linear constraints in n-dimensions, which has a wide variety of applications, such as Computer-Aided Design (CAD) [21, 22], image warping [23], and security pattern design [24]. However, this dissertation focuses on the application of TFC to differential equations.

Differential equations are used to model and simulate physics as well as design and refine ideas, objects, systems of objects, and systems of systems. Consequently, due to their general scope, differential equations are used across a diverse range of fields, such as engineering, finance, medicine, biology, and chemistry. Although ordinary differential equations (ODEs) will be discussed, this section's primary focus will be on partial differential equations (PDEs). Due to their wide applicability, a variety of methods exist to approximate the solutions of PDEs: chief among them is the finite element method (FEM) [25, 26, 27, 28]. Although FEM has been incredibly successful in solving PDEs, it does have some drawbacks.

FEM discretizes the domain into elements. This works well for low-dimensional cases, but the number of elements grows exponentially with the number of dimensions. Therefore, the discretization becomes prohibitive as the number of dimensions increases. Another issue is that FEM solves the PDE at discrete nodes, but if the solution is needed at locations other than these nodes, an interpolation scheme must be used. Moreover, extra numerical techniques are needed to perform further manipulation of the FEM solution.

Spectral methods [29, 30, 31], pseudo-spectral methods [32], and many of their variants avoid some of these issues by proposing an analytical solution form with unknown terms that can ultimately be used to reduce the residual of the PDE at a discrete set of training points[1] and simultaneously to reduce the error in the constraints. Since these techniques use an analytical solution form, they do not require an interpolation scheme for obtaining the

---

[1]Here, and throughout the dissertation, "training points" refers to the points used by the algorithm to estimate the solution of the differential equation.

solution at points not seen during training, and the PDE solution can be further manipulated afterward without any special techniques. However, these methods still rely on a set of basis functions, and as will be shown later, the number of basis functions required to obtain an accurate solution can become prohibitive, e.g., see the Navier-Stokes example in Section 3.4.

Using neural networks (NNs) to estimate the solution of PDEs can help circumvent this issue [33, 34, 35], as NNs have been proven to be extremely useful when approximating high-dimensional, nonlinear functions: for example, consider computer vision problems that contain thousands of dimensions or the 200 dimensional PDE estimated via NNs in Reference [35]. Similar to the spectral and pseudo-spectral methods, the NN techniques represent a closed-form, analytical estimation of the PDE, and therefore, do not require an interpolation scheme or other special techniques to further manipulate the estimated PDE solution. In most of these techniques, the constraints and minimization of the PDE residual are handled via the loss function that is minimized when training the NN. Although some of them do use a functional solution form that automatically satisfies the constraints, these functionals do not have the same mathematical guarantees as the TFC constrained expressions, see the theorems in Chapter 2, and cannot satisfy certain sets of constraints, such as the constraints in the Navier-Stokes problem in Section 3.4. In addition, the functionals of the other algorithms are multiplicative in nature, whereas TFC constrained expressions are additive in nature.

One property that all of these techniques lack is a mechanized way to satisfy the PDE constraints analytically. Such a technique is particularly important in physics-informed problems and/or when constraint information is known with a high degree of confidence [3, 36]. Fortunately, as shown in previous chapters, TFC provides a mechanized method to analytically satisfy constraints while maintaining a free function. Furthermore, with a little imagination, many of the concepts from the previously introduced PDE solution methods can be adopted and combined with the constrained expression to form useful PDE estimation methods. The benefits of these methods are:

- The constraints are analytically satisfied by the constrained expression, and therefore,

do not need to be handled in a separate way, such as augmenting the loss function, using an optimizer that handles constraints, or appending the constraints to the system of equations to be solved.

- A consequence of the previous benefit is that the TFC methods are typically faster than the competing algorithms.

- Improved convergence, especially when the initial guess is poor [37].

The methods that utilize the TFC constrained expression differ only in their choice of free function. Hence, a general methodology can be introduced that can be used for any differential equation with linear constraints, and one can switch between the PDE estimation methods simply by varying the free function. The general methodology can be summarized in five steps:

1. Derive the constrained expression associated with the differential equation's constraints.

2. Define the free function, $g(\boldsymbol{x})$.

3. Discretize the domain.

4. Formulate the loss function based on the residual of the differential equation.

5. Minimize the loss function in the previous step using the free function defined in step two.

Figure 3.1 visually depicts these steps via a flowchart.

In general, a differential equation can be represented by some function F of the independent variables, $\boldsymbol{x}$, the dependent variable u, and its derivatives, i.e.,

$$F\left(\boldsymbol{x}; u; \frac{\partial u}{\partial x_1}, \dots \frac{\partial u}{\partial x_n}; \frac{\partial^2 u}{\partial x_1 \partial x_1}, \dots \frac{\partial^2 u}{\partial x_1 \partial x_n}; \dots\right) = 0.$$
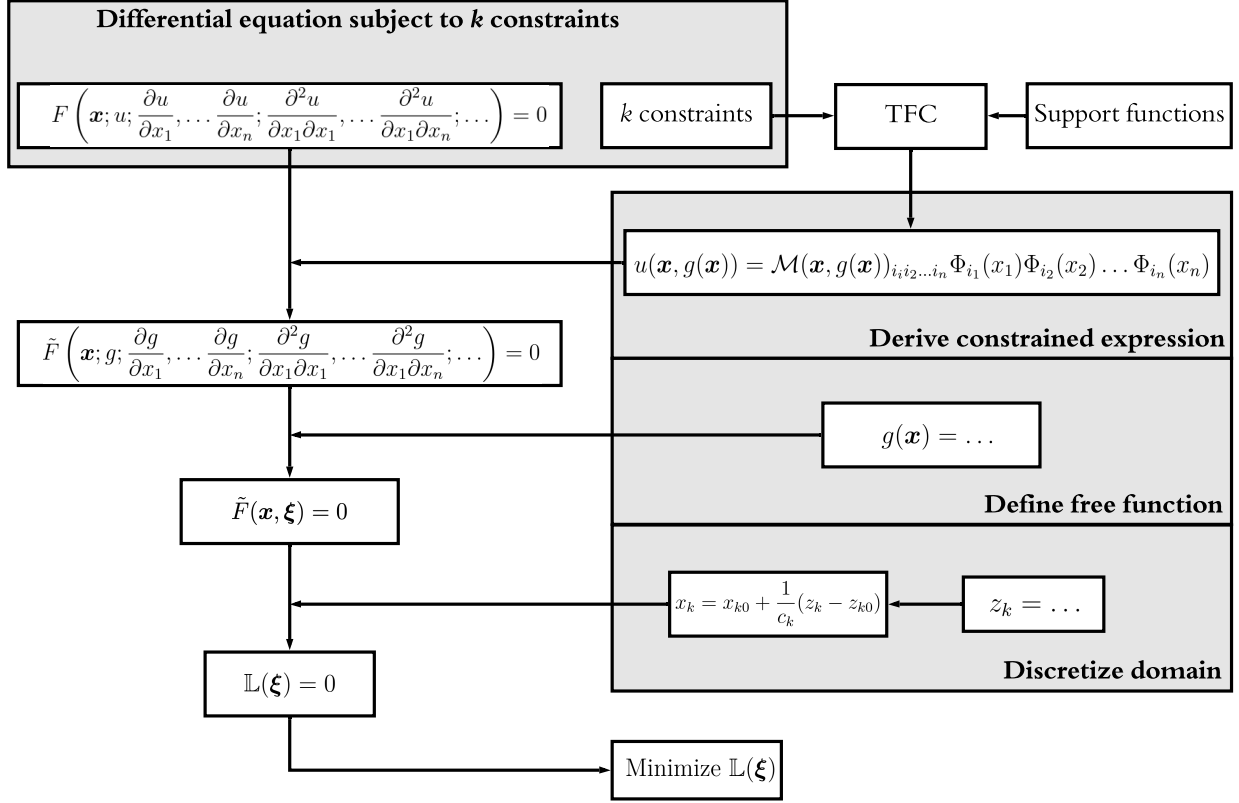
Figure 3.1: Differential equation solution estimation using TFC.

The constraints of this differential equation can be used to create a TFC constrained expression. Then, this constrained expression can be substituted into the differential equation to form a differential equation with no constraints, $\tilde{F}$, that is a function of the free function rather than the dependent variable.

Next, the free function is defined and substituted into the differential equation. In Figure 3.1, the unknown parameters in $g(\boldsymbol{x})$ are represented by the symbol $\boldsymbol{\xi}$, e.g., $\boldsymbol{\xi}$ represents θ if $g(\boldsymbol{x})$ is selected as a neural network, $\boldsymbol{\xi}$ represents $\boldsymbol{w}$ if $g(\boldsymbol{x})$ is selected as a LS-SVM, etc., see below for more details on each of these free function choices. Once the free function is substituted, the differential equation, $\tilde{F}$, becomes an algebraic equation that is a function of the independent variables $\boldsymbol{x}$ and the unknown parameters $\boldsymbol{\xi}$ only.

The dependence on the independent variables is removed by discretizing the domain. In

general, the domain of the free function may not coincide with the domain of the problem. For example, suppose the free function is selected as a linear combination of Chebyshev orthogonal polynomials which are defined on $[-1, 1]$. Let the free function be defined on $z \in [z_0, z_f]$ and the problem be defined on $x_k \in [x_{k_0}, x_{k_f}]$ where k corresponds to the dimension. In order to use the free function, a map between the basis function domain and problem domain must be created. The simplest map is a linear one,

$$z = z_0 + \frac{z_f - z_0}{x_{k_f} - x_{k_0}}(x - x_{k_0}) \quad \longleftrightarrow \quad x_k = x_{k_0} + \frac{x_{k_f} - x_{k_0}}{z_f - z_0}(z - z_0). \tag{3.1}$$

After discretizing the domain, the resultant set of algebraic equations is now only a function of the unknown parameters $\boldsymbol{\xi}$: this algebraic set of equations, also known as the loss function, is denoted by $\mathbb{L}(\boldsymbol{\xi})$. Thus, $\boldsymbol{\xi}$ are used to minimize the difference between $\mathbb{L}(\boldsymbol{\xi})$ and $\boldsymbol{0}$. Once the parameters that minimize the difference are found, they can be substituted back into the constrained expression to estimate the solution of the differential equation. Note that because the constrained expression is an analytical expression, it can be easily manipulated afterward, e.g., differentiated, integrated, etc.

3.1   Useful Free Function Choices

This section explains in detail some useful free function choices that are used in examples in later sections to solidify the reader's understanding of how TFC is applied to differential equations. Note that while it is included in this section for completeness and historical significance, as it paved the way for other machine learning algorithms, the Constrained Support Vector Machine (CSVM) methodology is no longer actively used as a free function choice, because it requires a complex analytical analysis for each new differential equation, and the resultant payoff in terms of solution error is overshadowed by the other free function choices.

3.1.1 Linear Combination of Basis Functions

A natural choice for the free function is a linear combination of basis functions, as this choice is capable of spanning the entire function space that the basis spans as the number of basis functions approaches infinity. For readers unfamiliar with univariate and multivariate basis functions, Appendix E provides a cursory overview.

Mathematically, a linear combination of m basis functions can be expressed as,

$$g(\boldsymbol{x}) = \boldsymbol{h}^{\mathrm{T}} \boldsymbol{\xi},$$

where $\boldsymbol{h} \in \mathbb{R}^m$ is a vector of the m basis functions evaluated at $\boldsymbol{x}$, and $\boldsymbol{\xi} \in \mathbb{R}^m$ is a vector of the unknown coefficients. The subsequent derivatives of the free function can be computed,

$$\frac{\partial^n g}{\partial x_k^n} = \left(\frac{\mathrm{d}z}{\mathrm{d}x_k} \right)^n \frac{\partial^n \boldsymbol{h}^{\mathrm{T}}}{\partial z^n} \boldsymbol{\xi}.$$

By defining,

$$c_k := \frac{\mathrm{d}z}{\mathrm{d}x_k} = \frac{z_f - z_0}{x_{k_f} - x_{k_0}},$$

the derivative computations can be written more succinctly as,

$$\frac{\partial^n g}{\partial x_k^n} = c_k^n \frac{\partial^n \boldsymbol{h}^{\mathrm{T}}}{\partial z^n} \boldsymbol{\xi}.$$

It follows that a partial derivative with respect to multiple independent variables, e.g., x_1 and x_2, can be written as,

$$\frac{\partial^2 g}{\partial x_1 \partial x_2} = c_1 c_2 \frac{\partial \boldsymbol{h}^{\mathrm{T}}}{\partial x_1 \partial x_2} \boldsymbol{\xi}.$$

This process applies to any derivative of the free function.

Throughout this dissertation, whenever the free function is taken to be a linear combination of basis functions, either the Chebyshev or Legendre orthogonal polynomials are used.

Thus, it is useful to mention that their optimal[2] discretization scheme is the Chebyshev-Gauss-Lobatto nodes [38, 39]. For N points, the Chebyshev-Gauss-Lobatto nodes are calculated using,

$$z_j = -\cos\left(\frac{j\pi}{N-1}\right) \quad \text{for} \quad j = 0, 1, 2, \cdots, N-1.$$

If least-squares is used as the optimization scheme, then the collocation point distribution results in a much slower increase, relative to the uniform distribution, of the condition number of the matrix to be inverted as the number of basis functions increases. The collocation points can be realized in the problem domain through the relationship provided in Equation (3.1).

A linear expansion of basis functions was the first free function used for solving differential equations using TFC and has been used extensively to solve ODEs [12, 16, 19, 20, 40, 41] and PDEs [1, 42]. However, one drawback of this free function choice is that it will become computationally prohibitive as the dimension increases. Compelling alternatives can be found in the machine learning community.

3.1.2 Support Vector Machines

Support vector machines (SVMs) were originally introduced to solve classification problems [43] like determining which class a given input x belongs to, where there are two possible classes x may belong to. The proposed solution was to find a decision boundary surface that separates the two classes. The equation of the separating boundary depended only on a few input vectors called the support vectors.

The training data is assumed to be separable by a linear decision boundary. Hence, a separating hyperplane, H, with equation $\boldsymbol{w}^{\text{T}}\boldsymbol{\varphi}(\boldsymbol{x}) + b = 0$, is sought. The parameters are rescaled such that the closest training point to the hyperplane H, $(\boldsymbol{x}_k, u_k)$, is on a parallel hyperplane H_1 with equation $\boldsymbol{w}^{\text{T}}\boldsymbol{\varphi}(x) + b = 1$. By using the formula for orthogonal projection, if $\boldsymbol{x}$ satisfies the equation of one of the hyperplanes, then the signed distance from the origin of the space to the corresponding hyperplane is given by $\boldsymbol{w}^{\text{T}}\boldsymbol{\varphi}(\boldsymbol{x})/\boldsymbol{w}^{\text{T}}\boldsymbol{w}$. Since

---

[2]Optimal here refers to minimizing the condition number of the matrix to invert when minimizing the residual of the differential equation via least-squares.

$\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\varphi}(\boldsymbol{x})$ equals $-b$ for H, and $1 - b$ for H_1, it follows that the distance between the two hyperplanes, called the "separating margin," is $1/\boldsymbol{w}^{\mathrm{T}}\boldsymbol{w}$. Thus, to find the largest separating margin, one needs to minimize $\boldsymbol{w}^{\mathrm{T}}\boldsymbol{w}$. The optimization problem becomes,

$$\min \frac{1}{2}\left(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{w}\right) \quad \text{subject to } u_i(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\varphi}(\boldsymbol{x}_i) + b) \geq 1, \quad i = 1, \ldots, n.$$

If a separable hyperplane does not exist, the problem is reformulated by taking into account the classification errors, or slack variables, Γ_i, and a linear or quadratic expression is added to the cost function. The optimization problem in the non-separable case is,

$$\min \frac{1}{2}\left(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{w}\right) + C\left(\sum \Gamma_i\right) \quad \text{subject to } u_i(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\varphi}(\Gamma_i) + b) \geq 1 - \Gamma_i.$$

When solving the optimization problem by using Lagrange multipliers, the function $\boldsymbol{\varphi}(x)$ always shows up as a dot product with itself; thus, the kernel trick [44] can be applied. In this dissertation, the kernel function chosen is the radial basis function (RBF) kernel proposed in [45]. Hence, the function $\boldsymbol{\varphi}(\boldsymbol{x})$ can be written using the kernel [44],

$$K(\boldsymbol{x}_i, \boldsymbol{x}_j) = \boldsymbol{\varphi}(\boldsymbol{x}_i)^{\mathrm{T}}\boldsymbol{\varphi}(\boldsymbol{x}_j) = \exp\left(-\frac{(\boldsymbol{x}_j - \boldsymbol{x}_i)^2}{\sigma^2}\right),$$

and its partial derivatives [45, 46],

$$K(\boldsymbol{x}_i, \boldsymbol{x}_j) = \boldsymbol{\varphi}(\boldsymbol{x}_i)^{\mathrm{T}}\boldsymbol{\varphi}(\boldsymbol{x}_j) = \exp\left(-\frac{(\boldsymbol{x}_i - \boldsymbol{x}_j)^2}{\sigma^2}\right)$$

$$K_1(\boldsymbol{x}_i, \boldsymbol{x}_j) = \boldsymbol{\varphi}'(\boldsymbol{x}_i)^{\mathrm{T}}\boldsymbol{\varphi}(\boldsymbol{x}_j) = -\frac{2(\boldsymbol{x}_i - \boldsymbol{x}_j)}{\sigma^2}\exp\left(-\frac{(\boldsymbol{x}_i - \boldsymbol{x}_j)^2}{\sigma^2}\right)$$

$$K_1^{\mathrm{T}}(\boldsymbol{x}_i, \boldsymbol{x}_j) = \boldsymbol{\varphi}(\boldsymbol{x}_i)^{\mathrm{T}}\boldsymbol{\varphi}'(\boldsymbol{x}_j) = \frac{2(\boldsymbol{x}_i - \boldsymbol{x}_j)}{\sigma^2}\exp\left(-\frac{(\boldsymbol{x}_i - \boldsymbol{x}_j)^2}{\sigma^2}\right)$$

$$K_{11}(\boldsymbol{x}_i, \boldsymbol{x}_j) = \boldsymbol{\varphi}'(\boldsymbol{x}_i)^{\mathrm{T}}\boldsymbol{\varphi}'(\boldsymbol{x}_j) = \frac{2}{\sigma^2} - \frac{4(\boldsymbol{x}_i - \boldsymbol{x}_j)^2}{\sigma^4}\exp\left(-\frac{(\boldsymbol{x}_i - \boldsymbol{x}_j)^2}{\sigma^2}\right).$$

The SVM free function choice was inspired by least-squares SVMs (LS-SVMs) and their

success in solving differential equations [45]. They can be written mathematically as,

$$g(\boldsymbol{x}) = \boldsymbol{w}^{\mathrm{T}} \boldsymbol{\varphi}(\boldsymbol{x}),$$

where $\boldsymbol{w}$ is a vector of weights used in the optimization process to reduce the residual of the differential equation and $\boldsymbol{\varphi}(\boldsymbol{x})$ is defined in terms of the kernel function. This free function choice was used in Reference [2] to solve ODEs and PDEs.

3.1.3 Neural Networks

The architecture of neural networks is a rich topic, and one could spend a lot of time analyzing how different architecture choices ultimately affect the accuracy of the differential equation solution. The author has chosen to leave this research effort to future work and instead focus on just one of the simpler architectures, a fully connected neural network. Each layer of a fully connected neural network consists of a nonlinear activation function composed with a linear transformation of the form $\mathcal{A} = W \cdot \boldsymbol{x} + \boldsymbol{b}$, where W is a matrix of the neuron weights, $\boldsymbol{b}$ is a vector of the neuron biases, and $\boldsymbol{x}$ is a vector of inputs from the previous layer (or the inputs to the neural network if it is the first layer). Then, each layer is composed to form the entire network. For the fully connected neural networks used in this dissertation, the last layer is simply a linear output layer. For example, a neural network with three hidden layers that each use the nonlinear activation function ψ and a linear output layer can be written mathematically as,

$$\mathcal{N}(\boldsymbol{x}; \theta) = W_4 \cdot \psi\left(W_3 \cdot \psi\left(W_2 \cdot \psi\left(W_1 \cdot \boldsymbol{x} + \boldsymbol{b}_1\right) + \boldsymbol{b}_2\right) + \boldsymbol{b}_3\right) + \boldsymbol{b}_4,$$

where $\mathcal{N}$ is the neural network function, $\boldsymbol{x}$ is the vector of inputs, W_k are the weight matrices, $\boldsymbol{b}_k$ are the bias vectors, and θ is a symbol that represents all trainable parameters of the neural network: the weights and biases of each layer constitute the trainable parameters. Note that the notation $\mathcal{N}(x, y, \ldots; \theta)$ is also used in this dissertation for independent variables

$x, y, \ldots$ and trainable parameters θ. In this dissertation, all neural networks' weights are initialized using the Glorot uniform initialization [47], and the biases are initialized as zeros. Whenever a neural network is used as the free function in a constrained expression to solve a differential equation, the overall technique is referred to as Deep-TFC. This technique was used in Reference [3] to solve a variety of PDEs.

3.1.4 Extreme Learning Machines

Extreme learning machines (ELMs) are a learning algorithm for single-hidden layer neural networks that randomly selects the hidden layer's input weights and biases and computes the output weights via least-squares [48]. Since the weights and biases of the hidden layer are not tuned during the training, i.e., they are not trainable parameters, the neural network is linear with respect to the trainable parameters; thus, they can be computed via least-squares. In terms of the neural network description given in the previous section, an ELM can be expressed mathematically as,

$$\mathcal{N}(\boldsymbol{x}; \theta) = W_2 \cdot \psi\big(W_1 \cdot \boldsymbol{x} + \boldsymbol{b}_1\big) \tag{3.2}$$

where θ consists of W_2 only. In this dissertation, the hidden layer's weights and biases, W_1 and b_1, respectively, are initialized using the uniform distribution $U(-10, 10)$ when solving ODEs and the uniform distribution on $U(-1, 1)$ when solving PDEs. Whenever an ELM is used as the free function in a constrained expression, the overall technique is referred to as X-TFC. X-TFC was used in Reference [49] to solve various ODEs and PDEs.

3.2 Useful Optimization Options

The optimization/minimization methods introduced in this section do not constitute an exhaustive list of optimizers that can be used with TFC. Rather, they form a short list of the optimization/minimization methods used for the problems and examples given in this dissertation. Many other optimization/minimization schemes could be used in conjunction with TFC to estimate the solutions of differential equations, and exploring them is a topic

of future work.

3.2.1 Least-Squares

When using basis functions, SVMs, or ELMs as the free function, the resultant minimization problem, $\mathbb{L}(\boldsymbol{\xi}) = 0$, can be solved via least-squares. For linear differential equations, the loss function can be written as,

$$\mathbb{L}(\boldsymbol{\xi}) = \mathbb{A}\boldsymbol{\xi} - \boldsymbol{b} = 0,$$

and a linear least-squares technique can be employed to solve,

$$\mathbb{A}\boldsymbol{\xi} = \boldsymbol{b}.$$

Appendix F describes some common methods to solve the linear least-squares problem. Of the approaches presented in Appendix F, the scaled QR method tends to have the lowest condition number and is thus the most numerically stable. In the specific case of ELMs, the $\mathbb{A}$ matrix tends to be ill-conditioned as the number of basis functions increases: as the number of basis functions increases, the probability of selecting nearly identical values for the weights and biases of two different neurons increases, which results in linearly dependent or nearly linearly dependent columns in $\mathbb{A}$. In this case, taking the pseudo-inverse using one of the previous techniques typically results in an inaccurate solution. To remedy this, a least-squares technique designed for ill-conditioned matrices is employed, such as the `lstsq` function in the NumPy package available for Python or the `lsqminnorm` function available in MATLAB.

If the differential equation is nonlinear, then a nonlinear least-squares, also known as iterative least-squares, method can be used. In this case, the loss function for the j-th iteration is approximated using the first two terms of the Taylor series,

$$\mathbb{L}(\boldsymbol{\xi}) \approx \mathbb{L}(\boldsymbol{\xi}_j) + \mathcal{J}\Delta\boldsymbol{\xi},$$

where $\Delta\boldsymbol{\xi} = \boldsymbol{\xi} - \boldsymbol{\xi}_j$ and

$$\mathcal{J}(\boldsymbol{\xi}_j) = \left.\frac{\partial\mathbb{L}(\boldsymbol{\xi})}{\partial\boldsymbol{\xi}}\right|_{\boldsymbol{\xi}_j},$$

is the Jacobian matrix of $\mathbb{L}(\boldsymbol{\xi})$. The goal is to drive the loss function to zero, i.e., set $\boldsymbol{\xi} = \boldsymbol{\xi}_{j+1}$ in the truncated Taylor series and set it equal to zero. The result is,

$$\boldsymbol{\xi}_{j+1} = \boldsymbol{\xi}_j + \Delta\boldsymbol{\xi},$$

where

$$\mathbb{L}(\boldsymbol{\xi}_j) + \mathcal{J}(\boldsymbol{\xi}_j)\Delta\boldsymbol{\xi} = 0$$

is used to determine the value of $\Delta\boldsymbol{\xi}$. Notice that the solution for $\Delta\boldsymbol{\xi}$ can be re-written as

$$\mathcal{J}(\boldsymbol{\xi}_j)\Delta\boldsymbol{\xi} = -\mathbb{L}(\boldsymbol{\xi}_j),$$

which can be solved using the previously described linear least-squares techniques. This method is repeated until the user-specified termination condition(s) are met. For a description of the termination conditions used in this dissertation, see Section G.6 of Appendix G.

3.2.2 Quasi-Newton Methods

Quasi-Newton methods approximate the Jacobian matrix—some also approximate the Hessian—of a given function and use them to minimize the given function. To use them with TFC, the loss function is first converted to a scalar by taking its norm: in this dissertation, that norm is either the L_1 or L_2 norm. Then, the new loss function is minimized using the Quasi-Newton method. In this dissertation, the only Quasi-Newton method used is the limited-memory Broyden-Fletcher-Goldfarb-Shanno [50] (L-BFGS) algorithm.

3.2.3 Gradient Descent

Gradient descent algorithms are an iterative method used to minimize a scalar loss function—as with Quasi-Newton methods, the TFC loss functions are converted to scalar functions by taking either their L_1 or L_2 norm—via its Jacobian matrix. In its simplest form,

$$\boldsymbol{\xi}_{j+1} = \boldsymbol{\xi}_j + \Delta\boldsymbol{\xi}$$

where

$$\Delta\boldsymbol{\xi} = -\lambda\mathcal{J}(\boldsymbol{\xi}_j)$$

and λ is some positive constant frequently referred to as the learning rate. However, the gradient descent can become more complex by, for example, randomly sampling a subset of the training points for each iteration as is done in stochastic gradient descent [51, 52], adapting the learning rate based on data from previous iterations as is done in AdaGrad [53], or using a concept analogous to linear momentum to inform the update along with gradient descent [54]. Variants of these techniques are utilized by the Adaptive Momentum Estimation (Adam) algorithm [55], which is the gradient descent algorithm used in this dissertation.

3.2.4 Constrained Least-Squares Support Vector Machines

The inspiration to use Least-Squares Support Vector Machines (LS-SVMs) stems from References [45] and [56], which used LS-SVMs to solve ODEs and PDEs, respectively. Essentially, this technique uses an LS-SVM to reduce the residual of the differential equation via least-squares, where Lagrange multipliers are used to enforce the differential equation constraints. When TFC is used with an SVM as the free function to solve the differential equation using the LS-SVM optimization technique, the overall methodology is referred to as a Constrained Support Vector Machine (CSVM) [2]. This methodology is best understood via an example.

Example 3.1: CSVM applied to a linear, first-order ODE

Consider a first-order, linear ODE,

$$\dot{y} - p(t)y = r(t), \quad \text{subject to} \quad y(t_0) = y_0,$$

where $\dot{y} = \frac{dy}{dt}$. The TFC constrained expression for the constraint is,

$$y(t, g(t)) = g(t) + y_0 - g(0).$$

Further, let the free function be defined as an SVM,

$$g(t) = \boldsymbol{w}^{\mathrm{T}} \boldsymbol{\varphi}(t),$$

so the constrained expression becomes,

$$y(t, \boldsymbol{w}) = \boldsymbol{w}^{\mathrm{T}} \big(\boldsymbol{\varphi}(t) - \boldsymbol{\varphi}(t_0) \big) + y_0. \tag{3.3}$$

Notice that a least-squares technique cannot be formed directly from the residual of the differential equation because $\boldsymbol{\varphi}(t)$ is only defined via the kernel trick, i.e., only dot products with itself are defined. Hence, a loss function is constructed,

$$\min \frac{1}{2} \left(\boldsymbol{w}^{\mathrm{T}} \boldsymbol{w} + \gamma \boldsymbol{e}^{\mathrm{T}} \boldsymbol{e} \right)$$

where γ is a positive, user-specified constant and

$$\boldsymbol{e} = e_i = \dot{y}(t_i, \boldsymbol{w}) - p(t_i) y_i(t_i, \boldsymbol{w}) - r(t_i).$$

This loss function contains the original term used to find the largest separating margin, $\boldsymbol{w}^{\mathrm{T}} \boldsymbol{w}$, as well as a term to reduce the residual of the ODE, $\gamma \boldsymbol{e}^{\mathrm{T}} \boldsymbol{e}$. Since the constrained

expression embeds the differential equation constraint, the only constraint that remains is the one associated with the error term, $\boldsymbol{e}$. Hence, the optimization problem is,

$$\min \frac{1}{2} \left(\boldsymbol{w}^{\mathrm{T}} \boldsymbol{w} + \gamma \boldsymbol{e}^{\mathrm{T}} \boldsymbol{e} \right)$$

$$\text{subject to} \quad \boldsymbol{w}^{\mathrm{T}} \dot{\boldsymbol{\varphi}}(t_i) - p(t_i) \left(\boldsymbol{w}^{\mathrm{T}} \boldsymbol{\varphi}(t_i) - \boldsymbol{w}^{\mathrm{T}} \boldsymbol{\varphi}(t_0) + y_0 \right) - r(t_i) - e_i = 0.$$

Using the method of Lagrange multipliers a term is introduced for the constraint on the residuals, which leads to the expression,

$$\mathbb{L}(\boldsymbol{w}, \boldsymbol{e}, \boldsymbol{\alpha}) = \frac{1}{2} \left(\boldsymbol{w}^{\mathrm{T}} \boldsymbol{w} + \gamma \boldsymbol{e}^{\mathrm{T}} \boldsymbol{e} \right)$$
$$- \sum_{i=1}^{N} \alpha_i \left[\boldsymbol{w}^{\mathrm{T}} \dot{\boldsymbol{\varphi}}(t_i) - p(t_i) \left(\boldsymbol{w}^{\mathrm{T}} \boldsymbol{\varphi}(t_i) - \boldsymbol{w}^{\mathrm{T}} \boldsymbol{\varphi}(t_0) + y_0 \right) - r(t_i) - e_i \right],$$

where α_i are the Lagrange multipliers. The values that force the gradients of $\mathbb{L}$ to be equal to zero give candidates for the minimum,

$$\frac{\partial \mathbb{L}}{\partial \boldsymbol{w}} = 0 \quad \rightarrow \quad \boldsymbol{w} = \sum_{i=1}^{N} \alpha_i \left[\dot{\boldsymbol{\varphi}}(t_i) - p(t_i) \left(\boldsymbol{\varphi}(t_i) - \varphi(t_0) \right) \right]$$

$$\frac{\partial \mathbb{L}}{\partial e_i} = 0 \quad \rightarrow \quad e_i = -\frac{\alpha_i}{\gamma}$$

$$\frac{\partial \mathbb{L}}{\partial \alpha_i} = 0 \quad \rightarrow \quad 0 = \boldsymbol{w}^{\mathrm{T}} \dot{\boldsymbol{\varphi}}(t_i) - p(t_i) \left(\boldsymbol{w}^{\mathrm{T}} \left(\boldsymbol{\varphi}(t_i) - \boldsymbol{\varphi}(t_0) \right) + y_0 \right) - r(t_i) - e_i.$$

Using,

$$\boldsymbol{w} = \sum_{j=1}^{N} \alpha_j \left[\dot{\boldsymbol{\varphi}}(t_j) - p(t_j) \left(\boldsymbol{\varphi}(t_j) - \varphi(t_0) \right) \right],$$

one obtains a new formulation of the approximate solution given by Equation (3.3) that can be expressed in terms of the kernel and its derivatives. One can combine the three equations obtained by setting the gradients of $\mathbb{L}$ equal to zero together to create

74

a linear system with unknowns α_j,

$$M_{ij}\alpha_j = r(t_i) + p(t_i)y_0.$$

The coefficient matrix, M_{ij}, is given by,

$$M_{ij} = K_{11}(t_i, t_j) - p(t_j)\left[K_1(t_i, t_j) - K_1(t_i, t_0)\right] - p(t_i)K_y(t_i, t_j) + \delta_{ij}/\gamma,$$

where,

$$K_4(t_i, t_j) = K(t_i, t_j) - K(t_j, t_0) - K(t_i, t_0) + 1,$$

$$K_y(t_i, t_j) = K_1(t_j, t_i) - K_1(t_j, t_0) - p(t_j)K_4(t_i, t_j).$$

Finally, in terms of the kernel matrix, the approximate solution at the training points, t_i, is given by,

$$y(t_i) = \sum_{j=1}^{N} \alpha_j K_y(t_i, t_j) + y_0,$$

and a formula for the approximate solution at an arbitrary point t is given by,

$$y(t) = \sum_{j=1}^{N} \alpha_j K_y(t, t_j) + y_0.$$

The CSVM technique creates a loss function based on the residual of the differential equation that can ultimately be solved via least-squares. Moreover, the least-squares system and the constrained expression can be rewritten in terms of the kernel function and its derivatives. Although not utilized in this dissertation, a similar derivation for first-order, nonlinear ODEs is included in Appendix H for completeness.

3.3 Numerical Implementation

Even for simple PDEs, taking the derivatives necessary to implement TFC analytically is tedious and error-prone. The errors in taking the derivatives can be reduced by using a

symbolic programming paradigm, but the results must still be copied into another framework, Python, MATLAB, etc., which is time-consuming and error-prone as well. Of course, TFC could be implemented directly in a symbolic program, but the computation speed would suffer: one of TFC's main benefits. If these were the only options, applying TFC to differential equations would be cumbersome, frustrating, and slow, and users would most likely choose other differential equation solution methods due to this pitfall. Fortunately, automatic differentiation alleviates the issues that plague the other implementation options.

Automatic differentiation utilizes the chain rule of differential calculus and modifies the variable types to calculate derivatives [57]. As such, automatic differentiation can be applied to code with minimal changes and can evaluate "derivatives at machine-level precision with only a small constant factor of overhead and ideal asymptotic efficiency" [57]. For example, Reference [58] utilizes a nilpotent algebra to calculate arbitrary order derivatives; this technique is an example of forward-mode automatic differentiation, where the derivative is calculated alongside the primary value. A second type is reverse-mode automatic differentiation [59], where the chain rule is traversed from the output backwards towards the input to calculate the derivative: also known as back-propagation. In general, a good rule of thumb is that reverse mode differentiation should be used for functions $f \colon \mathbb{R}^n \mapsto \mathbb{R}^m$ where $n \gg m$; otherwise, forward mode differentiation should be used. Hence, forward mode automatic differentiation will typically be used when implementing TFC. An in-depth understanding of automatic differentiation is not required to understand TFC's numerical implementation, so it will not be discussed here. However, if the reader is interested in learning more, they should consult Reference [57].

JAX [60, 61] is a framework for Python that combines the automatic differentiation power of Autograd [62] with XLA (Accelerated Linear Algebra) [63] to produce fast, composable transformations of NumPy/Python code. Moreover, a just-in-time compiler (JIT) allows one to easily convert their code into XLA-optimized kernels. Ultimately, this allows the user to easily compute the derivatives necessary to apply TFC to differential equations, and

JIT-ing the resultant code makes the run time fast: many of the differential equations in this dissertation were estimated via TFC in less than a second. Numerically implementing TFC in a JIT-able way via JAX was not a simple plug-and-play. Rather, the author wrote over 8,000 lines of C++ and Python code that interface with JAX to produce the final product. This code forms some general-use classes that can be used to apply TFC to a large variety of differential equations. The intricacies of this code are not germane to the topics covered in the body of this dissertation, and therefore are not included here; a more detailed description of the codebase can be found in Appendix G, and the reference documentation contains a complete description. Furthermore, this general-use code is publicly available on the TFC GitHub and includes the scripts used to generate many of the examples and results found in this dissertation [64].

The computations for all examples and results in this dissertation were performed in Python on a desktop computer running Ubuntu 20.04 with an Intel® Core™ i5-2400 and 16 GB of RAM. All run times were calculated using the `process_timer` function from the Python `time` package.

3.4   Simple PDE Example

To better understand each of the previously introduced free functions, this section applies each one to the same linear PDE:

$$u_{xx}(x, y) + u_{yy}(x, y) = e^{-x}(x - 2 + y^3 + 6y)$$

where $x, y \in [0, 1]$ and subject to,

$$
\begin{aligned}
u(0, y) &= y^3 \\
u(1, y) &= (1 + y^3)e^{-1} \\
u(x, 0) &= xe^{-x} \\
u(x, 1) &= e^{-x}(x + 1),
\end{aligned}
$$

which has the true solution $u(x, y) = e^{-x}(x + y^3)$. The true solution is shown in Figure 3.2. Following the step-by-step method given earlier, the PDE can be re-written as,
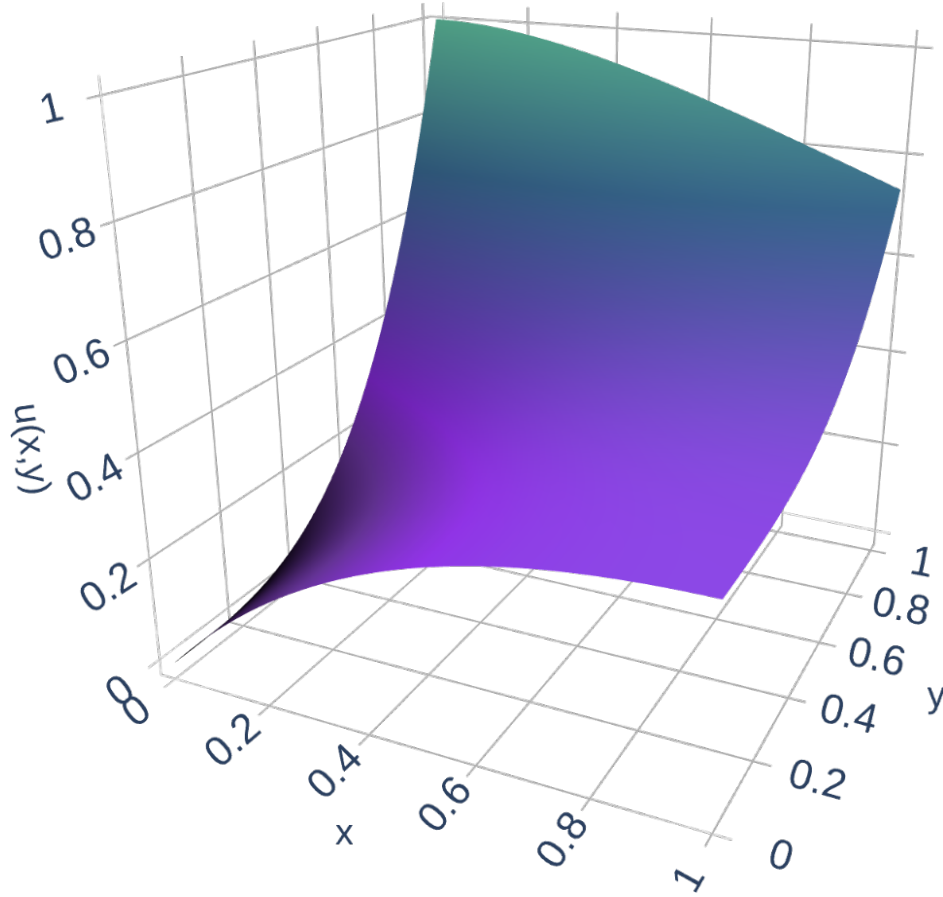
Figure 3.2: Analytical solution of the simple PDE.

$$F(x, y, u, u_{xx}, u_{yy}) = u_{xx}(x, y) + u_{yy}(x, y) - e^{-x}(x - 2 + y^3 + 6y) = 0.$$

The constraints can be embedded into a constrained expression,

$$^{(1)}u(x, y, g(x, y)) = g(x, y) + (1 - x)\left(y^3 - g(0, y)\right) + x\left((1 + y^3)e^{-1} - g(1, y)\right)$$

$$^{(2)}u(x, y, g(x, y)) = g(x, y) + (1 - y)\left(xe^{-x} - g(x, 0)\right) + y\left(e^{-x}(x + 1) - g(x, 1)\right)$$

where $^{(1)}u$ can be used as the free function in $^{(2)}u$ or $^{(2)}u$ can be used as the free function in $^{(1)}u$ to create the full constrained expression. The constrained expression written in tensor form is,

$$u(x, y) = g(x, y, g(x, y)) + \mathcal{M}(x, y, g(x, y))_{ij} \Phi_i(x) \Phi_j(y)$$

where

$$\mathcal{M}(x, y, g(x, y))_{ij} = \begin{bmatrix} 0 & xe^{-x} - g(x, 0) & e^{-x}(x + 1) - g(x, 1) \\ y^3 - g(0, y) & g(0, 0) & g(0, 1) - 1 \\ (1 + y^3)e^{-1} - g(1, y) & g(1, 0) - e^{-1} & g(1, 1) - 2e^{-1} \end{bmatrix},$$

$$\Phi_i(x) = \left\{ 1, \quad 1 - x, \quad x \right\}, \quad \text{and} \quad \Phi_j(y) = \left\{ 1, \quad 1 - y, \quad y \right\}.$$

Substituting the constrained expression into F yields $\tilde{F}(x, y, g(x, y))$, which does not have any constraints. Now, the various free function choices introduced earlier will be used to minimize $\tilde{F}$.

Example 3.2: Simple PDE solved using basis functions [1]

Let $g(x, y)$ be a linear expansion of Chebyshev orthogonal polynomials, and let m be the maximum degree of said polynomials. Remember, as shown in Appendix E, that the two-dimensional basis set is just a tensor product of the univariate Chebyshev orthogonal polynomials. Further, recall from the result of Theorem 8 that the basis functions linearly dependent to the support functions must be removed from the expansion: in the multivariate case, this also includes products of the support functions that include exactly one support function from each independent variable, e.g., $s_i(x_1)s_j(x_2)...s_k(x_n)$. To expound, suppose instead that the linear expansion for $g(x, y)$ was simply the set of monomials.

$$g(x, y) = \xi_1 + \xi_2 x + \xi_3 y + \xi_4 x^2 + \xi_5 xy + \xi_6 y^2 + \ldots$$

In this case, the terms 1, x, y, and xy need to be removed from the expansion, as 1, x, and y are used as support functions in the constrained expression. The same needs to be done for the Chebyshev orthogonal expansion used in this example.

Since the linear expansion is a tensor product and the terms linearly dependent to the support functions have to be removed, the degree of the expansion, m, and the number of basis functions in the expansion do not have a simple relationship. Therefore, the degree of the expansion, m, and the number of basis functions in the expansion are tabulated for this example in Table 3.1.

Table 3.1: Tabulated values for the degree of basis expansion and equivalent number of basis functions.

m	Number of Functions
5	17
10	62
15	132
20	227
25	347

Once the free function is substituted into the differential equation, $\tilde{F} = \tilde{F}(x, y, \boldsymbol{\xi}) = 0$. Next, the domain is discretized. Since Chebyshev orthogonal polynomials are used, the domain is discretized using Chebyshev-Gauss-Lobatto nodes. Let the number of points per independent variable be given by n. For example, a value of $n = 5$ would imply a 5×5 grid or 25 total training points. After the domain is discretized $\tilde{F}$ becomes $\mathbb{L}(\boldsymbol{\xi}) = 0$. The PDE in this example is linear, so $\mathbb{L}$ is linear in $\boldsymbol{\xi}$, and therefore, linear least-squares can be used to minimize $\mathbb{L}$.

Let the test set be a 100×100 grid of uniformly spaced points. Table 3.2 shows the maximum test set solution error,

$$e = \max_{(x,y) \in \text{test set}} |u(x,y) - u_{\text{true}}(x,y)|,$$

where $u_{\text{true}}(x,y)$ is the true solution given earlier, for different values of n and m. Table 3.2 shows that in general as the number of basis functions and training points increases, the maximum test set solution error decreases.

Table 3.2: Maximum test set solution error using TFC with Chebyshev orthogonal polynomials.

n \ m	5	10	15	20	25
5	6.26×10^{-4}	-	-	-	-
10	5.53×10^{-4}	1.20×10^{-10}	-	-	-
15	5.30×10^{-4}	1.17×10^{-10}	4.44×10^{-16}	-	-
20	5.20×10^{-4}	1.16×10^{-10}	5.00×10^{-16}	4.44×10^{-16}	-
25	5.13×10^{-4}	1.15×10^{-10}	7.22×10^{-16}	2.61×10^{-15}	5.55×10^{-16}
30	5.09×10^{-4}	1.14×10^{-10}	6.66×10^{-16}	8.88×10^{-16}	3.22×10^{-15}

Since the TFC method of solving differential equations is closely related to the spectral method—the only real difference is how the constraints are handled—it is worth comparing the two methods. To this end, Table 3.3 shows the same results as Table 3.2 but using the spectral method.

Table 3.3: Maximum test set solution error using spectral method with Chebyshev orthogonal polynomials.

n \ m	5	10	15	20	25
5	4.25×10^{-4}	-	-	-	-
10	3.40×10^{-4}	7.11×10^{-11}	-	-	-
15	3.16×10^{-4}	7.95×10^{-11}	1.41×10^{-12}	-	-
20	3.04×10^{-4}	7.77×10^{-11}	4.85×10^{-12}	5.75×10^{-12}	-
25	2.97×10^{-4}	7.69×10^{-11}	3.45×10^{-12}	9.91×10^{-12}	2.71×10^{-11}
30	2.92×10^{-4}	7.59×10^{-11}	3.12×10^{-12}	1.19×10^{-11}	1.79×10^{-11}

Comparing Tables 3.2 and 3.3 reveals that the spectral method is slightly more accurate—less than an order of magnitude—than TFC when the number of basis functions is low. However, as the number of basis functions increases, TFC becomes as many as five orders of magnitude more accurate than the spectral method. The accuracy difference between the two methods for a low number of basis functions stems from the fact that the spectral method can relax the error on the constraints in order to reduce the average error over the domain, whereas TFC is constrained to satisfy the constraints exactly, and so does not have the same freedom. The accuracy difference between the two methods for a high number of basis functions stems from the fact that TFC effectively has more information than spectral method, as it has the exact constraint information over the entire boundary, whereas spectral method only has information about the constraints at discrete points on the associated boundaries. Moreover, TFC is faster than the spectral method, as the matrix that is inverted during the least-squares process is smaller; it is smaller because it does not contain the extra equations that the spectral method needs to satisfy the constraints.

Example 3.3: Simple PDE solved using CSVM [2]

Let $g(x, y)$ be a SVM, so $\tilde{F}$ becomes $\tilde{F}(x, y, \boldsymbol{w})$. Then, the CSVM technique must be applied to rewrite the constrained expression and the optimization process in the dual form. For this example only, let superscripts denote a derivative with respect to the superscript variable and a subscript be a normal tensor index: this is done for clarity and compactness. For example, the symbol A_{ij}^{xx} would denote a second-order derivative of the second-order tensor A_{ij} with respect to the variable x, i.e., $\frac{\partial^2 A_{ij}}{\partial x^2}$. In the same spirit, for this example only, the arguments of most functions and functionals will be dropped.

The constrained expression shown earlier can be re-written as,

$$u = A_{ij}\Phi_i\Phi_j + w_j\varphi_j(x, y) - w_k B_{ijk}\Phi_i\Phi_j, \quad \text{where}$$

$$A_{ij} = \begin{bmatrix} 0 & xe^{-x} & e^{-x}(x+1) \\ y^3 & 0 & -1 \\ (1+y^3)e^{-1} & g(1,0) - e^{-1} & -2e^{-1} \end{bmatrix}$$

$$B_{ijk} = \begin{bmatrix} 0 & \varphi_k(x,0) & \varphi_k(x,1) \\ \varphi_k(0,y) & -\varphi_k(0,0) & -\varphi_k(0,1) \\ \varphi_k(1,y) & -\varphi_k(1,0) & -\varphi_k(1,1) \end{bmatrix}$$

$$\Phi_i = \left\{ 1, \quad 1-x, \quad x \right\}, \quad \text{and} \quad \Phi_j = \left\{ 1, \quad 1-y, \quad y \right\}.$$

Now, discretize the domain and use Lagrange multiplies to form $\mathbb{L}$,

$$\mathbb{L}(\boldsymbol{w}, \boldsymbol{\alpha}, \boldsymbol{e}) = \frac{1}{2}w_i w_i + \frac{\gamma}{2}e_I e_I - \alpha_I(u_I^{xx} + u_I^{yy} - f_I - e_I),$$

where u_I is a vector whose elements are $u(x_I, y_I, \boldsymbol{w})$ where (x_I, y_I) is the I-th training point. The gradients of $\mathbb{L}$ give candidates for the minimum,

$$\frac{\partial \mathbb{L}}{\partial w_k} = w_k - \alpha_I(\varphi_{Ik}^{xx} - B_{Iijk}^{xx}\Phi_i\Phi_j + \varphi_{Ik}^{yy} - B_{Iijk}^{yy}\Phi_i\Phi_j) = 0$$

$$\frac{\partial \mathbb{L}}{\alpha_I} = \hat{z}_I^{xx} + \hat{z}_I^{yy} - f_I - e_I = 0$$

$$\frac{\partial \mathbb{L}}{e_I} = \frac{\gamma}{2}e_I - \alpha_I = 0,$$

where φ_{Ik} is the second-order tensor composed of the vectors $\varphi_k(x_I, y_I)$ and B_{Iijk} is the fourth-order tensor composed of the third-order tensors $B(x_I, y_I)_{ijk}$. The gradients of $\mathbb{L}$ can be used to form a system of simultaneous linear equations to solve for the unknowns and write u in the dual form. The system of simultaneous linear equations is,

$$\mathcal{A}_{IJ}\alpha_J = \mathcal{B}_I,$$

where

$$\mathcal{A}_{IJ} = \varphi_{Ik}^{xx}\varphi_{Jk}^{xx} - \varphi_{Ik}^{xx}B_{Jijk}^{xx}\Phi_i\Phi_j + \varphi_{Ik}^{xx}\varphi_{Jk}^{yy} - \varphi_{Ik}^{xx}B_{Jijk}^{yy}\Phi_i\Phi_j - B_{Iijk}^{xx}\Phi_i\Phi_j\varphi_{Jk}^{xx}$$

$$+ B_{Iijk}^{xx}\Phi_i\Phi_j B_{Jmnk}^{xx}\Phi_m\Phi_n - B_{Iijk}^{xx}\Phi_i\Phi_j\varphi_{Jk}^{yy} + B_{Iijk}^{xx}\Phi_i\Phi_j B_{Jmnk}^{yy}\Phi_m\Phi_n + \varphi_{Ik}^{yy}\varphi_{Jk}^{xx}$$

$$- \varphi_{Ik}^{yy}B_{Jijk}^{xx}\Phi_i\Phi_j + \varphi_{Ik}^{yy}\varphi_{Jk}^{yy} - \varphi_{Ik}^{yy}B_{Jijk}^{yy}\Phi_i\Phi_j - B_{Iijk}^{yy}\Phi_i\Phi_j\varphi_{Jk}^{xx}$$

$$+ B_{Iijk}^{yy}\Phi_i\Phi_j B_{Jmnk}^{xx}\Phi_m\Phi_n - B_{Iijk}^{yy}\Phi_i\Phi_j\varphi_{Jk}^{yy} + B_{Iijk}^{yy}\Phi_i\Phi_j B_{Jmnk}^{yy}\Phi_m\Phi_n + \frac{1}{\gamma}\delta_{IJ}$$

$$\mathcal{B}_I = f_I - A_{Iij}^{xx}\Phi_i\Phi_j - A_{Iij}^{yy}\Phi_i\Phi_j$$

where $\Phi_m = \Phi_i$, $\Phi_n = \Phi_j$, and A_{Iijk} is the fourth order tensor composed of the third order tensors $A(x_I, y_I)_{ijk}$. The dual-form of the solution is,

$$u(x, y, \boldsymbol{\alpha}) = A_{ij}\Phi_i\Phi_j$$

$$+ \alpha_I \left[\varphi_{Ik}^{xx} \varphi(x,y)_k - B_{Iijk}^{xx} \Phi_i \Phi_j \varphi_k(x,y) + \varphi_{Ik}^{yy} \varphi_k(x,y) - B_{Iijk}^{yy} \Phi_i \Phi_j \varphi_k(x,y) \right]$$

$$- \alpha_I \left[\varphi_{Ik}^{xx} B_{ijk} \Phi_i \Phi_j - B_{Iijk}^{xx} \Phi_i \Phi_j B_{mnk} \Phi_m \Phi_n + \varphi_{Ik}^{yy} B_{ijk} \Phi_i \Phi_j - B_{Iijk}^{yy} \Phi_i \Phi_j B_{mnk} \Phi_m \Phi_n \right].$$

The system of simultaneous linear equations and the dual form of the solution can be written and were solved using the kernel matrix and its partial derivatives.

The attentive reader will notice two user-specified hyperparameters remain to be selected: σ, the user-specified parameter that modifies the kernel matrix, and γ, the parameter that appears in the loss function. In Reference [2], a grid search was used to find the pair of hyperparameters that minimized the residual of the differential equation on a set of validation points. Using that set of hyperparameters and 100 training points, the test set's maximum error was 5.561×10^{-8}.

Example 3.4: Simple PDE solved using Deep-TFC [3]

Let $g(x,y)$ be a neural network as defined earlier with nonlinear activation function $\psi = \tanh$. Further, let this neural network have six hidden layers with 15 neurons per layer and a linear output layer. Substituting the neural network as the free function into the constrained expression, then substituting the constrained expression into the differential equation, and finally discretizing the domain leads to a loss function, $\mathbb{L}(\theta)$, where θ are the trainable parameters of the neural network as defined earlier; the domain was discretized using a 10×10 grid of uniformly spaced points.

Neural networks are typically trained using gradient descent algorithms, but the author has found that for solving PDEs using Deep-TFC, the Quasi-Newton algorithms typically perform better. In this example, the L-BFGS algorithm was used to train the network.

Due to the inherent stochasticity of the Deep-TFC method, the problem was solved ten times, and the best solution was retained; that solution had a maximum error of 2.780×10^{-7} on the test set, a uniformly distributed 100×100 grid. The aforementioned stochasticity is easily visualized as a histogram: Figure 3.3 shows such a histogram for 100 Monte Carlo trials.
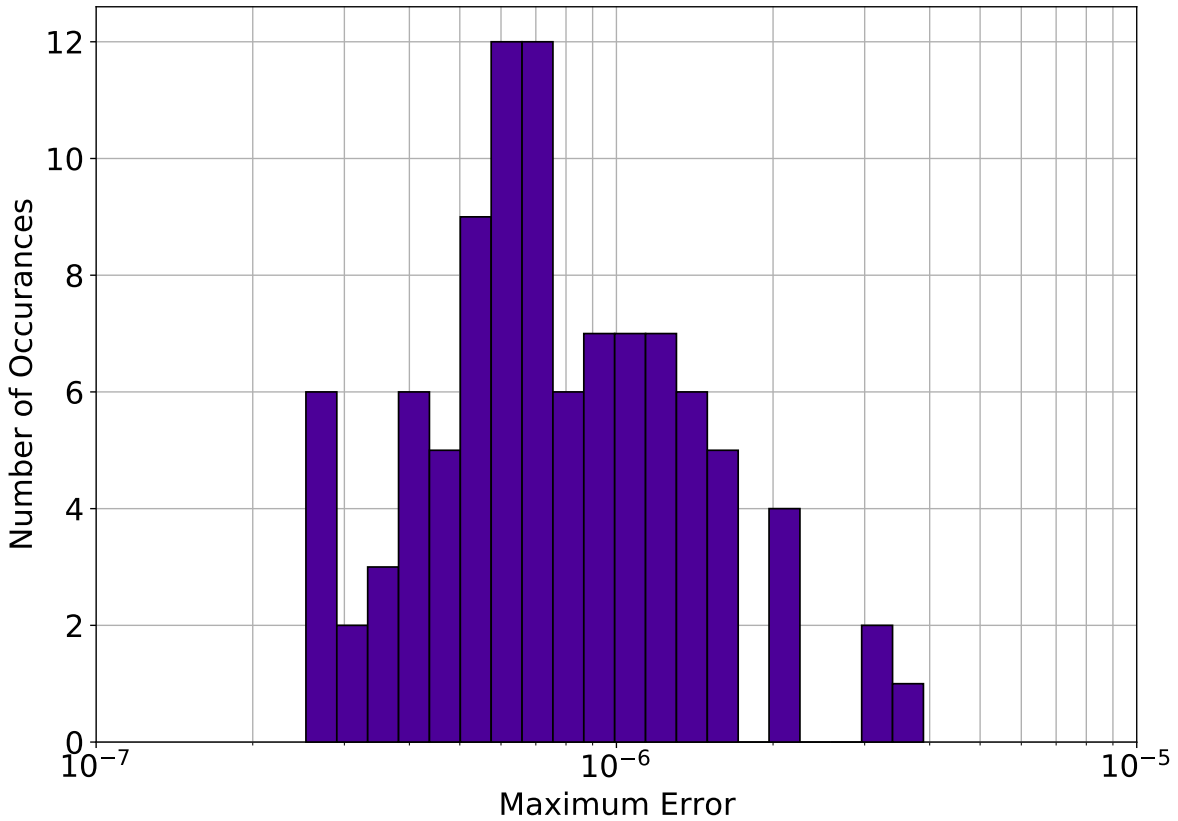
Figure 3.3: Histogram of the Deep-TFC maximum solution error on the test set for 100 Monte Carlo trials.

Figure 3.3 shows that Deep-TFC produces a solution at least as accurate as the solution reported earlier approximately 10% of the time; this aligns well with one's intuition, as the reported solution was the best of ten trials. The remaining 90% of the

time the solution error is larger, but Figure 3.3 shows that the Deep-TFC method is consistent: the maximum solution error in the 100 Monte Carlo trials was 3.891×10^{-6}, only an order of magnitude larger than the maximum solution error reported earlier [3].

Example 3.5: Simple PDE solved using X-TFC

Let $g(x, y)$ be an ELM as defined earlier with nonlinear activation function $\psi = \tanh$. Similar to the solution that used basis functions, Example 3.2, selecting $g(x, y)$ in this way ultimately results in a loss function, $\mathbb{L}(W_2)$, that can be solved via linear least-squares. As in the previous examples, let the test set of points be a 100×100 uniform grid. Table 3.4 shows the maximum test set solution error using X-TFC, where m corresponds to the number of neurons in the hidden layer of the ELM: note that the number of neurons in each column of Table 3.4 coincides with the number of basis functions in the corresponding columns of Table 3.2 from Example 3.2.

Table 3.4: Maximum test set solution error using X-TFC with the tanh activation function.

n \ m	17	62	132	227	347
5	1.74×10^{-5}	-	-	-	-
10	4.44×10^{-6}	1.49×10^{-10}	-	-	-
15	4.12×10^{-6}	1.11×10^{-10}	1.21×10^{-12}	-	-
20	3.95×10^{-6}	5.80×10^{-11}	4.40×10^{-13}	2.37×10^{-13}	-
25	3.84×10^{-6}	5.96×10^{-11}	5.46×10^{-13}	2.10×10^{-13}	2.14×10^{-13}
30	3.77×10^{-6}	5.50×10^{-11}	5.55×10^{-13}	1.66×10^{-13}	1.83×10^{-13}

Comparing Tables 3.2 and 3.4 reveals that when the number of basis functions is low, X-TFC outperforms TFC by as much as two orders of magnitude; however, as the number of basis functions increases, TFC outperforms X-TFC by as much as three orders of magnitude.

Figure 3.4: Histogram of X-TFC maximum solution error where $n = 30$ and $m = 347$ on the test set for 100 Monte Carlo trials.

Similar to Deep-TFC, X-TFC is inherently stochastic. Hence, Figure 3.4 shows a histogram of 100 Monte Carlo trials of the final case: $n = 30$, $m = 347$. Figure 3.4 shows that the corresponding value given in Table 3.4 is actually at the higher end of the distribution; however, it should be noted here that the deviation between Monte

Carlo trials when using X-TFC is only as large as approximately 2×10^{-13}. This relative deviation is much smaller than when using Deep-TFC, where cases varied by as much as an order of magnitude.

The small relative difference between X-TFC cases can be attributed to the large number of neurons relative to the initial distribution. In each of the Monte Carlo trials, the weights and biases are chosen using $U(-1, 1)$. Since there are 347 neurons, the sample space is well represented each time. If fewer neurons were used, the relative difference between test cases would be larger. To illustrate, Figure 3.5 shows a histogram of 100 Monte Carlo trials when $n = 30$ and $m = 17$.

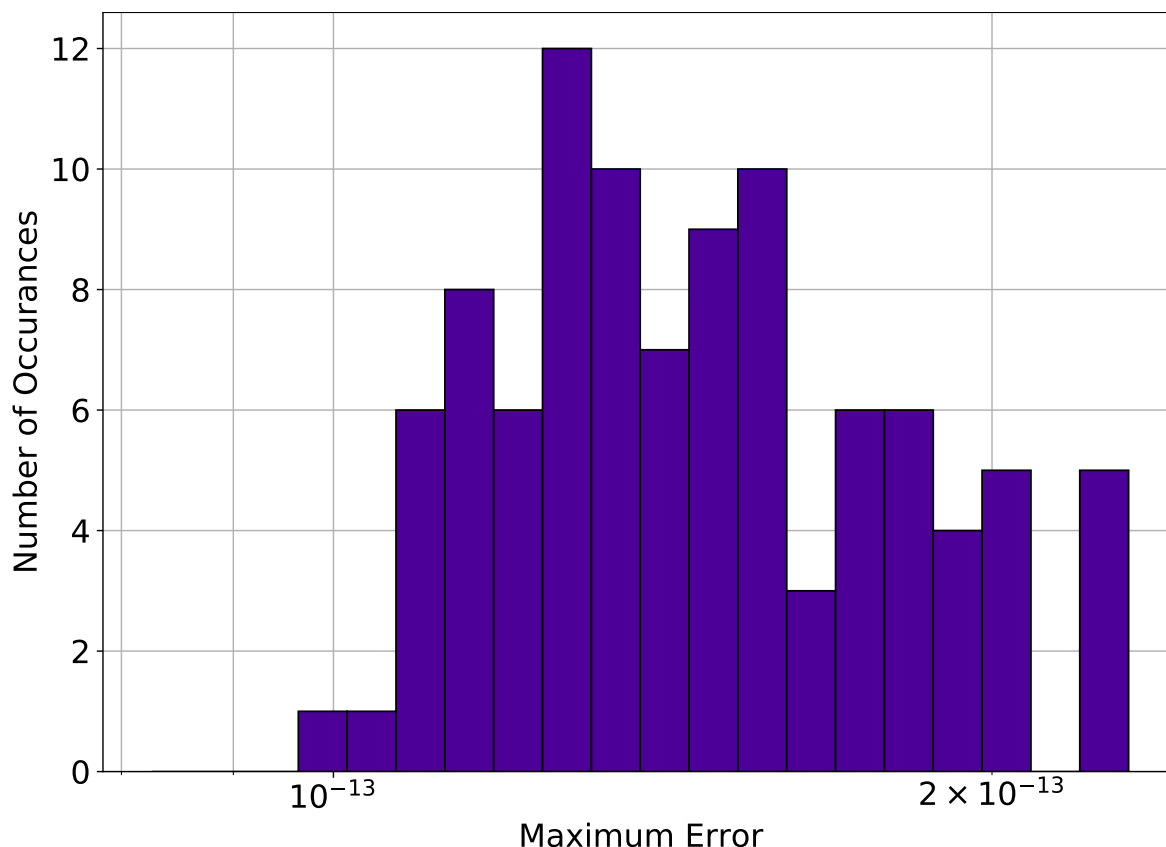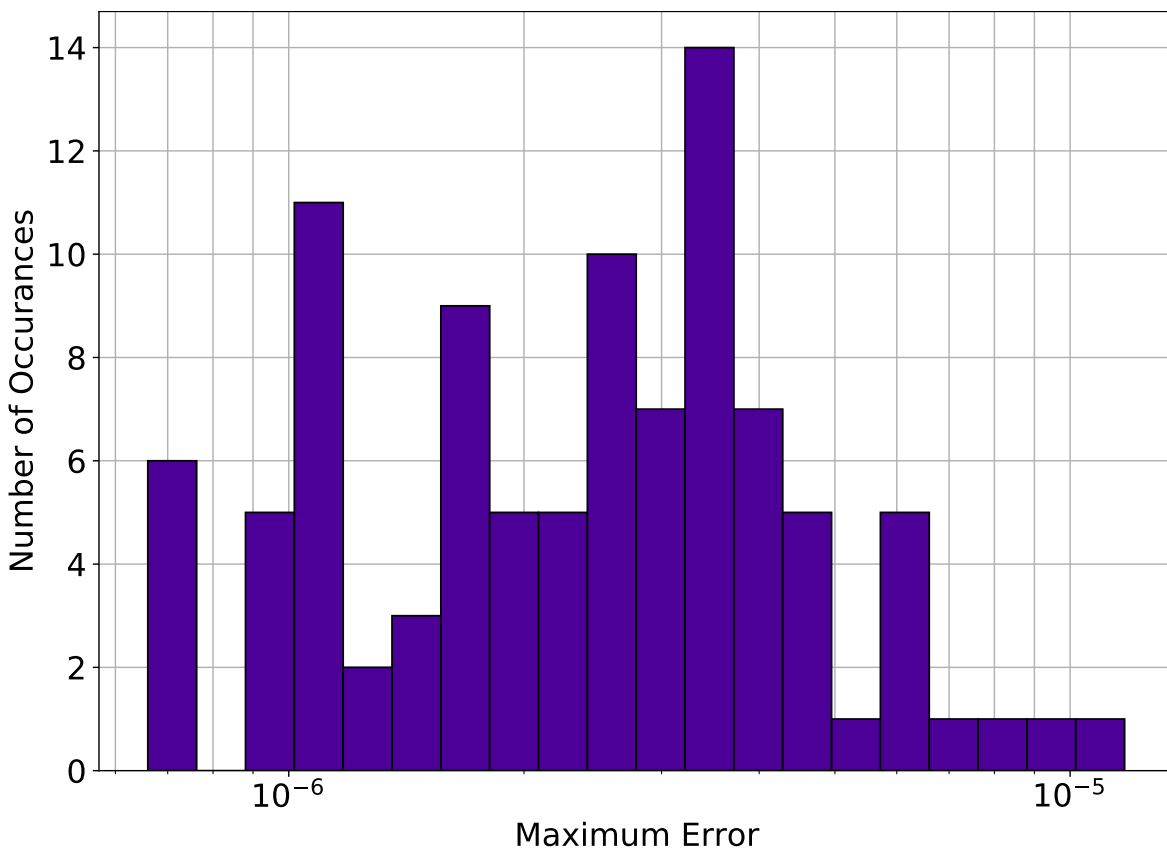

Figure 3.5: Histogram of X-TFC maximum solution error where $n = 30$ and $m = 17$ on the test set for 100 Monte Carlo trials.

In Figure 3.4, the maximum test set error in the worst case was approximately twice as large as the maximum test set error in the best case, whereas in Figure 3.5, the worst case is approximately ten times as large as the best case. Of course, other factors—such as the nonlinear relationship between the trainable parameters of Deep-TFC versus the linear relationship between the trainable parameters of X-TFC—play a role in the differences between the histograms of Figures 3.3 and 3.4 as well.

3.5 A Juxtaposition of TFC, CSVM, X-TFC, and Deep-TFC

The previous simple PDE examples—Examples 3.2 through 3.5—highlight some of the strengths and weaknesses of the four free function options discussed earlier; this section analyzes those strengths and weaknesses further. To aid in that analysis, Table 3.5 shows the maximum training and test set errors when using TFC and each of the four free function choices to solve the simple PDE. In addition, the solution errors of other state-of-the-art algorithms are included for reference: these algorithms include the well-known FEM, a neural-network-based method that analytically satisfies the constraints via a functional[3] [34], a Bernstein neural-network-based approach [65], and a Chebyshev neural-network-based approach [66]. Table 3.5 shows that the TFC methodology outperforms all the others in terms of accuracy on the training and test sets, followed by X-TFC. In terms of the test set error, these two methods are followed by CSVM, Deep-TFC, and then the other state-of-the-art algorithms. As for the training set error, FEM outperforms both Deep-TFC and CSVM, but the other state-of-the-art algorithms do not.

As mentioned earlier, the CSVM technique is no longer actively being used as a free function choice because it requires a complex analytical analysis for each new differential equation, and the resultant payoff in terms of solution error is overshadowed by the other free function choices. Example 3.3 demonstrates well the aforementioned complex analytical

---

[3]This functional is multiplicative in nature whereas the TFC constrained expression is additive in nature. Moreover, the multiplicative functional cannot satisfy certain sets of constraints, nor does it have the mathematical guarantees that TFC constrained expressions do.

Table 3.5: Comparison of maximum training set and test set errors between TFC methods and current state-of-the-art techniques.

Method	Training Set Maximum Error	Test Set Maximum Error
TFC [1]	2.22×10^{-16}	4.44×10^{-16}
X-TFC [49]	3.8×10^{-13}	5.1×10^{-13}
CSVM [2]	4.4×10^{-8}	5.6×10^{-8}
Deep-TFC [3]	2.7×10^{-7}	2.8×10^{-7}
FEM [34]	2×10^{-8}	1.5×10^{-5}
NN [34]	5×10^{-7}	5×10^{-7}
Bernstein NN [65]	-	2.4×10^{-4}
Chebyshev NN [66]	-	3.2×10^{-2}

analysis: the long expressions containing multiple 4-th and 5-th order tensors in the example are daunting enough, but the author reminds readers that these expressions are compact and simple compared to the five or so pages of work it took to derive them. Furthermore, Table 3.5 clearly shows the accuracy gained when using TFC or X-TFC rather than CSVM. In addition, because CSVM requires a grid search to find the two hyperparameters, the training time is longer than when using TFC or X-TFC. Although this free function choice has become antiquated, it laid the foundation for the synergy between TFC and machine learning algorithms and is therefore historically significant.

Looking at the error values alone in Table 3.5, the reader may wonder why Deep-TFC is useful. After all, one of the arguments against CSVM was the error relative to TFC and X-TFC; however, on more complex problems, Deep-TFC actually does better than X-TFC and TFC. Moreover, Deep-TFC does not require the same complex analytical analysis that CSVM did. To highlight Deep-TFC's performance on complex problems, consider low-speed, two-dimensional, developing channel flow governed by the Navier-Stokes equations and the

following boundary conditions:

$$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} = 0$$

$$\rho\left(\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y}\right) = -\frac{\partial P}{\partial x} + \mu\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right)$$

$$\rho\left(\frac{\partial v}{\partial t} + u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y}\right) = \mu\left(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2}\right)$$

subject to
$$\begin{cases} u(0,y,t) = \frac{\partial u}{\partial x}(L,y,t) = u(x,y,0) = 0 \\[2mm] u(x,\frac{H}{2},t) = u(x,-\frac{H}{2},t) = 0 \\[2mm] v(0,y,t) = \frac{\partial v}{\partial x}(L,y,t) = v(x,y,0) = 0 \\[2mm] v(x,\frac{H}{2},t) = v(x,-\frac{H}{2},t) = 0, \end{cases}$$

where u and v are velocities in the x and y directions respectively, H is the height of the channel, P is the pressure, ρ is the density, and μ is the viscosity. For this problem, the values $H = 1$ m, $\rho = 1$ kg/m^3, $\mu = 1$ Pa·s, and $\frac{\partial P}{\partial x} = -5$ N/m^3 were chosen.

The u and v dependent variables each have the same constraints; therefore, their constrained expressions are the same. Hence, just the constrained expression for u will be shown. In recursive form, the constrained expression for u is,

$$^{(1)}u(x,y,t,g^u(x,y,t)) = g^u(x,y,t) - g^u(0,y,t) - xg_x^u(L,y,t)$$
$$^{(2)}u(x,y,t,g^u(x,y,t)) = g^u(x,y,t) - \frac{H-2y}{2H}g^u\left(x,-\frac{H}{2},t\right) - \frac{H+2y}{2H}g^u\left(x,\frac{H}{2},t\right)$$
$$^{(3)}u(x,y,t,g^u(x,y,t)) = g^u(x,y,t) - g^u(x,y,0),$$

where $^{(1)}u$, $^{(2)}u$, and $^{(3)}u$ can be processed in any order to produce the full constrained expression. In tensor form, the constrained expression is,

$$u(x,y,t,g^u(x,y,t)) = g^u(x,y,t) + \mathcal{M}(x,y,t,g^u(x,y,t))_{ijk}\Phi_i(x)\Phi_j(y)\Phi_k(t)$$

where,

$$\mathcal{M}_{ij1}(x,y,t,g^u(x,y,t)) = \begin{bmatrix} 0 & -g^u(x,-\frac{H}{2},t) & -g^u(x,\frac{H}{2},t) \\ -g^u(0,y,t) & g^u(0,-\frac{H}{2},t) & g^u(0,\frac{H}{2},t) \\ -g^u_x(L,y,t) & g^u_x(L,-\frac{H}{2},t) & g^u_x(L,\frac{H}{2},t) \end{bmatrix}$$

$$\mathcal{M}_{ij2}(x,y,t,g^u(x,y,t)) = \begin{bmatrix} -g^u(x,y,0) & g^u(x,-\frac{H}{2},0) & g^u(x,\frac{H}{2},0) \\ g^u(0,y,0) & -g^u(0,-\frac{H}{2},0) & -g^u(0,\frac{H}{2},0) \\ g^u_x(L,y,0) & -g^u_x(L,-\frac{H}{2},0) & -g^u_x(L,\frac{H}{2},0) \end{bmatrix}$$

and

$$\Phi_i(x) = \left\{ 1, \quad 1, \quad x \right\}, \quad \Phi_j(y) = \left\{ 1, \quad \frac{H-2y}{2H}, \quad \frac{H+2y}{2H} \right\}, \quad \Phi_k(t) = \left\{ 1, \quad 1 \right\}.$$

For Deep-TFC, the training set used was $2,000$ independently and identically distributed (i.i.d.) points sampled from $x \in U(0,15)$, $y \in U(-H/2, H/2)$, and $t \in U(0,3)$. For X-TFC and TFC, the training set was a grid of $10 \times 10 \times 10$ uniformly spaced points. For each method, the test set consisted of a grid of 100×100 evenly spaced points in x and y at three different times: $t = 0.01$, $t = 0.1$ and $t = 3$. The test set can be analyzed in two different ways:

1. Qualitatively - The solution should be symmetric about the line $y = 0$, and the solution should develop spatially and temporally such that after a sufficient amount of time has passed and sufficiently far from the inlet, $x = 0$, the u-velocity will be equal, or very nearly equal, to the steady-state Poiseuille flow solution.

2. Quantitatively - The solution at $x = 15$ and $t = 3$ can be compared to the steady-state Poiseuille flow solution.

The neural network used for the Deep-TFC solution had four hidden layers and 30 neurons per layer, and the nonlinear activation function used was the hyperbolic tangent [3]. The X-TFC solution used 200 neurons and the hyperbolic tangent as the nonlinear activation function; adding additional neurons, up to 300, did not improve the solution over the case

with 200 neurons. The TFC solution used Chebyshev orthogonal polynomials up to degree ten; adding additional polynomials, up to degree 15, did not improve the solution over the case with polynomials up to degree ten.

A quantitative comparison of the three methods' errors with respect to the steady-state Poiseuille flow solution at $x = 15$ and $t = 3$ is shown in Table 3.6. In addition, a qualitative comparison is illustrated via Figures 3.6 through 3.14: Figures 3.6 through 3.8 correspond to the TFC solution, Figures 3.9 through 3.11 correspond to the X-TFC solution, and Figures 3.12 through 3.14 correspond to the Deep-TFC solution.

Table 3.6: Comparison of maximum and mean test set errors between TFC, X-TFC, and Deep-TFC.

Method	Test Set Maximum Error	Test Set Average Error
TFC	5.59×10^{-3}	3.68×10^{-3}
X-TFC	4.02×10^{-3}	1.89×10^{-3}
Deep-TFC [3]	5.38×10^{-4}	3.12×10^{-4}

Table 3.6 shows that the X-TFC solution does marginally better than the TFC solution in terms of error, and the Deep-TFC solution is approximately an order of magnitude better than X-TFC and TFC. This is reflected qualitatively in the figures as well.

The Deep-TFC figures, Figures 3.12 through 3.14, match the qualitative expectation given earlier. In contrast, the TFC and X-TFC solutions' figures do not: this difference is highlighted most in figures for $t = 3.0$.

In summation, this Navier-Stokes example demonstrates the utility of Deep-TFC as problems become sufficiently complex. These results coupled with those of Table 3.5 make good arguments for using Deep-TFC for complex problems and TFC for simpler problems. What about X-TFC?

For certain problems, X-TFC outperforms TFC: the two-dimensional wave equation in
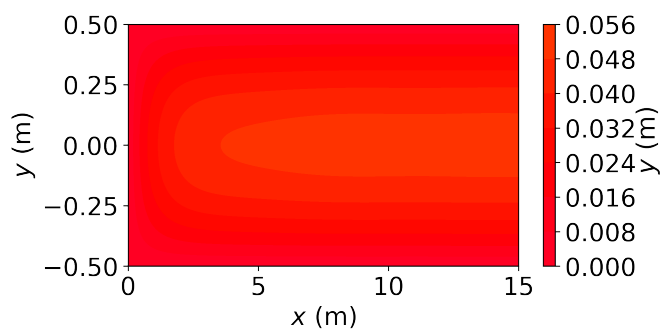
Figure 3.6: TFC solution at $t = 0.01$.
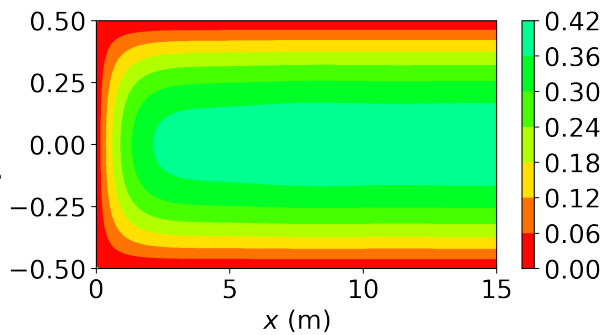
Figure 3.7: TFC solution at $t = 0.1$.
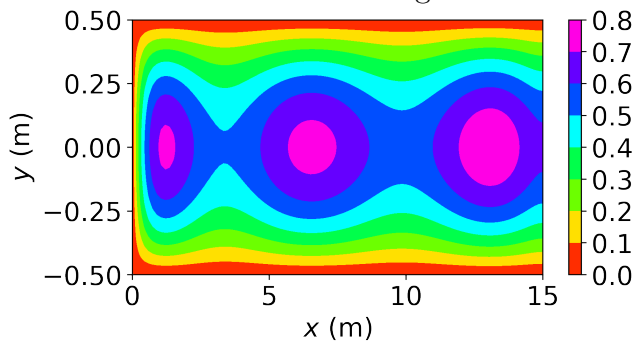
Figure 3.8: TFC solution at $t = 3.0$.
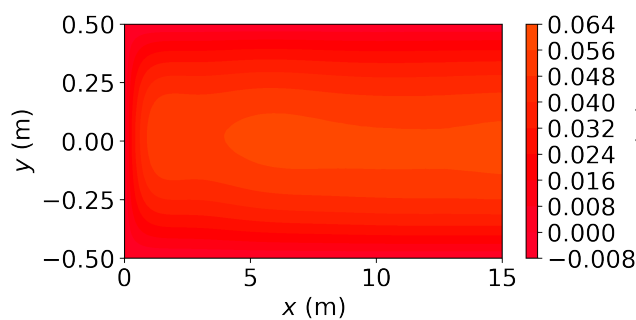
Figure 3.9: X-TFC solution at $t = 0.01$.
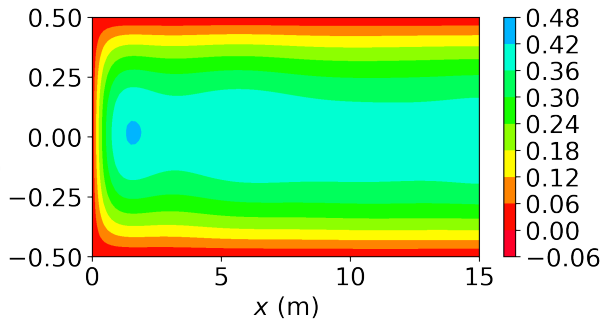
Figure 3.10: X-TFC solution at $t = 0.1$.
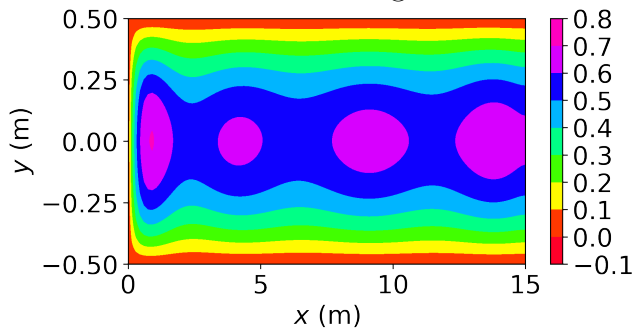
Figure 3.11: X-TFC solution at $t = 3.0$.

95
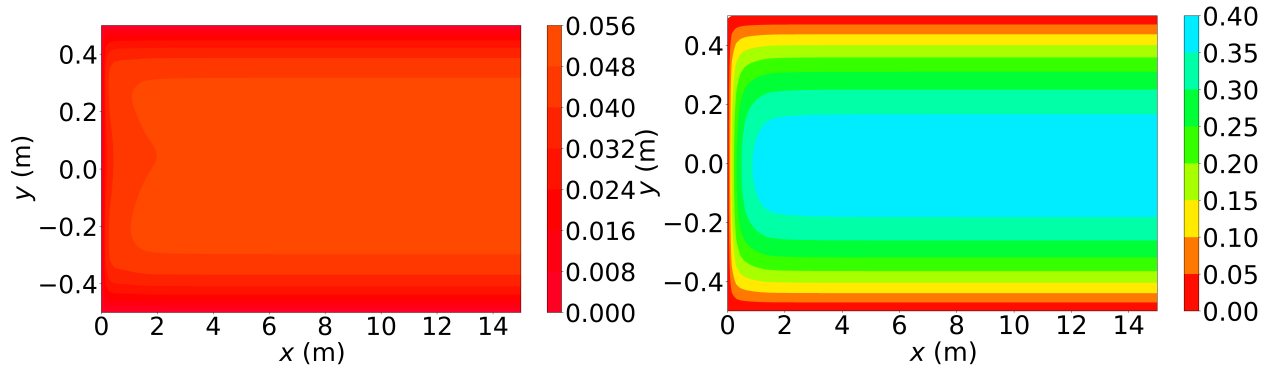
Figure 3.12: Deep-TFC solution at $t = 0.01$. Figure 3.13: Deep-TFC solution at $t = 0.1$.
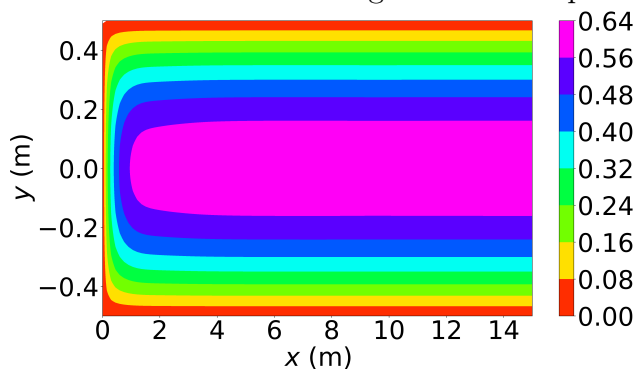
Figure 3.14: Deep-TFC solution at $t = 3.0$.

Section 4.2.2 is one example. Moreover, comparing the solution errors of the two methods—see Table 4.4 or compare the first two columns of Tables 3.2 and 3.4—shows that the difference between the two methods is the most significant for a low number of trainable parameters. This is another benefit of the X-TFC framework: lower solution error than other methods when a lower number of parameters is used.

Hence, for multidimensional problems in general, if a problem has a complex residual, use Deep-TFC; if the problem is simple, use X-TFC or TFC, and if the application is memory restrictive, i.e., a lower number of trainable parameters is required, use X-TFC. Oftentimes X-TFC or TFC is sufficient to estimate the solution, and the user is encouraged to try out both frameworks for their specific problem. In fact, the numerical implementation makes it extremely simple to switch between these two methods.

4. APPLICATIONS IN FLEXIBLE BODY PROBLEMS

The previous chapter explained how to apply TFC to differential equations. Consequently, TFC can be applied to a wide variety of problems spanning multiple fields and multiple disciplines within those fields. However, the author is particularly interested in some of the differential equations appearing in flexible body problems, and therefore, this chapter is dedicated to them exclusively. The flexible body problems solved in this chapter include:

- Natural tandem balloon shape - A set of four, coupled, first-order, nonlinear ODEs wherein both ends of the domain are themselves unknowns that must be solved simultaneously alongside the ODEs.

- One-dimensional wave equation - A two-dimensional (one space, one time), second-order, linear PDE.

- Two-dimensional wave equation - A three-dimensional (two space, one time), second-order, linear PDE.

- Biharmonic equation, Cartesian coordinates - A two-dimensional, fourth-order, linear PDE.

- Biharmonic equation, polar coordinates - A two-dimensional, fourth-order, linear PDE.

4.1   Natural Tandem Balloon Shape

Tandem balloons are useful scientific vessels for collecting terrestrial atmospheric data and are being considered for the same task on other planets and moons [67]. Naturally, the shape of these balloons is a critical component involved in simulating their trajectories. This section will provide a general overview of the problem [68]; a more detailed description can be found in References [69] and [70]. Figure 4.1 is a diagram of the tandem balloon and the

coordinate system used to describe its shape. Table 4.1 provides the nomenclature used to describe the tandem balloon.
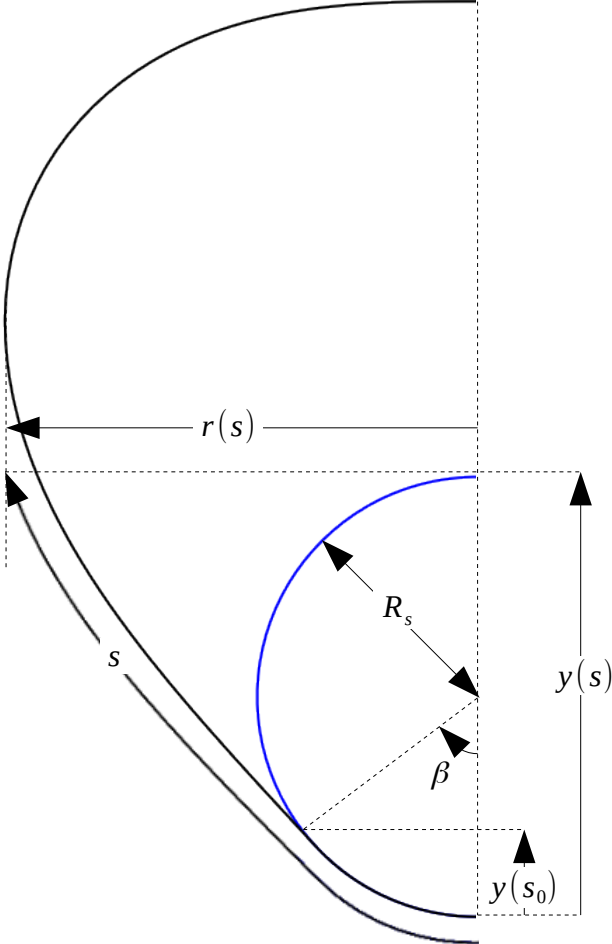
Figure 4.1: Tandem balloon diagram.

The differential equations that govern the balloon are summarized in Equation (4.1)

Table 4.1: Tandem balloon nomenclature.

Symbol	Description
A_s	Surface area of the super-pressure balloon
M_{atm}	Molecular weight of the atmosphere
M_g	Molecular weight of the lifting gas
R_s	Radius of the super-pressure balloon
T_0	Total vertical load felt at s_0
V_s	Volume of the super-pressure balloon
b	Specific density of the lifting gas
g	Acceleration due to gravity
m_{sg}	Total mass of the gas in the super pressure balloon
r	Coordinate that describes position perpendicular to axisymmetric line
s	Coordinate that describes position along the balloon film
s_0	Point where the zero-pressure balloon and super-pressure balloon come into contact
w	Zero-pressure balloon film mass per unit area
w_s	Super-pressure balloon film mass per unit area
y	Coordinate that describes position parallel to axisymmetric line
β	Angle measured from the center of the super-pressure balloon between the vertical and the point where the zero-pressure balloon and super-pressure balloon surfaces diverge
ℓ_d	Length of the balloon film
ρ	Atmospheric density
σ_c	Circumferential stress
σ_m	Meridional stress
θ	Angle in $[-\frac{\pi}{2}, \frac{\pi}{2}]$ between the vertical and a line tangent to the balloon's surface

[69, 70].

$$\frac{\mathrm{d}\theta}{\mathrm{d}s} = \frac{1}{\sigma_m}\left(\frac{\sigma_c}{r}\cos(\theta) - w\sin(\theta) - b\big(y - y(s_0)\big)\right)$$

$$\frac{\mathrm{d}\sigma_m}{\mathrm{d}s} = \frac{\sigma_c}{r}\sin(\theta) + w\cos(\theta) - \frac{\sigma_m}{r}\sin(\theta)$$

$$\frac{\mathrm{d}r}{\mathrm{d}s} = \sin(\theta)$$

$$\frac{\mathrm{d}y}{\mathrm{d}s} = \cos(\theta),$$

(4.1)

subject to the boundary constraints,

$$s_0 = R_s\beta$$

$$\theta(s_0) = \frac{\pi}{2} - \beta$$

$$\sigma_m(s_0) = \frac{T_0}{2\pi r \sin(\theta)}$$

$$r(s_0) = R_s \sin(\beta)$$

$$y(s_0) = R_s(1 - \cos(\beta))$$

$$\theta(\ell_d) = -\frac{\pi}{2}$$

$$r(\ell_d) = 0,$$

where

$$T_0 = L + g(w + w_s)A_{s0} + g\left(\frac{V_{s0}}{V_s}m_{sg} - \rho V_{s0}\right)$$

$$\text{if } \beta < \frac{\pi}{2} \rightarrow \begin{cases} A_{s0} = 2\pi R_s y(s_0) \\ \\ V_{s0} = \frac{\pi}{3}z^2(s_0)\left(3R_s - y(s_0)\right) \end{cases}$$

$$\text{if } \beta \geq \frac{\pi}{2} \rightarrow \begin{cases} h_0 = 2R_s - y(s_0) \\ A_{s0} = A_s - 2\pi R_s h_0 \\ V_{s0} = V_s - \frac{\pi}{3}h_0\left(3R_s - h_0\right) \end{cases}$$

$$A_s = 4\pi R_s^2$$

$$V_s = \frac{4}{3}\pi R_s^3$$

and

$$b = g\rho\left(1 - \frac{M_g}{M_{atm}}\right).$$

The simplest version of these differential equations is the natural balloon shape, which has zero circumferential stress, i.e., $\sigma_c = 0$. Even with this simplification, the set of coupled, nonlinear differential equations is challenging to solve as the problem domain, $s \in [s_0, \ell_d]$, is

variable on both ends: β and ℓ_d are variables to be solved alongside the differential equations, i.e., s_0 and ℓ_d are unknown. However, since TFC must map the free function domain to the problem domain anyway, the mapping parameter can be used in the least-squares when reducing the residual.

The meridional stress generates another complication, as $r(\ell_d) = 0$ and $\pm \frac{\mathrm{d}\sigma_m}{\mathrm{d}s} \to \infty$ as $r \to 0$ for non-zero σ_m, where the sign, $\pm$, depends on the sign of σ_m. Of course, this singularity does not exist in real life, as an infinite stress would rip the balloon apart, rather, it is due to the coordinate system chosen and assumptions made when deriving the differential equations. Fortunately, there is a change of variables that prevents a singularity in the dependent variables [69]. Let $q = \frac{1}{\sigma_m r}$, then, Equation (4.1) can be rewritten as,

$$
\begin{aligned}
\frac{\mathrm{d}\theta}{\mathrm{d}s} &= q\sigma_c \cos(\theta) - qrw \sin(\theta) - qrb(y - y(s_0)) \\
\frac{\mathrm{d}q}{\mathrm{d}s} &= -q^2 \Big(\sigma_c \sin(\theta) + wr \cos(\theta) \Big) \\
\frac{\mathrm{d}r}{\mathrm{d}s} &= \sin(\theta) \\
\frac{\mathrm{d}y}{\mathrm{d}s} &= \cos(\theta).
\end{aligned}
$$

The constrained expressions that embed the boundary constraints given previously are shown in Equation (4.2). Note that theses constrained expressions are written for the domain of the free function, because as mentioned earlier, the differential equations will ultimately be written in terms of the free function domain, z, so the mapping parameter can be used to solve for β and ℓ_d. For this problem, Chebyshev orthogonal polynomials are used, which have a domain of $z \in [-1, 1]$. Further, note that the domain is the only part of the problem being modified, so the right-hand side of the boundary conditions remains unchanged, e.g.,

$\theta(z = -1) = \theta(s_0)$.

$$\theta(z, g^\theta(z)) = g^\theta(z) + \frac{1-z}{2}\left(\theta(s_0) - g^\theta(-1)\right) + \frac{z+1}{2}\left(\theta(\ell_d) - g^\theta(1)\right)$$

$$r(z, g^r(z)) = g^r(z) + \frac{1-z}{2}\left(r(s_0) - g^r(-1)\right) + \frac{z+1}{2}\left(r(\ell_d) - g^r(1)\right) \tag{4.2}$$

$$q(z, g^q(z)) = g^q(z) + q(s_0) - g^q(-1)$$

$$y(z, g^y(z)) = g^y(z) + y(s_0) - g^y(-1)$$

The differential equations written on this domain are,

$$c\frac{\mathrm{d}\theta}{\mathrm{d}z} = q\sigma_c\cos(\theta) - qrw\sin(\theta) - qrb(y - y(s_0))$$

$$c\frac{\mathrm{d}q}{\mathrm{d}z} = -q^2\left(\sigma_c\sin(\theta) + wr\cos(\theta)\right)$$

$$c\frac{\mathrm{d}r}{\mathrm{d}z} = \sin(\theta)$$

$$c\frac{\mathrm{d}y}{\mathrm{d}z} = \cos(\theta),$$

where $c(\beta, \ell_d) = 2/\left(\ell_d - s_0(\beta)\right)$ is the mapping parameter. In this form, the residuals of the differential equations include β and ℓ_d as well as the $\boldsymbol{\xi}$ vectors for each of the independent variables. Therefore, the equations are in a form such that an optimization technique can be used to minimize the residuals: in this case, nonlinear least-squares was used.

Consider the Venus atmospheric data in Table 4.2, which was collected as part of the Venus Variable Altitude Aerobot project at JPL, and let the balloon constants be those given in Table 4.3. Using this data and the differential equations above, TFC was used to solve the natural balloon shapes: the results are shown in Figure 4.2. The average solution time for the natural balloon shapes in Figure 4.2 was 0.65 seconds, and the residual of the differential equation at all points was $\mathcal{O}(10^{-15})$. To compare, solving the same problem in Octave via a shooting method that uses `fzero` and `ode45` takes an average of 9.85 seconds per case. Of course, this comparison is not exactly one-to-one, because as mentioned earlier, TFC is implemented in JAX, and many of the functions have been JIT-ed.

Table 4.2: Tandem balloon atmospheric data.

Altitude (km)	Atmospheric Density ($\frac{kg}{m^3}$)	Super Pressure Balloon Gas Mass (kg)	Gravity ($\frac{m}{s^2}$)
52	1.28	11.62	8.719
53	1.15	10.74	8.716
54	1.03	9.97	8.713
55	0.921	9.29	8.71
56	0.818	8.67	8.707
57	0.721	8.12	8.704
58	0.629	7.58	8.702
59	0.545	7.14	8.699
60	0.469	6.812	8.696
61	0.41	6.675	8.693
62	0.341	6.2675	8.69

Table 4.3: Tandem balloon constants.

Parameter	Value and Units
$w \left(\frac{kg}{m^2} \right)$	0.095
$w_s \left(\frac{kg}{m^2} \right)$	0.215
$M_g \left(\frac{kg}{mol} \right)$	4×10^{-3}
$M_{atm} \left(\frac{kg}{mol} \right)$	4.34×10^{-2}
L (N)	$208g$

Another classic balloon shape scenario is one wherein the circumferential stress is constant. In this case, one can trade the unknown ℓ_d for σ_c if desired, i.e., the balloon has a fixed material length, and the circumferential stress is some unknown constant. Let the fixed material length $\ell_d = 18$ meters; Figure 4.3 shows the balloon shapes for this case. The average solution time for each case in Figure 4.3 was 0.75 seconds, and the residual of the differential equation at all points was $\mathcal{O}(10^{-15})$. Solving the same problem in Octave via a shooting method that uses `fsolve` and `ode45` takes an average of 45.7 seconds per case.
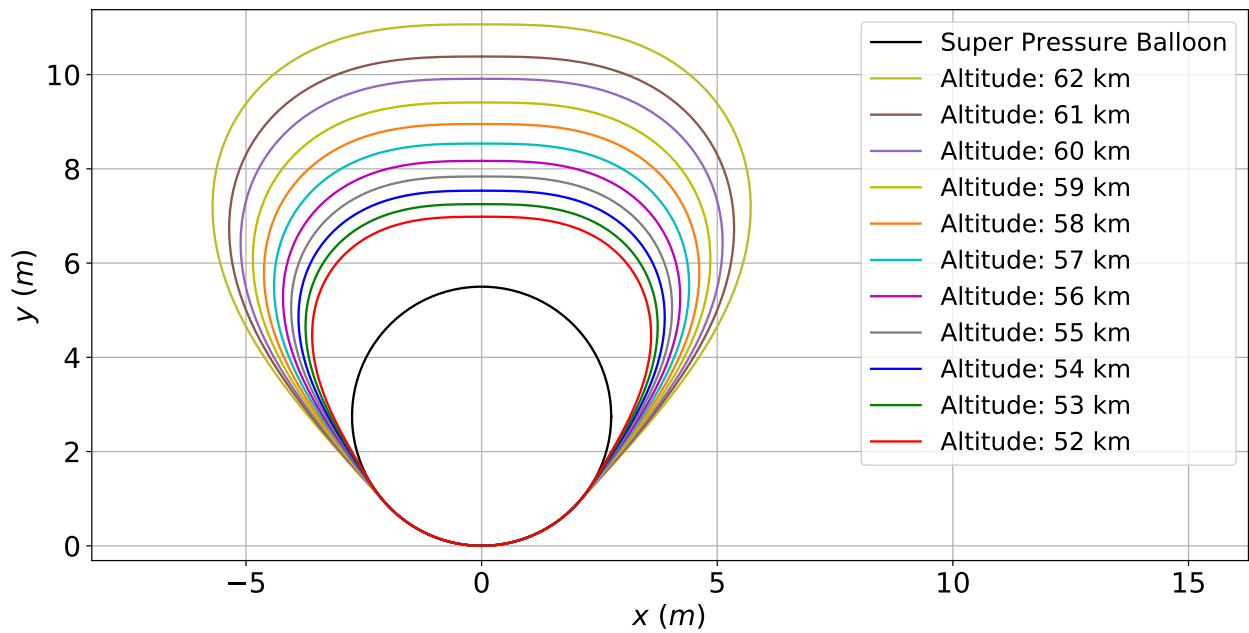
Figure 4.2: Natural balloon shapes on Venus for a range of altitudes from 52 km to 62 km.
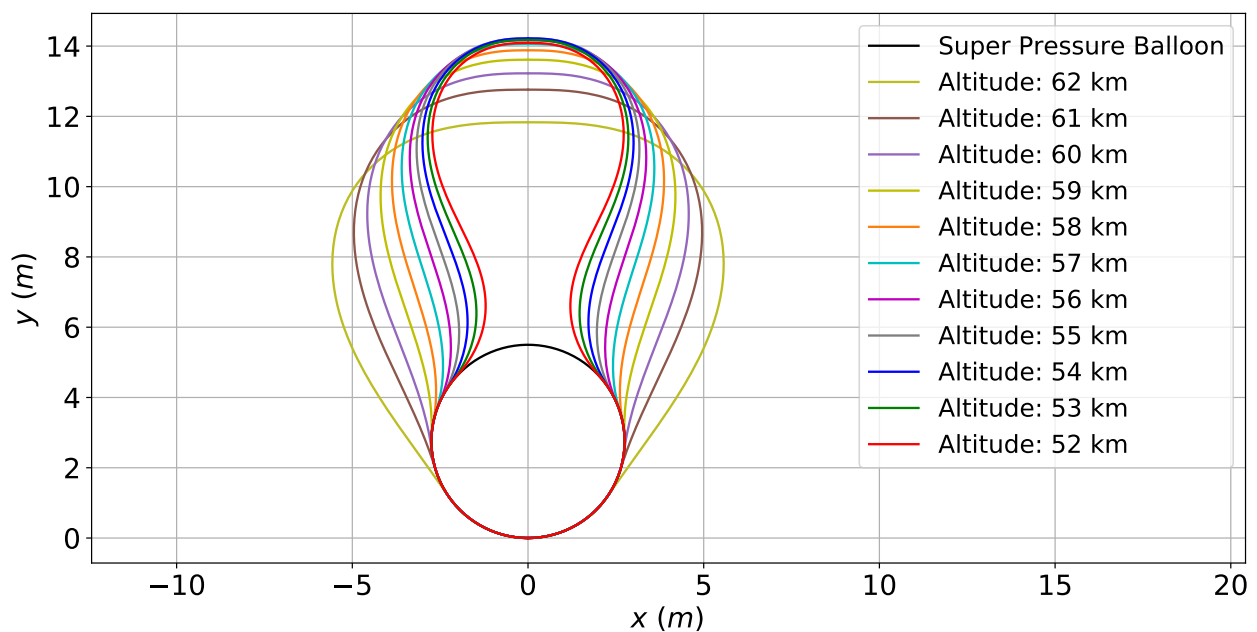
Figure 4.3: Constant circumferential stress balloon shapes with $\ell_d = 18$ meters on Venus for a range of altitudes from 52 km to 62 km.

4.2 Wave Equation

The wave equation is a well-known PDE that describes the propagation of waves, such as those found in a vibrating string or n-dimensional membrane. This section applies TFC to the one-dimensional (one spatial dimension and one time dimension) wave equation and the two-dimensional (two spatial dimensions and one time dimension) wave equation.

4.2.1 One-Dimensional Wave Equation

Consider the wave equation for a one-dimensional object,

$$u_{xx} = k^2 u_{tt},$$

for some constant k on the domain $(x, t) \in [0, 1] \times [0, 1]$ with the following boundary conditions,

$$u(0, t) = 0, \quad u(1, t) = 0, \quad u(x, 0) = \sin(\pi x), \quad \text{and} \quad u_t(x, 0) = 0.$$

One can physically imagine these boundary conditions as describing the string on a musical instrument that is fixed at both ends and free to vibrate with initial displacement $u(x, 0) = \sin(\pi x)$. Let the constant $k = 1$; then, the analytical solution is,

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

The analytical solution is shown in Figure 4.4.

The constrained expression written in recursive form is,

$$^{(1)}u(x, t, g(x, t)) = g(x, t) - (1 - x)g(0, t) - xg(1, t)$$

$$^{(2)}u(x, t, g(x, t)) = g(x, t) + \sin(\pi x) - g(x, 0) - tg_t(x, 0)$$

where $^{(1)}u$ can be used as the free function in $^{(2)}u$ or $^{(2)}u$ can be used as the free function in $^{(1)}u$ to create the full constrained expression. The constrained expression written in tensor
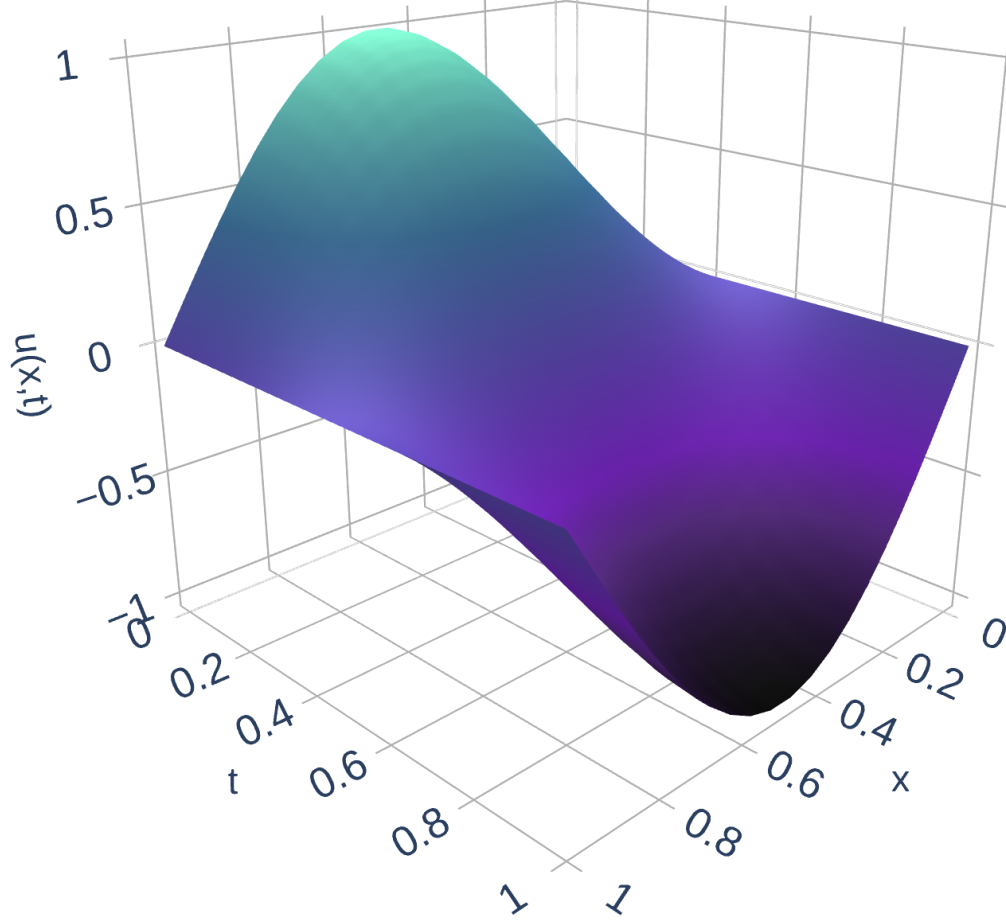
Figure 4.4: Analytical solution for the one-dimensional wave equation.

form is,

$$u(x, t, g(x, t)) = g(x, t) + \mathcal{M}_{ij}(x, t, g(x, t))\Phi_i(x)\Phi_j(t),$$

where

$$\mathcal{M}_{ij}(x, t, g(x, t)) = \begin{bmatrix} 0 & \sin(\pi x) - g(x, 0) & -g_t(x, 0) \\ -g(0, t) & g(0, 0) & g_t(0, 0) \\ -g(1, t) & g(1, 0) & g_t(1, 0) \end{bmatrix},$$

$$\Phi_i(x) = \left\{ 1, \quad 1 - x, \quad x \right\} \quad \text{and} \quad \Phi_j(t) = \left\{ 1, \quad 1, \quad t \right\}.$$

Using Legendre orthogonal polynomials up to degree 20 as the free function and a grid of 30×30 training points, the PDE solution was estimated using the TFC method. The

solution was obtained in 0.49 seconds, and the average error on a test set of 100×100 evenly spaced training points was 1.044×10^{-15}.

4.2.2 Two-Dimensional Wave Equation

The two-dimensional wave equation can be used to describe objects such as a flexible two-dimensional membrane. Consider such a membrane clamped at all sides with an initial deformation $u(x, y, 0) = \sin(\pi x) \sin(\pi y)$. Then, the governing PDE can be written as,

$$u_{xx} + u_{yy} = k^2 u_{tt},$$

on the domain $(x, t) \in [0, 1] \times [0, 1] \times [0, 1]$ with the following boundary conditions,

$$u(0, y, t) = 0, \quad u(1, y, t) = 0, \quad u(x, 0, t) = 0, \quad u(x, 1, t) = 0,$$

$$u(x, y, 0) = \sin(\pi x) \sin(\pi y), \quad \text{and} \quad u_t(x, y, 0) = 0.$$

Let $k = 8$, then the analytical solution is,

$$u(x, y, t) = \sin(\pi x) \sin(\pi y) \cos\left(\frac{\pi \sqrt{2}}{8} t\right).$$

The analytical solution at $t = 0.5$ is shown in Figure 4.5.

The constrained expression written in recursive form is,

$$^{(1)}u(x, y, t, g(x, y, t)) = g(x, y, t) - (1 - x)g(0, y, t) - xg(1, y, t)$$

$$^{(2)}u(x, y, t, g(x, y, t)) = g(x, y, t) - (1 - y)g(x, 0, t) - yg(x, 1, t)$$

$$^{(3)}u(x, y, t, g(x, y, t)) = g(x, y, t) + \sin(\pi x) \sin(\pi y) - g(x, y, 0) - tg_t(x, y, 0)$$

where $^{(1)}u$, $^{(2)}u$, and $^{(3)}u$ can be processed in any order to produce the full constrained

107
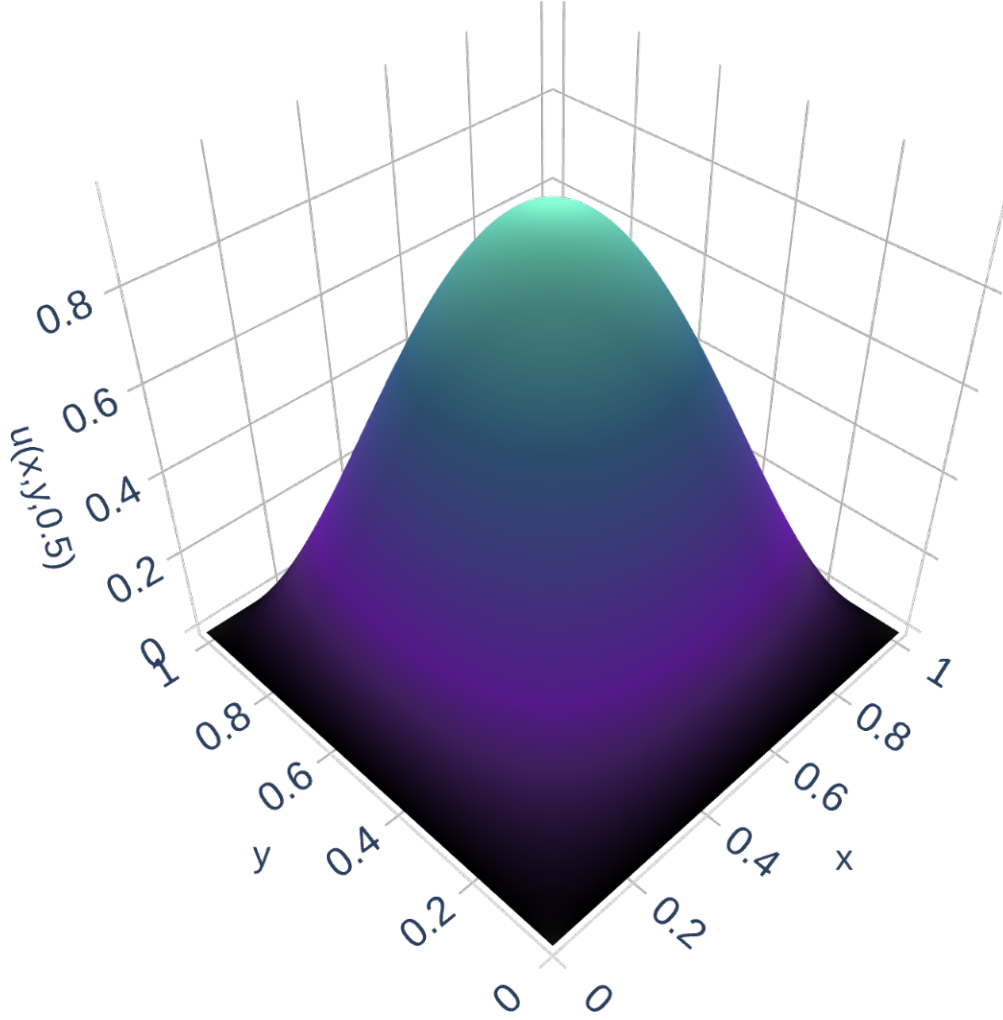
Figure 4.5: Two-dimensional wave equation analytical solution at $t = 0.5$.

expression. The tensor form of the constrained expression is,

$$u(x, y, t, g(x, y, t)) = g(x, y, t) + \mathcal{M}_{ijk}(x, y, t, g(x, y, t))\Phi_i(x)\Phi_j(y)\Phi_j(t),$$

where

$$\mathcal{M}_{ij1}(x, y, t, g(x, y, t)) = \begin{bmatrix} 0 & -g(x, 0, t) & -g(x, 1, t) \\ -g(0, y, t) & g(0, 0, t) & g(0, 1, t) \\ -g(1, y, t) & g(1, 0, t) & g(1, 1, t) \end{bmatrix}$$

$$\mathcal{M}_{ij2}(x,y,t,g(x,y,t)) = \begin{bmatrix} \sin(\pi x)\sin(\pi y) - g(x,y,0) & g(x,0,0) & g(x,1,0) \\ g(0,y,0) & -g(0,0,0) & -g(0,1,0) \\ g(1,y,0) & -g(1,0,0) & -g(1,1,0) \end{bmatrix}$$

$$\mathcal{M}_{ij3}(x,y,t,g(x,y,t)) = \begin{bmatrix} -g_t(x,y,0) & g_t(x,0,0) & g_t(x,1,0) \\ g_t(0,y,0) & -g_t(0,0,0) & -g_t(0,1,0) \\ g_t(1,y,0) & -g_t(1,0,0) & -g_t(1,1,0) \end{bmatrix}$$

and

$$\Phi_i(x) = \left\{ 1, \quad 1-x, \quad x \right\},$$
$$\Phi_j(y) = \left\{ 1, \quad 1-y, \quad y \right\},$$
$$\Phi_k(t) = \left\{ 1, \quad 1, \quad t \right\}.$$

For the two-dimensional wave equation, choosing ELMs as the free function led to a better estimate of the solution than Chebyshev or Legendre orthogonal polynomials on average: although the two were similar. To illustrate, Table 4.4 shows the maximum and mean errors when using the two methods for different numbers of basis functions: the number of basis functions corresponds to the number of Chebyshev polynomials there are on this problem when keeping all polynomials up to degree 3, 6, 9, 12, 15, and 18. The TFC method used Chebyshev orthogonal polynomials, and the X-TFC method used the hyperbolic tangent as the activation function. Each method used a grid of $11 \times 11 \times 11$ training points and a test set of $15 \times 15 \times 15$ uniformly spaced points.

Table 4.4 shows that while the two methods are similar, the X-TFC method performs slightly better. The differences between the two methods are the most pronounced when a lower number of basis functions is used. Consequently, ELMs were used to estimate the

Table 4.4: TFC and X-TFC solution errors for various numbers of basis functions when solving the two-dimensional wave equation.

m	TFC		X-TFC	
	Maximum Error	Mean Error	Maximum Error	Mean Error
12	5.32	5.16×10^{-1}	6.24×10^{-3}	6.98×10^{-4}
76	8.07×10^{-3}	1.04×10^{-3}	4.89×10^{-3}	4.56×10^{-4}
212	1.64×10^{-1}	1.55×10^{-2}	2.42×10^{-3}	2.29×10^{-4}
447	2.22×10^{-2}	2.09×10^{-3}	9.34×10^{-3}	8.79×10^{-4}
808	3.91×10^{-3}	3.67×10^{-4}	3.32×10^{-3}	3.12×10^{-4}
1322	3.90×10^{-3}	3.67×10^{-4}	3.34×10^{-3}	3.02×10^{-4}

solution of this differential equation. For one particular run using 650 neurons, the solution was obtained in 18.4 seconds, and the average solution error on the test set was 2.124×10^{-5}.

4.3 Biharmonic Equation

The biharmonic equation is a fourth-order linear PDE that appears in linear elasticity theory [71]. The PDE is given by,

$$\nabla^4 u(\boldsymbol{x}) = \nabla^2(\nabla^2 u(\boldsymbol{x})) = f(\boldsymbol{x}),$$

where u is the dependent variable of interest, f is a forcing term, and ∇^2 is the Laplacian operator. In two-dimensional plate problems, the variable u is related to the stress experienced by the plate, and $f(\boldsymbol{x})$ is related to the body forces acting on the plate.

4.3.1 Cartesian Coordinates

Consider the following forcing function,

$$\nabla^4 u(x, y) = 4\pi^2 \sin(\pi x) \sin(\pi y),$$

on the domain $(x, y) \in [0, 1] \times [0, 1]$ with the following boundary conditions,

$$u(0, y) = u(1, y) = u(x, 0) = u(y, 0) = 0$$

110

$$u_{xx}(0, y) = u_{xx}(1, y) = u_{yy}(x, 0) = u_{yy}(y, 0) = 0$$

The analytical solution to this problem is [72, 73],

$$u(x, y) = \frac{1}{\pi^2} \sin(\pi x) \sin(\pi y),$$

and is shown in Figure 4.6.
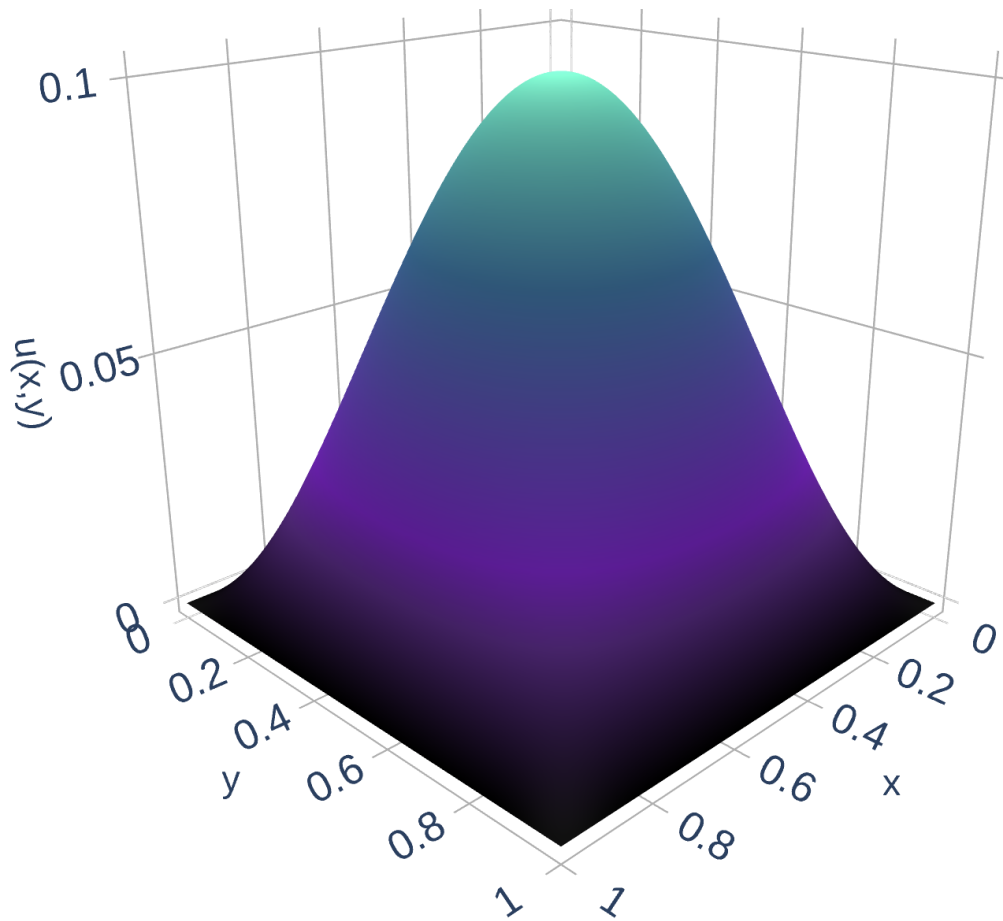
Figure 4.6: Biharmonic equation analytical solution in Cartesian coordinates.

The constrained expression for this problem in recursive form is,

$$^{(1)}u(x, y, g(x, y)) = g(x, y) - (1 - x)g(0, y) - xg(1, y)$$

$$- \frac{-x^3 + 3x^2 - 2x}{6} g_{xx}(0, y) - \frac{x^3 - x}{6} g_{xx}(1, y),$$

$$^{(2)}u(x, y, g(x, y)) = g(x, y) - (1 - y)g(x, 0) - yg(x, 1)$$

$$- \frac{y^3 - y}{6} g_{yy}(x, 1) - \frac{-y^3 + 3y^2 - 2y}{6} g_{yy}(x, 0),$$

where $^{(1)}u$ can be used as the free function in $^{(2)}u$ or $^{(2)}u$ can be used as the free function in $^{(1)}u$ to create the full constrained expression. In tensor form the constrained expression is,

$$u(x, y, g(x, t)) = g(x, y) + \mathcal{M}_{ij}(x, y, g(x, y))\Phi_i(x)\Phi_j(y),$$

where

$$\mathcal{M}_{ij}(x, y, g(x, y)) = \begin{bmatrix} 0 & -g(x, 0) & -g(x, 1) & -g_{yy}(x, 0) & -g_{yy}(x, 1) \\ -g(0, y) & g(0, 0) & g(0, 1) & g_{yy}(0, 0) & g_{yy}(0, 1) \\ -g(1, y) & g(1, 0) & g(1, 1) & g_{yy}(1, 0) & g_{yy}(1, 1) \\ -g_{xx}(0, y) & g_{xx}(0, 0) & g_{xx}(0, 1) & g_{xxyy}(0, 0) & g_{xxyy}(0, 1) \\ -g_{xx}(1, y) & g_{xx}(1, 0) & g_{xx}(1, 1) & g_{xxyy}(1, 0) & g_{xxyy}(1, 1) \end{bmatrix}$$

and

$$\Phi_i(x) = \left\{ 1, \quad 1 - x, \quad x, \quad \tfrac{-x^3 + 3x^2 - 2x}{6}, \quad \tfrac{x^3 - x}{6} \right\},$$

$$\Phi_i(y) = \left\{ 1, \quad 1 - y, \quad y, \quad \tfrac{-y^3 + 3y^2 - 2y}{6}, \quad \tfrac{y^3 - y}{6} \right\}.$$

Using Chebyshev orthogonal polynomials up to degree 26 as the free function and a grid of 20×20 training points, the PDE solution was estimated using the TFC method. The solution was obtained in 0.94 seconds, and the average error on a test set of 100×100

112

uniformly spaced points was 1.661×10^{-16}.

4.3.2 Polar Coordinates

Consider the following forcing function,

$$\nabla^4 u(r, \theta) = 0,$$

on the domain $(r, \theta) \in [1, 4] \times [0, 2\pi]$ with the following boundary conditions,

$$u(1, \theta) = \frac{1}{4}\sin(2\theta) + \frac{1}{16}\sin(3\theta) + \pi\cos(\theta) + \frac{1}{8}$$

$$u(4, \theta) = 4\sin(2\theta) + 4\sin(3\theta) + \frac{1}{4}\pi\cos(\theta) + 2$$

$$u_{rr}(1, \theta) = \frac{1}{2}\sin(2\theta) + \frac{3}{8}\sin(3\theta) + 2\pi\cos(\theta) + \frac{1}{4}$$

$$u_{rr}(4, \theta) = \frac{1}{2}\sin(2\theta) + \frac{3}{2}\sin(3\theta) + \frac{1}{32}\pi\cos(\theta) + \frac{1}{4}$$

$$u(r, 0) = u(r, 2\pi)$$

$$u_\theta(r, 0) = u_\theta(r, 2\pi)$$

$$u_{\theta\theta}(r, 0) = u_{\theta\theta}(r, 2\pi)$$

$$u_{\theta\theta\theta}(r, 0) = u_{\theta\theta\theta}(r, 2\pi).$$

The analytical solution to this problem is [74],

$$u(r, \theta) = \frac{r^3}{16}\sin(3\theta) + \frac{r^2}{4}\sin(2\theta) + \frac{r^2}{8} + \frac{\pi\cos(\theta)}{r},$$

and is shown in Figure 4.7. Note that the Laplacian in polar coordinates is,

$$\nabla^2 u(r, \theta) = u_{rr} + \frac{1}{r}u_r + \frac{1}{r^2}u_{\theta\theta},$$

113
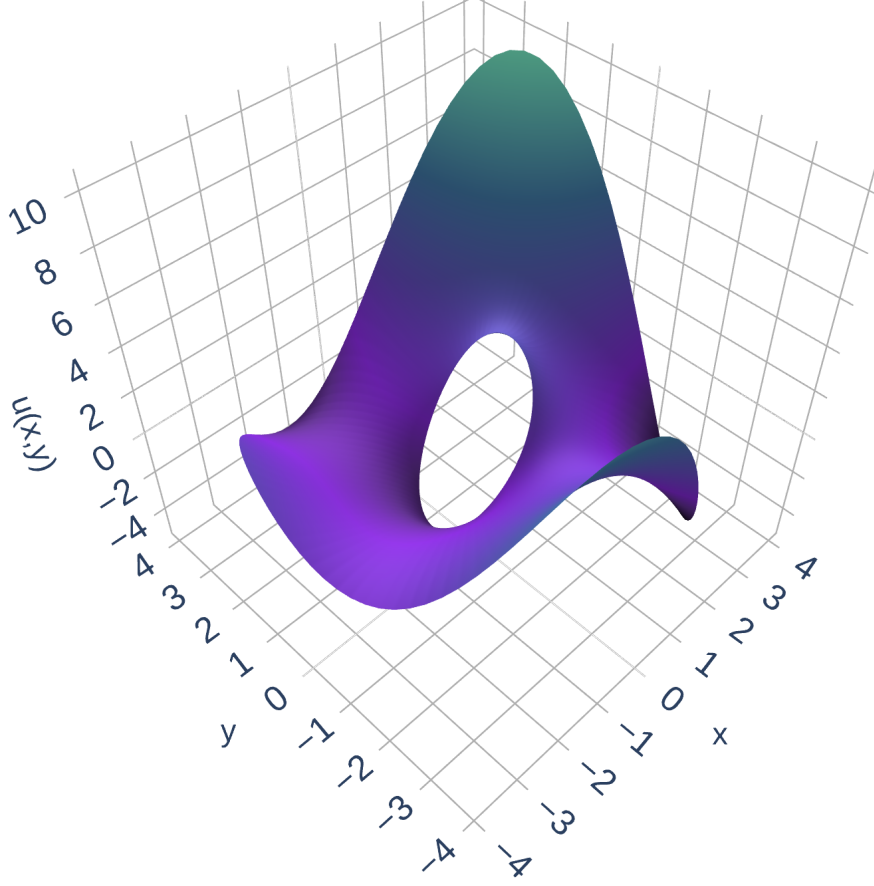
Figure 4.7: Biharmonic equation analytical solution in polar coordinates. Note, this figure contains an embedded, standalone HMTL version of the plot that can be viewed/downloaded by clicking on it. Doing so may require a dedicated PDF viewer such as Adobe Acrobat or Okular.

so the biharmonic operator in polar coordinates is [75],

$$\nabla^4 u(r,\theta) = u_{rrrr} + \frac{2}{r^2}u_{rr\theta\theta} + \frac{1}{r^4}u_{\theta\theta\theta\theta} + \frac{2}{r}u_{rrr} - \frac{2}{r^3}u_{r\theta\theta} - \frac{1}{r^2}u_{rr} + \frac{4}{r^4}u_{\theta\theta} + \frac{1}{r^3}u_r.$$

The constrained expression for this problem given in recursive form is,

$$^{(1)}u(r,\theta,g(r,\theta)) = g(r,\theta)$$
$$+ \frac{1}{3}(r-1)\left(4\sin(2\theta) + 4\sin(3\theta) + \frac{1}{4}\pi\cos(\theta) + 2 - g(4,\theta)\right)$$

114

$$+ \frac{1}{3}(4-r)\left(\frac{1}{4}\sin(2\theta) + \frac{1}{16}\sin(3\theta) + \pi\cos(\theta) + \frac{1}{8} - g(1,\theta)\right)$$

$$+ \frac{1}{18}\left(-r^3 + 12r^2 - 39r + 28\right)\left(-g_{rr}(1,\theta) + \frac{1}{2}\sin(2\theta) + \frac{3}{8}\sin(3\theta) + 2\pi\cos(\theta) + \frac{1}{4}\right)$$

$$+ \frac{1}{18}\left(r^3 - 3r^2 - 6r + 8\right)\left(\frac{1}{2}\sin(2\theta) + \frac{3}{2}\sin(3\theta) + \frac{1}{32}\pi\cos(\theta) + \frac{1}{4} - g_{rr}(4,\theta)\right),$$

$$^{(2)}u(r,\theta,g(r,\theta)) = g(r,\theta) - \frac{\theta}{2\pi}\Big(g(r,2\pi) - g(r,0)\Big) + \frac{2\pi\theta - \theta^2}{4\pi}\Big(g_r(r,2\pi) - g_r(r,0)\Big)$$

$$+ \frac{-\theta^3 + 3\pi\theta^2 - 2\pi^2\theta}{12\pi}\Big(g_{rr}(r,2\pi) - g_{rr}(r,0))\Big)$$

$$+ \frac{-\theta^4 + 4\pi\theta^3 - 4\pi^2\theta^2}{48\pi}\Big(g_{rrr}(r,2\pi) - g_{rrr}(r,0)\Big),$$

where $^{(1)}u$ can be used as the free function in $^{(2)}u$ or $^{(2)}u$ can be used as the free function in $^{(1)}u$ to create the full constrained expression. In tensor form the constrained expression is,

$$u(r,\theta,g(r,\theta)) = g(r,\theta) + \mathcal{M}_{ij}(r,\theta,g(r,\theta))\Phi_i(r)\Phi_j(\theta),$$

where

$$\mathcal{M}_{ij}(r,\theta,g(r,\theta)) = \begin{bmatrix} 0 \\ -g(1,\theta) + \frac{1}{4}\sin(2\theta) + \frac{1}{16}\sin(3\theta) + \pi\cos(\theta) + \frac{1}{8} \\ -g(4,\theta) + 4\sin(2\theta) + 4\sin(3\theta) + \frac{1}{4}\pi\cos(\theta) + 2 \\ -g_{rr}(1,\theta) + \frac{1}{2}\sin(2\theta) + \frac{3}{8}\sin(3\theta) + 2\pi\cos(\theta) + \frac{1}{4} \\ -g_{rr}(4,\theta) + \frac{1}{2}\sin(2\theta) + \frac{3}{2}\sin(3\theta) + \frac{1}{32}\pi\cos(\theta) + \frac{1}{4} \end{bmatrix}, \quad i = 1$$

$$\mathcal{M}_{ij}(r,\theta,g(r,\theta)) = \begin{bmatrix} g(r,2\pi) - g(r,0) & g_\theta(r,2\pi) - g_\theta(r,0) \\ g(1,0) - g(1,2\pi) & g_\theta(1,0) - g_\theta(1,2\pi) \\ g(4,0) - g(4,2\pi) & g_\theta(4,0) - g_\theta(4,2\pi) \\ g_{rr}(1,0) - g_{rr}(1,2\pi) & g_{rr\theta}(1,0) - g_{rr\theta}(1,2\pi) \\ g_{rr}(4,0) - g_{rr}(4,2\pi) & g_{rr\theta}(4,0) - g_{rr\theta}(4,2\pi) \end{bmatrix}, \quad i \in \{2,3\}$$

$$\mathcal{M}_{ij}(r, \theta, g(r, \theta)) = \begin{bmatrix} g_{\theta\theta}(r, 2\pi) - g_{\theta\theta}(r, 0) & g_{\theta\theta\theta}(r, 2\pi) - g_{\theta\theta\theta}(r, 0) \\ g_{\theta\theta}(1, 0) - g_{\theta\theta}(1, 2\pi) & g_{\theta\theta\theta}(1, 0) - g_{\theta\theta\theta}(1, 2\pi) \\ g_{\theta\theta}(4, 0) - g_{\theta\theta}(4, 2\pi) & g_{\theta\theta\theta}(4, 0) - g_{\theta\theta\theta}(4, 2\pi) \\ g_{rr\theta\theta}(1, 0) - g_{rr\theta\theta}(1, 2\pi) & g_{rr\theta\theta\theta}(1, 0) - g_{rr\theta\theta\theta}(1, 2\pi) \\ g_{rr\theta\theta}(4, 0) - g_{rr\theta\theta}(4, 2\pi) & g_{rr\theta\theta\theta}(4, 0) - g_{rr\theta\theta\theta}(4, 2\pi) \end{bmatrix}, \quad i \in \{4, 5\}$$

and

$$\Phi_i(r) = \left\{ 1, \frac{4-r}{3}, \frac{r-1}{3}, \frac{-r^3 + 12r^2 - 39r + 28}{18}, \frac{r^3 - 3r^2 - 6r + 8}{18} \right\},$$

$$\Phi_i(\theta) = \left\{ 1, -\frac{\theta}{2\pi}, \frac{2\pi\theta - \theta^2}{4\pi}, \frac{-\theta^3 + 3\pi\theta^2 - 2\pi^2\theta}{12\pi}, \frac{-\theta^4 + 4\pi\theta^3 - 4\pi^2\theta^2}{48\pi} \right\}.$$

Using Chebyshev orthogonal polynomials up to degree 30 as the free function and a grid of 30×30 training points, the PDE solution was estimated using the TFC method. The solution was obtained in 10.67 seconds, and the average error on a test set of 100×100 uniformly spaced points was 1.535×10^{-8}.

5.   SUMMARY AND CONCLUSIONS

This dissertation is titled "The Multivariate Theory of Functional Connections: An n-Dimensional Constraint Embedding Technique Applied to Partial Differential Equations" because it presents two main ideas: (1) the derivation and analysis of multivariate TFC constrained expressions (2) the estimation of PDE solutions using TFC.

The first of these main ideas is conveyed in Chapter 2, which is a self-contained presentation of multivariate TFC. It includes the derivation of multivariate constrained expressions starting from the univariate theory and all the associated mathematical theorems that have been proven to date. Although most readers will likely only be interested in using constrained expressions for value constraints, derivative constraints, and linear combinations thereof, as these are the most common types of constraints found in differential equations, integral and component constraints are included as well. Because the chapter is comprehensive, it serves as a convenient and useful reference for any reader interested in TFC, regardless of their familiarity with the subject. In addition, extensions of the theory to inequality constraints, nonlinear constraints, parallelotope domains, lower-dimensional constraints in n-dimensions, and to any field, i.e., beyond real numbers to other fields such as complex numbers, are covered in Appendices B, C, and I.

The second idea is conveyed in Chapter 3, which describes how to apply TFC to DEs, in particular, PDEs. In addition to describing the general methodology for solving DEs via TFC, this chapter discusses the common free function and optimization choices as well as their strengths and weaknesses. Similar to Chapter 2, Chapter 3 is also self-contained; consequently, it is a convenient and useful reference for any reader interested in solving DEs via TFC.

Although Chapters 2 and 3 contain all the information necessary to apply TFC to DEs, they show few complicated examples: such examples are useful, as they highlight and clarify some of the nuances of TFC discussed in these chapters. To that end, Chapter 4 contains

some complex problems in a field of particular interest to the author: flexible body dynamics. Furthermore, these complex problems showcase the power and convenience of the numerical implementation. As examples:

1. The natural tandem balloon shape problem is a complex system of four ODEs wherein both ends of the domain are themselves unknowns. The code for this problem is simple to read and write due to automatic differentiation—none of the derivatives have to be written out explicitly, which would clutter the code significantly—and the optimization of the free functions is simple too, despite the number of unknowns—a ξ vector for each of the four dependent variables and the two unknowns associated with the ends of the domain.

2. The polar biharmonic equation is a linear, fourth-order PDE with relative constraints up to the third derivative in one of the two independent variables. The numerical implementation allows the constrained expression to be written using the recursive format, which is short and easy to read. Furthermore, automatic differentiation makes creating the residual easy and straightforward even though it contains ninth-order partial derivatives (fifth-order partial derivatives in the constrained expression plus the fourth-order partial derivatives appearing in the residual itself).

The code for both of these problems and most of the problems and examples in this dissertation can be found for free on the [TFC GitHub](#) [64].

This dissertation showed that TFC is useful for solving differential equations. In many of the examples shown, the solution error found using TFC is multiple orders of magnitude lower than competing state-of-the-art methods. Moreover, most of the TFC solutions are found in seconds or fractions of a second, except for those found using Deep-TFC, which typically solves problems on the order of minutes. However, in its current state, TFC can only be applied to rectangular domains[1] and a limited number of non-rectangular domains

---

[1]Rectangular domains here means with respect to the coordinates being used, e.g., problems using polar coordinates such as the polar bi-harmonic problem appear to be on a cylindrical domain when viewed on a Cartesian grid, but are rectangular from the perspective of the polar coordinates.

[16, 17]: this restriction is not present in many of the competing state-of-the-art methods.

5.1 Future Work

In terms of constraints, TFC can currently embed value, derivative, integral, component, and linear constraints, and it can embed any number of these constraints on any number of dimensions. However, it cannot embed sets of integral constraints whose integration variables refer to one another, such as,

$$\int_0^1 u(x,0) \, \mathrm{d}x = 1 \quad \text{and} \quad \int_0^1 u(0,y) \, \mathrm{d}y = 1.$$

Finding a way to embed these constraints is a topic of future work. In addition, inequality constraints can currently be used in conjunction with value constraints only, see Appendix C. Integrating inequality constraints fully into the theory, i.e., finding a way to combine inequality constraints and the remaining types of linear constraints, remains a topic of further study. Also, as noted in the comparison with other methods, TFC is restricted to rectangular domains and a handful of irregular domains: extending TFC to all irregular domains is a topic of future research.

This dissertation focused on applying TFC to differential equations. However, there are a plethora of other applications that have yet to be explored, such as Computer-Aided Design (CAD) [21, 22], image warping [23], and security pattern design [24]. Yet, even in the application of differential equations, there are numerous directions future research can pursue:

- Hybrid basis functions - Combining two or more sets of basis functions and using the result as the free function, e.g., Fourier basis functions and Legendre orthogonal polynomials. Note that one could even combine X-TFC and basis functions in this way because both are simply a linear combination of functions.

- Optimizers - Only four optimizers are utilized in this dissertation: least-squares, L-BFGS, Adam, and CSVM. However, many optimizers could have been used instead,

some of which may outperform those used here.

- NN architecture - Deep-TFC has only used fully connected NNs up to this point. Like the optimizers, there are a variety of NN architectures that could be used, some of which may outperform the fully connected NNs used here.

This list is by no means exhaustive, and there are almost certainly research directions that have been excluded. Hence, the reader is encouraged to consider and pursue the research ideas that appear here as well as those that the author has not considered.

REFERENCES

[1] C. Leake, H. Johnston, and D. Mortari, "The multivariate theory of functional connections: Theory, proofs, and application in partial differential equations," *Mathematics*, vol. 8, p. 1303, Aug 2020.

[2] C. Leake, H. Johnston, L. Smith, and D. Mortari, "Analytically embedding differential equation constraints into least squares support vector machines using the theory of functional connections," *Machine Learning and Knowledge Extraction*, vol. 1, no. 4, pp. 1058–1083, 2019.

[3] C. Leake and D. Mortari, "Deep Theory of Functional Connections: A New Method for Estimating the Solutions of Partial Differential Equations," *Machine Learning and Knowledge Extraction*, vol. 2, no. 1, pp. 37–55, 2020.

[4] E. Waring, "Problems concerning interpolations. by edward waring, m. d. f. r. s. and of the institute of bononia, lucasian professor of mathematics in the university of cambridge," *Philosophical Transactions of the Royal Society of London*, vol. 69, pp. 59–67, 1779.

[5] N. Lam, "Spatial interpolation methods: a review," *American Cartographer*, vol. 10, pp. 129–149, 01 1983.

[6] J. Li and A. D. Heap, "A review of comparative studies of spatial interpolation methods in environmental sciences: Performance and impact factors," *Ecological Informatics*, vol. 6, no. 3, pp. 228 – 241, 2011.

[7] T. M. Lehmann, C. Gonner, and K. Spitzer, "Survey: interpolation methods in medical image processing," *IEEE Transactions on Medical Imaging*, vol. 18, no. 11, pp. 1049–1075, 1999.

[8] J. Steffensen, *Interpolation*. Chelsea Publishing Company, 1950.

[9] D. K. Hoffman, G. W. Wei, D. S. Zhang, and D. J. Kouri, "Interpolating distributed approximating functionals," *Phys. Rev. E*, vol. 57, pp. 6152–6160, May 1998.

[10] G. Wei, H. Wang, D. J. Kouri, M. Papadakis, I. A. Kakadiaris, and D. K. Hoffman, "On the mathematical properties of distributed approximating functionals," *Journal of Mathematical Chemistry*, vol. 30, no. 1, pp. 83–107, 2001.

[11] D. Mortari, "The theory of connections: Connecting points," *Mathematics*, vol. 5, no. 4, 2017.

[12] H. Johnston and D. Mortari, "Linear Differential Equations Subject to Relative, Integral, and Infinite Constraints," in *AIAA/AAS Astrodynamics Specialist Conference*, 8 2018.

[13] D. Mortari and R. Furfaro, "Univariate Theory of Functional Connections Applied to Component Constraints," *Mathematical and Computational Applications*, vol. 26, p. 9, Mar 2021.

[14] H. Johnston, C. Leake, and D. Mortari, "An Analysis of the Theory of Functional Connections Subject to Inequality Constraints," in *AIAA/AAS Astrodynamics Specialist Conference*, 8 2019.

[15] D. Mortari and C. Leake, "The multivariate theory of connections," *Mathematics*, vol. 7, no. 3, 2019.

[16] H. Johnston, C. Leake, Y. Efendiev, and D. Mortari, "Selected applications of the theory of connections: A technique for analytical constraint embedding," *Mathematics*, vol. 7, no. 6, 2019.

[17] D. Mortari and D. Arnas, "Bijective mapping analysis to extend the theory of functional connections to non-rectangular 2-dimensional domains," *Mathematics*, vol. 8, p. 1593, Sep 2020.

[18] J. A. Bondy and U. S. R. Murty, *Graph theory with applications*. New York: North Holland, 1976.

[19] D. Mortari, "Least-squares solution of linear differential equations," *Mathematics*, vol. 5, no. 4, 2017.

[20] D. Mortari, H. Johnston, and L. Smith, "High accuracy least-squares solutions of nonlinear differential equations," *Journal of Computational and Applied Mathematics*, vol. 352, pp. 293 – 307, 2019.

[21] T. Theoharis, G. Papaioannou, N. Platis, and N. M. Patrikalakis, *Graphics and Visualization: Principles & Algorithms*. Wellesley, Massachusetts/Boca Raton, FL: A K Peters/CRC Press, 2008.

[22] D. C. Llach, "Reconstructing "sketchpad" and the "coons patch"," in *ACM SIGGRAPH 2018 Art Gallery on - SIGGRAPH '18*, (Vancouver British Columbia Canada), pp. 429–430, ACM Press, 2018.

[23] P. S. Heckbert, *Bilinear Coons Patch Image Warping*, p. 438–446. USA: Academic Press Professional, Inc., 1994.

[24] W. Qi, X. Li, and B. Yang, "Bilinear coons patch and its application in security pattern design," in *2009 Fifth International Conference on Intelligent Information Hiding and Multimedia Signal Processing*, (Kyoto, Japan), pp. 881–884, IEEE, Sept. 2009.

[25] J. N. Reddy, "An Introduction to the Finite Element Method," *Journal of Pressure Vessel Technology*, vol. 111, pp. 348–349, 08 1989.

[26] J. Argyris and S. Kelsey, "Energy Theorems and Structural Analysis: A Generalized Discourse with Applications on Energy Principles of Structural Analysis Including the Effects of Temperature and Non-Linear Stress-Strain Relations," *Aircraft Engineering and Aerospace Technology*, vol. 26, no. 10, pp. 347–356, 1954.

[27] M. J. Turner, R. W. Clough, H. C. Martin, and L. J. Topp, "Stiffness and Deflection Analysis of Complex Structures," *Journal of the Aeronautical Sciences*, vol. 23, pp. 805–823, sep 1956.

[28] R. W. Clough, *The finite element method in plane stress analysis*. American Society of Civil Engineers, 1960.

[29] S. A. Orszag, "Numerical methods for the simulation of turbulence," *Physics of Fluids*, vol. 12, no. 12, p. II–250, 1969.

[30] J. P. Boyd, *Chebyshev and Fourier spectral methods*. Dover Publications, 2nd ed., rev ed., 2001.

[31] J. Shen, T. Tang, and L.-L. Wang, *Spectral methods: algorithms, analysis and applications*. Springer series in computational mathematics, Springer, 2011.

[32] B. Fornberg, *A practical guide to pseudospectral methods*. Cambridge University Press, 1996.

[33] N. Yadav, A. Yadav, and M. Kumar, *An Introduction to Neural Network Methods for Differential Equations*. Springer Netherlands, 2015.

[34] I. E. Lagaris, A. Likas, and D. I. Fotiadis, "Artificial neural networks for solving ordinary and partial differential equations," *IEEE Transactions on Neural Networks*, vol. 9, pp. 987–1000, Sept 1998.

[35] J. Sirignano and K. Spiliopoulos, "DGM: A deep learning algorithm for solving partial differential equations," *Journal of Computational Physics*, vol. 375, pp. 1339–1364, Dec. 2018.

[36] M. Raissi, P. Perdikaris, and G. E. Karniadakis, "Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations," *Journal of Computational Physics*, vol. 378, pp. 686–707, 2019.

[37] H. Johnston, "The Theory of Functional Connections A journey from theory to application." PhD Dissertation, Texas A&M University, 2021.

[38] C. Lanczos, *Applied Analysis*, p. 504. New York: Dover Publications, Inc., 1957.

[39] K. Wright, "Chebyshev Collocation Methods for Ordinary Differential Equations.," *The Computer Journal*, vol. 6, no. 1, pp. 358–365, 1964. Issue 4.

[40] H. Johnston, C. Leake, and D. Mortari, "Least-squares solutions of eighth-order boundary value problems using the theory of functional connections," *Mathematics*, vol. 8, p. 397, Mar 2020.

[41] H. Johnston and D. Mortari, "Weighted least-squares solutions of over-constrained differential equations," in *IAA SciTech-081 Forum on Space Flight Mechanics and Space Structures and Materials*, (Moscow, Russia), AAS/AIAA, 11 2018.

[42] C. Leake and D. Mortari, "An Explanation and Implementation of Multivariate Theory of Functional Connections via Examples," in *AIAA/AAS Astrodynamics Specialist Conference*, 8 2019.

[43] V. N. Vapnik, *Statistical Learning Theory*. Wiley-Interscience, sep 1998.

[44] S. Theodoridis, *Pattern Recognition*. Academic Press, nov 2008.

[45] T. F. S. Mehrkanoon and J. A. K. Suykens, "Approximate Solutions to Ordinary Differential Equations using Least-squares Support Vector Machines," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 23, no. 9, pp. 1356–1367, 2012.

[46] S. Mehrkanoon and J. A. Suykens, "LS-SVM Approximate Solution to Linear Time Varying Descriptor Systems," *Automatica*, vol. 48, no. 10, pp. 2502–2511, 2012.

[47] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics* (Y. W. Teh and M. Titterington, eds.), vol. 9 of *Proceedings of Machine Learning Research*, (Chia Laguna Resort, Sardinia, Italy), pp. 249–256, PMLR, 13–15 May 2010.

[48] G.-B. Huang, Q.-Y. Zhu, and C.-K. Siew, " Extreme learning machine: Theory and applications ," *Neurocomputing*, vol. 70, pp. 489–501, May 2006.

[49] E. Schiassi, C. Leake, M. D. Florio, H. Johnston, R. Furfaro, and D. Mortari, "Extreme theory of functional connections: A physics-informed neural network method for solving parametric differential equations," 2020.

[50] R. Fletcher, *Practical methods of optimization.* Chichester ; New York: Wiley, 2nd ed ed., 1987.

[51] H. Robbins and S. Monro, "A Stochastic Approximation Method," *The Annals of Mathematical Statistics*, vol. 22, pp. 400–407, Sept. 1951.

[52] J. Kiefer and J. Wolfowitz, "Stochastic Estimation of the Maximum of a Regression Function," *The Annals of Mathematical Statistics*, vol. 23, pp. 462–466, Sept. 1952.

[53] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *J. Mach. Learn. Res.*, vol. 12, p. 2121–2159, July 2011.

[54] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, pp. 533–536, Oct. 1986.

[55] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings* (Y. Bengio and Y. LeCun, eds.), 2015.

[56] S. Mehrkanoon and J. Suykens, "Learning solutions to partial differential equations using ls-svm," *Neurocomputing*, vol. 159, pp. 105–116, 07 2015.

[57] A. G. Baydin, B. A. Pearlmutter, A. A. Radul, and J. M. Siskind, "Automatic differentiation in machine learning: A survey," *J. Mach. Learn. Res.*, vol. 18, p. 5595–5637, Jan. 2017.

[58] A. D. Schutte, "A nilpotent algebra approach to lagrangian mechanics and constrained motion," *Nonlinear Dynamics*, vol. 88, p. 1001–1012, Apr 2017.

[59] S. Linnainmaa, "Taylor expansion of the accumulated rounding error," *BIT*, vol. 16, p. 146–160, Jun 1976.

[60] R. Frostig, M. Johnson, and C. Leary, "Compiling machine learning programs via high-level tracing," in *SysML Conference*, 2018.

[61] J. Bradbury, R. Frostig, P. Hawkins, M. J. Johnson, C. Leary, D. Maclaurin, and S. Wanderman-Milne, "JAX: composable transformations of Python+NumPy programs," 2018.

[62] D. Maclaurin, D. Duvenaud, M. Johnson, and J. Townsend, "Autograd," 2013.

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

[64] C. Leake and H. Johnston, "TFC: A Functional Interpolation Framework," 2020.

[65] H. Sun, M. Hou, Y. Yang, T. Zhang, F. Weng, and F. Han, "Solving Partial Differential Equation Based on Bernstein Neural Network and Extreme Learning Machine Algorithm," *Neural Processing Letters*, vol. 50, no. 2, pp. 1153–1172, 2019.

[66] S. Mall and S. Chakraverty, "Single Layer Chebyshev Neural Network Model for Solving Elliptic Partial Differential Equations," *Neural Processing Letters*, vol. 45, no. 3, pp. 825–840, 2017.

[67] J. L. Hall, J. Cameron, M. Pauken, J. Izraelevitz, M. W. Dominguez, and K. T. Wehage, "Altitude-controlled light gas balloons for venus and titan exploration," in *AIAA Aviation 2019 Forum*, American Institute of Aeronautics and Astronautics, Jun 2019.

[68] J. M. Cameron, "Natural shapes of tandem balloons," *Private communication*, 7 2020.

[69] F. Baginski, T. Williams, and W. Collier, "A parallel shooting method for determining the natural shape of a large scientific balloon," *SIAM Journal on Applied Mathematics*, vol. 58, pp. 961–974, June 1998.

[70] J. H. Smalley, "Determination of the shape of a free balloon," Tech. Rep. AD0611825, Air Force Cambridge Research Laboratories, Bedford, Massachusetts, April 1964.

[71] M. Lai, E. Krempl, and D. Ruben, *Introduction to Continuum Mechanics*. Elsevier, Sept. 2009.

[72] A. Logg, K.-A. Mardal, G. N. Wells, *et al.*, *Automated Solution of Differential Equations by the Finite Element Method*. Springer, 2012.

[73] A. Logg and G. N. Wells, "Dolfin: Automated finite element computing," *ACM Transactions on Mathematical Software*, vol. 37, p. 1–28, Apr 2010.

[74] J. H. Michell, "On the direct determination of stress in an elastic solid, with application to the theory of plates," *Proceedings of the London Mathematical Society*, vol. s1-31, pp. 100–124, Apr. 1899.

[75] W. Kaplan, *Advanced calculus*. Addison-Wesley, 3rd ed ed., 1984.

[76] R. N. Bracewell, *The Fourier transform and its applications*. McGraw-Hill series in electrical and computer engineering, McGraw Hill, 3rd ed ed., 2000.

[77] C. F. Dunkl and Y. Xu, *Orthogonal Polynomials of Several Variables*. Encyclopedia of Mathematics and its Applications, Cambridge University Press, 2nd ed., 2014.

[78] H. Triebel, *Theory of Function Spaces*. Modern Birkhäuser Classics, Springer Basel, 2010.

[79] G. Strang, *Introduction To Linear Algebra, 5th Edition*. Cambridge University Press, 12 2019.

[80] V. I. Bogachev, *Measure Theory*. Springer, 1 2007.

[81] J. Ye, Z. Gao, S. Wang, J. Cheng, W. Wang, and W. Sun, "Comparative Assessment of Orthogonal Polynomials for Wavefront Reconstruction over the Square Aperture," *Journal of the Optical Society of America A*, vol. 31, no. 10, pp. 2304–2311, 2014.

[82] Y. Xu, "Multivariate Orthogonal Polynomials and Operator Theory," *Transactions of the American Mathematical Society*, vol. 343, no. 1, pp. 193–202, 1994.

[83] H. P. Langtangen, *Computational Partial Differential Equations: Numerical Methods and Diffpack Programming.* Berlin, Heidelberg: Springer Berlin Heidelberg, 2003. OCLC: 851766084.

[84] O. Byer, F. Lazebnik, and D. L. Smeltzer, *Methods for Euclidean geometry.* Classroom resource materials, Mathematical Association of America, 2010.

APPENDIX A

GRAPH THEORY

This appendix provides a cursory overview of the graph theory concepts germane to determining the processing order of constrained expressions; readers who would like a more in-depth presentation of these topics should consult Reference [18]. First, a few different types of graphs are defined.

Definition A.1

A graph is a set of nodes connected by edges.

Figure A.1(a) shows an example of a graph.

Definition A.2

A multigraph is a graph wherein at least one pair of nodes is connected by more than one edge.

Figure A.1(b) shows an example of a multigraph.

Definition A.3

A directed graph is a graph wherein the edges have direction.

Figure A.1(c) shows an example of a directed graph: more specifically, a directed multigraph. Arrows denote the directions of the edges. For a directed edge, the target node is the node at the head of the arrow, and the source node is the node at the tail of the arrow.

Definition A.4

A cyclic graph is a graph that contains at least one cycle.

Definition A.5

An acyclic graph is a graph that contains no cycles.

Definition A.6

A cycle is a path on the graph wherein the only repeated nodes are the first and last nodes.

Figure A.1(d) shows an example of a cyclic graph: more specifically, a directed cyclic multi-graph.

(a) Graph

(b) Multigraph

(c) Directed acyclic multigraph

(d) Directed cyclic multigraph
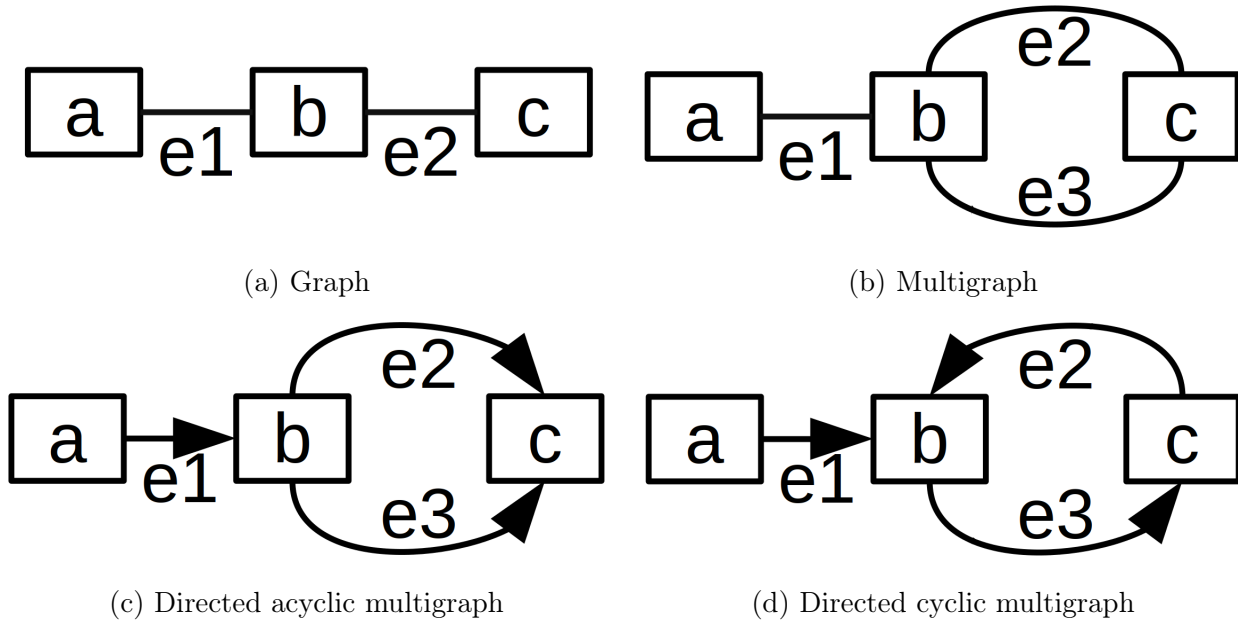
Figure A.1: Example graphs.

It is also convenient to define some node types.

Definition A.7

A root node is a node in a directed graph that is not the target of any edges.

Definition A.8

A leaf node is a node in a directed graph that is not the source of any edges.

In Fig. A.1, node a is a root node in (c) and (d), and node c is a leaf node in (c) but not in (d).

Definition A.9

A parent node of node i is any node j in a directed graph such that an edge exists where i is the target and j is the source.

Definition A.10

A child node of node i is any node j in a directed graph such that an edge exists where i is the source and j is the target.

In Fig. A.1, node a is a parent of node b and node b is a child of a in (c) and (d).

Lastly, the concept of the adjacency matrix, A_{ij}, for a directed graph is introduced. The adjacency matrix can be constructed using,

$$A_{ij} = \begin{cases} 1, & \text{if node } i \text{ is a parent of node } j \\ 0, & \text{otherwise.} \end{cases}$$

The adjacency matrix has many uses, but in the context of TFC, the adjacency matrix is used to determine if a graph is acyclic or not. If A_{ij} is nilpotent, then the graph is acyclic [18].

APPENDIX B

EXTENSION TO NONLINEAR CONSTRAINTS

This appendix extends TFC to simple nonlinear constraints and parameterized nonlinear constraints. The extension is accomplished by transforming the nonlinear constraints into linear constraints by introducing extra variables into the constrained expression, which are found in the κ terms. These extra variables are constants from the perspective of the constraint operators. Hence, the rest of the theory—derivation of switching functions and projection functionals, extension to n-dimensions, and associated mathematical theorems— remains unchanged and still applies as presented in Chapter 2.

This is one of the critical aspects of abstracting the constrained expression into the symbols associated with the switching-projection form: if one can rewrite constraints as $\mathfrak{C}[y] = \kappa$ and maintain the relevant mathematical properties, the rest of the theory still applies. This core idea is used in many of the extensions to the TFC theory, e.g., extending beyond the field of real numbers to all mathematical fields, as shown in Appendix I.

B.1 Simple Nonlinear Constraints

Definition B.1

Simple nonlinear constraints are those that can be written as,

$$\psi\Big[\mathfrak{C}[y]\Big] = \hat{\kappa}$$

for some nonlinear function or operator ψ that has a well-defined inverse, i.e., ψ^{-1} exists and can be calculated.

Notice that these types of constraints can be rewritten as linear constraints by applying

ψ^{-1} to each side:

$$\mathfrak{C}[y] = \psi^{-1}[\hat{\kappa}] = \kappa,$$

where $\kappa = \psi^{-1}[\hat{\kappa}]$. However, doing so may result in multiple solutions. That is, there may be be multiple κ values that satisfy $\kappa = \psi^{-1}[\hat{\kappa}]$ or even an infinite number of κ values.

One option is to write a constrained expression for each κ solution. However, this is impossible for an infinite number of κ solutions and quickly becomes burdensome when there are multiple simple nonlinear constraints; the total number of constrained expressions required for multiple nonlinear constraints is $\prod_i \mathrm{num}(\kappa_i)$ where $\mathrm{num}(\kappa_i)$ is the number of κ solutions for the i-th nonlinear constraint.

Fortunately, by introducing new variables into the constrained expression, one can combine the set of constrained expressions into one constrained expression. The following examples highlight some common cases: solutions with $\pm$, a finite number of solutions, and a countably infinite number of solutions.

Example B.1: Constraints with solutions that contain $\pm$

Consider the constraint $y^2(0) = 3$. For this constraint, the nonlinear operator raises the function to the power of two, e.g., $\psi[f(x)] = f^2(x)$. Inverting the nonlinear operator results in the solutions $y(0) = \pm\sqrt{3}$. One could write the set of constrained expressions that satisfy these constraints,

$$y(x, g(x)) = g(x) + \sqrt{3} - g(0) \tag{B.1}$$

$$y(x, g(x)) = g(x) - \sqrt{3} - g(0). \tag{B.2}$$

However, by including a variable n in the constrained expression, these two constrained expressions can be combined into one,

$$y(x, n, g(x)) = g(x) + (-1)^{\mathbb{1}_0(n)}\sqrt{3} - g(0),$$

where $\mathbb{1}_0(x)$ is the unit step function where the step occurs at $x = 0$. If $n < 0$, then Equation (B.2) is recovered, and if $n > 0$, then Equation (B.1) is recovered. Hence, $n \in \mathbb{R}$ is now just a variable, and its value dictates which constrained expression branch is used.

Figure B.1 shows the constrained expression plotted with randomly chosen values of n and free functions chosen as polynomials with random coefficients. The solutions of $y(0)$ that satisfy the constraint are shown as black points.
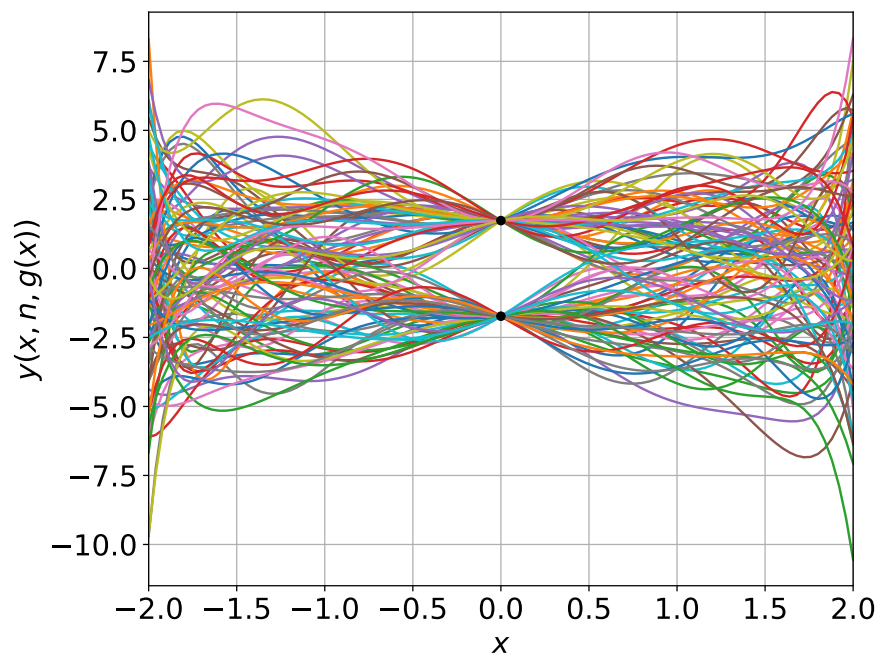
Figure B.1: Squared constraint example for randomly chosen $g(x)$ and n.

Example B.2: Constraints with a finite number of solutions

Consider the constraint

$$y^3(0) - 6y^2(0) + 11y(0) = 6.$$

Applying $\psi_1^{-1}[\hat{\kappa}]$ yields $\kappa = \{1, 2, 3\}$, a finite set. The set of constrained expressions that satisfy these solutions can be written compactly as,

$$y(x, n, g(x)) = g(x) - \kappa[n] - g(0),$$

where $n \in \mathbb{Z}/3\mathbb{Z}$ and $\kappa[n]$ is the n-th solution in the set of κ that satisfies the constraints—one may think of $\kappa[n]$ as the indexing operation of a zero-indexed array κ that contains the solutions of $\psi_1^{-1}[\hat{\kappa}]$. Alternatively, if one prefers to keep $n \in \mathbb{R}$, then this constrained expression can be rewritten as,

$$y(x, n, g(x)) = g(x) - \kappa\left[\lfloor \mathrm{mod}(n, 2) \rceil\right] - g(0).$$
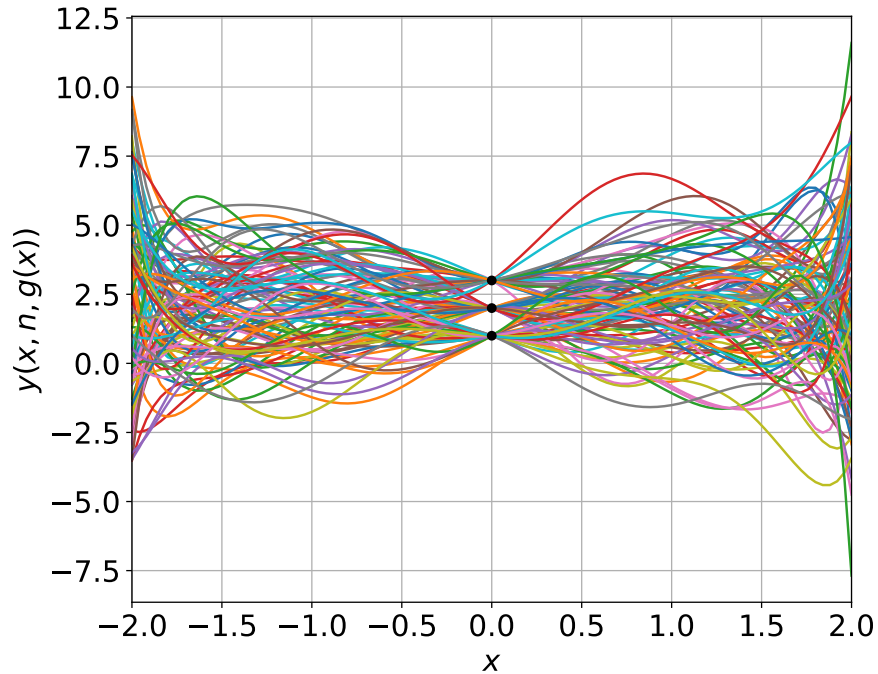
Figure B.2: Polynomial constraint example for randomly chosen $g(x)$ and n.

136

The term $\lfloor (\mathrm{mod}\,(n, 2)) \rceil$, where $\lfloor x \rceil$ rounds x to the nearest integer, forces $n \in \mathbb{R}$ to lie on $\mathbb{Z}/3\mathbb{Z}$.

Figure B.2 shows the constrained expression plotted with randomly chosen values of n and free functions chosen as polynomials with random coefficients. The solutions of $y(0)$ that satisfy the constraint are shown as black points.

Example B.3: Constraints with a countably infinite number of solutions

Consider the constraint,

$$\sin\big(y(1)\big) = 0.$$

Applying $\psi_1^{-1}[\hat{\kappa}]$ yields $\kappa = n\pi$ where $n \in \mathbb{Z}$: a countably infinite number of solutions. Similar to the previous example, one can write the constrained expression as,
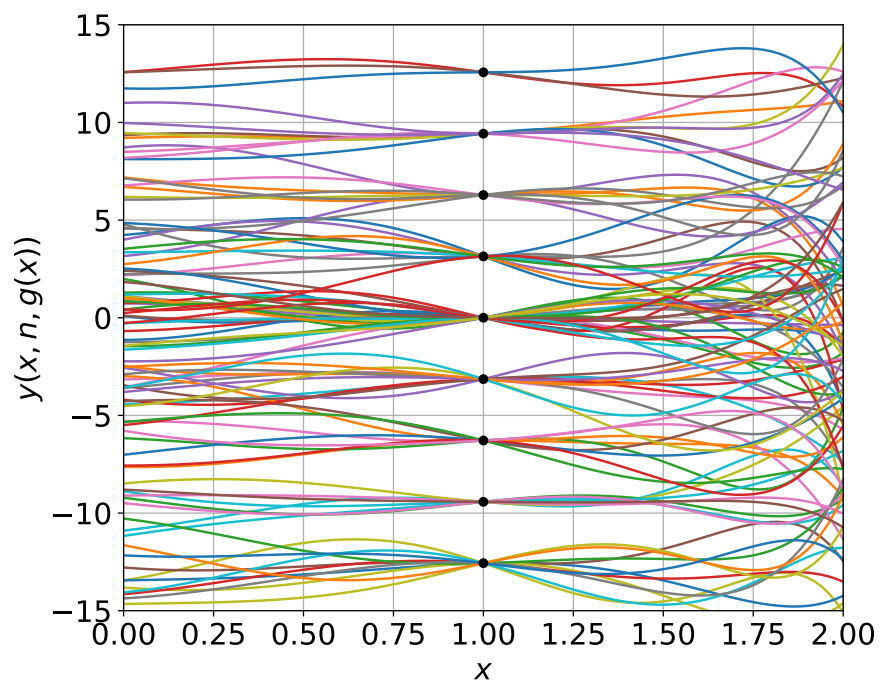
Figure B.3: Sine constraint example for randomly chosen $g(x)$ and n.

$$y(x, n, g(x)) = g(x) - n\pi - g(1),$$

where $n \in \mathbb{Z}$ or as,

$$y(x, n, g(x)) = g(x) - \lfloor n \rfloor \pi - g(1),$$

where $n \in \mathbb{R}$.

Figure B.3 shows the constrained expression plotted with randomly chosen values of n and free functions chosen as polynomials with random coefficients. A subset of the solutions of $y(1)$ that satisfy the constraint is shown via black points.

B.2 Parameterized Nonlinear Constraints

Oftentimes, one can parameterize the solution space of the nonlinear constraints and rewrite them as a set of linear constraints. Examples B.4 and B.5 demonstrate the idea.

Example B.4: Simple parameterized nonlinear constraint

Consider the nonlinear constraint,

$$y^2(0) + (y_x(0) - 4)^2 = 9.$$

The solutions to this nonlinear constraint lie on a circle that can be parameterized in terms of an unknown, θ:

$$y(0) = 3\sin(\theta) \quad \text{and} \quad y_x(0) = 3\cos(\theta) + 4.$$

Then, these linear constraints can be embedded into a constrained expression using the usual method,

$$y(x, \theta, g(x)) = g(x) + 3\sin(\theta) - g(0) + x\left(3\cos(\theta) + 4 - g_x(0)\right),$$

where $\theta \in \mathbb{R}$.

Example B.5: Parameterized nonlinear constraints on conics

Consider the following nonlinear constraints,

$$x^2(0) + \frac{16}{9}y^2(0) + \frac{1}{4}z^2(0) = 1 \quad \text{and} \quad 100\big(x(3) - 3\big)^2 + 100y^2(3) - \frac{100}{9}z^2(3) = -1.$$

The solutions of these nonlinear constraints form an ellipsoid at $t = 0$ and a hyperboloid of two sheets at $t = 3$. Hence, theses nonlinear constraints can be parameterized as,

$$x(0) = \sin(\phi)\cos(\theta) \qquad\qquad x(3) = \frac{1}{10}\sinh(|v|)\cos(\psi) + 3$$

$$y(0) = \frac{3}{4}\sin(\phi)\sin(\theta) \qquad\qquad y(3) = \frac{1}{10}\sinh(|v|)\sin(\psi)$$

$$z(0) = 2\cos(\phi) \qquad\qquad z(3) = (-1)^{\mathbb{1}_0(n)}\frac{3}{10}\cosh(|v|),$$

where $\phi, \theta, v, \psi, n \in \mathbb{R}$. These parameterized constraints can be embedded into constrained expressions:

$$
\begin{aligned}
x(t, \phi, \theta, v, \psi, g(t)) &= g(t) + \frac{3-t}{3}\Big(\sin(\phi)\cos(\theta) - g(0)\Big) \\
&\quad + \frac{t}{3}\Big(\frac{1}{10}\sinh(|v|)\cos(\psi) + 3 - g(3)\Big) \\
y(t, \phi, \theta, v, \psi, g(t)) &= g(t) + \frac{3-t}{3}\Big(\frac{3}{4}\sin(\phi)\sin(\theta) - g(0)\Big) \\
&\quad + \frac{t}{3}\Big(\frac{1}{10}\sinh(|v|)\sin(\psi) - g(3)\Big) \\
z(t, \phi, \theta, v, \psi, g(t)) &= g(t) + \frac{3-t}{3}\Big(2\cos(\phi) - g(0)\Big) \\
&\quad + \frac{t}{3}\Big((-1)^{\mathbb{1}_0(n)}\frac{3}{10}\cosh(|v|) - g(3)\Big).
\end{aligned}
$$

Figure B.4 shows the constrained expressions plotted with randomly chosen values of ϕ, θ, v, ψ, and n and free functions that were chosen as polynomials with random

coefficients. The nonlinear constraint surfaces are shown as a black ellipse and purple hyperboloid.
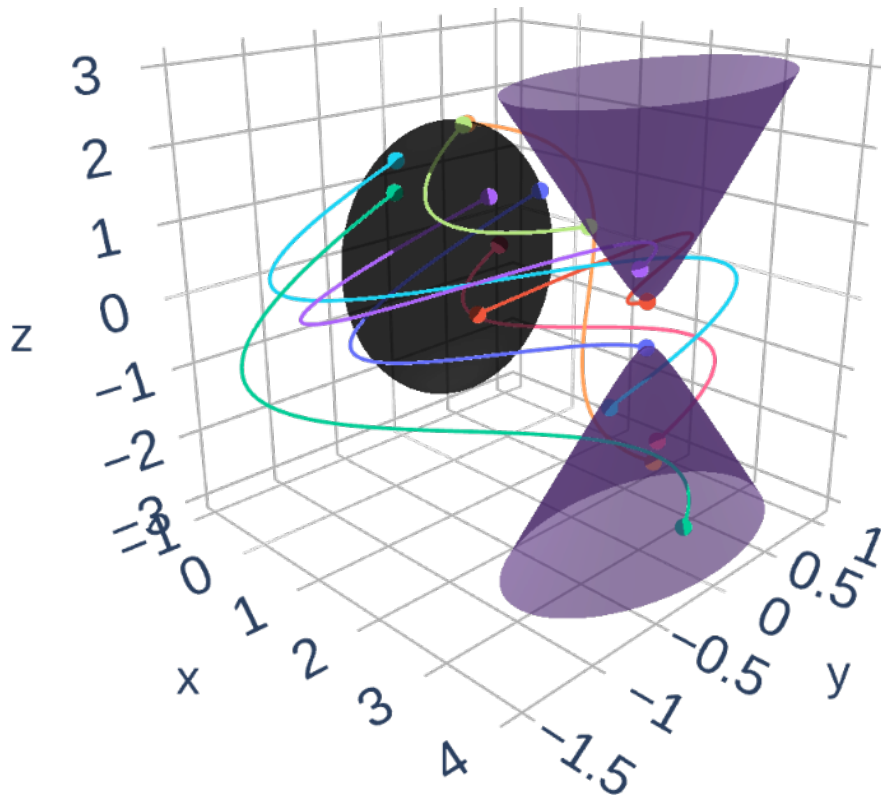
Figure B.4: Parameterized nonlinear constraints on conics. Note, this figure contains an embedded, standalone HMTL version of the plot that can be viewed/downloaded by clicking on it. Doing so may require a dedicated PDF viewer such as Adobe Acrobat or Okular.

ADDING INEQUALITY CONSTRAINTS TO CONSTRAINED EXPRESSIONS

Inequality constraints such as $y(x) < f_u(x) \; \forall x \in \Omega$ where y is the dependent variable, $f_u(x)$ is some function that specifies an upper bound, and Ω is the domain of interest, can also be added to TFC constrained expressions. However, at the time this dissertation is written, the method for incorporating inequality constraints cannot be used in combination with all other constraint types introduced previously, nor have all the mathematical theorems related to the constrained expression been extended to include inequality constraints; thus, they are included here as an appendix, rather than in the main body of the text.

Early attempts to incorporate inequality constraints utilized the sigmoid function to satisfy the inequality constraints approximately [14], but the introduction of an automatic differentiation framework allows them to be incorporated exactly. The enabling component of the automatic differentiation framework is the concept of primitives, which allow a user to specify both a function and its derivative as black boxes: meaning that the derivative specified does not have to be the actual mathematical derivative of the original function. The Heaviside function is a prime example [61]; the function value is encoded as,

$$\mathbb{1}(x, x_1) = \begin{cases} 0, & x < x_1 \\ x_1, & x = 0 \\ 1, & x > x_1 \end{cases}$$

but the derivative is encoded as,

$$\frac{\mathrm{d}\mathbb{1}}{\mathrm{d}x} = 0.$$

Mathematically this is not correct as the derivative is really the Dirac delta function [76], but encoding it in this way allows one to incorporate inequality constraints into constrained

expressions.

Let $\mathbb{1}_0(x) = \mathbb{1}(x, 0)$, which is equivalent to the unit step function where the step occurs at $x = 0$. One can think of this function as the mathematical equivalent of a switch or gate when it is composed with multiplication. For example,

$$f(x) = g(x)\mathbb{1}(x, 0) = g(x)\mathbb{1}_0,$$

will return $g(x)$ when $x > 0$ and 0 otherwise. This switching behavior is exactly the desired behavior needed to implement inequality constraints.

Consider the following two inequality constraints,

$$y(x) > f_\ell(x) \quad \text{and} \quad y(x) < f_u(x),$$

where $f_\ell(x)$ and $f_u(x)$ are the lower and upper bound functions respectively. Using the switch-like behavior of $\mathbb{1}_0$, it is straightforward to write a functional that maintains a free function, i.e., a constrained expression, and obeys these two inequality constraints,

$$y(x, g(x)) = g(x) + \Big(f_u(x) - g(x)\Big)\mathbb{1}_0\Big(g(x) - f_u(x)\Big) + \Big(f_\ell(x) - g(x)\Big)\mathbb{1}_0\Big(f_\ell(x) - g(x)\Big). \quad \text{(C.1)}$$

One can prove that this form satisfies the two inequality constraints via brute-force by checking the three possible cases:

1. $g(x) < f_\ell(x) \rightarrow y(x, g(x)) = g(x) + \Big(f_u(x) - g(x)\Big)(0) + \Big(f_\ell(x) - g(x)\Big)(1) = f_\ell(x)$

2. $f_\ell(x) \leq g(x) \leq f_u(x) \rightarrow y(x, g(x)) = g(x) + \Big(f_u(x) - g(x)\Big)(0) + \Big(f_\ell(x) - g(x)\Big)(0)$
$$= g(x)$$

3. $f_u(x) < g(x) \rightarrow y(x, g(x)) = g(x) + \Big(f_u(x) - g(x)\Big)(1) + \Big(f_\ell(x) - g(x)\Big)(0) = f_u(x)$

Furthermore, the Heaviside derivative overridden by the automatic differentiation program

produces the desired behavior in the derivative of the bounded constrained expression,

$$
y_x(x, g(x)) = \begin{cases} \frac{\mathrm{d}f_\ell}{\mathrm{d}x}(x), & g(x) < f_\ell(x) \\ g_x(x), & f_\ell(x) \le g(x) \le f_u(x) \\ \frac{\mathrm{d}f_u}{\mathrm{d}x}(x), & f_u(x) < g(x); \end{cases}
$$

that is, the derivative of the constrained expression is equal to the derivatives of the lower and upper bound functions when they are active and equal to the derivative of $g(x)$ otherwise.

Although inequality constraints cannot yet be combined with all the constraint types introduced earlier, they can be combined with the most commonly occurring constraint: point constraints. Let $\hat{y}(x, g(x))$ be a constrained expression satisfying some set of point constraints that are consistent with the inequality constraints,

$$
y(x) > f_\ell(x) \quad \text{and} \quad y(x) < f_u(x).
$$

Then, $\hat{y}(x, g(x))$ can be substituted as the free function into the inequality constrained expression given earlier,

$$
\begin{aligned}
y(x, g(x)) = \hat{y}(x, g(x)) + \Big(f_u(x) - \hat{y}(x, g(x)) \Big) \mathbb{1}_0 \Big(\hat{y}(x, g(x)) - f_u(x) \Big) \\
+ \Big(f_\ell(x) - \hat{y}(x, g(x)) \Big) \mathbb{1}_0 \Big(f_\ell(x) - \hat{y}(x, g(x)) \Big),
\end{aligned} \tag{C.2}
$$

and the result satisfies both the equality and inequality constraints. The inequality constraints are satisfied because Equation (C.1) satisfies them for any free function, including $\hat{y}(x, g(x))$, and the equality constraints can be shown to be satisfied by simply applying the constraint operator to the constrained expression (similar to the proof of Theorem 1),

$$
\begin{aligned}
\mathfrak{C}_i[y(x, g(x))] &= \mathfrak{C}_i[\hat{y}(x, g(x))] + \mathfrak{C}_i\Big[\Big(f_u(x) - g(x) \Big)(0) \Big] + \mathfrak{C}_i\Big[\Big(f_\ell(x) - g(x) \Big)(0) \Big] \\
&= \kappa_i + 0 + 0 = \kappa_i.
\end{aligned}
$$

143

Note that for any consistent set of constraints, the $\mathbb{1}_0$ functions must yield zero at the equality constraint locations. The theorem on the existence of $g(x)$ (Theorem 2) can easily be extended as well. Let $f(x)$ be some function that satisfies the constraints, then,

$$y(x, f(x)) = \hat{y}(x, f(x)) + \Big(f_u(x) - f(x)\Big)(0) + \Big(f_\ell(x) - f(x)\Big)(0)$$

$$= f(x) + 0 + 0$$

$$= f(x).$$

Thus, for any function satisfying the constraints, $f(x)$, there exists at least one free function $g(x) = f(x)$, such that the constrained expression is equal to the function satisfying the constraints, i.e., $y(x, f(x)) = f(x)$.

Figures C.1 and C.2 visually show that Equations C.1 and C.2 work, respectively. Each figure contains randomly generated upper and lower bounds, shown as black, dashed lines, and randomly generated free functions, shown as solid, colored lines. In addition, Figure C.2 contains randomly generated point constraints shown via black dots.
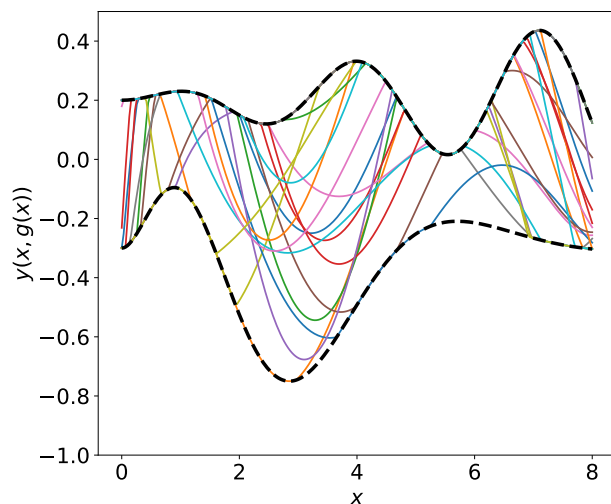
Figure C.1: Inequality constraints only.
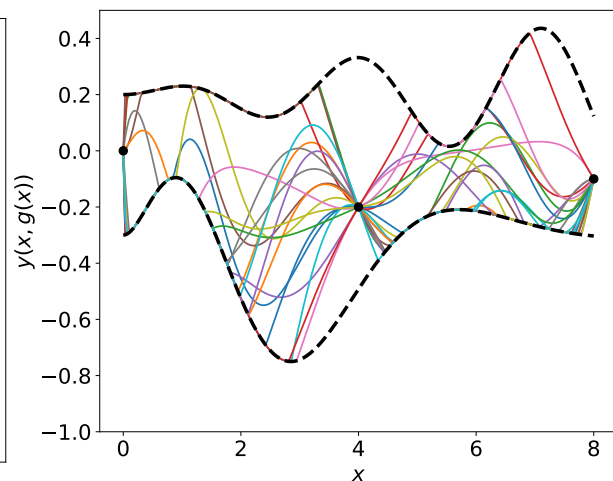
Figure C.2: Inequality and value-level equality constraints.

APPENDIX D

SPLITTING THE DOMAIN

When the solution of a differential equation has steep gradients, it is oftentimes difficult to describe the solution over the whole domain using one constrained expression. In these cases, it is convenient to split the domain into sections and enforce continuity at the intersections via the constrained expressions. For example, consider the following differential equation that describes convection and diffusion processes:

$$y_{xx} - P_e y_x = 0$$

subject to,

$$y(0) = 1 \quad \text{and} \quad y(1) = 0,$$

where $x \in [0, 1]$, P_e is the Peclet number, and the analytical solution is,

$$y = \frac{1 - e^{P_e(x-1)}}{1 - e^{-P_e}}.$$

As the Peclet number increases, the solution's gradients become larger. To illustrate, Figure D.1 shows the analytical solutions to the convection-diffusion equation with two different Peclet numbers, $P_e = 1$ and $P_e = 10^6$. When the Peclet number is low enough, one can estimate the solution well using only one constrained expression for the entire domain,

$$y(x, g(x)) = g(x) + (1 - x)(1 - g(0)) - xg(1).$$

However, as the Peclet number increases, a better estimation is obtained with two constrained expressions. These two constrained expressions are for the subdomains $x_1 \in [0, x_p]$ and $x_2 \in [x_p, 1]$, where x_p is the point of intersection between the two. For this differential
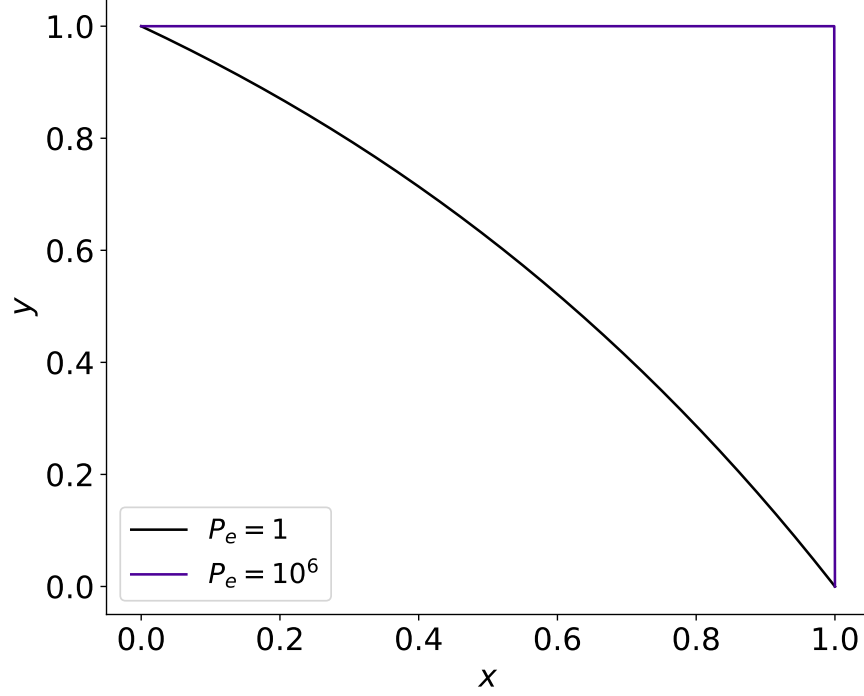
Figure D.1: Analytical solutions of the convection-diffusion equation with different Peclet numbers.

equation, choosing the point x_p arbitrarily does not increase the accuracy noticeably; hence, the point x_p is an unknown that will be solved as part of the solution process.

Since the domains of the two constrained expressions are dependent on x_p, it is simplest to write them on the basis function domain,

$$
\begin{aligned}
^{(1)}y(z, g(z)) &= {}^{(1)}g(z) + \frac{1 - 2z + z^2}{4}\left(1 - {}^{(1)}g(z_0)\right) + \frac{3 + 2z - z^3}{4}\left(y_p - {}^{(1)}g(z_f)\right) \\
&+ \frac{z^2 - 1}{2}\left(\mathrm{d}y_p/c_1 - \frac{\mathrm{d}\,{}^{(1)}g}{\mathrm{d}z}(z_f)\right) \\
^{(2)}y(z, g(z)) &= {}^{(2)}g(z) + \frac{3 - 2z - z^2}{4}\left(y_p - {}^{(2)}g(z_0)\right) - \frac{1 + 2z + z^2}{4}\,{}^{(2)}g(z_f) \\
&+ \frac{1 - z^2}{2}\left(\mathrm{d}y_p/c_2 - \frac{\mathrm{d}\,{}^{(2)}g}{\mathrm{d}z}(z_0)\right)
\end{aligned}
$$

where $^{(1)}g(z)$ is the free function for the first constrained expression, $^{(2)}g(z)$ is the free function for the second constrained expression, y_p and $\mathrm{d}y_p$ are the value and derivative of

146

the intersection point, $z \in [z_0, z_f]$ is the free function domain, and c_1 and c_2 are the constants in the linear maps from the problem domain to the basis function domain; for this problem, Legendre orthogonal polynomials are used, so $z \in [-1, 1]$. In addition, the constants in the mapping function can be expressed as,

$$c_1 = \frac{2}{x_p} \quad \text{and} \quad c_2 = \frac{2}{1 - x_p}.$$

Since the two constrained expressions are written on the basis function domain, the differential equation must be modified,

$$c_k^2 y_{zz} - c_k P_e y_z = 0,$$

where $k = 1$ if $z < z_p$ and $k = 2$ if $z > z_p$: z_p can be calculated based on x_p using the linear map between the problem domain and the basis function domain.

Now, the unknown coefficients in the two free functions, $^{(1)}\boldsymbol{\xi}$ and $^{(2)}\boldsymbol{\xi}$, and the intersection point and derivative values, x_p, y_p, and $\mathrm{d}y_p$, can be used to reduce the residual of the differential equation at each point in the discretized domain: the domain here is broken up into 200 points per constrained expression, for a total of 400 points. However, there is one more nuance to this problem: solving the problem as-is with nonlinear least-squares tends to diverge unless a good initial guess is provided. This divergent behavior is related to trying to solve for x_p, which frequently takes on values outside the domain if left unchecked. The author has identified two options that fix this divergent behavior:

1. Remove x_p from the nonlinear least-squares optimizer and estimate it using a separate, exterior optimization scheme such as a genetic algorithm.

2. Modify the nonlinear least squares to bound the values that x_p can take.

In this section, the second option is used, but the author has verified that the first method also works. When using the second option, it is tempting to simply perform an update after

147

each iteration of the nonlinear least-squares that bounds x_p to values within the domain. However, this simple change still results in divergent behavior much of the time. In terms of convergence, a better option is to use concepts from inequality constraint embedding (see appendix C for more details) to bound x_p. For this differential equation, x_p was chosen to be,

$$x_p = \hat{x}_p + (f_u - \hat{x}_p)\mathbb{1}_0(\hat{x}_p - f_u) + (f_\ell - \hat{x}_p)\mathbb{1}_0(f_\ell - \hat{x}_p),$$

where x_p is the value used in the constrained expression and $\hat{x}_p$ is the unknown used in the nonlinear least squares; $f_\ell = 1 \times 10^{-3}$ and $f_u = 1 - 1 \times 10^{-3}$ are the lower and upper bounds, respectively, on x_p. This simple change results in a nonlinear least-squares that converges.

To demonstrate the benefits of the domain splitting technique, Table D.1 shows the maximum and mean error on a test set of $1,000$ evenly spaced data points per constrained expression when using a single constrained expression for the whole domain and when using two constrained expression and a split domain as described above. The results are shown for $P_e = 1$ and $P_e = 10^6$. Each case used 200 training points per constrained expression, and Legendre polynomials up to degree 190 as the free function. Table D.1 shows that the split

Table D.1: Convection-diffusion equation error: whole vs. split domain.

P_e	Whole Domain		Split Domain	
	Maximum Error	Mean Error	Maximum Error	Mean Error
1	2.22×10^{-16}	5.62×10^{-17}	4.44×10^{-16}	8.33×10^{-17}
10^6	1.00	4.99×10^{-1}	8.61×10^{-12}	1.10×10^{-14}

domain and whole domain approaches produce similar error values for the $P_e = 1$ case when the solution's gradients are relatively small. However, when $P_e = 10^6$ and the gradients are larger, the split domain's errors are orders of magnitude lower than when using the whole domain: the maximum error is 12 orders of magnitude lower, and the average error is 13 orders of magnitude lower.

ORTHONORMAL BASIS FUNCTIONS

This appendix provides the reader with an elementary understanding of orthogonal basis functions. Any reader interested in this subject may refer to Reference [77] for a more in-depth understanding. In essence, basis functions are for a function space what vectors are for a vector space. In other words, a linear combination of basis functions spans the function space, just as a linear combination of basis vectors spans the vector space. Thus, a linear combination of basis functions is a useful free function choice for optimization problems.

E.1   Mathematical Preliminaries

This section introduces some mathematical preliminaries needed to understand the properties of basis functions, and in particular, the properties of orthogonal basis functions. The content introduced here is designed to give the reader a basic understanding and will only scratch the surface of this field of mathematics. As such, when appropriate, references will be provided so that the reader can delve deeper into these topics if desired. Moreover, this section assumes the reader is familiar with the properties of vector spaces. If the reader is unfamiliar with these topics, then they may consider reading Reference [78] for function spaces and reviewing the portion of Reference [79] dedicated to vector spaces.

This dissertation is primarily concerned with function spaces that can be used to describe continuous, non-infinite functions, as these will be particularly useful for describing the solutions of differential equations: the extended Lebesgue spaces, also known as L^{pe} spaces, are the function spaces that contain these functions. The extended Lebesgue spaces are defined based on a generalization of the p-norm used to describe vector spaces. Recall that the p-norm for a vector is

$$||\boldsymbol{x}||_p = \left(\sum_{k=1}^{n} |x_k|^p \right)^{1/p},$$

where $\boldsymbol{x} \in \mathbb{R}^n$ is an arbitrary vector, x_k are the components of $\boldsymbol{x}$, and $p \geq 1$. The p-norm of functions is,

$$||f(z)||_p = \left(\int_\Omega |f(z)|^p \, \mathrm{d}z \right)^{1/p},$$

for some arbitrary function $f(z)$ defined on the domain Ω. Note that this p-norm can also be defined with a measure $\mathrm{d}\mu(z)$, in which case the p-norm is written as

$$||f(z)||_p = \left(\int_\Omega |f(z)|^p \, \mathrm{d}\mu(z) \right)^{1/p}.$$

The rigorous mathematical definition of a measure will not be discussed here; the interested reader can refer to Reference [80] for more information. For the material in this dissertation, it is sufficient to note that the measure $\mathrm{d}\mu(z) = W(z) \, \mathrm{d}z$ where $W(z) \geq 0 \, \forall \, z \in \Omega$. The measure for a function is analogous to the weights in a weighted vector norm. An arbitrary function, $f(z)$, defined over the domain Ω is part of the $L^{pe}(\Omega, \mu)$ space if

$$||f(z)||_p = \left(\int_\Omega |f(z)|^p \, \mathrm{d}\mu(z) \right)^{1/p} < \infty.$$

This appendix will focus on basis functions in the L^{2e} space, i.e., for $p = 2$.

The generalization of the p-norm is sufficient for describing which functions are in the L^{pe} space. However, the p-norm gives no information about the orthogonality of two functions. For this, an inner product is needed. Fortunately, the $L^{2e}(\Omega, \mu)$ space already comes equipped with an inner product,

$$\langle f, g \rangle = \int_\Omega f(z) \, g(z) \, \mathrm{d}\mu(z),$$

where $f(z)$ and $g(z)$ are arbitrary functions in the $L^{2e}(\Omega, \mu)$ space, and $\langle f, g \rangle$ is used to denote an inner product between these functions. The functions f and g are considered orthogonal if $\langle f, g \rangle = 0$. Just as orthogonal basis vectors can be convenient for describing an arbitrary vector in a vector space, so too are orthogonal basis functions for describing an

arbitrary function in a function space[1].

Of course, spanning the entirety of L^{2e} space would require an infinite number of basis functions, as the dimension of the L^{2e} space is infinite. Thus, to make problems computationally tractable, a finite number, m, of basis functions is used. In general, as the number m increases, the error between the estimated and actual solution will decrease. Finally, note that the basis set domain need not coincide with the domain of the problem. If a bijective map exists that transforms the basis function domain into the problem domain, then that basis may be used to describe the problem's solution. This notion is used frequently throughout this dissertation.

Based on the description of orthogonal basis function sets thus far, one has two parameters that can be used to describe a basis set for L^{2e}:

1. The domain on which the basis is defined, Ω.

2. The measure used for the basis, μ.

In the following sections, some frequently used orthogonal basis sets will be presented. The presentation will include the domain and measure for each set and recursive generating functions for the set if they exist. The section that follows explains how to extend these basis sets to the multivariate case and concludes with a table that summarizes all the basis functions presented.

E.2   Chebyshev Orthogonal Polynomials

Chebyshev orthogonal polynomials are two sets of basis functions, the first and the second kind. They are usually indicated as $T_k(z)$ and $U_k(z)$, respectively. This section summarizes the main properties of the first kind, $T_k(z)$, only, which are defined on the domain $z \in [-1, +1]$ and with the measure $d\mu(z) = \dfrac{1}{\sqrt{1 - z^2}} \, dz$. These polynomials can be generated

---

[1]In addition, choosing orthogonal basis functions can also guarantee certain solution properties.

using the following useful recursive function,[2]

$$T_{k+1} = 2\,z\,T_k - T_{k-1} \qquad \text{starting from:} \quad \begin{cases} T_0 = 1, \\ T_1 = z. \end{cases}$$

Also, all the derivatives of Chebyshev orthogonal polynomials can be computed recursively, starting from

$$\frac{\mathrm{d}T_0}{\mathrm{d}z} = 0, \quad \frac{\mathrm{d}T_1}{\mathrm{d}z} = 1 \quad \text{or} \quad \frac{\mathrm{d}^d T_0}{\mathrm{d}z^d} = \frac{\mathrm{d}^d T_1}{\mathrm{d}z^d} = 0 \quad \forall\, d > 1,$$

and then using,

$$\frac{\mathrm{d}T_{k+1}}{\mathrm{d}z} = 2\left(T_k + z\,\frac{\mathrm{d}T_k}{\mathrm{d}z}\right) - \frac{\mathrm{d}T_{k-1}}{\mathrm{d}z}$$

$$\frac{\mathrm{d}^2 T_{k+1}}{\mathrm{d}z^2} = 2\left(2\,\frac{\mathrm{d}T_k}{\mathrm{d}z} + z\,\frac{\mathrm{d}^2 T_k}{\mathrm{d}z^2}\right) - \frac{\mathrm{d}^2 T_{k-1}}{\mathrm{d}z^2}$$

$$\vdots$$

$$\frac{\mathrm{d}^d T_{k+1}}{\mathrm{d}z^d} = 2\left(d\,\frac{\mathrm{d}^{d-1} T_k}{\mathrm{d}z^{d-1}} + z\,\frac{\mathrm{d}^d T_k}{\mathrm{d}z^d}\right) - \frac{\mathrm{d}^d T_{k-1}}{\mathrm{d}z^d} \quad \forall\, d \geq 1.$$

for $k \geq 1$. The integral of $T_k(z)$ has the following useful property,

$$\int_{-1}^{+1} T_k(z)\,\mathrm{d}z = \begin{cases} = 0 & \text{if} \quad k = 1 \\ = \dfrac{(-1)^k + 1}{1 - k^2} & \text{if} \quad k \neq 1 \end{cases}$$

---

[2]Note that in this recursive formulation and those that follow, the z argument is dropped for clarity, i.e., $T_k(z) \to T_k$.

while the inner product of two Chebyshev orthogonal polynomials satisfies the orthogonality property,

$$\langle T_i(z), T_j(z) \rangle = \int_{-1}^{+1} T_i(z)\, T_j(z)\, \frac{1}{\sqrt{1-z^2}}\, \mathrm{d}z = \begin{cases} = 0 & \text{if} \quad i \neq j \\ = \pi & \text{if} \quad i = j = 0 \\ = \pi/2 & \text{if} \quad i = j \neq 0 \end{cases} \cdot$$

Figure E.1 shows the first five Chebyshev orthogonal polynomials.
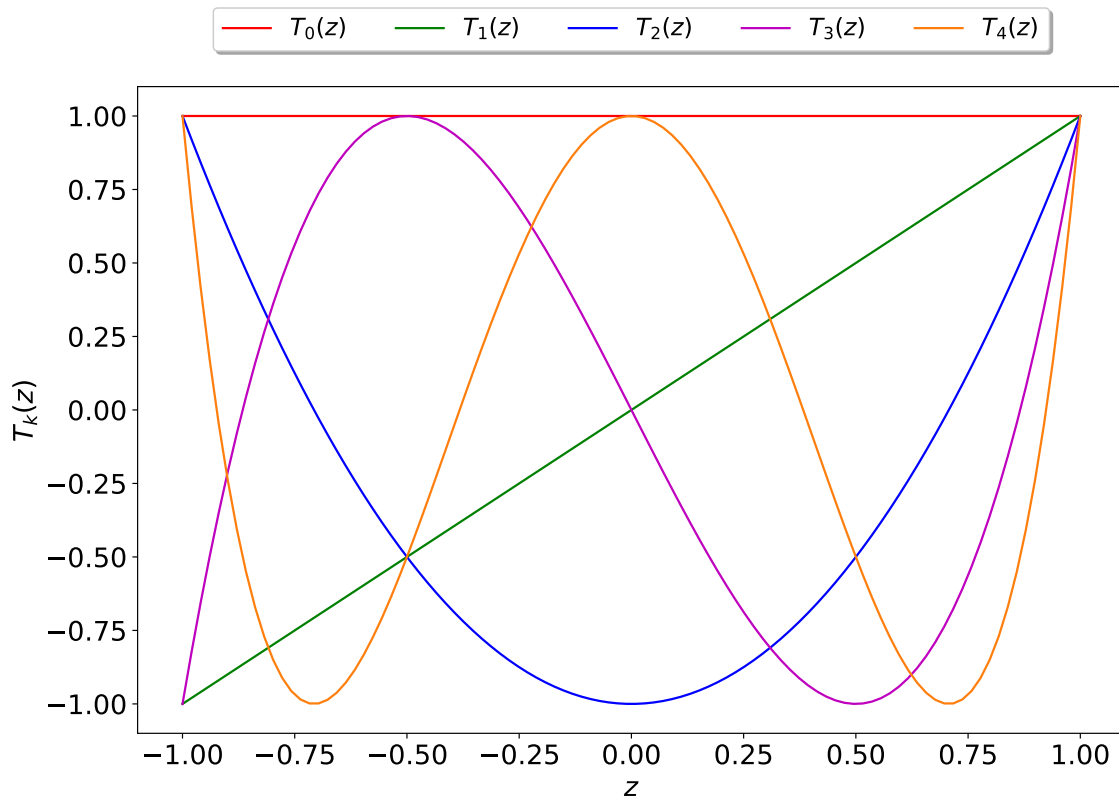
Figure E.1: First five Chebyshev orthogonal polynomials.

E.3 Legendre Orthogonal Polynomials

The Legendre orthogonal polynomials, $L_k(z)$, are defined on the domain $z \in [-1, +1]$ with measure $d\mu(z) = dz$. These polynomials can also be generated recursively by,

$$L_{k+1} = \frac{2k+1}{k+1} z\, L_k - \frac{k}{k+1} L_{k-1} \qquad \text{starting with:} \begin{cases} L_0 = 1 \\ L_1 = z. \end{cases}$$

All derivatives of Legendre orthogonal polynomials can be computed in a recursive way, starting from,

$$\frac{dL_0}{dz} = 0, \quad \frac{dL_1}{dz} = 1 \qquad \text{or} \qquad \frac{d^d L_0}{dz^d} = \frac{d^d L_1}{dz^d} = 0 \quad \forall\, d > 1,$$

and continuing with,

$$\frac{dL_{k+1}}{dz} = \frac{2k+1}{k+1}\left(L_k + z\frac{dL_k}{dz} \right) - \frac{k}{k+1}\frac{dL_{k-1}}{dz}$$

$$\frac{d^2 L_{k+1}}{dz^2} = \frac{2k+1}{k+1}\left(2\frac{dL_k}{dz} + z\frac{d^2 L_k}{dz^2} \right) - \frac{k}{k+1}\frac{d^2 L_{k-1}}{dz^2}$$

$$\vdots$$

$$\frac{d^d L_{k+1}}{dz^d} = \frac{2k+1}{k+1}\left(d\frac{d^{d-1} L_k}{dz^{d-1}} + z\frac{d^d L_k}{dz^d} \right) - \frac{k}{k+1}\frac{d^d L_{k-1}}{dz^d} \quad \forall\, d \geq 1,$$

for $k \geq 1$. In addition, the inner products of the Legendre polynomials highlight their orthogonality,

$$\langle L_i(z), L_j(z) \rangle = \int_{-1}^{+1} L_i(z)\, L_j(z)\ dz = \frac{2}{2i+1}\, \delta_{ij}.$$

Figure E.2 shows the first five Legendre orthogonal Polynomials.
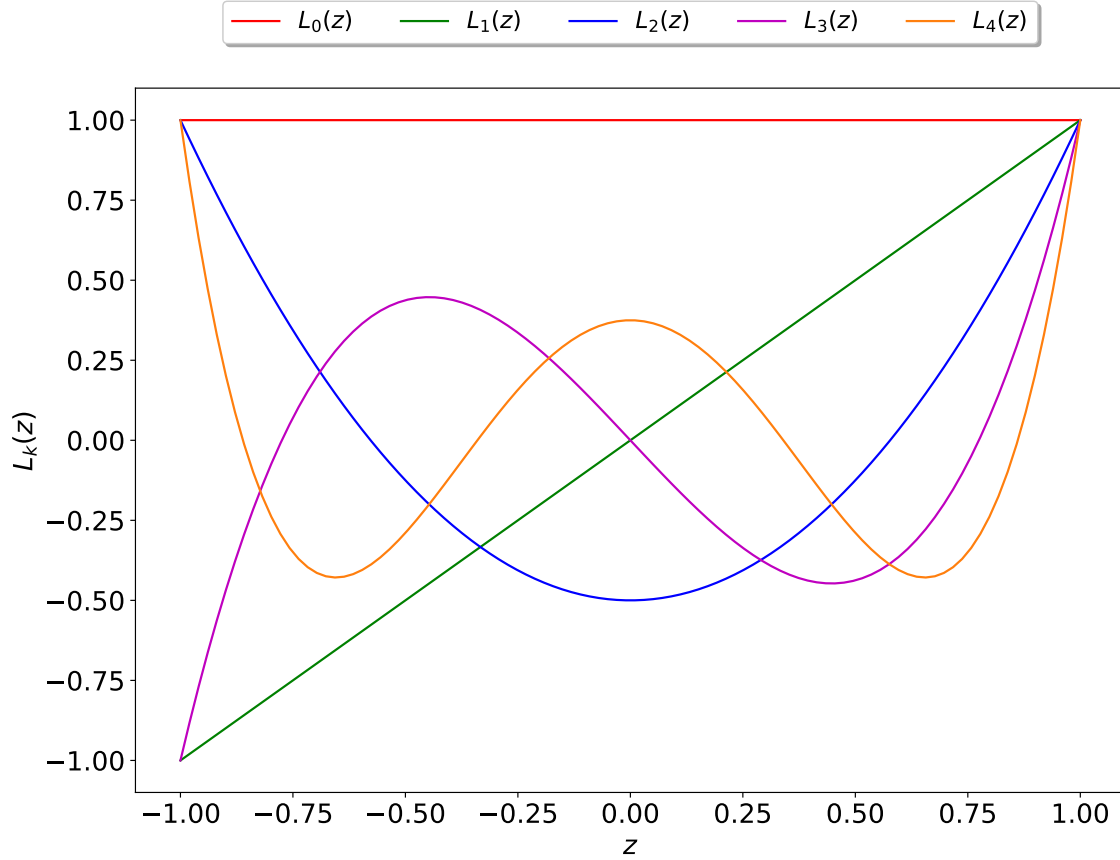
Figure E.2: First five Legendre orthogonal polynomials.

E.4   Laguerre Orthogonal Polynomials

Laguerre orthogonal polynomials, $L_k(z)$, are defined on the domain $[0, \infty)$ and by the measure $\mathrm{d}\mu(z) = e^{-z} \, \mathrm{d}z$. They are generated using the recursive function,

$$L_{k+1} = \frac{2k + 1 - z}{k + 1} \, L_k - \frac{k}{k + 1} \, L_{k-1} \qquad \text{starting with:} \quad \begin{cases} L_0 = & 1, \\[2mm] L_1 = & 1 - z. \end{cases}$$

All derivatives of Laguerre orthogonal polynomials can be computed recursively, starting from

$$\frac{\mathrm{d}L_0}{\mathrm{d}z} = 0, \quad \frac{\mathrm{d}L_1}{\mathrm{d}z} = -1 \qquad \text{or} \qquad \frac{\mathrm{d}^d L_0}{\mathrm{d}z^d} = \frac{\mathrm{d}^d L_1}{\mathrm{d}z^d} = 0 \quad \forall \, d > 1,$$

then using

$$\frac{\mathrm{d}L_{k+1}}{\mathrm{d}z} = \frac{2k+1-z}{k+1}\frac{\mathrm{d}L_k}{\mathrm{d}z} - \frac{1}{k+1}L_k - \frac{k}{k+1}\frac{\mathrm{d}L_{k-1}}{\mathrm{d}z}$$

$$\frac{\mathrm{d}^2L_{k+1}}{\mathrm{d}z^2} = \frac{2k+1-z}{k+1}\frac{\mathrm{d}^2L_k}{\mathrm{d}z^2} - \frac{2}{k+1}\frac{\mathrm{d}L_k}{\mathrm{d}z} - \frac{k}{k+1}\frac{\mathrm{d}^2L_{k-1}}{\mathrm{d}z^2}$$

$$\vdots$$

$$\frac{\mathrm{d}^dL_{k+1}}{\mathrm{d}z^d} = \frac{2k+1-z}{k+1}\frac{\mathrm{d}^dL_k}{\mathrm{d}z^d} - \frac{d}{k+1}\frac{\mathrm{d}^{d-1}L_k}{\mathrm{d}z^{d-1}} - \frac{k}{k+1}\frac{\mathrm{d}^dL_{k-1}}{\mathrm{d}z^d} \quad \forall\, d \geq 1,$$

for $k \geq 1$.

Figure E.3 shows the first five Laguerre orthogonal Polynomials.
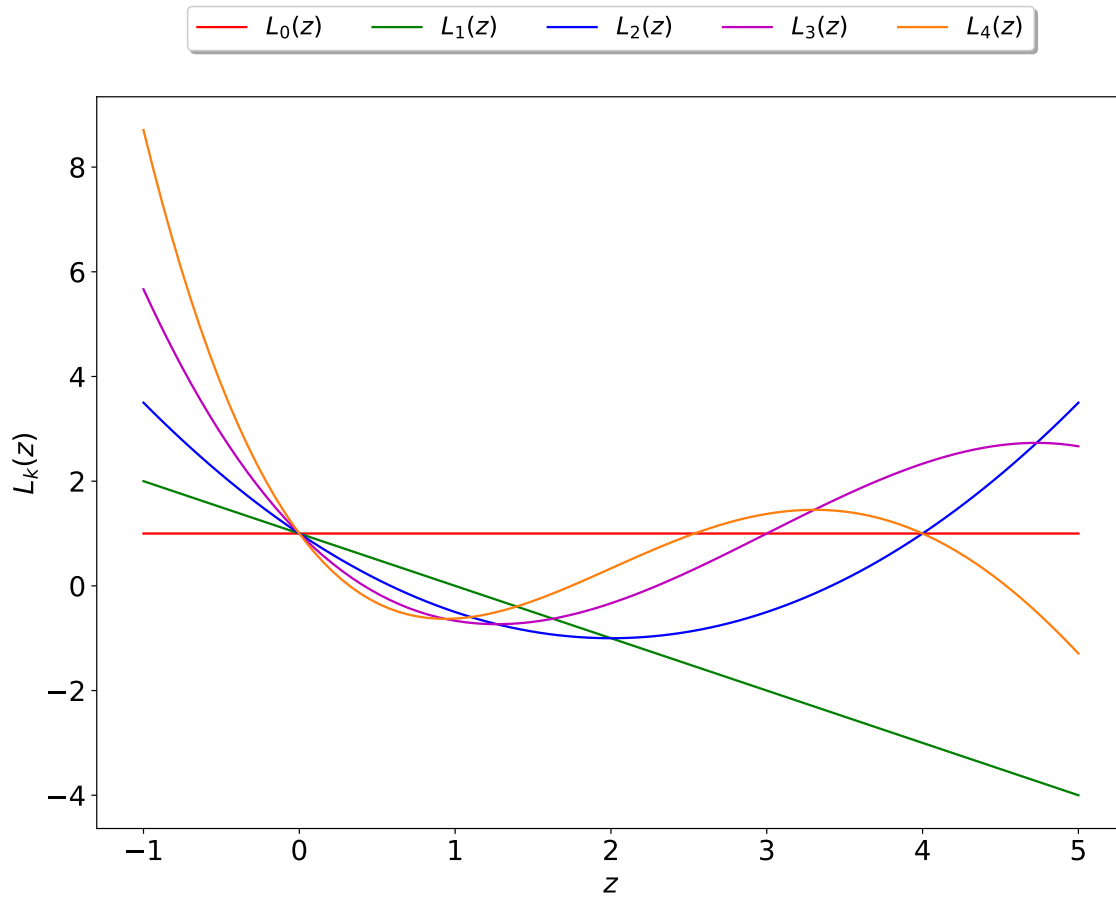
Figure E.3: First five Laguerre orthogonal polynomials.

E.5 Hermite Orthogonal Polynomials

There are two Hermite orthogonal polynomials, the probabilists, indicated by $E_k(z)$, and the physicists, indicated by $H_k(z)$. The probabilists are defined on the domain $z \in (-\infty, \infty)$ and with the measure $d\mu(z) = e^{-(z^2/2)} \, dz$, and the physicists are defined on the domain $z \in (-\infty, \infty)$ and with the measure $d\mu(z) = e^{-z^2} \, dz$. They are both generated using recursive functions.

The probabilists' polynomials can be defined recursively by,

$$E_{k+1} = z \, E_k - k E_{k-1} \qquad \text{starting with:} \quad \begin{cases} E_0 = & 1 \\ E_1 = & z. \end{cases}$$

All derivatives can be computed recursively, starting from

$$\frac{dE_0}{dz} = 0, \quad \frac{dE_1}{dz} = 1 \qquad \text{or} \qquad \frac{d^d E_0}{dz^d} = \frac{d^d E_1}{dz^d} = 0 \quad \forall \, d > 1,$$

then using,

$$\frac{dE_{k+1}}{dz} = E_k + z \frac{dE_k}{dz} - k \frac{dE_{k-1}}{dz}$$

$$\frac{d^2 E_{k+1}}{dz^2} = 2 \frac{dE_k}{dz} + z \frac{d^2 E_k}{dz^2} - k \frac{d^2 E_{k-1}}{dz^2}$$

$$\vdots$$

$$\frac{d^d E_{k+1}}{dz^d} = d \frac{d^{d-1} E_k}{dz^{d-1}} + z \frac{d^d E_k}{dz^d} - k \frac{d^d E_{k-1}}{dz^d} \quad \forall \, d \geq 1,$$

for $k \geq 1$.

The physicists' polynomials can be defined by the recursive relationship,

$$H_{k+1} = 2z \, H_k - 2k \, H_{k-1} \qquad \text{starting with:} \quad \begin{cases} H_0 = & 1 \\ H_1 = & 2z. \end{cases}$$

All derivatives can be computed recursively, starting from

$$\frac{\mathrm{d}H_0}{\mathrm{d}z} = 0, \quad \frac{\mathrm{d}H_1}{\mathrm{d}z} = 2 \qquad \text{or} \qquad \frac{\mathrm{d}^d H_0}{\mathrm{d}z^d} = \frac{\mathrm{d}^d H_1}{\mathrm{d}z^d} = 0 \quad \forall\, d > 1,$$

then using,

$$\frac{\mathrm{d}H_{k+1}}{\mathrm{d}z} = 2H_k + 2z\frac{\mathrm{d}H_k}{\mathrm{d}z} - 2k\frac{\mathrm{d}H_{k-1}}{\mathrm{d}z}$$

$$\frac{\mathrm{d}^2 H_{k+1}}{\mathrm{d}z^2} = 4\frac{\mathrm{d}H_k}{\mathrm{d}z} + 2z\frac{\mathrm{d}^2 H_k}{\mathrm{d}z^2} - 2k\frac{\mathrm{d}^2 H_{k-1}}{\mathrm{d}z^2}$$

$$\vdots$$

$$\frac{\mathrm{d}^d H_{k+1}}{\mathrm{d}z^d} = 2d\frac{\mathrm{d}^{d-1} H_k}{\mathrm{d}z^{d-1}} + 2z\frac{\mathrm{d}^d H_k}{\mathrm{d}z^d} - 2k\frac{\mathrm{d}^d H_{k-1}}{\mathrm{d}z^d} \quad \forall\, d \geq 1,$$

for $k \geq 1$.

Figure E.4 shows the first five probabilists' and physicists' Hermite orthogonal polynomials.
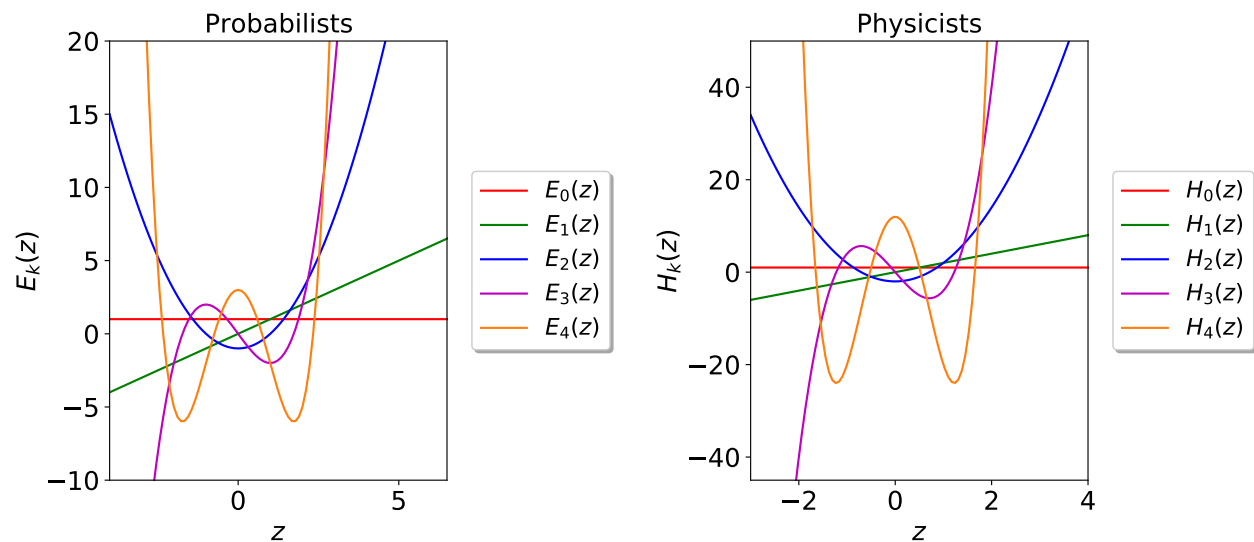
Figure E.4: First five Hermite orthogonal polynomials.

E.6 Fourier Basis

The Fourier basis is defined on the domain $z \in [-\pi, \pi]$ and with the measure $\mathrm{d}\mu(z) = \mathrm{d}z$. The basis does not have a recursive generating function. Rather, the basis can be mathematically written as,

$$
g_k(z) = \begin{cases} 1, & k = 0 \\ \cos(\lceil k/2 \rceil z), & k \text{ is even} \\ \sin(\lceil k/2 \rceil z), & k \text{ is odd} \end{cases}
$$

where $\lceil x \rceil$ rounds x to the next largest integer and $k = 0, \ldots, m$. There is no recursive relationship to compute the subsequent derivatives of Fourier bases. However, the n-th derivative can be computed using,

$$
\frac{\mathrm{d}^d g(z)}{\mathrm{d}z^d} = \begin{cases} \begin{cases} 0, & k = 0 \\ \lceil k/2 \rceil^d \cos(\lceil k/2 \rceil z), & k \text{ is even} \\ \lceil k/2 \rceil^d \sin(\lceil k/2 \rceil z), & k \text{ is odd} \end{cases} & \mod(d, 4) = 0 \\[2em] \begin{cases} 0, & k = 0 \\ -\lceil k/2 \rceil^d \sin(\lceil k/2 \rceil z), & k \text{ is even} \\ \lceil k/2 \rceil^d \cos(\lceil k/2 \rceil z), & k \text{ is odd} \end{cases} & \mod(d, 4) = 1 \\[2em] \begin{cases} 0, & k = 0 \\ -\lceil k/2 \rceil^d \cos(\lceil k/2 \rceil z), & k \text{ is even} \\ -\lceil k/2 \rceil^d \sin(\lceil k/2 \rceil z), & k \text{ is odd} \end{cases} & \mod(d, 4) = 2 \\[2em] \begin{cases} 0, & k = 0 \\ \lceil k/2 \rceil^d \sin(\lceil k/2 \rceil z), & k \text{ is even} \\ -\lceil k/2 \rceil^d \cos(\lceil k/2 \rceil z), & k \text{ is odd} \end{cases} & \mod(d, 4) = 3 \end{cases}
$$

159

whenever $d > 0$. Figure E.5 shows the first five Fourier basis functions.
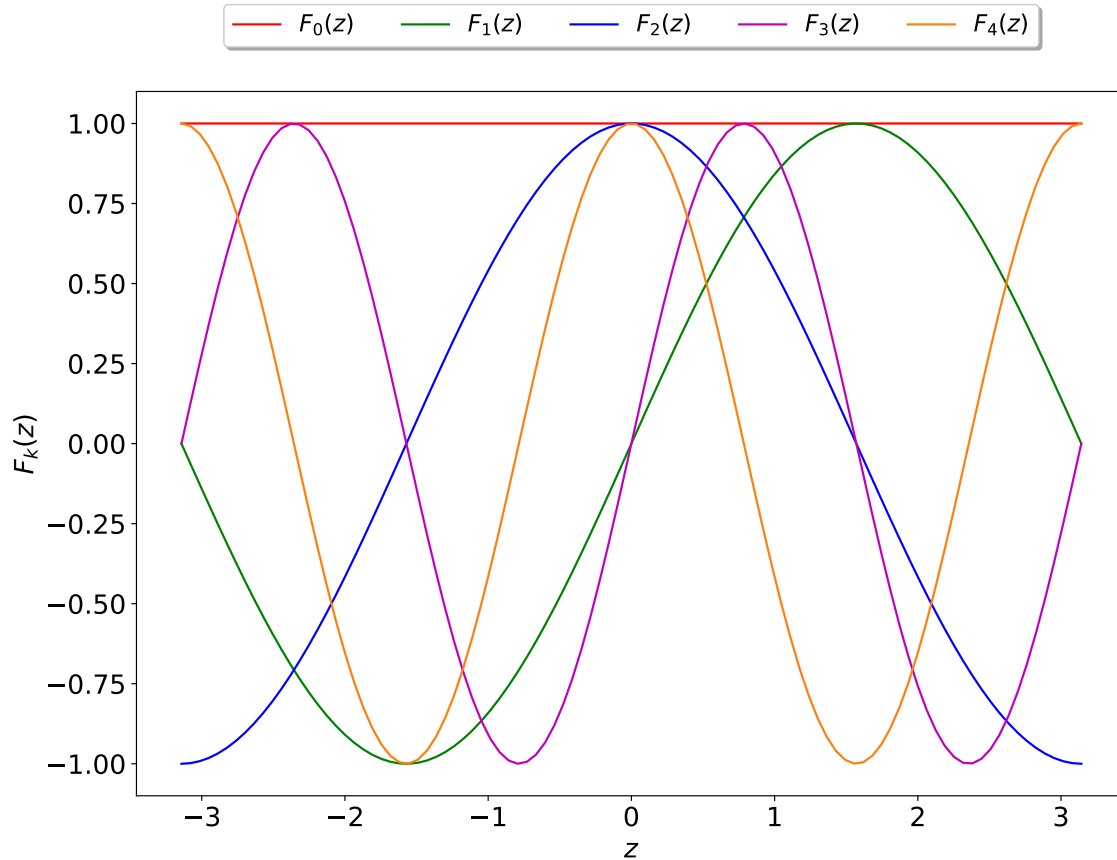
Figure E.5: First five Fourier basis functions.

E.7 Extension to Multivariate Domains

In general, multivariate orthogonal basis sets can be created by taking all possible products of functions in the basis sets that make up the individual variables. The measure that makes up this new basis set will be the product of measures of the individual basis sets, and the domain of the multivariate basis set will be the union of the domains that make up the individual basis sets. More details and insights on the 2-dimensional and n-dimensional orthogonal basis functions are contained in Reference [81] and References [77, 82], respectively.

Consider n independent variables in the vector $\boldsymbol{x} = \{x_1, x_2, \cdots, x_n\}^{\mathrm{T}}$. Moreover, let

the orthogonal basis set for each of these independent variables be denoted by $^{(k)}B_j$, where the subscript j denotes the j-th basis function and the pre-superscript k denotes the k-th independent variable. For example, the third basis function for x_2 would be $^{(2)}B_3$. The measure of the multivariate basis set will be denoted by $\mu(\boldsymbol{x}) = \prod_{k=1}^{n} {}^{(k)}\mu(x_k)$ where $^{(k)}\mu(x_k)$ is the measure for the k-th independent variable. The domain of the multivariate basis will be denoted by $\Omega = {}^{(1)}\Omega \times {}^{(2)}\Omega \times \cdots \times {}^{(n)}\Omega$, where the generic $^{(k)}\Omega$ denotes the domain of the k-th basis set. Then, an arbitrary basis function for the multivariate domain can be written as,

$$\mathcal{B}_{i_1 i_2 \ldots i_n} = {}^{(1)}B_{i_1}\,{}^{(2)}B_{i_2} \cdots {}^{(n)}B_{i_n}, \tag{E.1}$$

where $i_1, \cdots, i_n \in \mathbb{Z}^+$. In other words, Equation (E.1) generates a multivariate basis via a tensor product of univariate basis functions [83]. If one were to use all possible products of the functions in the individual basis sets which span $L^{2e}\big({}^{(k)}\Omega, {}^{(k)}\mu\big)$, i.e., use all possible combinations of $i_1, \cdots, i_n \in \mathbb{Z}^+$, an infinite set, then the resulting multivariate basis would span the multivariate function space $L^{2e}(\Omega, \mu)$. Of course, in practice this is not possible, so a finite number of basis functions from the set is used.

Consider the inner product of two different basis functions $\mathcal{B}_{i_1 \ldots i_n}$ and $\mathcal{B}_{j_1 \ldots j_n}$ where at least one $i_k \neq j_k$,

$$\langle \mathcal{B}_{i_1 \ldots i_n}, \mathcal{B}_{j_1 \ldots j_n} \rangle = \int_\Omega \mathcal{B}_{i_1 \ldots i_n}\,\mathcal{B}_{j_1 \ldots j_n}\,\mathrm{d}\mu = \prod_{k=1}^{n} \int_{\Omega_k} {}^{(k)}B_{i_k}\,{}^{(k)}B_{j_k}\,\mathrm{d}\mu_k. \tag{E.2}$$

Since these are different basis functions, there must be some $k = \kappa$ such that $i_\kappa \neq j_\kappa$. For $k = \kappa$, the integral

$$\int_{\Omega_\kappa} {}^{(\kappa)}B_{i_\kappa}\,{}^{(k)}B_{j_\kappa}\,\mathrm{d}\mu_\kappa = 0,$$

and thus, the product of integrals in Equation (E.2) is equal to zero. It follows that,

$$\langle \mathcal{B}_{i_1 \ldots i_n}, \mathcal{B}_{j_1 \ldots j_n} \rangle = 0 \quad \text{if} \quad \exists\, \kappa \mid i_\kappa \neq j_\kappa.$$

Hence, the resulting multivariate basis set is orthogonal.

Just as in the univariate case, the problem being solved must be made tractable by choosing basis functions up to some finite degree m. All the multivariate basis functions of order m are defined by choosing $i_1, \cdots, i_n$ to be on the set,

$$\{\boldsymbol{i} \mid i_k \in \mathbb{Z}^+, \sum_{k=1}^{n} (i_k - 1) = m\},$$

where i_k denotes the elements of $\boldsymbol{i}$.

Table E.1 summarizes the orthogonal basis sets described in this section.

Table E.1: Univariate orthogonal basis functions summary.

Basis function name	Domain, Ω	Measure, $\mathrm{d}\mu(z)$
Chebyshev polynomials	$[-1, 1]$	$\dfrac{1}{1 - z^2}\,\mathrm{d}z$
Legendre polynomials	$[-1, 1]$	$\mathrm{d}z$
Laguerre polynomials	$[0, \infty)$	$e^{-z}\,\mathrm{d}z$
Hermite probabilists polynomials	$(-\infty, \infty)$	$e^{-(z^2/2)}\,\mathrm{d}z$
Hermite physicists polynomials	$(-\infty, \infty)$	$e^{-z^2}\,\mathrm{d}z$
Fourier series	$[-\pi, \pi]$	$\mathrm{d}z$

LINEAR LEAST-SQUARES METHODS

There are different numerical techniques to compute the linear least-squares (LS) solution of $\mathbb{A}\,\boldsymbol{\xi} = \boldsymbol{b}$. These are:

- The Moore–Penrose inverse,

$$\boldsymbol{\xi} = (\mathbb{A}^\mathrm{T}\,\mathbb{A})^{-1}\,\mathbb{A}^\mathrm{T}\,\boldsymbol{b}.$$

- QR decomposition,

$$\mathbb{A} = Q\,R \qquad \rightarrow \qquad \boldsymbol{\xi} = R^{-1}\,Q^\mathrm{T}\,\boldsymbol{b},$$

where Q is an orthogonal matrix and R an upper triangular matrix.

- SVD decomposition,

$$\mathbb{A} = U\,\Sigma\,V^\mathrm{T} \qquad \rightarrow \qquad \boldsymbol{\xi} = \mathbb{A}^+\,\boldsymbol{b} = V\,\Sigma^+\,U^\mathrm{T}\,\boldsymbol{b}$$

where U and V are two orthogonal matrices and Σ^+ is the pseudo-inverse of Σ, which is formed by replacing every non-zero diagonal entry by its reciprocal and transposing the resulting matrix.

- Cholesky decomposition,

$$\mathbb{A}^\mathrm{T}\mathbb{A}\,\boldsymbol{\xi} = U^\mathrm{T}U\boldsymbol{\xi} = \mathbb{A}^\mathrm{T}\,\boldsymbol{b} \qquad \rightarrow \qquad \boldsymbol{\xi} = U^{-1}\left(U^{-\mathrm{T}}\mathbb{A}^\mathrm{T}\,\boldsymbol{b}\right),$$

where U is an upper triangular matrix, and consequently, U^{-1} and $U^{-\mathrm{T}}$ are easy to compute.

One can reduce the condition number of the matrix to be inverted by scaling the columns

of $\mathbb{A}$,

$$\mathbb{A}\left(SS^{-1}\right)\boldsymbol{\xi} = (\mathbb{A}S)\left(S^{-1}\boldsymbol{\xi}\right) = \mathbb{B}\,\boldsymbol{\eta} = \boldsymbol{b} \;\rightarrow\; \boldsymbol{\xi} = S\,\boldsymbol{\eta} = S\,(\mathbb{B}^{\mathrm{T}}\mathbb{B})^{-1}\mathbb{B}^{\mathrm{T}}\boldsymbol{b},$$

where S is the $m \times m$ scaling diagonal matrix whose diagonal elements are the inverse of the norms of the corresponding columns of $\mathbb{A}$: $s_{kk} = |\boldsymbol{a}_k|^{-1}$ or the maximum absolute value, $s_{kk} = \max_i |a_{ki}|$.

In this dissertation, the least-squares problem is solved using two methods: (1) the SVD decomposition introduced above (2) a combination of QR decomposition and the previously mentioned scaling, called the scaled QR approach. This approach performs the QR decomposition of the scaled matrix,

$$\mathbb{B} = \mathbb{A}\,S = Q\,R \qquad \rightarrow \qquad \boldsymbol{\xi} = S\,R^{-1}\,Q^{\mathrm{T}}\,\boldsymbol{b}.$$

APPENDIX G

TFC NUMERICAL IMPLEMENTATION IN JAX

This appendix provides a more detailed description of the code package used to implement TFC in JAX than was given in the main body of the text and some of the major challenges in doing so. In addition, a summary of the main classes available in the package is provided. For a more detailed explanation and tutorials on how to use them see either the code documentation or the TFC GitHub [64]. Note that while not discussed in detail here, this package also contains some convenience classes and functions that the reader may find useful; for example, the `MakePlot` class assists the user in creating journal-ready plots, and the `Latex` class can be used to export `NumPy` arrays to LaTeX tables.

G.1 Basis Function Classes

When applying TFC to a differential equation, the free function is used to minimize the differential equation's residual. As described in Section 3.1, two popular choices for the free function are a linear combination of n-dimensional basis functions and ELMs. Notice that these two free function choices can both be written as,

$$g(\boldsymbol{x}) = \boldsymbol{h}^{\mathrm{T}}\boldsymbol{\xi},$$

where the calculation for $\boldsymbol{h}$ evaluated at $\boldsymbol{x}$ is dictated by the basis function set chosen, if using basis functions, or by the activation function, if using an ELM. Hence, these two free function choices have a lot in common, and in the code are derived from the same abstract, parent class called `BasisFunc`. Figure G.1 shows the inheritance diagram for the basis function classes. From this base class, the univariate basis function classes are derived—`CP` for Chebyshev polynomials, `LeP` for Legendre polynomials, `FS` for Fourier series, `LaP` for Laguerre polynomials, `HoPphy` for the physicists' Hermite polynomials, and `HoPpro` for the

165
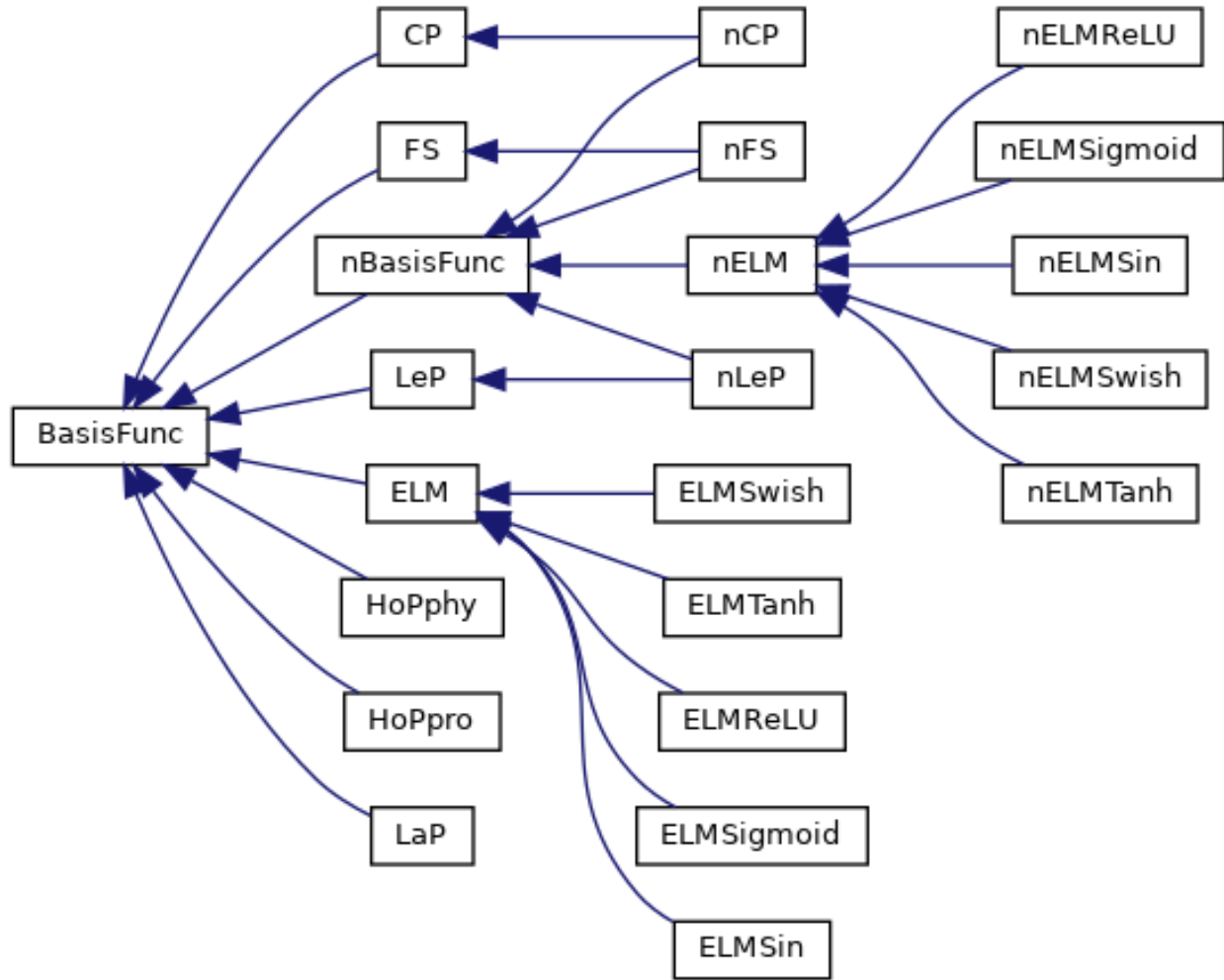
Figure G.1: Basis function class hierarchy.

probabilists' Hermite polynomials. In addition, an n-dimensional abstract class is derived from the base class, from which n-dimensional versions of Chebyshev polynomial, Legendre polynomial, and Fourier series classes are derived, called `nCP`, `nLeP`, and `nFS` respectively. Furthermore, an ELM abstract base class is derived from `BasisFunc`, from which five concrete univariate ELM classes are derived; each of these five classes implements a different activation function: `ELMSin` implements the sine activation function, `ELMSwish` implements the swish activation function, `ELMTanh` implements the hyperbolic tangent activation function, `ELMSigmoid` implements the sigmoid activation function, and `ELMReLU` implements the rectified linear activation function. The n-dimensional versions of the five univariate ELM

166

classes are derived from an abstract parent class called `nELM`, which is derived from the abstract n-dimensional basis function class mentioned earlier. The only real difference between the ELM classes and the basis function classes, besides the functions they are implementing, is the addition of randomly generated weights and biases for the hidden layer, i.e., W_1 and b_1 in Equation (3.2).

All of the classes shown in Figure G.1 are programmed in C++. The attentive reader may wonder why not just write them in Python, as the scripts that use this core code are ultimately compiled via a JIT anyway? The underlying reasons lie within the restrictions JAX has on JIT. To best understand, the reader must first recall what these basis function classes need to do:

- Calculate the basis functions and their derivatives. Recall that this requires using the linear map from $x \rightarrow z$, as the problem domain, x, and the basis function domain, z, do not generally coincide. Furthermore, recall that many of the basis functions are created via a recursion, see Appendix E.

- While the default behavior is to neglect the terms linearly independent to the support functions when calculating the basis function matrix and its derivatives, there may be times when the user wants to ignore this default behavior. This functionality is accomplished through the argument `full` that shows up as a required input argument in the C++ API and an optional keyword argument in the Python API.

Naturally, the argument `full` requires an if statement to be implemented. Unfortunately, the JIT does not allow tracing through if statements; hence, the basis function classes must be implemented as primitives. Furthermore, if built using JAX directly, the primitive recursions used by many of the basis functions would require using lax operations, as one cannot modify JAX arrays in-place. These lax operations are pure functions; rather than modifying arrays in-place, they return a new array with the updated indices. Hence, the basis function array would be copied to a new array on each iteration of the recursion: not an efficient solution.

One may be tempted to implement these using the original NumPy library, but doing so would not provide a function to XLA, which is needed for the JIT. Therefore, to make the final result JIT-able, one must write the basis functions in a lower-level language: the author chose C++.

Writing the basis functions in C++ is a fairly straightforward task, but integrating the result with the JAX JIT compiler, which is XLA, was non-trivial. Since JAX uses XLA, there is a convenient `register_custom_call_target` function that can be used to integrate C++ functions with XLA and ultimately allow them to be JIT-able. However, using this custom call function requires wrapping the C++ function into a PyCapsule object, which means the object being placed in the PyCapsule must be of type `void*`; hence, the C++ function must be cast as a `void*`. However, the C++ functions to be wrapped are methods of classes, which means they are not regular functions. Therefore, they do not have standard function pointers that can be easily cast to a `void*`.

Fortunately, this predicament can be overcome through the use of polymorphism. Notice that all the TFC free function classes are derived from `BasisFunc`, see Figure G.1. Hence, all basis function classes can be referenced using a `BasisFunc` pointer. Therefore, a static `std::vector` was added to the `BasisFunc` class that contains a pointer to each `BasisFunc` class as it is created. Furthermore, each class contains a unique integer identifier that corresponds to its pointer's position in said `std::vector`. That means that given this integer, one could access the correct element of the aforementioned `std::vector` and call the correct member function. Thus, a simple C++ wrapper function was created with the correct function signature that takes in said integer and the required inputs for the member function, calls the member function, and returns the result. Moreover, this simple wrapper function has a standard function pointer that can easily be cast to a `void*`, which subsequently can be added to a PyCapsule, and finally added to the set of JIT-able functions via `register_custom_call_target`.

G.2 Univariate TFC Class

The univariate TFC class, called `utfc`, is used to create univariate TFC expressions and solve ODEs. The class's inputs are:

- N - Number of points to use when discretizing the domain.

- nC - Number of basis functions to remove from the linear expansion. This variable is used to account for basis functions that are linearly dependent to the support functions used in the construction of the constrained expression. The constraints for each dimension can be expressed in one of two ways. Note that a value of -1 is used to indicate that no constraints exist for a particular dimension.

  1. As an integer. When expressed as an integer, the first nC basis functions are removed from the free function.

  2. As a set of integers. The basis functions corresponding to the numbers given in the set are removed from the free function.

- `m` - Degree of the basis function expansion. This number is one less than the number of basis functions used.

- `basis` - This optional string argument specifies the basis functions that will be used as the free function. The default is Chebyshev orthogonal polynomials.

- x_0 - This optional argument specifies the beginning of the DE domain. The default value of 0 will result in a DE domain that begins at 0.

- x_f - This required keyword argument specifies the end of the DE domain.

The class creates a set of N discretized points in the domains $x \in [x_0, x_f]$ and $z \in [z_0, z_f]$, where the values of z_0 and z_f are automatically chosen based on the user-specified value of the optional keyword argument `basis`. The problem domain values, x, and basis domain values, z, are both made available to the user as public variables associated with the class.

169

In addition, the TFC class creates an instance of the basis function class specified by `basis` and creates the required JAX primitives to take gradients, Jacobians, and JIT the basis functions. The basis function values and their derivatives are made available to the user via methods associated with the TFC class. These methods contain the optional keyword `full` discussed earlier, see section G.1.

G.3 Multivariate TFC Class

The multivariate TFC class is the multidimensional extension of the univariate TFC class. The input arguments to the class are the same, but they must be specified for each dimension, e.g., N specifies the number of discretization points per dimension via a list or array of the proper size. Furthermore, one additional keyword argument, `dim`, is used to specify the number of dimensions: the default is two.

G.4 Elementwise Gradients

The Autograd package contains a function called `egrad`, which stands for elementwise gradient. This function does not exist explicitly in JAX, but can easily be recreated using the same methodology as the original `egrad` function available in Autograd [62]: extract the diagonal elements of the Jacobian via a Jacobian-vector product or vector-Jacobian product. Using the tree utilities available in JAX, this function can easily be extended to pytrees, i.e., one can take elementwise gradients with respect to nested sets of Python containers.

Typically, one can use `vmap` to transform calls to the gradient function, `grad`, in JAX into elementwise gradients. However, since the basis function class must output a vector for each input, the `grad` JAX transform cannot be used. Hence, the necessity for a separate elementwise gradient function.

G.5 Extending Ordered Dictionaries

As mentioned earlier, JAX allows one to take gradients and Jacobians with respect to pytrees. This is particularly useful for coupled systems of differential equations or when splitting the domain into sub-domains, as all the unknowns can be combined into one pytree,

and the gradient and/or Jacobian of the loss function with respect to all unknowns can be written in one line.

However, if one is using an iterative least-squares optimization technique, a complication arises. Ideally, one wants to perform the following iteration,

$$\boldsymbol{\xi}_{j+1} = \boldsymbol{\xi}_j + \Delta\boldsymbol{\xi},$$

where

$$\mathbb{L}(\boldsymbol{\xi}_j) + \mathcal{J}(\boldsymbol{\xi}_j)\Delta\boldsymbol{\xi} = 0,$$

but the Jacobian in JAX computed from a dictionary actually returns a dictionary of Jacobians. This can be easily overcome using list comprehension, such as,

```
j = jacfwd(L,0)
J = np.hstack([k for k in j.values()]).
```

This allows one to calculate $\Delta\boldsymbol{\xi}$; however, this action will not be repeatable unless an ordered dictionary is used: without an ordered dictionary, the concatenation of the Jacobians might happen in a different order each time. While an ordered dictionary solves the Jacobian creation problem, it still does not allow for $\boldsymbol{\xi}_j + \Delta\boldsymbol{\xi}$ to be performed, as $\Delta\boldsymbol{\xi}$ is a `NumPy` array and $\boldsymbol{\xi}_j$ is an ordered dictionary. Therefore, the ordered dictionary is extended via operator overloading to include methods that allow for this in the `TFCDict` class.

The `TFCDict` class is designed for ordered dictionaries that have flat arrays as values. However, when dealing with vector differential equations, it is often convenient to express the free functions' unknown values associated with each component of the vector in one matrix [37]. For these types of situations, a second class, `TFCDictRobust`, has been created that is similar to `TFCDict`, but works for both flat and non-flat arrays.

G.6   Nonlinear Least-Squares

Nonlinear least squares is used throughout this dissertation to minimize the residuals of differential equations via the unknowns in the free functions. Since this method is used so often, a function called `NLLS` is included that runs the nonlinear least squares. A class called `NllsClass` exists as well for cases where the nonlinear least-squares needs to be called multiple times; the inputs to these two are similar, so only the `NLLS` function will be covered here.

The inputs to the function are:

- `xiInit` - Initial guess for the unknown parameters, ξ.

- `res` - Loss function, $\mathbb{L}$.

- `*args` - Any additional arguments taken by $\mathbb{L}$..

In addition, the following are optional keyword arguments,

- J - User-specified Jacobian. The default value is the Jacobian of $\mathbb{L}$ with respect to ξ.

- tol - Tolerance for stopping the while loop. Default is 1×10^{-13}.

- maxIter - Maximum number of nonlinear least-squares iterations. Default is 50.

- method - Method used to invert the matrix at each iteration. The default is `pinv`. The two options are:

  1. `pinv` - Uses `np.linalg.pinv` to perform the inversion.

  2. `lstsq` - Uses `np.linalg.lstsq` to perform the inversion.

- User specified condition function. Default is None, which results in a condition that checks the three stopping conditions described below.

172

- body - User specified body function. Default is None, which results in a body function that performs least-squares using the method provided and updates ξ, $\Delta\xi$ and $\mathtt{it}$, the current number of iterations.

- $\mathtt{timer}$ - Setting this to True will time the non-linear least squares using Python's $\mathtt{time.process\_time}$ timer. Note that doing so adds a slight increase in run time, as one iteration of the non-linear least squares is run first to avoid timing the JAX trace. The default is False.

- $\mathtt{printOut}$ - Setting this to true prints out the iteration number and value of $\max(|\mathbb{L}|_\infty)$ at each iteration.

- $\mathtt{printOutEnd}$ - This string argument is passed to the $\mathtt{end}$ keyword argument of the print function used in $\mathtt{printOut}$. The default value is "\n" (newline).

The outputs of the function are

1. ξ - The value of ξ at the end of the nonlinear least squares.

2. $\mathtt{it}$ - The number of iterations.

3. $\mathtt{time}$ - If the keyword argument $\mathtt{timer\ =\ True}$, then the third output is the time required to run the nonlinear least-squares as measured by the timer; otherwise, there is no third output.

When using the default condition function $\mathtt{NLLS}$ checks the following conditions; if any of the conditions are true, then the nonlinear least-squares stops iterating:

1. $|\mathbb{L}|_\infty < \mathtt{tol}$

2. $|\Delta\xi|_\infty < \mathtt{tol}$

3. Number of iterations $> \mathtt{maxIter}$

APPENDIX H

NONLINEAR SVM DERIVATION

This appendix shows how the CSVM method can be used to solve a first-order, nonlinear ODE. Consider the first-order nonlinear ODE with an initial value boundary condition,

$$\dot{y}(t) = f(t, y), \quad y(t_0) = y_0, \quad t \in [t_0, t_f].$$

Similar to the linear case, the constrained expression is,

$$y(t) = \boldsymbol{w}^{\mathrm{T}} \left[\boldsymbol{\varphi}(t) - \boldsymbol{\varphi}(t_0) \right] + y_0,$$

and the domain is discretized into N training points $t_0, t_1, \ldots, t_N$. Again, let e_i be the residual at t_i,

$$e_i = \dot{y}(t_i) - f(t_i, y(t_i)).$$

To minimize the error, the sum of the squares of the residuals is minimized. As in the linear case, the regularization term $\boldsymbol{w}^{\mathrm{T}}\boldsymbol{w}$ is added to the expression to be minimized. Now, the problem can be formulated as an optimization problem, where the Lagrange multipliers

$$
\begin{aligned}
\mathcal{L}(\boldsymbol{w}, b, \boldsymbol{e}, \boldsymbol{y}, \boldsymbol{\alpha}, \beta, \boldsymbol{\eta}) = & \frac{1}{2}(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{w} + \gamma \boldsymbol{e}^{\mathrm{T}}\boldsymbol{e}) - \sum_{i=1}^{N} \alpha_i \left[\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\varphi}'(t_i) - f(t_i, y_i) - e_i \right] \\
& - \beta[\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\varphi}(t_0) + b - y_0] - \sum_{i=1}^{N} \eta_i \left[\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\varphi}(t_i) + b - y_i \right],
\end{aligned}
\tag{H.1}
$$

$\boldsymbol{\alpha}$, β, and $\boldsymbol{\eta}$ are used to enforce the constraints: see Equation (H.1). The variables y_i are introduced into the optimization problem to keep track of the nonlinear function f at the

values corresponding to the training points.

The values where $\mathcal{L}$ are zero give candidates for the minimum.

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{w}} = \boldsymbol{0} \qquad \rightarrow \qquad \boldsymbol{w} = \sum_{i=1}^{N} \alpha_i \boldsymbol{\varphi}'(t_i) + \sum_{i=1}^{N} \eta_i \boldsymbol{\varphi}(t_i) + \beta \boldsymbol{\varphi}(t_0)$$

$$\frac{\partial \mathcal{L}}{\partial e_i} = 0 \qquad \rightarrow \qquad \gamma e_i = -\alpha_i$$

$$\frac{\partial \mathcal{L}}{\partial \alpha_i} = 0 \qquad \rightarrow \qquad \boldsymbol{w}^{\mathrm{T}} \boldsymbol{\varphi}'(t_i) = f(t_i, y_i) + e_i$$

$$\frac{\partial \mathcal{L}}{\partial \eta_i} = 0 \qquad \rightarrow \qquad y_i = \boldsymbol{w}^{\mathrm{T}} \boldsymbol{\varphi}(t_i) + b$$

$$\frac{\partial \mathcal{L}}{\partial \beta} = 0 \qquad \rightarrow \qquad \boldsymbol{w}^{\mathrm{T}} \boldsymbol{\varphi}(t_0) + b = y_0$$

$$\frac{\partial \mathcal{L}}{\partial b} = 0 \qquad \rightarrow \qquad \beta + \sum_{i=1}^{N} \eta_i = 0$$

$$\frac{\partial \mathcal{L}}{\partial y_i} = 0 \qquad \rightarrow \qquad \alpha_i f_y(t_i, y_i) + \eta_i = 0$$

A system of equations can be constructed by substituting the results found by differentiating $\mathcal{L}$ with respect to $\boldsymbol{w}$ and e_i into the remaining five equations. This leads to a set of $3N + 2$ equations and $3N + 2$ unknowns, which are α_i, η_i, y_i, β, and b: this system of equations is shown in Equation (H.2).

$$\sum_{j=1}^{N} \alpha_j \boldsymbol{\varphi}'(t_j)^{\mathrm{T}} \boldsymbol{\varphi}'(t_i) + \sum_{j=1}^{N} \eta_j \boldsymbol{\varphi}(t_j)^{\mathrm{T}} \boldsymbol{\varphi}'(t_i) + \beta \boldsymbol{\varphi}(t_0)^{\mathrm{T}} \boldsymbol{\varphi}'(t_i) + \frac{\alpha_i}{\gamma} = f(t_i, y_i)$$

$$\sum_{j=1}^{N} \alpha_j \boldsymbol{\varphi}'(t_j)^{\mathrm{T}} \boldsymbol{\varphi}(t_i) + \sum_{j=1}^{N} \eta_j \boldsymbol{\varphi}(t_j)^{\mathrm{T}} \boldsymbol{\varphi}(t_i) + \beta \boldsymbol{\varphi}(t_0)^{\mathrm{T}} \boldsymbol{\varphi}(t_i) + b - y_i = 0$$

$$\sum_{j=1}^{N} \alpha_j \boldsymbol{\varphi}'(t_j)^{\mathrm{T}} \boldsymbol{\varphi}(t_0) + \sum_{j=1}^{N} \eta_j \boldsymbol{\varphi}(t_j)^{\mathrm{T}} \boldsymbol{\varphi}(t_0) + \beta \boldsymbol{\varphi}(t_0)^{\mathrm{T}} \boldsymbol{\varphi}(t_0) + b = y_0 \qquad \text{(H.2)}$$

$$\beta + \sum_{i=j}^{N} \eta_j = 0$$

$$\alpha_i f_y(t_i, y_i) + \eta_i = 0$$

where $i = 1, ..., N$. This system of equations can be written in the dual form, in terms of

175

the Kernel matrix and is derivatives, and can be solved using least-squares. Once the set of equations has been solved, the model solution is given in the dual form by,

$$\hat{y}(t) = \sum_{i=1}^{N} \alpha_i \boldsymbol{\varphi}'(t_i)^{\mathrm{T}} \boldsymbol{\varphi}(t) + \sum_{i=1}^{N} \eta_i \boldsymbol{\varphi}(t_i)^{\mathrm{T}} \boldsymbol{\varphi}(t) + \beta \boldsymbol{\varphi}(t_0)^{\mathrm{T}} \boldsymbol{\varphi}(t) + b.$$

VARIOUS TFC EXTENSIONS

This appendix consists of various TFC concepts and extensions that did not fit elsewhere in the dissertation. Naturally, these concepts and extensions are encountered less frequently than those in the main body of the text.

I.1 Extension to Parallelotopes via Affine Transformations

Theorem 12.7 of Reference [84] shows that affine transformations map parallelotopes (n-dimensional parallelograms) to parallelotopes. This affine transformation can serve as a bijective map between a general parallelotope and an n-dimensional unit hypercube: where multivariate TFC can be applied.

Consider a general n-dimensional parallelotope with n sides emanating from each vertex. Pick a vertex arbitrarily and label it p_0. Then, for each of the n sides emanating from p_0, label the vertices at the opposite ends p_1 through p_n; again, the order in which they are labeled is arbitrary. Now, an affine transformation from the unit hypercube to the parallelotope can be defined as,

$$
\begin{Bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{Bmatrix} = \underbrace{\begin{bmatrix} p_1 - p_0 & p_2 - p_0 & \cdots & p_n - p_0 \end{bmatrix}}_{A} \begin{Bmatrix} X_1 \\ X_2 \\ \vdots \\ X_N \end{Bmatrix} + p_0,
$$

where $p_0, \ldots, p_n$ are written as column vectors, $x_1, \ldots, x_n$ are the coordinates in the parallelotope space, and $X_1, \ldots, X_n$ are the familiar Cartesian coordinates of the unit hypercube. The matrix that appears in the affine transformation will appear later and has thus been given the symbol, A. Let the entire affine transformation be denoted by $\mathcal{A}$, i.e., $\boldsymbol{x} = \mathcal{A}(\boldsymbol{X})$. Additionally, let $\mathcal{A}(X_k)$ denote the x_k portion of $\boldsymbol{x} = \mathcal{A}(\boldsymbol{X})$.

The affine transformation can be used to map functions from the parallelotope to the unit hypercube. For example, suppose there is some function $f(\boldsymbol{x})$ on the parallelotope, then, the function on the unit hypercube, $F(\boldsymbol{X})$, is

$$F(\boldsymbol{X}) = (f \circ \mathcal{A})(\boldsymbol{X}) = f(\mathcal{A}(\boldsymbol{X})).$$

Similarly, the inverse of the affine transformation can be used to map functions from the unit hypercube to the parallelotope,

$$f(\boldsymbol{x}) = (F \circ \mathcal{A}^{-1})(\boldsymbol{x}) = F(\mathcal{A}^{-1}(\boldsymbol{x})).$$

Of significance to TFC are the directions in which constraints can be specified in the parallelotope space that correspond to TFC-embeddable constraints in the unit hypercube space. Recall that the constraint operators in multivariate TFC can only operate on one independent variable— except integral constraints, see Section 2.3.1.1—else, the structure of the recursive form breaks down. For value-level constraints, a constraint that operates on X_i corresponds to the direction in parallelotope space parallel to the side containing p_i and p_0, which corresponds to the coordinate x_i; this comes merely from observing the i-th column of the affine transformation. Similarly, the allowed constraint derivative directions in the parallelotope space are those parallel to one of the parallelotope's sides:

$$\begin{aligned}
\frac{\partial F}{\partial X_i} &= \frac{\partial f}{\partial x_j} \frac{\partial x_j}{\partial X_i} \\
&= \frac{\partial f}{\partial x_j} A_{ji}.
\end{aligned} \tag{I.1}$$

Notice that the right-hand side is just the gradient of f dotted with one of the columns of A; this is precisely a directional derivative in the parallelotope space where the direction corresponds to the side containing p_i and p_0. Similarly, integral constraints must integrate in a direction parallel to one of the parallelotope's sides. It follows that general linear

178

constraints must correspond with directions parallel to one of the parallelotope's sides.

In addition, Equation I.1 has important ramifications with regards to scaling the derivatives and integrals. Let $\boldsymbol{n}_i = p_i - p_0$, $n_i = ||\boldsymbol{n}_i||$, and $\hat{\boldsymbol{n}}_i = \frac{\boldsymbol{n}_i}{n_i}$, then, Equation I.1 can be rewritten as,

$$\frac{\partial F}{\partial X_i} = \nabla f \cdot \boldsymbol{n}_i$$

and after some algebraic simplification,

$$\nabla f \cdot \hat{\boldsymbol{n}}_i = \frac{1}{n_i} \frac{\partial F}{\partial X_i}.$$

Hence, a directional derivative in the $\hat{\boldsymbol{n}}_i$ direction of the parallelotope space corresponds to a derivative in the unit hypercube space in the direction of X_i and scaled by $\frac{1}{n_i}$. Similarly, integrating in the parallelotope space will require scaling the corresponding integral by n_i,

$$\int f \, \mathrm{d}\hat{\boldsymbol{n}}_i = n_i \int F \, \mathrm{d}X_i.$$

This knowledge allows one to rewrite all the constraints on the dependent variables in the parallelotope space as constraints in the unit hypercube space. The constrained expression can be developed for the unit hypercube and then transformed back into the parallelotope space via the affine transform.

Example I.1: Parallelotope example

Throughout this example, capital letters will be used to denote quantities in the unit hypercube space, and lowercase letters will be used to denote quantities in the parallelotope space, e.g., U is the constrained expression in the unit hypercube space while $u = U \circ \mathcal{A}^{-1}$ is the constrained expression in the parallelotope space. Consider the parallelotope and constraints shown in Figure I.1.
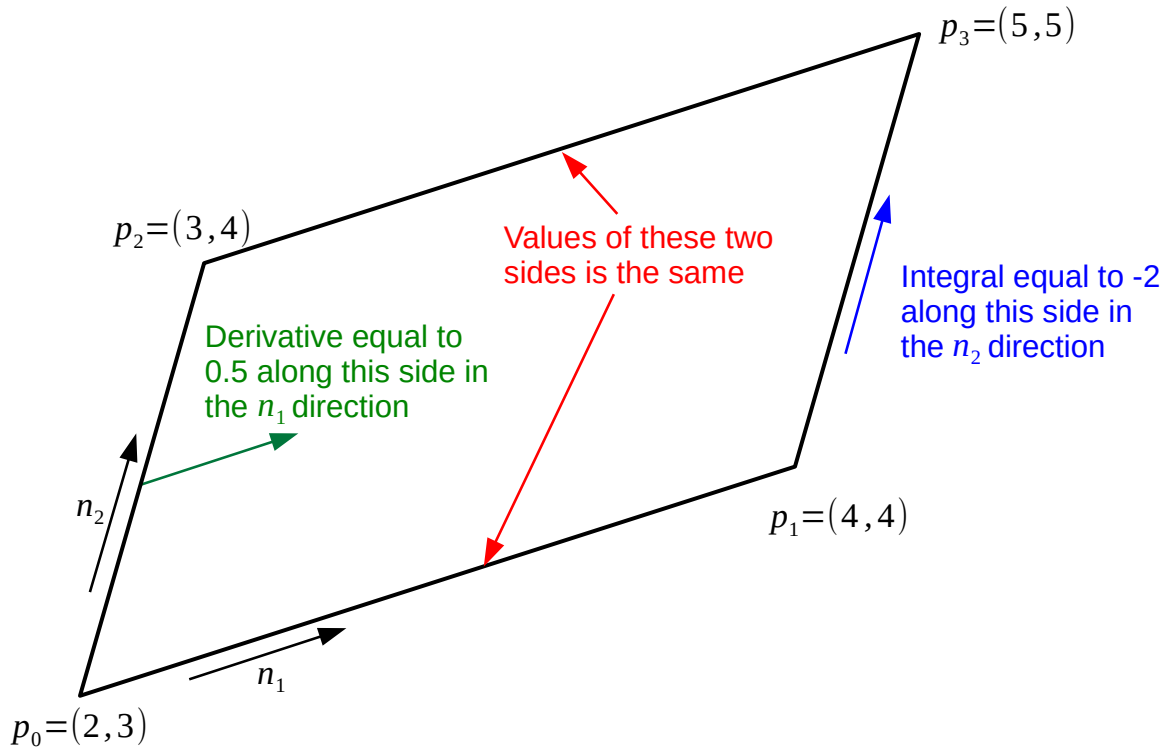
Figure I.1: Parallelotope constraints.

Using the affine transformation previously described, the parallelotope's constraints in the unit hypercube space can be written as,

$$\frac{1}{n_1}U_X(0,Y) = 0.5, \quad n_2\int_0^1 U(1,Y)\,\mathrm{d}Y = -2, \quad \text{and} \quad U(X,0) = U(X,1).$$

Using $S_1(X) = 1$ and $S_2(X) = X$ as the support functions, the univariate constrained expression for the constraints on X is,

$$^{(1)}U(X,Y,G(X,Y)) = G(X,Y) + n_1(X-1)\Big(0.5 - \frac{1}{n_1}G_X(0,Y)\Big)$$
$$+ \frac{1}{n_2}\Big(-2 - n_2\int_0^1 G(1,\tau)\,\mathrm{d}\tau\Big).$$

Utilizing the affine transformation, this constrained expression can be transformed back into the parallelotope space,

$$^{(1)}u(x, y, g(x, y)) = (^{(1)}U \circ \mathcal{A}^{-1})(x, y, g(x, y))$$

$$= g(x, y) + n_1(\mathcal{A}^{-1}(x) - 1)\left(0.5 - g_{\hat{\boldsymbol{n}}_1}(\mathcal{A}(0, \mathcal{A}^{-1}(y)))\right)$$

$$+ \frac{1}{n_2}\left(-2 - \int_0^1 g(\mathcal{A}(1, \mathcal{A}^{-1}(\tau))) \, d\hat{\boldsymbol{n}}_2\right)$$
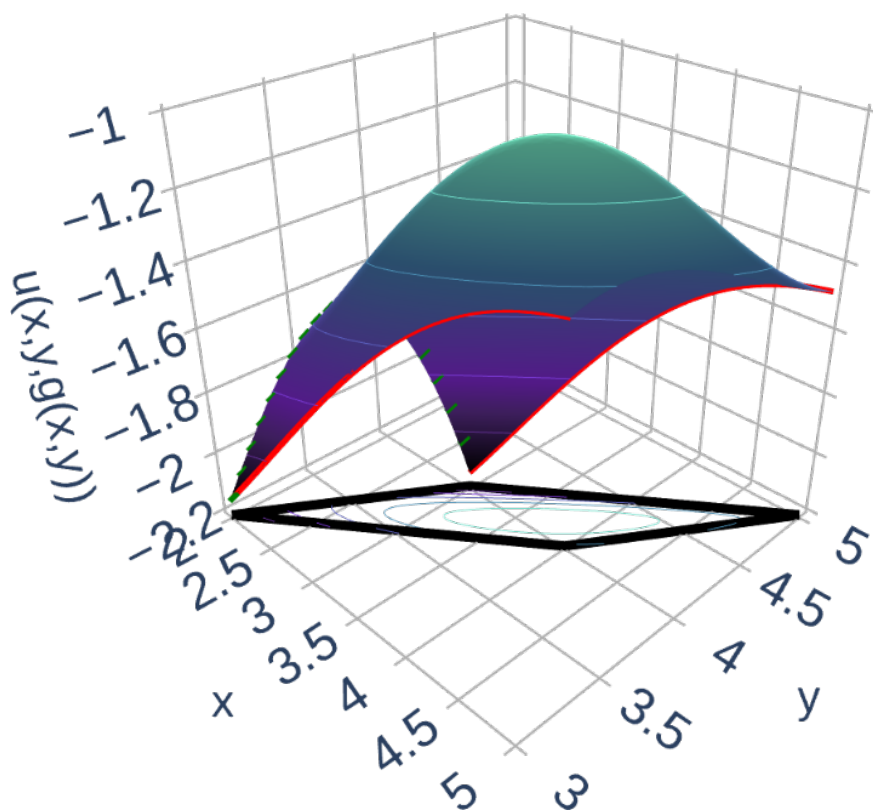

Figure I.2: Parallelotope constrained expression example. Note, this figure contains an embedded, standalone HMTL version of the plot that can be viewed/downloaded by clicking on it. Doing so may require a dedicated PDF viewer such as Adobe Acrobat or Okular.

where $g_{\hat{\boldsymbol{n}}_1}$ represents the derivative of g with respect to $\hat{\boldsymbol{n}}_1$, $\mathrm{d}\hat{\boldsymbol{n}}_2$ in the integral constraint is given in terms of the dummy variable τ (this dummy variable takes the place of y), and affine transformations have been simplified where possible, e.g., $G \circ \mathcal{A}^{-1} = g \circ \mathcal{A} \circ \mathcal{A}^{-1} = g$.

Similarly, the univariate constrained expression for the constraints on Y can be transformed into the parallelotope space,

$$^{(2)}u(x, y, g(x, y)) = \left(\frac{1}{2} - \mathcal{A}^{-1}(y)\right)\Big(g(\mathcal{A}(\mathcal{A}^{-1}(x), 1)) - g(\mathcal{A}(\mathcal{A}^{-1}(x), 0))\Big),$$

and the two can be combined using the recursive method to form a multivariate constrained expression that satisfies all of the constraints.

Figure I.2 shows the multivariate constrained expression evaluated using $g(x, y) = \sin(x)\cos(y)$. The constraint on y is shown in red, and the derivative constraint on x is shown via green lines; the integral constraint is not easily visualized but is satisfied nonetheless. The details of the surface can be a bit difficult to distinguish, so the surface's contours have been projected onto the x-y plane, and the boundaries of the surface projected onto the x-y plane are shown in black. In addition, an interactive HTML version of the plot has been embedded into the PDF and can be opened using a dedicated PDF viewer such as Adobe Acrobat or Okular.

I.2 Lower-Dimensional Constraints in n-Dimensions

There are times in n-dimensions when the constraints are not written as n-1 dimensional manifolds. For example, consider this point constraint in two-dimensional space: $u(0, 0) = 5$. Notice that these types of constraints can be written using a series of constraint operators,

$$_k\mathfrak{C}_i\Big[\cdots\big[_j\mathfrak{C}_i[u]\big]\cdots\Big] = \kappa_i$$

where the pre-subscript in front of the constraint operator represents that variable's contribution to the constraint, e.g., $_k\mathfrak{C}_i$ is x_k's contribution to the i-th constraint. For example, again consider the constraint $u(0,0) = 5$ and suppose it is the i-th constraint, then,

$$\mathfrak{C}_i[u(x_1, x_2)] = {}_1\mathfrak{C}_i\left[{}_2\mathfrak{C}_i[u(x_1, x_2)]\right] = {}_1\mathfrak{C}_i\left[u(x_1, 0)\right] = u(0,0).$$

Utilizing lower-dimensional constrained expressions, these lower-dimensional constraints can be embedded into n-dimensional constrained expressions by modifying the projection functionals.

Recall that the projection functional for the i-th constraint on the k-th dimension is written as,

$$^{(k)}\rho_i(\boldsymbol{x}, g(\boldsymbol{x})) = {}^{(k)}\kappa_i - {}^{(k)}\mathfrak{C}_i[g(\boldsymbol{x})].$$

For lower-dimensional constraints, the projection functionals are written as,

$$^{(k)}\rho_i(\boldsymbol{x}, g(\boldsymbol{x})) = {}^{(k)}_j\phi_i(x_j)\, {}^{(k)}_j\rho_i(\boldsymbol{x}, {}^{(k)}_k\mathfrak{C}_i[g(\boldsymbol{x})])$$

where $^{(k)}_j\phi_i(x_j)$ and $^{(k)}_j\rho_i(\boldsymbol{x}, g(\boldsymbol{x}))$ are the switching function and projection functional of a lower-dimensional constrained expression—there is no implied sum over i or j on the right-hand side of this equation as i and j are used here as identifying symbols rather than as indices. This lower-dimensional constrained expression is built using all of the constraint operators associated with the constraint that do not operate on x_k, i.e., $^{(k)}_j\mathfrak{C}_i$ such that $j \neq k$. Suppose that the constraint is just one dimension lower than the n-dimensional constrained expression. Then, expanding the projection functional gives a form similar to the original,

$$^{(k)}\rho_i(\boldsymbol{x}, g(\boldsymbol{x})) = {}^{(k)}_j\phi_i(x_j)\kappa_i - {}^{(k)}_j\phi_i(x_j)\,{}^{(k)}_j\mathfrak{C}_i\left[{}^{(k)}_k\mathfrak{C}_i[g(\boldsymbol{x})]\right] = {}^{(k)}_j\phi_i(x_j)\left(\kappa_i - {}^{(k)}\mathfrak{C}_i[g]\right),$$

but κ and $^{(k)}\mathfrak{C}_i[g(\boldsymbol{x})]$ are multiplied by $^{(k)}_j\phi_i$—again, no sum is implied over i or j.

Notice that if the lower-dimensional constraint is multiple dimensions lower than the n-dimensional constrained expression, then the projection functional of the lower-dimensional constrained expression will itself contain an even lower-dimensional constrained expression. Working out the algebra and simplifying results in the following form for the projection functional,

$$^{(k)}\rho_i(\boldsymbol{x}, g(\boldsymbol{x})) = {}^{(k)}\rho_i(\boldsymbol{x}, g(\boldsymbol{x})) = \left({}^{(k)}_j\phi(x_j) \cdots {}^{(k)}_h\phi(x_h) \right) \left(\kappa_i - {}^{(k)}\mathfrak{C}_i[g] \right), \qquad (\text{I.2})$$

where $j, \ldots, h$ are the dimensions associated with the constraint operators ${}^{(k)}_j\mathfrak{C}_i, \ldots, {}^{(k)}_h\mathfrak{C}_i$ that make up the constraint excluding the k-th dimension. Furthermore, the reader should note that if one has multiple lower-dimensional constraints that share the same operator, ${}^{(k)}_k\mathfrak{C}_i$, then these constraints can be written into the same projection functional, i.e., they can be collected into the same lower-dimensional constrained expression that the projection functional projects $g(\boldsymbol{x})$ to. This will modify Equation (I.2) by adding an additional two terms for each constraint: a term for the lower-dimensional switching functions and a term for the lower-dimensional projection function.

In the end, these modifications to the projection functional project $g(\boldsymbol{x})$ to the set of functions that satisfy the constraint value while maintaining two critical properties of the projection functional: (1) the projection functional is constant with respect to x_k, i.e.,

$$^{(k)}_k\mathfrak{C}_i[{}^{(k)}\phi_i(x_k) \, {}^{(k)}\rho_i(\boldsymbol{x}, g(\boldsymbol{x}))] = {}^{(k)}_k\mathfrak{C}_i[{}^{(k)}\phi_i(x_k)] \, {}^{(k)}\rho_i(\boldsymbol{x}, g(\boldsymbol{x}))$$

and (2) if $g(\boldsymbol{x})$ satisfies the constraints, then ${}^{(k)}\rho_i(\boldsymbol{x}, g(\boldsymbol{x})) = 0$, i.e., Property 3 still holds. These properties are crucial, as they are necessary and sufficient conditions for the proofs of the constrained expression theorems shown in Chapter 2 to hold. Hence, all of the constrained expression theorems shown in Chapter 2 still apply to these lower-dimensional constraints.

To help solidify these concepts, the following two examples are provided.

Example I.2: Single point constraint in two dimensions

Consider the point constraint proposed earlier: $u(0,0) = 5$. Suppose the constraint is chosen, arbitrarily, to be embedded into the constraints on x. Using the support function $s_1(x) = 1$, the switching function for x is $^{(1)}\phi_1(x) = 1$, and the resulting constrained expression is,

$$u(x, y, g(x, y)) = {}^{(1)}u(x, y, g(x, y)) = g(x, y) + {}^{(1)}\rho_1(x, y, g(x, y)).$$

Since the constraint is embedded into x, $^{(1)}\rho_1$ needs to project $g(x, y)$ to the univariate constrained expression on y. That is,

$$^{(1)}\rho_1(x, y, g(x, y)) = {}^{(1)}_2\phi_1(y) \, {}^{(1)}_2\rho_1(x, y, {}^{(1)}_1\mathcal{C}_1[g(x, y)]).$$

Let the support function for this lower-dimensional constrained expression be $s_1(y) = 1.$, then, $^{(1)}_2\phi_1(y) = 1$. The lower-dimensional projection functional is,

$$^{(1)}_2\rho_1(x, y, {}^{(1)}_1\mathcal{C}_1[g(x, y)]) = 5 - {}^{(1)}_2\mathcal{C}_1\left[{}^{(1)}_1\mathcal{C}_1[g(x, y)]\right] = 5 - {}^{(1)}\mathcal{C}_1[g(x, y)] = 5 - g(0, 0).$$

Putting everything together yields the multivariate constrained expression,

$$u(x, y, g(x, y)) = g(x, y) + 5 - g(0, 0).$$

For this example, it is simple to verify that the constrained expression satisfies the constraints for any free function.

185

Example I.3: Lower-dimensional constraints in three dimensions

Consider the following constraints in three-dimensional space,

$$u(x, y, 1) = \sin(x) \cos(y), \quad u(0, y, 0) = e^y, \quad u(1, 0, 0) = 3, \quad \text{and} \quad u(1, 1, 0) = 5.$$

Notice that the last three lower-dimensional constraints all share the same constraint operator ${}^{(3)}_3\mathfrak{C}_i$, where $i = \{2, 3, 4\}$. Therefore, they can be embedded into the same lower dimensional constraint. Utilizing multivariate TFC, the constrained expression for u can be written as,

$$u(x, y, z, g(x, y, z)) = g(x, y, 1) + z\Big(\sin(x) \cos(y) - g(x, y, z) \Big)$$
$$+ (1 - z) \, {}^{(3)}\rho_2(x, y, z, g(x, y, 0))$$

where $s_1(z) = 1$ and $s_2(z) = z$ were chosen for the support functions and ${}^{(3)}\rho_2(x, y, z, g(x, y, z))$ is the projection functional for the lower-dimensional constraints.

The lower-dimensional constrained expression is,

$$u(x, y, 0, g(x, y, z)) = g(x, y, 0) + (1 - x)\Big(e^y - g(0, y, 0) \Big) + x \, {}^{(3)}_1\rho_2(x, y, 0, g(x, y, z))$$

where $s_1(x) = 1$ and $s_2(x) = x$ were chosen for the support functions and ${}^{(3)}_1\rho_2(x, y, 0, g(x, y, 0))$ is the projection functional for the two point constraints. The univariate constrained expression for the two point constraints is,

$$u(1, y, 0, g(x, y, z)) = g(1, y, 0) + (1 - y)\Big(3 - g(1, 0, 0) \Big) + y\Big(5 - g(1, 1, 0) \Big).$$

Utilizing the switching functions and projection functionals from the univariate constrained expression, the projection functional for the two-dimensional constrained ex-

pression can be derived,

$$^{(3)}_{1}\rho_2(x, y, 0, g(x, y, 0)) = (1 - y)\Big(3 - g(1, 0, 0)\Big) + y\Big(5 - g(1, 1, 0)\Big).$$

Likewise, the now complete two-dimensional constrained expression can be used to create the projection functional for the full constrained expression,

$$^{(3)}\rho_2(x, y, z, g(x, y, z)) = (1 - x)\Big(e^y - g(0, y, 0)\Big) + x\Big((1 - y)\big(3 - g(1, 0, 0)\big)$$
$$+ y\big(5 - g(1, 1, 0)\big)\Big),$$

so the full constrained expression is,

$$u(x, y, z, g(x, y, z)) = g(x, y, z) + z\Big(\sin(x)\cos(y) - g(x, y, 1)\Big)$$
$$+ (1 - z)\Big((1 - x)\big(e^y - g(0, y, 0)\big) + x\big((1 - y)\big(3 - g(1, 0, 0)\big)$$
$$+ y\big(5 - g(1, 1, 0)\big)\big)\Big).$$

I.3 Conjecture: TFC Extends to any Field

The author of this dissertation believes that TFC constrained expressions as presented here are not restricted to the field of real numbers and extend to any mathematical field. However, the author does not feel confident enough in their abstract algebra knowledge to say this for certain, i.e., to write it as a formal theorem. Hence, rather than presenting this as a theorem in the main body of the text, it is presented here as a conjecture with supporting evidence.

The steps to write a univariate constrained expression can be succinctly summarized for a given set of constraints $\kappa_i = \mathfrak{C}_i[y(x)]$ as:

1. $\alpha_{ij} = \Big(\mathfrak{C}_i[s_j]\Big)^{-1}$

2. $\phi_i(x) = s_j(x)\alpha_{ji}$

3. $\rho_i(x, g(x)) = \kappa_i - \mathfrak{C}_i[g(x)]$

4. $y(x, g(x)) = g(x) + \phi_i(x)\rho_i(x, g(x))$

These steps and those used to prove the theorems related to the univariate constrained expression shown in Section 2.2.4 only use operations defined for a field and matrices consisting of elements of that field. Throughout the body of this dissertation, the field of real numbers was used, i.e., $x \in \mathbb{R}$, but the algebra remains the same for $x \in \mathbb{F}$ where $\mathbb{F}$ is any mathematical field. Of course, the types of constraints that can be embedded for a given field are restricted to the types of constraints that can be calculated on that field. For example, one cannot embed integral constraints into a TFC constrained expression for a finite field because one cannot calculate integrals on finite fields. Furthermore, multivariate constrained expressions are constructed via recursive application of the univariate theory, and the associated proofs also only utilize mathematical operations defined for a field. Hence, multivariate constrained expressions and their associated theorems extend to all mathematical fields as well.

To further provide evidence for this conjecture, the following two examples are provided.

Example I.4: TFC on a finite field

Consider the finite field containing the four elements $\{0, 1, A, B\}$ with the addition and multiplication tables shown in Tables I.1 and I.2.[a] Consider the following constraints,

$$u(0, y) = A, \quad u(B, y) = 1, \quad \text{and} \quad u(x, 0) = u(x, B).$$

Utilizing the theory as described in Chapter 2, the multivariate constrained expression that satisfies these constraints can be derived. In this example, the univariate constrained expression that satisfies the constraints on x is derived step by step; notice

that the steps have not been modified from those used to derive constrained expressions for real numbers. Let the support functions be $s_1(x) = 1$ and $s_2(x) = 2$, then,

Table I.1: Addition table

+	0	1	A	B
0	0	1	A	B
1	1	0	B	A
A	A	B	0	1
B	B	A	1	0

Table I.2: Multiplication table

*	0	1	A	B
0	0	0	0	0
1	0	1	A	B
A	0	A	B	1
B	0	B	1	A

$$\alpha_{ij} = \left({}^{(1)}\mathfrak{C}_i[s_j(x)] \right)^{-1} = \begin{bmatrix} 1 & 0 \\ 1 & B \end{bmatrix}^{-1}$$

$$\alpha_{ij} = \begin{bmatrix} 1 & 0 \\ A & A \end{bmatrix}.$$

The switching functions are defined as,

$$^{(1)}\phi_i = s_i(x)\alpha_{ij};$$

thus,

$$^{(1)}\phi_1(x) = Ax + 1 \quad \text{and} \quad ^{(1)}\phi_2(x) = Ax.$$

The projection functionals are defined as

$$^{(1)}\rho_i(\boldsymbol{x}, g(\boldsymbol{x})) = \kappa_i - {}^{(1)}\mathfrak{C}_i[g(\boldsymbol{x})];$$

thus,

$$^{(1)}\rho_1(x, y, g(x, y)) = A - g(0, y) \quad \text{and} \quad ^{(1)}\rho_2(x, y, g(x, y)) = 1 - g(B, y).$$

Combining the pieces yields the univariate constrained expression,

$$^{(1)}u(x, y, g(x, y)) = g(x, y) + (Ax + 1)(A - g(0, y)) + Ax(1 - g(B, y)).$$

The univariate constrained expression for the constraints on y is derived in a similar fashion,

$$^{(2)}u(x, y, g(x, y)) = g(x, y) + Ay\big(g(x, B) - g(x, 0)\big).$$

Just as with the real numbers, these two univariate constrained expressions are com-

bined recursively to yield the multivariate constrained expression:

$$u(x, y, g(x, y)) = {}^{(2)}u(x, y, {}^{(1)}u(x, y, g(x, y)))$$

$$= g(x, y) + Ax(1 - g(B, y)) + (Ax + 1)(A - g(0, y))$$

$$+ Ay\Big((Ax + 1)(A - g(0, B)) + Ax(1 - g(B, 0)) + Ax(1 - g(B, B))$$

$$- (Ax + 1)(A - g(0, 0)) + g(x, B) - g(x, 0)\Big).$$

Table I.3 shows the output of this constrained expression for $g(x, y) = Ax + xy + y$. The bottom row of the table gives the x value, and the left-most column gives the y value. The remaining table entries give the output, i.e., $u(x, y, g(x, y))$. As expected, the constraints are satisfied.

Table I.3: Finite field constrained expression output for $g(x, y) = Ax + xy + y$.

B	A	B	0	1
A	A	B	0	1
1	A	B	0	1
0	A	B	0	1
$y \diagdown x$	0	1	A	B

^aIf the reader is familiar with finite fields, they will recognize this field as $GF(2)[x]/(x^2 + x + 1)$.

Example I.5: TFC using complex numbers

Consider the following constraints,

$$y\left(\frac{i}{2}\right) = 1 + \pi i, \quad y(1) = y(i), \quad \text{and} \quad y(2 + i) + y_x(1) = 2i.$$

Let the support functions be $s_1(x) = 1$, $s_2(x) = x$, and $s_3(x) = x^2$. Then,

$$
\alpha_{ij} = \left({}^{(1)}\mathfrak{C}_i[s_j(x)] \right)^{-1} = \begin{bmatrix} 1 & \frac{i}{2} & -\frac{1}{4} \\ 0 & -1+i & -2 \\ 1 & 3+i & 5+4i \end{bmatrix}^{-1}
$$

$$
\alpha_{ij} = \begin{bmatrix} \frac{132}{125} - \frac{24i}{125} & -\frac{82}{125} + \frac{49i}{125} & -\frac{7}{125} + \frac{24i}{125} \\ \frac{52}{125} + \frac{36i}{125} & -\frac{129}{250} - \frac{397i}{250} & -\frac{52}{125} - \frac{36i}{125} \\ -\frac{44}{125} + \frac{8i}{125} & \frac{69}{125} + \frac{67i}{125} & \frac{44}{125} - \frac{8i}{125} \end{bmatrix},
$$

and

$$
\phi_1(x) = \left(-\frac{44}{125} + \frac{8i}{125} \right) x^2 + \left(\frac{52}{125} + \frac{36i}{125} \right) x + \left(\frac{132}{125} - \frac{24i}{125} \right)
$$

$$
\phi_2(x) = \left(\frac{69}{125} + \frac{67i}{125} \right) x^2 - \left(\frac{129}{250} + \frac{397i}{250} \right) x + \left(-\frac{82}{125} + \frac{49i}{125} \right)
$$

$$
\phi_3(x) = \left(\frac{44}{125} - \frac{8i}{125} \right) x^2 - \left(\frac{52}{125} + \frac{36i}{125} \right) x + \left(-\frac{7}{125} + \frac{24i}{125} \right).
$$

The projection functionals are,

$$
\rho_1(x) = 1 + \pi i - g\left(\frac{i}{2} \right)
$$

$$
\rho_2(x) = g(1) - g(i)
$$

$$
\rho_3(x) = 2i - g(2+i) - g_x(1).
$$

Using the switching functions and projection functionals, the full constrained expression is,

$$
y(x, g(x)) = g(x)
$$

$$+ \left(\left(-\frac{44}{125} + \frac{8i}{125} \right) x^2 + \left(\frac{52}{125} + \frac{36i}{125} \right) x + \left(\frac{132}{125} - \frac{24i}{125} \right) \right) \left(1 + \pi i - g \left(\frac{i}{2} \right) \right)$$

$$+ \left(\left(\frac{69}{125} + \frac{67i}{125} \right) x^2 - \left(\frac{129}{250} + \frac{397i}{250} \right) x + \left(-\frac{82}{125} + \frac{49i}{125} \right) \right) \left(g(1) - g(i) \right)$$

$$+ \left(\left(\frac{44}{125} - \frac{8i}{125} \right) x^2 - \left(\frac{52}{125} + \frac{36i}{125} \right) x + \left(-\frac{7}{125} + \frac{24i}{125} \right) \right) \left(2i - g(2+i) - g_x(1) \right).$$

Figures I.3 and I.4 show the real and imaginary portions of the constrained expression respectively for $g(x) = \frac{1}{4}x + 0.3i \cos(x/4)$. In these figures, $\text{Re}[\cdot]$ is used to denote the real portion, and $\text{Im}[\cdot]$ is used to denote the imaginary portion. The first constraint is plotted as a red point, and the second constraint is plotted as green points. The third constraint is harder to visualize but is satisfied nonetheless.
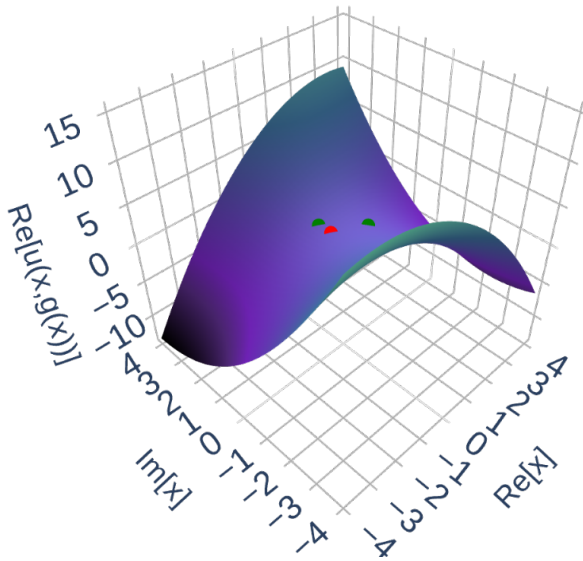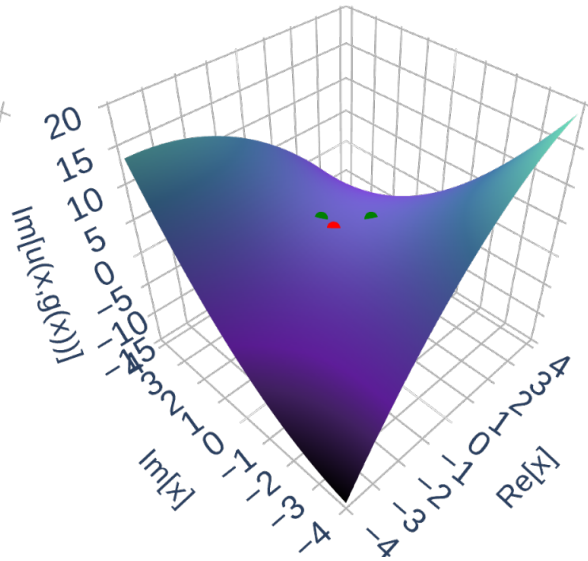

Figure I.3: Complex constrained expression, real portion.

Figure I.4: Complex constrained expression, imaginary portion.