mathematics

MDPI

Article

The Multivariate Theory of Functional Connections: Theory, Proofs, and Application in Partial Differential Equations

Carl Leake *, Hunter Johnston and Daniele Mortari

Aerospace Engineering, Texas A&M University, College Station, TX 77843, USA; hunterjohnston@tamu.edu (H.J.); mortari@tamu.edu (D.M.)

* Correspondence: leakec@tamu.edu

Received: 8 July 2020; Accepted: 4 August 2020; Published: 6 August 2020

Abstract: This article presents a reformulation of the Theory of Functional Connections: a general methodology for functional interpolation that can embed a set of user-specified linear constraints. The reformulation presented in this paper exploits the underlying functional structure presented in the seminal paper on the Theory of Functional Connections to ease the derivation of these interpolating functionals—called constrained expressions—and provides rigorous terminology that lends itself to straightforward derivations of mathematical proofs regarding the properties of these constrained expressions. Furthermore, the extension of the technique to and proofs in n-dimensions is immediate through a recursive application of the univariate formulation. In all, the results of this reformulation are compared to prior work to highlight the novelty and mathematical convenience of using this approach. Finally, the methodology presented in this paper is applied to two partial differential equations with different boundary conditions, and, when data is available, the results are compared to state-of-the-art methods.

Keywords: Theory of Functional Connections; partial differential equations; least-squares; functional interpolation

1. Introduction

The Theory of Functional Connections (TFC) is a mathematical framework used to construct functionals, functions of functions, that represent the family of all possible functions that satisfy some user-defined constraints; these functionals are referred to as "constrained expressions" in the context of the TFC. In other words, the TFC is a framework for performing functional interpolation. In the seminal paper on TFC [1], a univariate framework was presented that could construct constrained expressions for constraints on the values of points or arbitrary order derivatives at points. Furthermore, Reference [1] showed how to construct constrained expressions for constraints consisting of linear combinations of values and derivatives at points, called linear constraints; for example, $y(x_1) + 3\pi\, y_{xx}(x_2) = 2\,e$, for some points x_1 and x_2, where y_{xx} symbolizes the second order derivative of y with respect to x. In the current formulation, the univariate constrained expression has been used for a variety of applications, including solving linear and non-linear differential equations [2,3], hybrid systems [4], optimal control problems [5,6], in quadratic and nonlinear programming [7], and other applications [8].

Recently, the TFC method has been extended to n-dimensions [9]. This multivariate framework can provide functionals representing all possible n-dimensional manifolds subject to constraints on the value and arbitrary order derivative of $n-1$ dimensional manifolds. However, Reference [9] does not discuss how the multivariate framework can be used to construct constrained expressions for linear constraints. Regardless, these multivariate constrained expressions have been used to embed constraints into machine learning frameworks [10–12] for use in solving partial differential equations (PDEs). Moreover, it was shown that this framework may be combined with orthogonal basis functions to solve PDEs [13]; this is essentially the n-dimensional equivalent of the ordinary differential equations (ODEs) solved using the univariate formulation [2,3].

The contributions of this article are threefold. First, this article examines the underlying structure of univariate constrained expressions and provides an alternative method for deriving them. This structure is leveraged to derive mathematical proofs regarding the properties of univariate constrained expressions. Second, using the aforementioned structure, this article extends the multivariate formulation presented in Reference [9] to include linear constraints by introducing the recursive application of univariate constrained expressions as a method for generating multivariate constrained expressions. Further, mathematical proofs are provided that prove the resultant constrained expressions indeed represent all possible manifolds subject to the given constraints. Thirdly, this article presents how the multivariate constrained expressions can be combined with a linear expansion of n-dimensional orthogonal basis functions to numerically estimate the solutions of PDEs. While Reference [13] showed that solving PDEs with the multivariate TFC is possible, it merely gave a cursory overview, skipping some rather important details; this article fills in those gaps.

The remainder of this article is structured as follows. Section 2 introduces the univariate constrained expression, examines its underlying structure, and provides an alternative method to derive univariate constrained expressions. Then, in Section 3, this structure is leveraged to rigorously define the univariate TFC constrained expression and provide some related mathematical proofs. In Section 4, this new structure and the mathematical proofs are extended to n-dimensions, and a compact tensor form of the multivariate constrained expression is provided. Section 5 discusses how to combine the multivariate constrained expression with multivariate basis functions to estimate the solutions of PDEs. Then, in Section 6, this method is used to estimate the solution of two PDEs, and the results are compared with state-of-the-art methods when data is available. Finally, Section 7 summarizes the article and provides some potential future directions for follow-on research.

2. Univariate TFC

Extending the multivariate TFC to include linear constraints requires recursive applications of the univariate TFC. Hence, it is paramount the reader understand univariate TFC before moving to the multivariate case. First, the general form of the univariate constrained expression will be presented, followed by a few examples. These examples serve to solidify the readers understanding of the univariate constrained expression, as well as highlight nuances of deriving constrained expressions. In addition, this section includes mathematical proofs that univariate TFC constrained expressions indeed represent the family of all possible functions that satisfy the constraints.

Given a set of k constraints, the univariate constrained expression takes the following form,

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

where $g(x)$ is a free function, $s_j(x)$ are k mutually linearly independent functions called support functions, and η_j 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.

The following examples start from Equation (1), the framework proposed in the seminal paper on TFC [1], and highlight a unified structure that underlies the univariate TFC constrained expressions. Following these examples is a section that rigorously defines this structure and provides important mathematical proofs.

2.1. Univariate Example # 1: Constraints at a Point

Constraints at a point consist of constraints on the value at a point and constraints on a derivative at a point. Take for example the follow constraints,

$$y(0) = 1, \quad y_x(1) = 2, \quad y(2) = 3.$$

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

$$\begin{aligned} y(0) &= 1 = g(0) + \eta_1 \\ y_x(1) &= 2 = g_x(1) + 2\eta_2 + 3\eta_3 \\ y(2) &= 3 = g(2) + \eta_1 + 4\eta_2 + 8\eta_3. \end{aligned}$$

Solving this set of equations for the unknowns η_j leads to the solution,

$$\begin{aligned} \eta_1 &= 1 - g(0) \\ \eta_2 &= \frac{10 - 3g(0) + 3g(2) - 8g_x(1)}{4} \\ \eta_3 &= \frac{g(0) - g(2) + 2g_x(1)}{2}. \end{aligned}$$

Substituting the coefficient functionals back into Equation (1) and simplifying yields,

$$\begin{aligned} y(x, g(x)) = g(x) &+ \frac{-2x^3 + 3x^2 + 4}{4}\Big(1 - g(0)\Big) + \Big(- x^3 + 2x^2\Big)\Big(2 - g_x(1)\Big) \\ &+ \frac{2x^3 - 3x^2}{4}\Big(3 - g(2)\Big). \end{aligned} \tag{2}$$

It is simple to verify that regardless of how $g(x)$ is chosen, provided $g(x)$ exists at the constraint points, Equation (2) always satisfies the given constraints.

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

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

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 $s_1 = 1$, $s_2 = x$, and $s_3 = x^2$ 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 singularity of the support function matrix is dependent 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 for 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),

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

The first term in the product, $\phi_1(x)$, is called a *switching function*—Reference [1] introduced these switching functions as "coefficient" functions, $\boldsymbol{\beta}_k$—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. 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 that constraint point. It is called the projection functional because it projects the free function to the set of functions that vanish at the constraint. 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 projection functional, $\rho_1(x, g(x))$, is just the difference between the constraint $y(0) = 1$ and the free function evaluated at the constraint point, $g(0)$. This structure is important, as it shows up in the other constraint types too. Property 1 follows from the definition of the projection functional.

Property 1. *The projection functionals for constraints at a point are always equal to zero if the free function, $g(x)$, is selected such that it satisfies the associated constraint.*

For example, if $g(x)$ is selected such that $g(0) = 1$, then the first projection functional in this example becomes $\rho_1(x, g(x)) = 1 - g(0) = 0$. Based on this structure, an alternate way to define the constrained expression, shown in Equation (3), can be derived,

$$y(x, g(x)) = g(x) + \sum_{j=1}^{k} \phi_j(x)\, \rho_j(x, g(x)). \tag{3}$$

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. Based on this definition, 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, we will use the same constraints and support functions and rederive the constrained expression shown in Equation (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 paper, 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,

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

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.

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

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 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). This approach to derive constrained expressions using switching functions has, similar to the first approach, the risk of obtaining a singular matrix if the support functions selected are not able to interpolate the constraints. However, as will be demonstrated in the coming sections, this approach can be easily extended to multivariate domains via recursive applications of the univariate theory, and this approach lends itself nicely to mathematical proofs.

2.2. Univariate Example # 2: Linear Constraints

Linear constraints consist of linear combinations of the previous types of constraints. Note that by this definition, relative constraints such as $y(0) = y(1)$ are just a special case of linear constraints. Take for example the following two constraints,

$$y(0) = y(1), \qquad \text{and} \qquad 3 = 2y(2) + \pi y_{xx}(0).$$

To generate a constrained expression, the projection functionals and switching functions must be found. Similar to the constraints at a point, first the constraints are arranged such that one side of the constraint is equal to zero; for example,

$$y(0) - y(1) = 0 \qquad \text{and} \qquad 3 - 2y(2) - \pi y_{xx}(0) = 0.$$

Then, the projection functionals can be defined by replacing $y(x)$ with $g(x)$. Thus,

$$\rho_1(x, g(x)) = g(0) - g(1) \quad \text{and} \quad \rho_2(x, g(x)) = 3 - 2g(2) - \pi g_{xx}(0).$$

The switching functions are again defined such that they are equal to 1 when evaluated with their associated constraint, and equal to 0 when evaluated at all other constraints. The word "evaluation" in the previous sentence requires clarification. Substitution of the constrained expression back into the constraint should result in the expression $0 = 0$ (i.e., the constraint is satisfied). When doing so, the switching functions, $\phi(x)$, will be evaluated in the same way $y(x)$ is evaluated in the constraint. Thus, the constants within the constraint are not used in the evaluation. Moreover, because the projection functional is designed to exactly cancel the values of the free function in the constraint, the switching function equations should have the opposite sign. Hence, evaluation means to replace the function with the switching function, remove any terms not multiplied by the switching function, and multiply the entire equation by -1. Any reader confused by this linguistic definition of switching function evaluation may refer to Property 4, which defines the switching function evaluation mathematically. For this example, this leads to,

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

for the first switching function, and

$$\phi_2(1) - \phi_2(0) = 0, \qquad 2\phi_2(2) + \pi \frac{\partial^2 \phi_2}{\partial x^2}(0) = 1,$$

for the second switching function. Note that while this "evaluation" definition may seem convoluted at first, it is in fact exactly what was done for the constraints at a point case. However, in that case, due to the simple nature of the constraints and the way the projection functionals were defined, this was simply the switching function evaluated at the point.

Similar to the constraints at a point case, the switching functions are defined as a linear combination of support functions with unknown coefficients. Again, this can be written compactly in matrix form. For this example, the support functions $s_1(x) = 1$ and $s_2(x) = x$ are chosen. Then,

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

which results in the switching functions,

$$\phi_1(x) = x - 2, \quad \phi_2(x) = \frac{1}{2}.$$

Substituting the switching and projection functionals back into the constrained expression form given in Equation (3) yields,

$$y(x, g(x)) = g(x) + (x - 2)\Big(g(0) - g(1)\Big) + \frac{1}{2}\Big(3 - 2g(2) - \pi g_{xx}(0)\Big).$$

By substituting this expression for y back into the constraints, one can verify that indeed this constraint expression satisfies the constraints regardless of the choice of free function $g(x)$. Property 2 extends Property 1 to linear constraints.

Property 2. *The projection functionals for linear constraints are always equal to zero if the free function is selected such that it satisfies the associated constraint.*

For example, if $g(x)$ is selected such that $g(1) = g(0)$, then the first projection functional in this example becomes $\rho_1(x, g(x)) = g(1) - g(0) = 0$.

3. General Formulation of the Univariate TFC

This section rigorously defines the TFC constrained expression and provides some relevant proofs. First, a functional is defined and its properties are investigated.

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 function as defined here is coincident 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 function, $g(x)$, as its input and produces a function, $h(x) = f(x, g(x))$ for any specified $g(x)$, as its output. Since this article is focused on constraint embedding, or in other words functional interpolation, we will not concern ourselves with the domain/range of the input and output functions. Rather, we will discuss functionals only 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,* $f(x, g(x))$*, is said to be 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 said to be surjective if for every function in the codomain,* $h(x)$*, there exists at least one* $g(x)$ *such that* $h(x) = f(x, g(x))$*.*

Definition 4. *A functional,* $f(x, g(x))$*, is said to be bijective if it is both injective and surjective.*

To elaborate, Figure 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 $h(x)$ in the codomain there is at most one $g(x)$ that maps $f(x, g(x))$ to $h(x)$.

However, the functional is not surjective, because the functional does not span the space of the codomain. For example, consider the desired output function $h(x) = -2$: 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 the set of all smooth functions $h(x)$ such that $h(0) = 0$. This functional is surjective because it spans the space 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 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 the typical definition of a projection matrix $P^n = P$ for some $n \in \mathbb{Z}^+$. In other words, when P operates on itself, it produces itself: a projection property for functionals can be defined similarly.

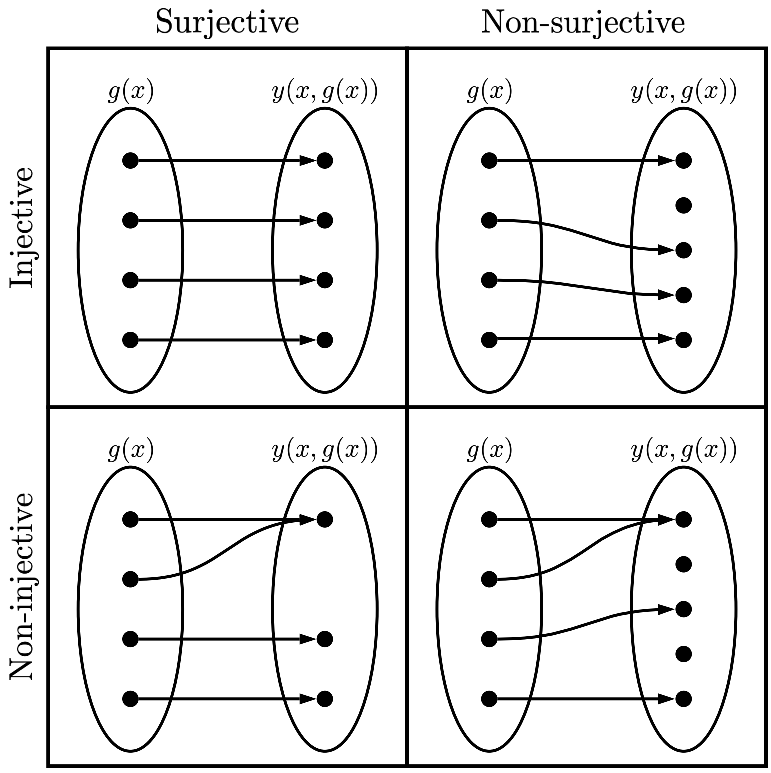

Figure 1. Graphical representation of injective and surjective functionals.

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, $f(x, f(x, g(x)))$. Then, 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: for example, $f(x, f(x, f(x, g(x))) = f(x, f(x, g(x))) = f(x, g(x))$, and so on.

Now that a functional and some properties of a functional have been investigated, the notation used in the prior section can be leveraged to rigorously define TFC related concepts. First, it is useful to define the constraint operator, denoted by the symbol $\mathfrak{C}$.

Definition 6. *The constraint operator, $\mathfrak{C}^i$, is a linear operator that when operating on a function returns the function evaluated at the i-th specified constraint.*

As an example, consider the 2nd linear constraint ($i = 2$) given in Section 2.2, $3 = 2y(2) + \pi y_{xx}(0)$. For this problem, it follows that,

$$\mathfrak{C}^2[y(x)] = 2y(2) + \pi y_{xx}(0).$$

The constraint operator is a linear operator, as it satisfies the two properties of a linear operator: (1) $\mathfrak{C}^i[f(x) + g(x)] = \mathfrak{C}^i[f(x)] + \mathfrak{C}^i[g(x)]$ and (2) $\mathfrak{C}^i[ag(x)] = a\mathfrak{C}^i[g(x)]$. For example, again consider the 2^{nd} linear constraint given in Section 2.2,

$$\begin{aligned}\mathfrak{C}^2[f(x) + g(x)] &= \mathfrak{C}^2[f(x)] + \mathfrak{C}^2[g(x)] = 2f(2) + \pi f_{xx}(0) + 2g(2) + \pi g_{xx}(0)\\ \mathfrak{C}^2[af(x)] &= a\mathfrak{C}^2[f(x)] = a\Big(2f(2) + \pi f_{xx}(0)\Big).\end{aligned}$$

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

Property 3. *The constraint operator acting on the support functions $s_j(x)$ produces the matrix*

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

Again, consider the example from Section 2.2 where the support functions were $s_1(x) = 1$ and $s_2(x) = x$. By applying the constraint operator,

$$\mathbb{S}_{ij} = \mathfrak{C}^i[s_j(x)] = \begin{bmatrix} \mathfrak{C}^1[s_1(x)] & \mathfrak{C}^1[s_2(x)] \\ \mathfrak{C}^2[s_1(x)] & \mathfrak{C}^2[s_2(x)] \end{bmatrix} = \begin{bmatrix} s_1(1) - s_1(0) & s_2(1) - s_2(0) \\ 2s_1(2) + \pi s_{1_{xx}}(0) & 2s_2(2) + \pi s_{2_{xx}}(0) \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 2 & 4 \end{bmatrix},$$

which is identical to the matrix derived in Section 2.2. In fact, the matrix $\mathbb{S}_{ij}$ is simply the matrix multiplying the α_{ij} matrix in all the previous examples. Therefore, 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 are precisely the inverse of the constraint operator operating on the support functions.

Property 4. *The constraint operator acting on the switching functions $\phi_j(x)$ produces the Kronecker delta.*

$$\mathfrak{C}^i[\phi_j(x)] = \delta_{ij}$$

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

Definition 7. *Let $g(x)$ be the free function where $g : \mathbb{R} \to \mathbb{R}$, and let $\kappa_i \in \mathbb{R}$ be the numerical portion of the i^{th} constraint. Then,*

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

Following the example from Section 2.2, the projection functional for the second constraint is,

$$\rho_2(x, g(x)) = \kappa_2 - \mathfrak{C}^2[g(x)] = 3 - 2g(2) - \pi g_{xx}(0).$$

Note that in the univariate case κ_i is a scalar value, but in the multivariate case κ_i is a function.

Theorem 1. *For any function, $f(x)$, satisfying the constraints, there exists at least one free function, $g(x)$, such that the TFC constrained expression $y(x, g(x)) = f(x)$.*

Proof. As highlighted in Properties 1 and 2, 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 $y(x, g(x)) = g(x) + \rho_i(x, g(x))\,\phi_i(x) = g(x) + 0_i\,\phi_i(x) = g(x)$. 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., $y(x, f(x)) = f(x)$. □

Theorem 2. *The TFC univariate constrained expression is a projection functional.*

Proof. To prove Theorem 2, one must show that $y(x, y(x, g(x))) = y(x, g(x))$. By definition, the constrained expression returns a function that satisfies the constraints. In other words, for any $g(x)$, $y(x, g(x))$ is a function that satisfies the constraints. From Theorem 1, 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. □

Theorem 3. *For a given function, $f(x)$, satisfying the constraints, the free function, $g(x)$, in the TFC constrained expression $y(x, g(x)) = f(x)$ is not unique. In other words, the TFC constrained expression is a surjective functional.*

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

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

Substituting the chosen $g(x)$ yields,

$$y(x) = f(x) + \beta_j\, s_j(x) + \phi_i(x)\, \rho_i(x, f(x) + \beta_j\, s_j(x)).$$

Now, according to Definition 7 of the projection functional,

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

Since the constraint operator $\mathfrak{C}^i$ is a linear operator,

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

Since $f(x)$ is defined as a function satisfying the constraints, then $\mathfrak{C}^i[f(x)] = \kappa_i$, and,

$$y(x) = f(x) + \beta_j s_j(x) - \phi_i(x)\mathfrak{C}^i[s_j(x)]\beta_j.$$

Now, according to Property 3 of the constraint operator, and by decomposing the switching functions ϕ_i,

$$y(x) = f(x) + \beta_j\, s_j(x) - \alpha_{ki}\, s_k(x)\mathbb{S}_{ij}\, \beta_j.$$

Collecting terms results in,

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

However, $\mathbb{S}_{ki}\alpha_{ij} = \delta_{kj}$ because α_{ij} is the inverse of $\mathbb{S}_{ki}$. Therefore, by the definition of inverse, $\mathbb{S}_{ki}\alpha_{ij} = \alpha_{ki}\mathbb{S}_{ij} = \delta_{kj}$, and thus,

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

Simplifying yields the result,

$$y(x) = f(x),$$

which is independent of the $\beta_j s_j(x)$ terms in the free function. Therefore, the free function 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, prior research focused on using this method to solve ODEs [2,3] through a basis expansion of $g(x)$ and least-squares, and the basis terms linearly dependent to the support functions had to be omitted from $g(x)$ in order to maintain full rank matrices in the least-squares.

The previous proofs coupled with the functional and functional property definitions given earlier provide a more rigorous definition for the TFC constrained expression: the TFC constrained expression is a surjective, projection functional whose domain is the space of all real-valued functions that are defined at the constraints and whose codomain is the space of all real-valued functions that satisfy the constraints. It is surjective because it spans the space of all functions that satisfy the constraints, its codomain, based on Theorem 1, but is not injective, because Theorem 3 shows that functions in the codomain are the image of more than one function in the domain: the functional is thus not bijective either because it is not injective. Moreover, the TFC constrained expression is a projection functional as shown in Theorem 2.

4. Multivariate TFC

Consider the general multivariate function $F : \mathbb{R}^n \to \mathbb{R}^m$ where $F = (f_1, f_2, \cdots, f_m)$. In this definition, F is composed of the real-values functions $f_i : \mathbb{R}^n \to \mathbb{R}$ such that $f_i = f_i(x_1, x_2, \cdots, x_n)$ where x_i are the independent variables. In terms of the TFC, the functions f_i can be expressed as individual constrained expressions, and therefore the extension to multidimensional functions only involves extending the original method developed in Section 2 for $f(x)$ to $f(x_1, x_2, \cdots, x_n)$. Once completed, the extension to the original definition of F is immediate: simply write a multivariate constrained expression for every f_i in F.

In the following section, the multivariate TFC is developed using a recursive application of the univariate TFC. In this manner, it can be shown that this approach is a generalization of the original theory, and that all mathematical proofs for the univariate constrained expressions can easily be extended to the multivariate constrained expressions. Then, a tensor form of the multivariate constrained expression is introduced by simplifying the recursive method. The tensor formulation provides a succinct way to write multivariate constrained expressions.

4.1. Recursive Application of Univariate TFC

As discussed above, our extension to the multivariate case is concerned with deriving the constrained expression for the form $f = f(x_1, x_2, \cdots, x_n)$. For a set of constraints in the multivariate case, one can first create the constrained expression all constraints on x_1 using the univariate TFC formulation. The resulting univariate constrained expression, which we can denote as, ${}^1f$ is then used as the free function in a constrained expression that includes all the constraints on x_2 to produce the expression ${}^2f$. This method carries on until the final independent variable, x_n, is reached and the expression ${}^nf = f$ and is the multivariate constrained expression.

This concept is best shown through some simple examples. These examples have two spatial dimensions only one dependent variable (i.e., $F : \mathbb{R}^2 \to \mathbb{R}$) we adopt the following notation:

$$\begin{aligned} F = f_1 &:= u \\ (x_1, x_2) &:= (x, y). \end{aligned}$$

4.1.1. Multivariate Example # 1: Value and Derivative Constraints

Take for example the following constraints in two dimensions,

$$u(0, y) = \sin(2\pi y), \quad u_x(0, y) = 0, \quad u(x, 0) = x^2, \quad \text{and} \quad u(x, 1) = \cos(x) - 1.$$

First, the constrained expression is built for the constraints involving x. Using, the univariate TFC, this can be written as,

$${}^1u(x, y, g(x, y)) = g(x, y) + \sin(2\pi y) - g(0, y) - x g_x(0, y).$$

Then, ${}^1u(x,y)$ is used as the free function in the constrained expression involving the constraints on y. Since this problem is two-dimensional, the resultant expression is the multivariate TFC constrained expression.

$$u(x,y,{}^1u(x,y)) = {}^1u(x,y) + (1-y)\Big(x^2 - {}^1u(x,0)\Big) + y\Big(\cos(x) - 1 - {}^1u(x,1)\Big) \tag{4}$$

Substituting ${}^1u$ into Equation (4) and simplifying yields,

$$\begin{aligned} u(x,y,g(x,y)) = g(x,y) + \sin(2\pi y) - g(0,y) - xg_x(0,y) + (1-y)\Big(x^2 - g(x,0) + g(0,0) \\ + xg_x(0,0)\Big) + y\Big(\cos(x) - 1 - g(x,1) + g(0,1) + xg_x(0,1)\Big). \end{aligned} \tag{5}$$

Alternatively, one could first write the expression for the constraints on y,

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

and use ${}^2u(x,y)$ as the free function in a constrained expression for the constraints on x,

$$u(x,y,{}^2u(x,y)) = {}^2u(x,y) + \sin(2\pi y) - {}^2u(0,y) - x\frac{\partial({}^2u)}{\partial x}(0,y). \tag{6}$$

Substituting ${}^2u(x,y)$ into Equation (6) and simplifying yields,

$$\begin{aligned} u(x,y,g(x,y)) = g(x,y) + \sin(2\pi y) - g(0,y) - xg_x(0,y) + (1-y)\Big(x^2 - g(x,0) + g(0,0) \\ + xg_x(0,0)\Big) + y\Big(\cos(x) - 1 - g(x,1) + g(0,1) + xg_x(0,1)\Big), \end{aligned}$$

the exact same result as in Equation (5). Therefore, it does not matter in what order recursive univariate TFC is applied to produce multivariate constrained expressions, as the final result will be the same.

Figure 2 shows the constrained expression evaluated with the free function $g(x,y) = x^2 cos(y) + 4$. The constraints that can be visualized easily are shown as black lines. As expected, the TFC constrained expression satisfies these constraints exactly.

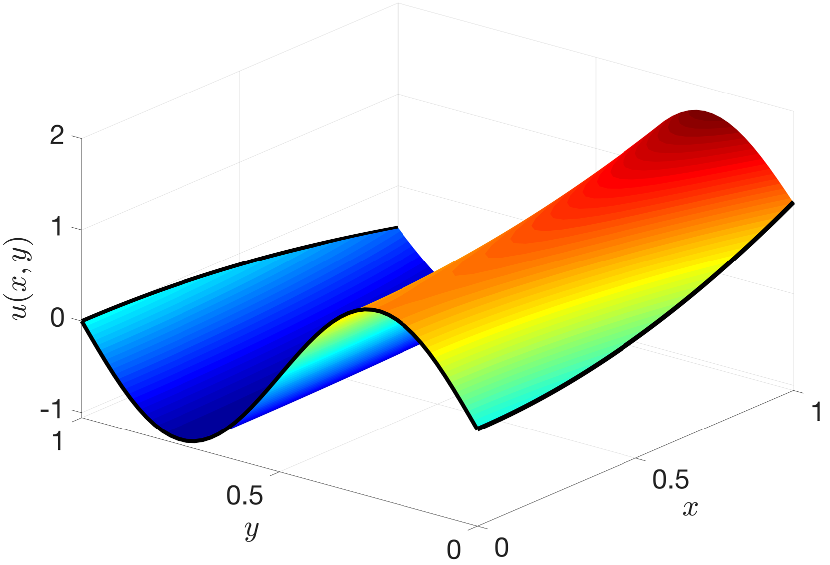

Figure 2. Multivariate example # 1 constrained expression evaluated using $g(x,y) = x^2 cos(y) + 4$.

4.1.2. Multivariate Example # 2: Linear Constraints

Take for example the following constraints in two dimensions,

$$u(0,y) = y^2\sin(\pi y), \quad u(1,y)+u(2,y) = y\sin(\pi y), \quad u_y(x,0) = 0, \quad \text{and} \quad u(x,0) = u(x,1).$$

As in the first example, the univariate constrained expression is built for the constraints in x,

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

Then, ${}^1u(x,y)$ is used as the free function in the constrained expression for the constraints in y,

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

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

$$\begin{aligned} u(x,y,{}^1u(x,y)) = g(x,y) &+ \Big(y-y^2\Big)\left(\frac{3-2x}{3}g_y(0,0) - \frac{x}{3}\left(-g_y(1,0) - g_y(2,0)\right) - g_y(x,0)\right) \\ &- y^2\Big(\frac{3-2x}{3}g(0,0) - \frac{3-2x}{3}g(0,1) - \frac{x}{3}(-g(1,0) - g(2,0)) + \frac{x}{3}(-g(1,1) \\ &- g(2,1)) - g(x,0) + g(x,1)\Big) + \frac{3-2x}{3}\Big(y^2\sin(\pi y) - g(0,y)\Big) \\ &+ \frac{x}{3}\Big(-g(1,y) - g(2,y) + y\sin(\pi y)\Big). \end{aligned} \tag{7}$$

Just as in the previous example, one could first write the constrained expression for the constraints in y, call it ${}^2u(x,y)$, and then use ${}^2u(x,y)$ as the free function in the constrained expression for the constraints in x: the result, after simplifying, would be identical to Equation (7). Figure 3 shows the constrained expression for the specific $g(x,y) = x^2\cos y + \sin(2x)$, 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.

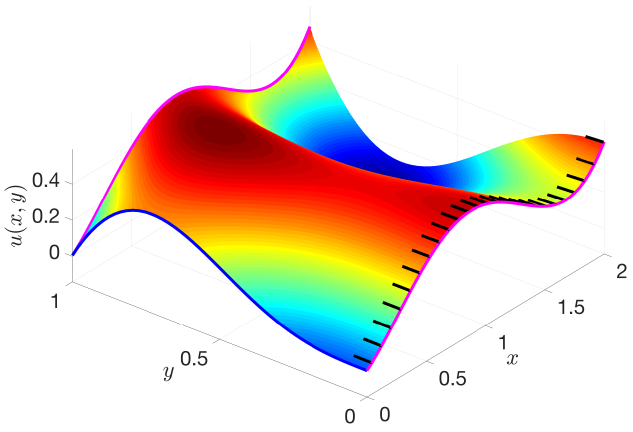

Figure 3. Multivariate example # 2 constrained expression evaluated using $g(x,y) = x^2\cos y + \sin(2x)$. The blue line signifies the constraint on $u(0,y)$, the black lines signify the derivative constraint on $u_y(x,0)$ and the magenta lines signify the relative constraint $u(x,0) = u(x,1)$. The linear constraint $u(1,y)+u(2,y) = y\sin(\pi y)$ is not easily visualized, but is nonetheless satisfied by the constrained expression.

4.1.3. Multivariate Constrained Expression Theorems

Theorem 4. *For any function, $f(\boldsymbol{x})$, satisfying the constraints, there exists at least one free function, $g(\boldsymbol{x})$, such that the multivariate TFC constrained expression $u(\boldsymbol{x}, g(\boldsymbol{x})) = f(\boldsymbol{x})$.*

Proof. Based on Theorem 1, the univariate constrained expression will return the free function if the free function satisfies the constraints. Let ${}^1u(\boldsymbol{x})$ represent the univariate constrained expression for the independent variable x_1 that uses the free function $g(\boldsymbol{x})$, ${}^2u(\boldsymbol{x})$ represent the univariate constrained expression for the independent variable x_2 that uses the free function ${}^1u(\boldsymbol{x})$, and so on up to ${}^nu(\boldsymbol{x})$ which is simply the total constrained expression $u(\boldsymbol{x})$. If we choose $g(\boldsymbol{x}) = f(\boldsymbol{x})$, then based on Theorem 1 ${}^1u(\boldsymbol{x}) = f(\boldsymbol{x})$. Applying Theorem 1 recursively leads to ${}^2u(\boldsymbol{x}) = f(\boldsymbol{x})$ and so on until ${}^nu(\boldsymbol{x}) = u(\boldsymbol{x}) = f(\boldsymbol{x})$. 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., $u(\boldsymbol{x}, f(\boldsymbol{x})) = f(\boldsymbol{x})$. □

Theorem 5. *The TFC multivariate constrained expression is a projection functional.*

Proof. To prove Theorem 5, one must show that $u(\boldsymbol{x}, u(\boldsymbol{x}, g(\boldsymbol{x}))) = u(\boldsymbol{x}, g(\boldsymbol{x}))$. By definition, the 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 4, 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. □

Theorem 6. *For a given function, $f(\boldsymbol{x})$, satisfying the constraints, the free function, $g(\boldsymbol{x})$, in the TFC constrained expression $u(\boldsymbol{x}, g(\boldsymbol{x})) = f(\boldsymbol{x})$ is not unique. In other words, the multivariate TFC constrained expression is a surjective functional.*

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

Just 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, 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 only one support function from each independent variable, e.g., $s_i(x_1)s_j(x_2)...s_k(x_n)$.

In addition, just as in the univariate case, Theorems 4, 5, and 6 allow for a more rigorous definition of the multivariate TFC constrained expression. The multivariate TFC constrained expression is a surjective, projection functional whose domain is the space of all real-valued functions that are defined at the constraints and whose codomain is the space of all real-valued functions that satisfy the constraints.

4.2. Tensor Form

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

$$u(\boldsymbol{x}) = g(\boldsymbol{x}) + \mathcal{M}(\rho(\boldsymbol{x}, g(\boldsymbol{x}))_{i_1 i_2 \ldots i_n} \Phi_{i_1}(x_1)\Phi_{i_2}(x_2)\ldots\Phi_{i_n}(x_n) \qquad (8)$$

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 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,
$$\mathcal{M}_{1\ldots i_k \ldots 1} = \left\{0, \quad {}^k\rho_1, \quad \cdots, \quad {}^k\rho_{\ell_k}\right\},$$
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,
$$\mathcal{M}_{i_1 i_2 \ldots i_n} = {}^j\mathfrak{C}^{i_j - 1}\left[{}^k\mathfrak{C}^{i_k - 1}\left[\cdots\left[{}^h\rho_{i_h - 1}\right]\cdots\right]\right](-1)^m, \tag{9}$$
where i_j, i_k, $\ldots$, i_h are the indices of $\mathcal{M}_{i_1 i_2 \ldots i_n}$ that are not equal to one and m is equal to the number of non-one indices. If the constraint functions and free function are differential up to the order of derivatives required to compute Equation (9), then by multiple applications of Clairaut's Theorem the constraint operators can be freely permuted [9]. For example, Equation (9) could be re-written as,
$$\mathcal{M}_{i_1 i_2 \ldots i_n} = {}^h\mathfrak{C}^{i_h - 1}\left[{}^j\mathfrak{C}^{i_j - 1}\left[\cdots\left[{}^k\rho_{i_k - 1}\right]\cdots\right]\right](-1)^m.$$

The elements of the vectors Φ_{i_k} are simply 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, the constrained expressions for the two multivariate examples are re-derived using the tensor form.

4.2.1. Multivariate Example # 1: Value and Derivative Constraints Using the Tensor Form

The constraints from the first multivariate example are copied below for the reader's convenience.
$$u(0,y) = \sin(2\pi y), \quad u_x(0,y) = 0, \quad u(x,0) = x^2, \quad \text{and} \quad u(x,1) = \cos(x) - 1$$

The projection functionals are defined based on the constraints,
$$\begin{gathered} {}^1\rho_1(x,y,g(x,y)) = \sin(2\pi y) - g(0,y), \quad {}^1\rho_2(x,y,g(x,y)) = -g_x(0,y), \\ {}^2\rho_1(x,y,g(x,y)) = x^2 - g(x,0), \quad \text{and} \quad {}^2\rho_2(x,y,g(x,y)) = \cos(x) - 1 - g(x,1). \end{gathered}$$

Then, the first step in constructing the $\mathcal{M}$ tensor can be completed.
$$\mathcal{M}_{ij}(x,y,g(x,y)) = \begin{bmatrix} 0 & x^2 - g(x,0) & \cos(x) - 1 - g(x,1) \\ \sin(2\pi y) - g(0,y) & \text{-} & \text{-} \\ \cos(x) - 1 - g(x,1) & \text{-} & \text{-} \end{bmatrix}$$

The remaining elements of the $\mathcal{M}$ tensor are found in step 2 by calculating the geometric intersection of the projection functionals. For example,

$$\begin{aligned}\mathcal{M}_{22} &= {}^{1}\mathfrak{C}^{1}\left[{}^{2}\rho_1\right](-1)^2 = -\left[x^2 - g(x,0)\right]\Big|_{x=0} = g(0,0)\\ &= {}^{2}\mathfrak{C}^{1}\left[{}^{1}\rho_1\right](-1)^2 = -\left[\sin(2\pi y) - g(0,y)\right]\Big|_{y=0} = g(0,0),\end{aligned}$$

where functional arguments have been dropped for clarity. The remaining elements are computed in a similar fashion to produce,

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

The Φ vectors are assembled by concatenating a 1 with the switching functions for that independent variable. Hence,

$$\begin{aligned}\Phi_i(x) &= \begin{Bmatrix}1 & 1 & x\end{Bmatrix}\\ \Phi_j(y) &= \begin{Bmatrix}1 & 1-y & y\end{Bmatrix}.\end{aligned}$$

Now, the tensor form of the constrained expression can be compactly written as,

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

Note that expanding this expression produces the exact same constrained expression as the recursive method.

4.2.2. Multivariate Example # 2: Linear Constraints Using the Tensor Form

The constraints from the second multivariate example are copied below for the reader's convenience.

$$u(0,y) = y^2\sin(\pi y), \quad u(1,y) + u(2,y) = y\sin(\pi y), \quad u_y(x,0) = 0, \quad \text{and} \quad u(x,0) = u(x,1)$$

The projection functionals are defined based on the constraints,

$$\begin{aligned}{}^{1}\rho_1(x,y,g(x,y)) &= y^2\sin(\pi y) - g(0,y), \quad {}^{1}\rho_2(x,y,g(x,y)) = y\sin(\pi y) - g(1,y) - g(2,y),\\ {}^{2}\rho_1(x,y,g(x,y)) &= -g_y(x,0), \quad \text{and} \quad {}^{2}\rho_2(x,y,g(x,y)) = g(x,1) - g(x,0),\end{aligned}$$

and the first step in constructing the $\mathcal{M}$ tensor is complete,

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

Then, just as in the previous example, remaining elements of the $\mathcal{M}$ tensor are found by calculating the geometric intersection of the projection functionals. For example,

$$\begin{aligned}\mathcal{M}_{33} = {}^{1}\mathfrak{C}^{2}\left[{}^{2}\rho_{2}\right](-1)^{2} &= -\Big[g(x,1)-g(x,0)\Big]\Big|_{x=1} - \Big[g(x,1)-g(x,0)\Big]\Big|_{x=2}\\ &= g(1,0)-g(1,1)+g(2,0)-g(2,1)\\ = {}^{2}\mathfrak{C}^{2}\left[{}^{1}\rho_{2}\right](-1)^{2} &= \Big[y\sin(\pi y)-g(1,y)-g(2,y)\Big]\Big|_{y=1} - \Big[y\sin(\pi y)-g(1,y)-g(2,y)\Big]\Big|_{y=0}\\ &= -g(1,1)-g(2,1)+g(1,0)+g(2,0),\end{aligned}$$

where functional arguments have been dropped for clarity. The complete $\mathcal{M}$ tensor for this example is,

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

The Φ vectors are again assembled by concatenating a 1 with the switching functions for the associated independent variable.

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

Now, the tensor form of the constrained expression can be compactly written as,

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

Note that expanding this expression produces the exact same constrained expression as the recursive method.

5. Applications to PDEs

In this article, orthogonal bases in n-dimensions, namely Chebyshev orthogonal polynomials of the first kind and Legendre orthogonal polynomials, are leveraged to approximate the solutions of PDEs with the TFC. For completeness, the equations to compute these polynomials are provided in Appendix A.

In general, multivariate basis sets can be created by using all possible products of the functions in the univariate basis sets. The measure that makes up the new multivariate basis set will be the product of measures of the univariate basis sets, and the domain of the multivariate basis set will be the union of the domains that make up the univariate basis sets. More details and insights regarding two-dimensional and n-dimensional orthogonal basis functions can be found in Refs. [14–16].

In this article, the free function will be defined as a linear combination of some multivariate basis set with unknown coefficients. The resultant constrained expression and its derivatives are substituted into the PDE. Since the free function consists of known basis functions and unknown coefficients, the PDE is transformed into an algebraic equation. This algebraic equation is discretized over the problem domain, and the unknown coefficients are used to minimize the residual of the PDE over the set of discretized points. The following subsections provide a detailed explanation of each major step, and a summary of the entire process is given in Figure 4.

5.1. Defining the Free Function $g(\boldsymbol{x})$

Let us define n independent variables in the vector $\boldsymbol{x} = \{x_1, \cdots, x_k, \cdots, x_n\}^{\mathrm{T}}$. Moreover, let the orthogonal basis set for each of these independent variables be denoted by B_k^m, where the superscript m denotes the m-th basis function and the subscript k denotes the k-th independent variable. For example, the third basis function for x_2 would be denoted as B_2^3. The domain of the multivariate basis will be denoted by $\Omega = \Omega_1 \times \Omega_2 \times \cdots \times \Omega_n$, where the generic Ω_k denotes the domain of the k-th basis set. Then, an arbitrary basis function for the multivariate domain can be written as,

$$\mathcal{B} = B_1^{m_1} B_2^{m_2} \cdots B_n^{m_n}, \tag{10}$$

where $m_1, \cdots, m_n \in \mathbb{Z}^+$. In other words, Equation (10) generates a multivariate basis via a tensor product of univariate basis functions [17]. If one were to use all possible products of the functions in the individual basis sets, i.e., use all possible combinations of $m_1, \cdots, m_n \in \mathbb{Z}^+$, an infinite set, then the resulting multivariate basis would span the union of the individual univariate basis sets' function spaces. However, when creating this expansion, one must pay attention to the results of Theorem 6. Theorem 6 shows that the functions used to constrained the expression must be omitted from the formulation of $\mathcal{B}$. This ensures a full rank system in the later optimization steps.

As previously stated, the free function is chosen to be a linear combination of this multivariate basis with unknown coefficients. Mathematically, this can be expressed as,

$$g(\boldsymbol{x}) = \boldsymbol{h}^{\mathrm{T}} \boldsymbol{\xi}, \tag{11}$$

where $\boldsymbol{h} \in \mathbb{R}^{\left(\sum_{k=1}^{n} m_k\right)}$ whose elements are elements of $\mathcal{B}$, and $\boldsymbol{\xi}$ is a same-sized vector of the unknown coefficients.

5.2. Derivatives of the Free Function

In most applications, the domains Ω_k of the basis sets do not coincide with the domain of the problem (e.g., for Chebyshev and Legendre polynomials the functions are defined on $[-1, +1]$). Let the basis functions 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 basis functions, a map between the basis function domain and problem domain must be created. The simplest map is a linear one,

$$z = z_0 + \frac{z_f - z_0}{x_{k_f} - x_{k_0}}(x - x_{k_0}) \quad \longleftrightarrow \quad x_k = x_{k_0} + \frac{x_{k_f} - x_{k_0}}{z_f - z_0}(z - z_0). \tag{12}$$

The subsequent derivatives of the free function can be computed,

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

but by defining,

$$c_k := \frac{\mathrm{d}z}{\mathrm{d}x_k} = \frac{z_f - z_0}{x_{k_f} - x_{k_0}}, \tag{13}$$

the derivative computations can be simply written as,

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

The immediate result is if the derivative of the function $g(\boldsymbol{x})$ is taken with respect to the x_k variable, then along with taking the derives of the basis functions with respect to this coordinate, the product must also be multiplied by the c_k mapping coefficient. From this, 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.

5.3. Discretization

In order to solve problems numerically, the problem domain (and therefore the basis function domain) must be discretized. Since this article uses Chebyshev and Legendre orthogonal polynomials, an optimal discretization scheme is the Chebyshev-Gauss-Lobatto nodes [18,19]. For, $N+1$ points, the discrete points are calculated as,

$$z_j = -\cos\left(\frac{j\pi}{N}\right) \quad \text{for} \quad j = 0, 1, 2, \cdots, N. \tag{14}$$

When compared with the uniform distribution, the collocation point distribution results in a much slower increase of the condition number of the matrix to be inverted in the least-squares as the number of basis functions, m, increases. The collocation points can be realized in the problem domain through the relationship provided in Equation (12).

5.4. Summary of the Major Steps to Solving PDEs

To summarize, consider a PDE for the function $u(\boldsymbol{x})$ such that

$$F\left(\boldsymbol{x}; \frac{\partial u}{\partial x_1}, \dots \frac{\partial u}{\partial x_n}; \frac{\partial^2 u}{\partial x_1 \partial x_1}, \dots \frac{\partial^2 u}{\partial x_1 \partial x_n}; \dots\right) = 0, \tag{15}$$

subject to k constraints. In general, the TFC approach to solving differential equations can be broken down into four major steps: (1) derive the constrained expression, (2) define the free function, (3) discretize the domain, and (4) minimize the residual of the differential equation. The flow chart in Figure 4 outlines these steps with all of the relevant equations.

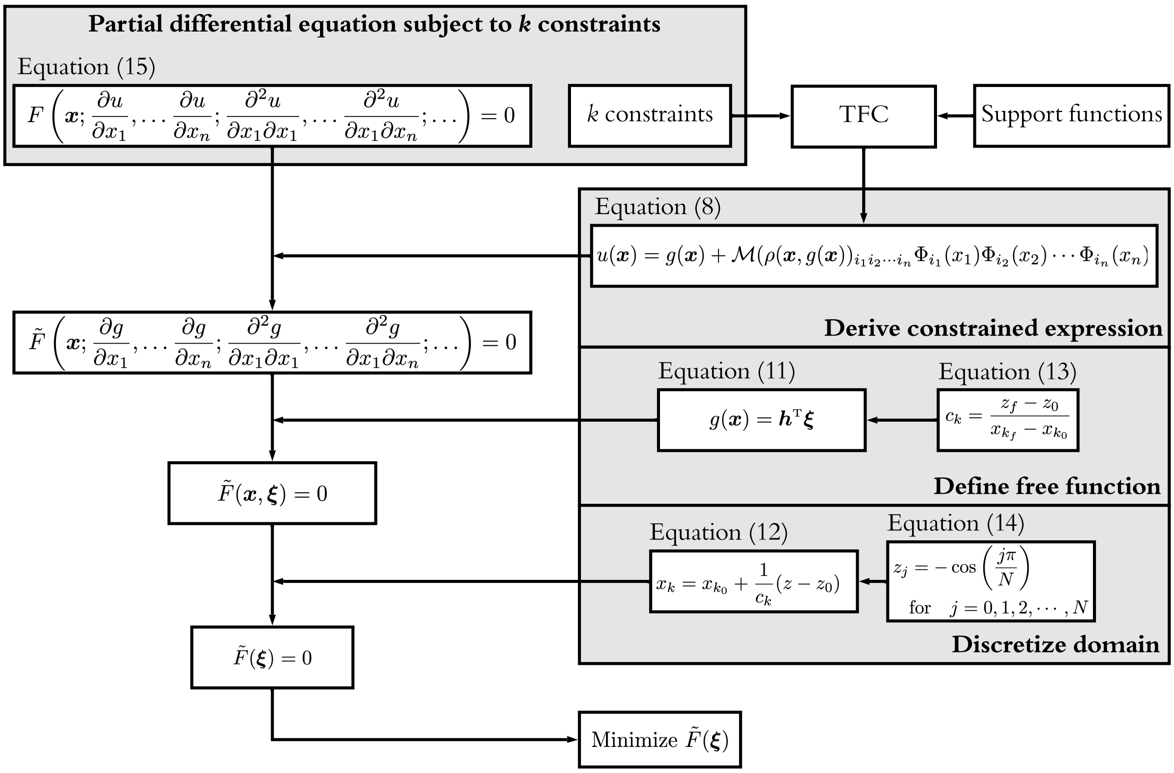
Partial differential equation subject to k constraints
Equation (15)
$F\left(\boldsymbol{x}; \frac{\partial u}{\partial x_1}, \dots \frac{\partial u}{\partial x_n}; \frac{\partial^2 u}{\partial x_1 \partial x_1}, \dots \frac{\partial^2 u}{\partial x_1 \partial x_n}; \dots\right) = 0$
k constraints
TFC
Support functions
Equation (8)
$u(\boldsymbol{x}) = g(\boldsymbol{x}) + \mathcal{M}(\rho(\boldsymbol{x}, g(\boldsymbol{x}))_{i_1 i_2 \cdots i_n} \Phi_{i_1}(x_1)\Phi_{i_2}(x_2)\cdots\Phi_{i_n}(x_n)$
Derive constrained expression
$\tilde{F}\left(\boldsymbol{x}; \frac{\partial g}{\partial x_1}, \dots \frac{\partial g}{\partial x_n}; \frac{\partial^2 g}{\partial x_1 \partial x_1}, \dots \frac{\partial^2 g}{\partial x_1 \partial x_n}; \dots\right) = 0$
Equation (11)
$g(\boldsymbol{x}) = \boldsymbol{h}^{\mathrm{T}}\boldsymbol{\xi}$
Equation (13)
$c_k = \frac{z_f - z_0}{x_{k_f} - x_{k_0}}$
Define free function
$\tilde{F}(\boldsymbol{x}, \boldsymbol{\xi}) = 0$
Equation (12)
$x_k = x_{k_0} + \frac{1}{c_k}(z - z_0)$
Equation (14)
$z_j = -\cos\left(\frac{j\pi}{N}\right)$
for $j = 0, 1, 2, \cdots, N$
Discretize domain
$\tilde{F}(\boldsymbol{\xi}) = 0$
Minimize $\tilde{F}(\boldsymbol{\xi})$

Figure 4. General flowchart of the TFC approach to solving partial differential equations.

In order to approximate the solution of Equation (15), the constrained expression must be created: this is done using the theory developed in earlier sections. This constrained expression embeds the constraints of the differential equation, and by substituting the constrained expression into

Equation (15), transforms the original differential equation into an algebraic equation of the free function subject to no constraints. This transformed expression is denoted by the ($\sim$) symbol. Next, by defining the free function $g(x)$ according to Equation (11), the differential equation becomes an algebraic equation in the unknown coefficient vector $\boldsymbol{\xi}$. Lastly, this equation is discretized according to Equation (14), leading to a linear system of $\boldsymbol{\xi}$ if the PDE is linear and a nonlinear system of $\boldsymbol{\xi}$ if the PDE is nonlinear, which can be solved by many available optimization techniques. Previous work [2,3] as well as the results provided in the following section utilize a least-squares approach.

6. Results

This section applies the method to two PDEs. For each problem, the PDE and associated constraints are summarized along with the equations needed to construct the constrained expression. All numerical results were performed in Python, and utilized the autograd package [20] to perform derivatives via automatic differentiation [21]. All computations were performed on a MacBook Pro (2016) macOS Version 10.15.3 with a 3.3 GHz Dual-Core Intel® Core™ i7 and with 16 GB of RAM. All run times were calculated using the default_timer function in the Python `timeit` package. In all cases, matrix inversion was handled with NumPy's `pinv` function.

Consequently, two specific computation times are provided (and tabulated in Appendix B), (1) the full run-time of the problem and (2) the computational time associated with the least-squares. In these tables, the full run-time is drastically affected by the computational overhead from autograd, with full run times on the order of 0.5–50 seconds while the computation time for the least squares and nonlinear least squares is on the order of 0.5–185 milliseconds.

For the results in the following sections, the accuracy of the method was determined according to the following process:

1. Training
   (a) Estimate the solution of the PDE (i.e., determine the coefficients $\boldsymbol{\xi}$) using the TFC method with n points per independent variable and basis functions up to the to the m-th degree.
   (b) Maximum training error: Using the training set discretization, determine the absolute error of the estimated solution compared with the true solution and record the maximum value.
2. Test
   (a) Using converged $\boldsymbol{\xi}$ parameters from the training phase, refine the discretization of domain with $n = 100$ equally spaced points per dimension and evaluate estimated solution at these points.
   (b) Maximum test error: Using the test set discretization, determine the absolute error of the estimated solution compared with the true solution and record the maximum value.

Additionally, for both numerical tests, the method was completed over a varying range of discretization points per independent variable, n, and degree of basis expansion, m. The results in this section and in Appendix B are reported as a function of these two parameters. For example, a value of $n = 5$ would imply a 5 × 5 grid or 25 points. Likewise, a value of $m = 5$ would imply that all the univariate basis functions, and combinations thereof, are at most quintic functions. However, the number of coefficients to be solved, and therefore the size of the matrix to be inverted, is dependent on the constrained expression, since some terms need to be removed from the expression of $g(x, y)$ (see Theorem 6). The total number of basis functions associated with each degree for both Problems #1 and #2 are displayed in Table 1.

Table 1. Equivalence of number of basis function compared to degree of basis expansion for both Problems #1 and #2.

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

6.1. Problem # 1

Consider the PDE solved in Lagaris et al. [22], Mall & Chakraverty [23], Sun et al. [24], and Schiassi et al. [12],

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

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 $u(x,y) = e^{-x}(x+y^3)$. Using the proposed method, the constrained expression can be derived and written in its tensor form as,

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

where $g(x,y)$ is defined according to Equation (11) and for these numerical tests is implemented with either Chebyshev or Legendre polynomials. Furthermore, for this problem,

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

and

$$\begin{aligned} \Phi_i(x) &= \Big\{1, \quad 1-x, \quad x\Big\}, \\ \Phi_j(y) &= \Big\{1, \quad 1-y, \quad y\Big\}. \end{aligned}$$

It follows that the expanded constrained expression is,

$$\begin{aligned} u(x,y) = g(x,y) &- (x-1)\left(y(-g(0,0) + g(0,1) - 1) + g(0,0) + y^3\right) + (x-1)g(0,y) \\ &+ x(yg(1,1) - (y-1)g(1,0)) - xg(1,y) + (y-1)g(x,0) - yg(x,1) + \frac{xy\left(y^2-1\right)}{e} \\ &+ e^{-x}(x+y). \end{aligned}$$

Figure 5 shows the analytical solution for this problem and Tables 2 and 3 display the maximum error over the domain for the test set using both Chebyshev and Legendre polynomials.

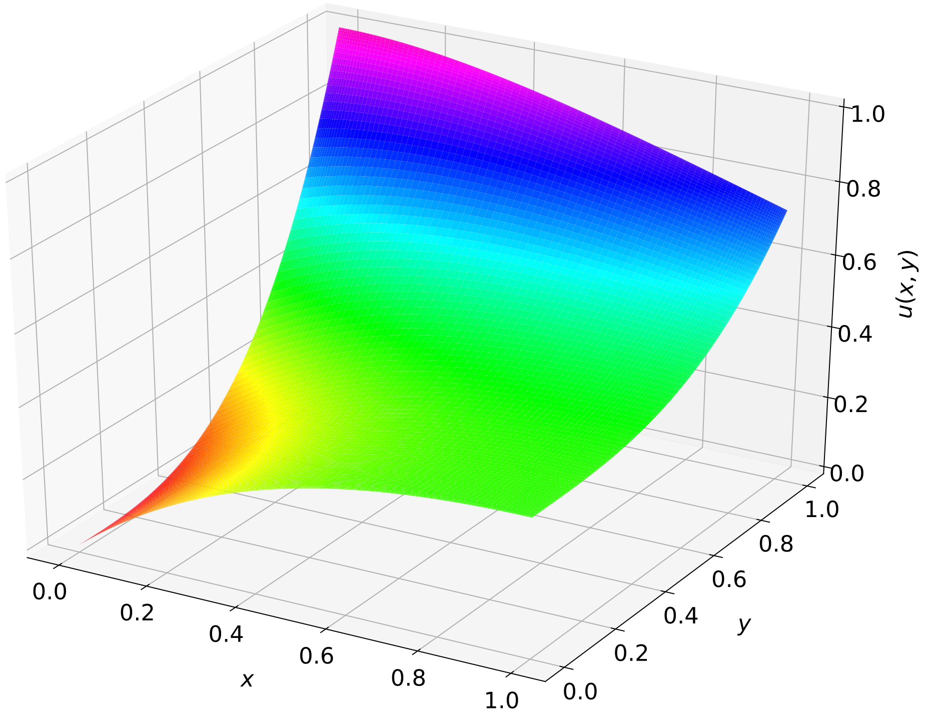

Figure 5. Problem #1 analytical solution.

It can be seen that the difference in accuracy of Chebyshev polynomials versus Legendre polynomials is negligible: the solutions that use Legendre polynomials maintain a slightly more accurate estimation than those that use Chebyshev polynomials when using higher order terms.

Table 2. Maximum test set solution error using Chebyshev polynomials for Problem # 1.

m **n**	**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}	3.2210^{-15}

Table 3. Maximum test set solution error using Legendre polynomials for Problem # 1.

m **n**	**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}	5.55×10^{-16}	4.44×10^{-16}	-
25	5.13×10^{-4}	1.15×10^{-10}	4.44×10^{-16}	4.44×10^{-16}	5.55×10^{-16}
30	5.09×10^{-4}	1.14×10^{-10}	4.44×10^{-16}	4.44×10^{-16}	5.55×10^{-16}

Finally, the results of the numerical test for Problem #1 are compared to the other approaches in Table 4, where the maximum training and test errors are presented. It can be seen that this method produces an estimate at machine level precision that is at least 3 orders of magnitudes more accurate than the other methods. In fact, the next closest method is a TFC based approach where the free function is expressed using an Extreme Learning Machine [25] in the paper by Schiassi et al. [12].

Table 4. Comparison of maximum training and test error of TFC with current state-of-the-art techniques for Problem # 1.

Method	**Training Set** **Maximum Error**	Test Set **Maximum Error**
TFC	2.22×10^{-16}	4.44×10^{-16}
X-TFC [12]	3.8×10^{-13}	5.1×10^{-13}
FEM	2×10^{-8}	1.5×10^{-5}
Reference [22]	5×10^{-7}	5×10^{-7}
Reference [23]	-	3.2×10^{-2}
Reference [24]	-	2.4×10^{-4}

6.2. Problem #2

Problem #2 is a PDE with a linear constraint created by the authors.

$$u_{xx}(x,y) + u_x(x,y)u_y(x,y) = 2\cos(y) - 2x^3\sin(y)\cos(y)$$

where $(x,y) \in [0,1] \times [0,2\pi]$ and subject to,

$$\begin{aligned} u(0,y) &= 0 \\ u(1,y) &= \cos(y) \\ u(x,0) &= u(x,1), \end{aligned}$$

which has the true solution $u(x,y) = x^2\cos(y)$.

Using the TFC, the constrained expression for these boundary conditions can be written in the tensor form as,

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

where

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

and

$$\begin{aligned} \Phi_i(x) &= \begin{Bmatrix} 1 & 1-x & x \end{Bmatrix}^{\mathrm{T}}, \\ \Phi_j(y) &= \begin{Bmatrix} 1 & \frac{y}{2\pi} \end{Bmatrix}^{\mathrm{T}}, \end{aligned}$$

or in its expanded form as,

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

Figure 6 shows the analytical solution of Problem #2, and Tables 5 and 6 show the maximum test set error over the domain using Chebyshev and Legendre polynomials respectively.

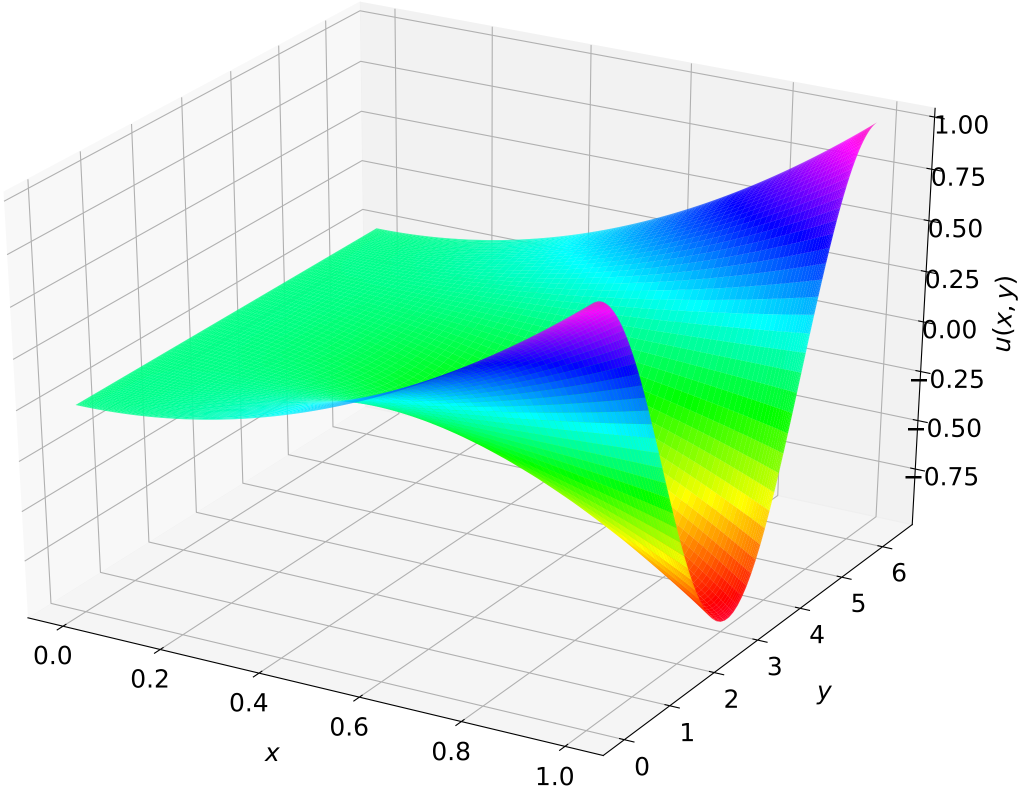

Figure 6. Problem #2 analytical solution.

Table 5. Maximum test set solution error using Chebyshev polynomials for Problem # 2.

n \ m	5	10	15	20	25
5	1.03×10^{-1}	-	-	-	-
10	9.20×10^{-2}	2.49×10^{-5}	-	-	-
15	9.03×10^{-2}	1.54×10^{-5}	4.34×10^{-9}	-	-
20	8.94×10^{-2}	1.52×10^{-5}	4.56×10^{-9}	5.33×10^{-15}	-
25	8.88×10^{-2}	1.50×10^{-5}	4.53×10^{-9}	2.72×10^{-15}	4.44×10^{-16}
30	8.85×10^{-2}	1.49×10^{-5}	4.51×10^{-9}	2.72×10^{-15}	3.33×10^{-16}

Table 6. Maximum test set solution error using Legendre polynomials for Problem # 2.

n \ m	5	10	15	20	25
5	1.03×10^{-1}	-	-	-	-
10	9.20×10^{-2}	2.49×10^{-5}	-	-	-
15	9.03×10^{-2}	1.54×10^{-5}	4.34×10^{-9}	-	-
20	8.94×10^{-2}	1.52×10^{-5}	4.56×10^{-9}	5.33×10^{-15}	-
25	8.88×10^{-2}	1.50×10^{-5}	4.53×10^{-9}	2.78×10^{-15}	5.55×10^{-16}
30	8.85×10^{-2}	1.49×10^{-5}	4.51×10^{-9}	2.73×10^{-15}	5.55×10^{-16}

Tables 5 and 6 show that the difference in error between Chebyshev and Legendre polynomials for Problem #2 is small: the Chebyshev polynomials perform slightly better than the Legendre polynomials as the number of points in the domain and number of basis functions increase.

7. Conclusions

This article illustrated that the structure of the univariate TFC constrained expression is composed of a free function and constraint terms, which contain products of projection functionals and switching functions. A method to calculate the projection functionals and switching functions was demonstrated, and their properties were defined. Then, these properties were used as the foundation for mathematical proofs related to the univariate constrained expression.

In addition, the projection/switching perspective of the univariate constrained expression lead directly to a multivariate extension via recursive application of the univariate theory. Since these multivariate constrained expressions where built from univariate constrained expressions, it was fairly simple to extend the mathematical proofs to the multivariate case as well. In the end, it was concluded that the univariate and multivariate TFC constrained expressions are surjective, projection functionals whose domain is the space of all real-valued functions that are defined at the embedded constraints and whose codomain is the space of all real-valued functions that satisfy said constraints. Additionally, a method for compactly writing multivariate constrained expressions via tensors was provided.

After introducing the multivariate TFC in this way, a methodology for solving PDEs via the TFC was presented. This methodology included choosing the free function to be a linear combination of multivariate orthogonal polynomials, discretizing the domain via collocation points, and finally minimizing the residual of the PDE via least-squares. Two example PDEs solved using this methodology were presented, and when available, the TFC solution accuracy was compared with other state-of-the-art methods.

While this article focused on using orthogonal basis functions, namely, Chebyshev and Legendre orthogonal polynomials, as the free function $g(\boldsymbol{x})$, and ultimately a linear/nonlinear least-square technique to find the unknown parameters $\boldsymbol{\xi}$, the technique is not limited to this. At its heart, the TFC approach is a way to derive functionals which analytically satisfy the specified constraints. In other words, when solving ODEs or PDEs, these functionals transform a constrained optimization problem into an unconstrained optimization problem, and therefore a myriad valid definitions of $g(\boldsymbol{x})$ and optimization schemes exist. For example, deep neural networks, support vector machines, and extreme learning machines have been used as free functions in the past. These other free function choices and associated optimization schemes are useful, as for sufficiently complex problems, the number of multivariate basis functions needed to estimate a PDE with sufficient accuracy can become computationally prohibitive.

As defined in this article, the TFC multivariate constrained expressions are capable of embedding value constraints, derivative constraints, and linear combinations thereof. However, other constraint types such as integral, component, and inequality constraints were not discussed. Future work will focus on incorporating these other constraint types. In addition, a more in-depth comparison between TFC and other state-of-the-art methods on a variety of PDEs is likely forthcoming.

Author Contributions: Conceptualization, C.L. and H.J.; Formal analysis, C.L. and H.J.; Methodology, C.L. and H.J.; Software, C.L. and H.J.; Supervision, D.M.; Validation, C.L. and H.J.; Writing—original draft, C.L. and H.J.; Writing—review & editing, C.L., H.J. and D.M. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by a NASA Space Technology Research Fellowship, Leake [NSTRF 2019] Grant #: 80NSSC19K1152 and Johnston [NSTRF 2019] Grant #: 80NSSC19K1149.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

BVP	Boundary Value Problem
FEM	Finite Element Method
ODE	Ordinary Differential Equation
PDE	Partial Differential Equation
TFC	Theory of Functional Connections
X-TFC	Extreme Theory of Functional Connections

Appendix A. Orthogonal Polynomials

Appendix A.1. 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 subsection 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 $\mathrm{d}\mu(z) = \dfrac{1}{\sqrt{1-z^2}}\,\mathrm{d}z$. These polynomials can be generated using the following useful recursive function,

$$T_{k+1} = 2\,z\,T_k - T_{k-1} \qquad \text{starting from: } \begin{cases} T_0 = 1, \\ T_1 = z. \end{cases}$$

Also, all the derivatives of Chebyshev orthogonal polynomials can be computed in a recursive way, starting from

$$\frac{\mathrm{d}T_0}{\mathrm{d}z} = 0, \quad \frac{\mathrm{d}T_1}{\mathrm{d}z} = 1 \qquad \text{and} \qquad \frac{\mathrm{d}^d T_0}{\mathrm{d}z^d} = \frac{\mathrm{d}^d T_1}{\mathrm{d}z^d} = 0 \quad (\forall\, d > 1),$$

while the subsequent derivatives of $T_{k+1}(z)$ can be derived directly from the recursive definition. For $k \geq 1$ they are,

$$\begin{array}{ccc}
\dfrac{\mathrm{d}T_{k+1}}{\mathrm{d}z} & = & 2\left(T_k + z\,\dfrac{\mathrm{d}T_k}{\mathrm{d}z}\right) & -\dfrac{\mathrm{d}T_{k-1}}{\mathrm{d}z} \\
\dfrac{\mathrm{d}^2T_{k+1}}{\mathrm{d}z^2} & = & 2\left(2\,\dfrac{\mathrm{d}T_k}{\mathrm{d}z} + z\,\dfrac{\mathrm{d}^2T_k}{\mathrm{d}z^2}\right) & -\dfrac{\mathrm{d}^2T_{k-1}}{\mathrm{d}z^2} \\
\vdots & & \vdots & \vdots \\
\dfrac{\mathrm{d}^dT_{k+1}}{\mathrm{d}z^d} & = & 2\left(d\,\dfrac{\mathrm{d}^{d-1}T_k}{\mathrm{d}z^{d-1}} + z\,\dfrac{\mathrm{d}^dT_k}{\mathrm{d}z^d}\right) & -\dfrac{\mathrm{d}^dT_{k-1}}{\mathrm{d}z^d}; \quad (\forall\, d \geq 1).
\end{array}$$

The inner product of two Chebyshev orthogonal polynomials satisfies the orthogonality property,

$$\langle T_i(z), T_j(z) \rangle = \int_{-1}^{+1} T_i(z)\,T_j(z)\,\frac{1}{\sqrt{1-z^2}}\,\mathrm{d}z = \begin{cases} = 0 & \text{if} \quad i \neq j \\ = \pi & \text{if} \quad i = j = 0\,. \\ = \pi/2 & \text{if} \quad i = j \neq 0 \end{cases}$$

where the orthogonality appears when taking two distinct Chebyshev orthogonal polynomials with indices, $i \neq j$.

Appendix A.2. Legendre Orthogonal Polynomials

The Legendre orthogonal polynomials, $L_k(z)$, are also defined on the domain $z \in [-1, +1]$, with measure $\mathrm{d}\mu(z) = \mathrm{d}z$. These polynomials can also be generated recursively by,

$$L_{k+1} = \frac{2k+1}{k+1}\, z\, L_k - \frac{k}{k+1}\, L_{k-1} \qquad \text{starting with: } \begin{cases} L_0 = 1 \\ L_1 = z \end{cases} \tag{A1}$$

All derivatives of Legendre orthogonal polynomials can be computed in a recursive way, starting from,

$$\frac{\mathrm{d}L_0}{\mathrm{d}z} = 0, \quad \frac{\mathrm{d}L_1}{\mathrm{d}z} = 1 \quad \text{and} \quad \frac{\mathrm{d}^d L_0}{\mathrm{d}z^d} = \frac{\mathrm{d}^d L_1}{\mathrm{d}z^d} = 0 \quad (\forall\, d > 1),$$

while the subsequent derivatives of Equation (A1) for $k \geq 1$ can be computed in cascade,

$$\begin{aligned}
\frac{\mathrm{d}L_{k+1}}{\mathrm{d}z} &= \frac{2k+1}{k+1}\left(L_k + z\frac{\mathrm{d}L_k}{\mathrm{d}z}\right) && -\frac{k}{k+1}\frac{\mathrm{d}L_{k-1}}{\mathrm{d}z} \\
\frac{\mathrm{d}^2 L_{k+1}}{\mathrm{d}z^2} &= \frac{2k+1}{k+1}\left(2\frac{\mathrm{d}L_k}{\mathrm{d}z} + z\frac{\mathrm{d}^2 L_k}{\mathrm{d}z^2}\right) && -\frac{k}{k+1}\frac{\mathrm{d}^2 L_{k-1}}{\mathrm{d}z^2} \\
&\;\;\vdots \qquad\qquad\qquad \vdots && \qquad\qquad \vdots \\
\frac{\mathrm{d}^d L_{k+1}}{\mathrm{d}z^d} &= \frac{2k+1}{k+1}\left(d\frac{\mathrm{d}^{d-1} L_k}{\mathrm{d}z^{d-1}} + z\frac{\mathrm{d}^d L_k}{\mathrm{d}z^d}\right) && -\frac{k}{k+1}\frac{\mathrm{d}^d L_{k-1}}{\mathrm{d}z^d}; \quad (\forall\, d \geq 1).
\end{aligned}$$

Additionally, the orthogonality of Legendre polynomials is given by,

$$\langle L_i(z), L_j(z)\rangle = \int_{-1}^{+1} L_i(z)\, L_j(z)\; \mathrm{d}z = \frac{2}{2i+1}\,\delta_{ij} \qquad \text{where} \qquad \delta_{ij} = \begin{cases} 1, & i = j \\ 0, & i \neq j \end{cases} \quad .$$

Appendix B. Solution Times

Appendix B.1. Problem #1 Solution Times

Table A1. Least-squares solution time using Chebyshev polynomials in milliseconds for Problem #1.

n \ m	5	10	15	20	25
5	0.46	-	-	-	-
10	0.49	1.45	-	-	-
15	0.61	1.88	4.90	-	-
20	0.68	2.42	6.50	13.83	-
25	0.92	3.93	8.23	18.07	35.63
30	0.97	4.09	13.23	22.09	50.88

Table A2. Total solution time using Chebyshev polynomials in seconds for Problem #1.

m n	5	10	15	20	25
5	0.053	-	-	-	-
10	0.119	0.128	-	-	-
15	0.210	0.288	0.430	-	-
20	0.399	0.584	0.824	1.114	-
25	0.697	1.011	1.582	2.344	3.609
30	1.045	1.871	2.909	4.683	8.164

Table A3. Least-squares solution time using Legendre polynomials in milliseconds for Problem #1.

m n	5	10	15	20	25
5	0.77	-	-	-	-
10	0.58	1.41	-	-	-
15	0.56	1.82	4.70	-	-
20	0.65	2.56	6.62	13.74	-
25	0.86	3.15	7.93	17.71	35.61
30	1.32	3.53	9.47	22.06	46.26

Table A4. Total solution time using Legendre polynomials in seconds for Problem #1.

m n	5	10	15	20	25
5	0.061	-	-	-	-
10	0.111	0.164	-	-	-
15	0.214	0.298	0.368	-	-
20	0.394	0.613	0.863	1.108	-
25	0.692	1.095	1.606	2.389	3.505
30	1.031	1.689	2.765	4.696	7.063

Appendix B.2. Problem #2 Solution Times

Table A5. Least-squares solution time using Chebyshev polynomials in milliseconds for Problem #2.

m n	5	10	15	20	25
5	1.76	-	-	-	-
10	2.14	5.56	-	-	-
15	2.38	7.79	27.30	-	-
20	2.99	10.17	36.38	71.75	-
25	3.90	13.00	42.40	93.11	144.5
30	4.62	18.20	48.68	105.7	185.8

Table A6. Total solution time using Chebyshev polynomials in seconds for Problem #2.

m n	5	10	15	20	25
5	0.547	-	-	-	-
10	1.905	1.384	-	-	-
15	4.045	3.337	4.460	-	-
20	7.460	6.753	9.771	10.97	-
25	11.93	12.95	17.43	21.15	22.88
30	18.94	21.98	28.90	38.92	45.82

Table A7. Least-squares solution time using Legendre polynomials in milliseconds for Problem #2.

m n	5	10	15	20	25
5	1.64	-	-	-	-
10	1.90	5.46	-	-	-
15	2.24	10.63	21.52	-	-
20	3.27	11.74	29.56	69.51	-
25	3.79	13.82	36.32	91.89	145.1
30	4.17	13.73	51.42	112.4	181.4

Table A8. Total solution time using Legendre polynomials in seconds for Problem #2.

m n	5	10	15	20	25
5	0.508	-	-	-	-
10	1.684	1.330	-	-	-
15	3.875	4.489	3.849	-	-
20	7.284	7.882	8.078	10.56	-
25	11.63	12.86	15.50	21.35	23.15
30	17.57	18.44	28.46	40.53	45.60

References

1. Mortari, D. The Theory of Connections: Connecting Points. *Mathematics* **2017**, *5*, 57. doi:10.3390/math5040057.
2. Mortari, D. Least-Squares Solution of Linear Differential Equations. *Mathematics* **2017**, *5*, 48. doi:10.3390/math5040048.
3. Mortari, D.; Johnston, H.; Smith, L. High Accuracy Least-squares Solutions of Nonlinear Differential Equations. *J. Comput. Appl. Math.* **2019**, *352*, 293–307. doi:10.1016/j.cam.2018.12.007.
4. Johnston, H.; Mortari, D. Least-squares Solutions of Boundary-value Problems in Hybrid Systems. *arXiv* **2019**, arXiv:math.OC/1911.04390.
5. Furfaro, R.; Mortari, D. Least-squares Solution of a Class of Optimal Space Guidance Problems via Theory of Connections. *ACTA Astronaut.* **2019**. doi:10.1016/j.actaastro.2019.05.050.
6. Johnston, H.; Schiassi, E.; Furfaro, R.; Mortari, D. Fuel-Efficient Powered Descent Guidance on Large Planetary Bodies via Theory of Functional Connections. *arXiv* **2020**, arXiv:math.OC/2001.03572.
7. Mai, T.; Mortari, D. Theory of Functional Connections Applied to Nonlinear Programming under Equality Constraints. In Proceeding of the 2019 AAS/AIAA Astrodynamics Specialist Conference, Portland, ME, USA, 11–15 August 2019.
8. Johnston, H.; Leake, C.; Efendiev, Y.; Mortari, D. Selected Applications of the Theory of Connections: A Technique for Analytical Constraint Embedding. *Mathematics* **2019**, *7*, 537. doi:10.3390/math7060537.
9. Mortari, D.; Leake, C. The Multivariate Theory of Connections. *Mathematics* **2019**, *7*, 296. doi:10.3390/math7030296.

10. Leake, C.; Johnston, H.; Smith, L.; Mortari, D. Analytically Embedding Differential Equation Constraints into Least Squares Support Vector Machines Using the Theory of Functional Connections. *Mach. Learn. Knowl. Extr.* **2019**, *1*, 1058–1083. doi:10.3390/make1040060.
11. Leake, C.; Mortari, D. Deep Theory of Functional Connections: A New Method for Estimating the Solutions of Partial Differential Equations. *Mach. Learn. Knowl. Extr.* **2020**, *2*, 37–55.
12. Schiassi, E.; Leake, C.; Florio, M.D.; Johnston, H.; Furfaro, R.; Mortari, D. Extreme Theory of Functional Connections: A Physics-Informed Neural Network Method for Solving Parametric Differential Equations. *arXiv* **2020**, arXiv:cs.LG/2005.10632.
13. Leake, C.; Mortari, D. An Explanation and Implementation of Multivariate Theory of Functional Connections via Examples. In Proceeding of the AIAA/AAS Astrodynamics Specialist Conference, Portland, ME, USA, 11–15 August 2019.
14. Ye, J.; Gao, Z.; Wang, S.; Cheng, J.; Wang, W.; Sun, W. Comparative Assessment of Orthogonal Polynomials for Wavefront Reconstruction over the Square Aperture. *J. Opt. Soc. Am. A* **2014**, *31*, 2304–2311.
15. Dunkl, C.F.; Xu, Y. *Orthogonal Polynomials of Several Variables*, 2nd ed.; Encyclopedia of Mathematics and Its Applications; Cambridge University Press: Cambridge, UK, 2014. doi:10.1017/CBO9781107786134.
16. Xu, Y. Multivariate Orthogonal Polynomials and Operator Theory. *Trans. Am. Math. Soc.* **1994**, *343*, 193–202.
17. Langtangen, H.P. *Computational Partial Differential Equations: Numerical Methods and Diffpack Programming*; OCLC: 851766084; Springer: Berlin/Heidelberg, Germany, 2003;
18. Lanczos, C. Applied Analysis. In *Progress in Industrial Mathematics at ECMI 2008*; Dover Publications, Inc.: New York, NY, USA, 1957; p. 504.
19. Wright, K. Chebyshev Collocation Methods for Ordinary Differential Equations. *Comput. J.* **1964**, *6*, 358–365.
20. Maclaurin, D.; Duvenaud, D.; Johnson, M.; Townsend, J. Autograd. 2013. Available online: https://github.com/HIPS/autograd (accessed on 1 July 2020).
21. Baydin, A.G.; Pearlmutter, B.A.; Radul, A.A.; Siskind, J.M. Automatic Differentiation in Machine Learning: A Survey. *J. Mach. Learn. Res.* **2018**, *18*, 1–43.
22. Lagaris, I.E.; Likas, A.; Fotiadis, D.I. Artificial neural networks for solving ordinary and partial differential equations. *IEEE Trans. Neural Netw.* **1998**, *9*, 987–1000. doi:10.1109/72.712178.
23. Mall, S.; Chakraverty, S. Single Layer Chebyshev Neural Network Model for Solving Elliptic Partial Differential Equations. *Neural Process. Lett.* **2017**, *45*, 825–840. doi:10.1007/s11063-016-9551-9.
24. Sun, H.; Hou, M.; Yang, Y.; Zhang, T.; Weng, F.; Han, F. Solving Partial Differential Equation Based on Bernstein Neural Network and Extreme Learning Machine Algorithm. *Neural Process. Lett.* **2019**, *50*, 1153–1172. doi:10.1007/s11063-018-9911-8.
25. Huang, G.B.; Zhu, Q.Y.; Siew, C.K. Extreme learning machine: Theory and applications. *Neurocomputing* **2006**, *70*, 489–501. doi:10.1016/j.neucom.2005.12.126.