Article

Not peer-reviewed version

An Overview of the Euler-Type Universal Numerical Integrator (E-TUNI): Applications in Non-Linear Dynamics and Predictive Control

Paulo M. Tasinaffo *, Gildárcio S. Gonçalves, Johnny C. Marques, Luiz A. V. Dias, Adilson M. da Cunha

Posted Date: 13 August 2025

doi: 10.20944/preprints202508.0719.v1

Keywords: Adams-Bashforth neural network; Euler-type universal numerical integrator; neural differential equation; Runge-Kutta neural network; universal numerical integrator

 doi:10.20944/preprints202508.0719.v1

Article

An Overview of the Euler-Type Universal Numerical Integrator (E-TUNI): Applications in Non-Linear Dynamics and Predictive Control

Paulo M. Tasinaffo *, Gildárcio S. Gonçalves, Johnny C. Marques, Luiz A. V. Dias and Adilson M. da Cunha

Instituto Tecnológico de Aeronáutica (ITA), São José dos Campos/SP, Brazil
* Correspondence: tasinaffo@ita.br

Abstract: A Universal Numerical Integrator (UNI) is a computational framework that combines a classical numerical integration method, such as Euler, Runge-Kutta, or Adams-Bashforth, with a universal approximator of functions, such as a feed-forward neural network (including MLP, SVM, RBF, among others) or a fuzzy inference system. The Euler-Type Universal Numerical Integrator (E-TUNI) is a particular case of UNI based on the first-order Euler integrator and is designed to model nonlinear dynamic systems observed in real-world scenarios accurately. The UNI framework can be organized into three primary methodologies: the NARMAX model (Nonlinear AutoRegressive Moving Average with eXogenous input), the mean derivatives approach (which characterizes E-TUNI), and the instantaneous derivatives approach. The E-TUNI methodology relies exclusively on mean derivative functions, distinguishing it from techniques that employ instantaneous derivatives. Although it is based on a first-order scheme, the E-TUNI achieves an accuracy level comparable to that of higher-order integrators. This performance is made possible by the incorporation of a neural network acting as a universal approximator, which significantly reduces the approximation error. This article provides a comprehensive overview of the E-TUNI methodology, focusing on its application to the modeling of nonlinear autonomous dynamic systems and its use in predictive control. Several computational experiments are presented to illustrate and validate the effectiveness of the proposed method.

Keywords: Adams-Bashforth neural network; Euler-type universal numerical integrator; neural differential equation; Runge-Kutta neural network; universal numerical integrator

1. Introduction

One of the first works on artificial neural networks is credited to McCulloch and Pitts in [1] and is a work from 1943. In these eighty years of development of artificial intelligence and artificial neural networks, many interesting things have emerged in this area of human knowledge. In [2], a 1957 work credited to the great Russian mathematician Kolmogorov, it is demonstrated that any n-dimensional function can be represented by linearly combining one-dimensional non-linear functions. In [3], Kolmogorov's work from 1957 was extensively recognized, and the term Kolmogorov neural networks was introduced in the literature. In [4], the classic back-propagation algorithm is proposed for training Multi-Layer Perceptron (MLP) networks with at least one inner layer. In [5,6], it is demonstrated, independently, that MLP networks are universal approximators of functions. A very detailed summary of the main scientific knowledge achieved by artificial neural networks in the 20th century can be found at [7].

One of the great applications of artificial neural networks is in the modeling of non-linear dynamic systems in the real world. Thus, next, a bibliographical review is reported, only on this theme, considering only neural networks with feed-forward architecture trained through supervised learning.

In [8], the concept of Universal Numerical Integrator (UNI) is defined. A UNI is nothing more than the coupling of a conventional numerical integrator (e.g., Euler, Runge-Kutta, predictive-corrector,

 doi:10.20944/preprints202508.0719.v1

among others) with a universal approximator of functions (e.g., MLP, SVM, RBF, and wavelet networks, fuzzy inference system, paraconsistent inference system, among others). However, it is essential to note that this definition only took into account neural networks with feed-forward architecture trained exclusively through supervised learning using input/output training patterns.

Also in [8], an interesting classification for UNIs is given. According to this reference, the UNIs can be divided into three large classes, thus giving rise to three distinct methodologies, namely: (i) the NARMAX model, (ii) the mean derivative methodology (e.g., E-TUNI), and (iii) the instantaneous derivatives methodology (e.g., Runge-Kutta neural network, Adams-Bashforth neural network, predictive-corrector neural network, among others). In [8], these methodologies are also presented in some depth, and a detailed table comparing these three methodologies is presented. It is important to note that in the classification of UNIs, the NARMAX model was included. The reason for this is that, although in the NARMAX model, the neural network is not included in the numerical integrator structure, the neural network behaves as if it were a numerical integrator.

In [9–11], they are all classic 20th-century books on conventional numerical integrators. The term "conventional" is used here to inform the reader that such structures do not have a universal approximator of functions coupled to them. In [12,13], the NARMAX methodology is discussed in detail using artificial neural networks. In [14] is Leonard Euler's original work on the first-order integrator that bears his name. This is a 1768 work and initially uses instantaneous derivatives rather than mean derivatives. Everyone knows that the first-order Euler integrator, although mathematically quite simple, is relatively imprecise. However, when the instantaneous derivative functions are exchanged for the mean derivative functions, the Euler integrator becomes as accurate as any other higher-order integrator. In this case, such structures are given the special name of Euler-Type Universal Numerical Integrator (E-TUNI).

Furthermore, a correction must be given here. In [8], the E-TUNI is called Euler Neural Networks and is a 2019 reference. However, in the 2009 reference [15], the term Euler Neural Network is also used, but in a different context. Therefore, to avoid any confusion, we decided to name the first-order UNI of E-TUNI. That said, the references [16–18] refer directly or indirectly to E-TUNI. However, since reference [16] is relatively old, the term E-TUNI is referred to there only as Euler Neural Integrator or Mean Derivative Methodology.

In [16], the E-TUNI was coupled to a neural network with an MLP architecture and applied in a predictive control structure. In [17], a qualitative analysis on the design of a Universal Numerical Integrator (UNI) is carried out in detail. In this reference, the E-TUNI is compared with the NARMAX Model and the Runge-Kutta Neural Network (RKNN). Reference [18] is a continuation of reference [17]. In reference [18], a quantitative analysis of the design of a universal numerical integrator is presented in some mathematical depth. In [19], an important work is done involving E-TUNI in a backward integration process. It is worth noticing that E-TUNI, as well as the NARMAX model, are dynamic models that are naturally discrete and not continuous.

The first work we find in the UNI literature is credited to Wang and Lin in [20]. This reference is from 1998, and nothing before that has been found in the literature. This work is about the Runge-Kutta Neural Network (RKNN). In [21], the Runge-Kutta neural network is used in control. It is interesting to note that there is a gap of more than 20 years between the [20] and [21] references. It is also essential to note that the Predictive-Corrector Neural Network (PCNN) still does not exist in the literature. In [22], it is an essential and current work on neural differential equations. In [23], it is another work on the Runge-Kutta neural network applied in control.

It is vital to know that both the NARMAX model and the mean derivatives methodology (E-TUNI) are inherently of fixed integration step Δt. This property means that if it is desired to change the Δt integration step in these two methodologies, then it is necessary to do a new neural training. However, in the instantaneous derivatives methodology (RKNN, ABNN, PCNN, among others) the integration step Δt can be changed, within certain limits imposed by the order of the referred integrator used,

 doi:10.20944/preprints202508.0719.v1

without having to carry out a new training [8,17,18,20]. Finally, some small theoretical models of non-linear ordinary differential equations can be taken from [24] to test any UNI.

The original contributions of this paper are as follows: (i) it presents an overview of how E-TUNI works to solve dynamics and control problems, (ii) it presents three studies of practical technological applications of the proposed model (e.g., including the orbit transfer problem and the non-linear simple pendulum problem), (iii) it presents a practical example of how to generate an approximate continuous solution for the E-TUNI model (see the nonlinear simple pendulum problem), which is naturally discrete in solution, and (iv) it presents a correct proof of the general expression for E-TUNI, since in [16] it is not correct. Furthermore, the E-TUNI has its origins in [16].

Accordingly, the remainder of this article is organized as follows. Section 2 describes in detail the meaning of the main discrete and continuous variables, which are used to describe the proposed E-TUNI model. Section 3 presents a concise yet comprehensive theoretical overview of the Euler-Type Universal Numerical Integrator (E-TUNI), outlining its foundational principles and key mathematical formulations. Section 4 details three numerical simulations that serve to validate the proposed methodology and demonstrate its practical effectiveness. Finally, Section 5 summarizes the main conclusions, highlighting the contributions and potential future directions of this research.

2. Preliminaries and Symbols Used

To facilitate a deeper understanding of the theoretical framework developed in this paper, we present a complete definition of all symbols and variables used throughout the work. The proposed methodology is named Euler Type Universal Numerical Integrator (E TUNI), and it is based on a forward integration scheme that operates in a discrete-time setting. This approach is designed to approximate the behavior of continuous-time autonomous ordinary differential equations by means of a discrete formulation. For clarity, all notation used in this context is listed below and classified into two main categories: (i) variables defined in continuous time and (ii) variables defined in discrete time.

(i) Continuous Variables

$\dot{y} = f(y) \cdots$ System of continuous differential equations.
$y = [y_1 \quad y_2 \cdots y_n]^T \cdots$ State Variables.
$f(y) = [f_1(y) \quad f_2(y) \cdots f_n(y)]^T \cdots$ Instantaneous derivative functions.
$y_j^i(t) = g_j^i(t) \cdots$ Particular continuous and differentiable curve of a family of solution curves of the dynamical system $\dot{y} = f(y)$.
$\dot{y}_j^i(t) = \dot{g}_j^i(t) \cdots$ First derivative of $y_j^i(t)$.

(ii) Discrete Variables

${}^k y^i = y^i(t_0 + k \cdot \Delta t) \cdots$ Vector of state variables at time t_k.
${}^k y_j^i \cdots$ Scalar state variable for $j = 1, 2, \cdots, n$ at time t_k. It is a generic discretization point of the state variables generated by the integers i, j, and k.
$n \cdots$ Total number of state variables.
${}^k u = u(t_0 + k \cdot \Delta t) \cdots$ Vector of control variables at time t_k.
${}^k u_j \cdots$ Scalar control variable for $j = 1, 2, \cdots, m$ at time t_k.
$m \cdots$ Total number of control variables.
${}^{k+1} y^i = y^i[t_0 + (k+1) \cdot \Delta t] \cdots$ Exact vector of state variables at time t_{k+1}.
${}^{k+1} y_j^i \cdots$ Exact scalar state variable for $j = 1, 2, \cdots, n$ at time t_{k+1}.
${}^{k+1} \hat{y}^i = \hat{y}^i[t_0 + (k+1) \cdot \Delta t] \cdots$ Estimated Vector of state variables by UNI or E-TUNI at time t_{k+1}.
${}^{k+1} \hat{y}_j^i \cdots$ Scalar state variable estimated by UNI or E-TUNI for $j = 1, 2, \cdots, n$ at time t_{k+1}.
${}^{k+1} \tilde{y}^i \cdots$ Estimated Vector of state variables when using only the integrator and without using the neural network at time t_{k+1}.

 doi:10.20944/preprints202508.0719.v1

$tan_{\Delta t}{}^k\alpha^i = tan_{\Delta t}{}^k\alpha^i = [tan_{\Delta t}{}^k\alpha_1^i \quad tan_{\Delta t}{}^k\alpha_2^i \cdots tan_{\Delta t}{}^k\alpha_n^i]^T \cdots$ Exact vector of positive mean derivative functions at time t_k.
$tan_{\Delta t}{}^k\alpha_j^i = tan_{\Delta t}{}^k\alpha_j^i = \frac{{}^{k+1}y_j^i - {}^k y_j^i}{\Delta t} \cdots$ Scalar positive mean derivative functions for $j = 1, 2, \cdots, n$ at time t_k.
$tan_{\Delta t}{}^k\hat{\alpha}^i = [tan_{\Delta t}{}^k\hat{\alpha}_1^i \quad tan_{\Delta t}{}^k\hat{\alpha}_2^i \cdots tan_{\Delta t}{}^k\hat{\alpha}_n^i]^T \cdots$ Estimated vector of positive mean derivative functions by the E-TUNI at time t_k.
$tan^k\theta^i = [tan^k\theta_1^i \quad tan^k\theta_2^i \cdots tan^k\theta_n^i]^T \cdots$ Vector of positive instantaneous derivatives at time t_k.
$tan^k\theta_j^i = \lim_{\Delta t \to 0} \frac{{}^{k+1}y_j^i - {}^k y_j^i}{\Delta t} \cdots$ Scalar positive instantaneous derivative for $j = 1, 2, \cdots, n$ at instant t_k.
$t_k \cdots$ Time instant $t_k = t_0 + k \cdot \Delta t$.
$t_{k+1} \cdots$ Time instant $t_{k+1} = t_0 + (k+1) \cdot \Delta t$.
$\Delta t \cdots$ Integration step.
$i \cdots$ Over-index that enumerates a particular curve from the family of curves of the dynamical system to be modelled ($i = 1, 2, \cdots, q$).
$j \cdots$ Under-index that enumerates the state and control variables.
$k \cdots$ Over-index that enumerates the discrete time instants ($k = 1, 2, \cdots, r$).
$r \cdots$ Total number of horizons of the time variable.
$q \cdots$ Total number of curves from the family of curves curves of the dynamic system to be modelled.
$t_k^* \cdots$ Instant of time within the interval $[t_k, t_{k+1}]$ as a result of the Differential Mean Value Theorem (see Theorem 1).
$t_k^x \cdots$ Instant of time within the interval $[t_k, t_{k+1}]$ as a result of the Integral Mean Value Theorem (see Theorem 2).

To interpret the notation used in this work, it is essential to distinguish between the variables ${}^{k+1}y^i$, ${}^{k+1}\tilde{y}^i$, and ${}^{k+1}\hat{y}^i$. The vector variable ${}^{k+1}y^i$ is the exact value of the state variables at time $t_{k+1} = t_0 + (k+1) \cdot \Delta t$. The vector variable ${}^{k+1}\tilde{y}^i$ is the estimated value of the state variables using only the numerical integrator. Finally, the vector variable ${}^{k+1}\hat{y}^i$ is the estimated value of the state variables using the numerical integrator coupled to an artificial neural network. Figure 1 follows this convention and helps to understand better the notations adopted here. Additionally, it is important to understand the meaning of the auxiliary variables i, j, and k. This is accomplished in the next paragraph.

A key point in this notation is the interpretation of the over-indexes k and i and the under-index j in the variables ${}^k y_j^i$ and $tan_{\Delta t}{}^k\alpha_j^i$. When the auxiliary variables i, j, and k are used simultaneously, they uniquely identify the secant ($tan_{\Delta t}{}^k\alpha_j^i$) at the point ${}^k y_j^i$. Note that the over-index k indicates the time instant $t_k = t_0 + k \cdot \Delta t$ of the secant, the under-index j indicates the state variable in question, where $j = 1, 2, \cdots, n$, and the over-index i ($i = 1, 2, \cdots, q$) indicates the particular curve where the respective secant is located, from the family of possible curves of the system of differential equations considered. The value of q can be as large as desired. The larger the value of q, the more different curves the neural network trains on, and the better its generalization. This convention is quite useful to fully understand the formal proof of the general expression of E-TUNI that is performed in section 3.2.

3. Mathematical Development

In this section, we present a concise and complete description of the Euler-Type Universal Numerical Integrator (E-TUNI). To this end, we present a formal mathematical proof for the general expression that governs E-TUNI's operation to generate discrete solutions for autonomous nonlinear dynamical systems governed by ordinary differential equations. Additionally, we also present the correct way to use E-TUNI in a predictive control framework. We conclude this section by obtaining an approximate mathematical expression for E-TUNI to provide a continuous solution, rather than a discrete solution, for autonomous dynamical systems.

 doi:10.20944/preprints202508.0719.v1

3.1. Basic Mathematical Development of E-TUNI

We provide a brief mathematical description of E-TUNI below. Then, in the following sub-section, we perform a formal mathematical demonstration of the general expression of the first-order Euler Integrator designed with mean derivative functions. So, by definition, the secants or mean derivatives $tan_{\Delta t}{}^{k}\alpha_j^i$ for $j = 1, 2, \cdots, n$ between the points ${}^{k+1}y_j^i$ and ${}^{k}y_j^i$ are given by:

$$tan_{\Delta t}{}^{k}\alpha_j^i = \frac{{}^{k+1}y_j^i - {}^{k}y_j^i}{\Delta t} \tag{1}$$

where, ${}^{k+1}y_j^i = y_j^i[t + (k+1)\cdot\Delta t]$ is the forward state of the dynamic system, ${}^{k}y_j^i = y_j^i[t + k\cdot\Delta t]$ is the present state of the dynamical system, the over-index k on the left indicates the instant k, the over-index i on the right suggests a discretization of the continuum, the sub-index j on the right indicates the j-th state variable, n is the total number of state variables and Δt is the integration step. We talk more about the over-index i later in this article.

A geometric and intuitive difference between the mean and the instantaneous derivative functions is shown in Figure 1. In line with the Figures 1 and 2, the E-TUNI is entirely based on the concept of mean derivative functions and not on the idea of instantaneous derivative functions. As will be seen throughout this article, this change is quite significant when incorporated adequately into the Euler-type first-order integrator. Furthermore, the mean derivative functions can also be obtained from supervised training with input/output patterns using a neural network with any feed-forward architecture (MLP, RBF, SVM, among others).

It is also essential to note that it is possible to train E-TUNI in two different ways, namely: a) through the direct approach or b) through the indirect or empirical approach. In the direct approach, the neural network is trained decoupled from the Euler-type integrator, while in the indirect or empirical approach, the neural network is trained coupled to the structure of the first-order Euler integrator. For this reason, in the indirect approach, the back-propagation algorithm needs to be modified slightly. In the references [8,17,18] this is explained in more detail.

In this way, be a non-linear dynamic system of n simultaneous first-order equations with dependent variables $y_1, y_2, \cdots, y_n$. If each of these variables satisfies a given initial condition for the same value a of t, then we have an initial value problem for a first-order system, and we can write:

$$\begin{cases} \dot{y}_1 = f_1(y_1, y_2, \cdots y_n), & y_1(a) = \eta_1 \\ \dot{y}_2 = f_2(y_1, y_2, \cdots y_n), & y_2(a) = \eta_2 \\ \qquad\vdots & \qquad\vdots \\ \dot{y}_n = f_n(y_1, y_2, \cdots y_n), & y_n(a) = \eta_n \end{cases} \tag{2}$$

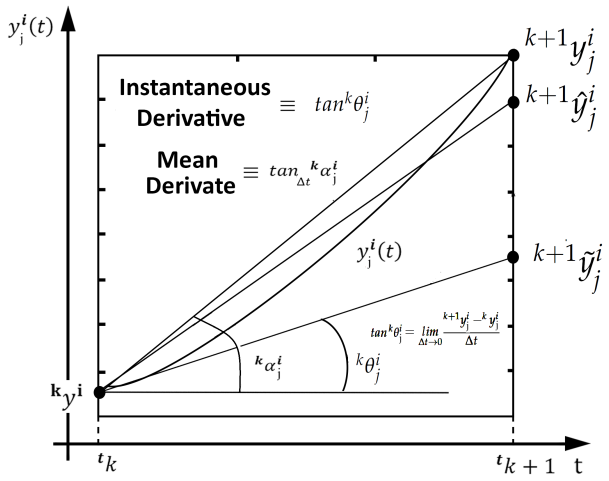

Figure 1. Difference between mean derivative and instantaneous derivative functions (Source: see [19]).

 doi:10.20944/preprints202508.0719.v1

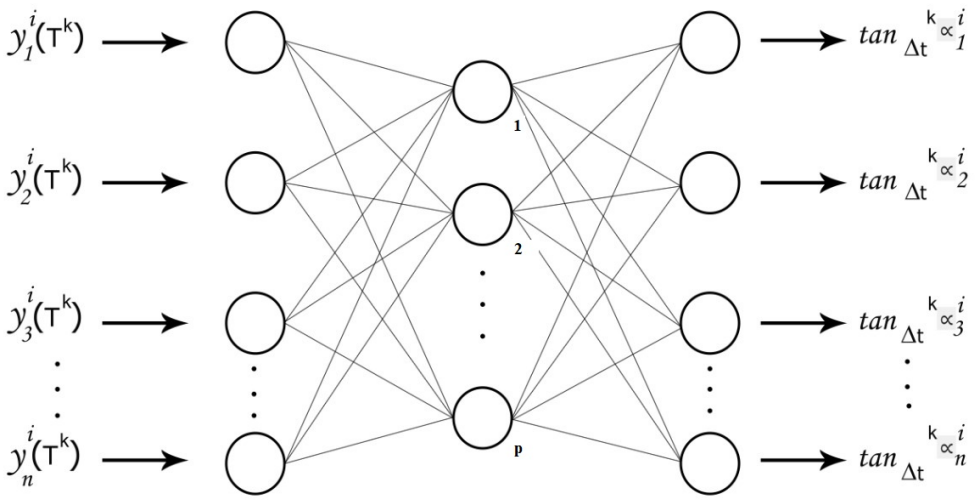

Figure 2. A feed-forward neural network project with the concept of mean derivative functions.

In [14], the mathematician Leonard Euler himself proposed in 1768 the first numerical integrator, in the history of mathematics, to approximately solve non-linear dynamical systems governed by the equation (2). This solution, which is well-known to everyone, is given by:

$$\begin{cases} {}^{k+1}y_1^i \cong tan^k\theta_1^i \cdot \Delta t + {}^k y_1^i \\ {}^{k+1}y_2^i \cong tan^k\theta_2^i \cdot \Delta t + {}^k y_2^i \\ \vdots \\ {}^{k+1}y_n^i \cong tan^k\theta_n^i \cdot \Delta t + {}^k y_n^i \end{cases} \tag{3}$$

where, $tan^k\theta_j^i$ for $j = 1, 2, \cdots, n$ are instantaneous derivative functions, according to the graphical notation shown in Figure 1. On the other hand, there is an attempt in [18] to prove mathematically that if you exchange the instantaneous derivatives in (3) for the mean derivatives, then the solution proposed by Leonard Euler becomes accurate, i.e.,

$$\begin{cases} {}^{k+1}y_1^i = tan_{\Delta t}{}^k\alpha_1^i \cdot \Delta t + {}^k y_1^i \\ {}^{k+1}y_2^i = tan_{\Delta t}{}^k\alpha_2^i \cdot \Delta t + {}^k y_2^i \\ \vdots \\ {}^{k+1}y_n^i = tan_{\Delta t}{}^k\alpha_n^i \cdot \Delta t + {}^k y_n^i \end{cases} \tag{4}$$

Comparing equations (3) and (4), it is observed that the instantaneous derivative functions do not depend on the integration step, but the mean derivative functions do. Additionally, the equations in (4) can also be expressed in a more compact notation, given by:

$${}^{k+1}y^i = tan_{\Delta t}{}^k\alpha^i \cdot \Delta t + {}^k y^i \tag{5}$$

where, ${}^{k+1}y^i = [\ {}^{k+1}y_1^i\ {}^{k+1}y_2^i \cdots {}^{k+1}y_n^i\ ]^T$, $tan_{\Delta t}{}^k\alpha^i = [\ tan_{\Delta t}{}^k\alpha_1^i\ tan_{\Delta t}{}^k\alpha_2^i \cdots tan_{\Delta t}{}^k\alpha_n^i\ ]^T$ e ${}^k y^i = [\ {}^k y_1^i\ {}^k y_2^i \cdots {}^k y_n^i\ ]^T$. The generalization of the mean derivatives methodology to multiple backward inputs and/or multiple forward outputs is relatively easy. This expression can be obtained by following the equation:

$${}^{k+p}y_j^i = \sum_{m=0}^{p-1} tan_{\Delta t}{}^{k+m}\alpha_j^i \cdot \Delta t + {}^k y_j^i \tag{6}$$

where p is the number of backward and/or forward instants and $j = 1, 2, \cdots, n$. For example, if $p = p_1 + p_2$ then it is possible to design an E-TUNI with p_1 inputs in the role of backward mean derivatives and p_2 outputs in the role of forward mean derivative. In this way, the reader is free to choose the

 doi:10.20944/preprints202508.0719.v1

values of p_1 and p_2 as long as the previous equality is satisfied. However, it should be noted that the first input of the neural network must be an absolute value, not a relative one.

On the other hand, notice that if the reader tries to compare the NARMAX model with the E-TUNI structure, one can see that the former is very similar to the latter. In the NARMAX model, for example, the output of the universal approximator of functions is the forward instant $y_j(t + \Delta t)$. However, in the E-TUNI model, the output of the universal approximator of functions is the mean derivative function ${}^{k}tan_{\Delta t}\alpha_j^i$ at the present instant. This description is the only practical difference between these two methodologies. However, the E-TUNI may be a little more computationally efficient than the NARMAX model, as explained in the following paragraph.

Considering the direct approach to training the mean derivative functions required by the E-TUNI structure, it is then possible to perform a theoretical analysis of the local error committed by this first-order universal numerical integrator. Thus, let the exact value ${}^{k+1}\bar{y}_j^i$ and the estimated value ${}^{k+1}\hat{y}_j^i$ be obtained, respectively, by the equations (7) and (8) of a given solution of a generic dynamical system.

$$ {}^{k+1}\bar{y}_j^i = {}^{k}tan_{\Delta t}\alpha_j^i \cdot \Delta t + {}^{k}y_j^i \tag{7} $$

$$ {}^{k+1}\hat{y}_j^i = ({}^{k}tan_{\Delta t}\alpha_j^i \pm e_m) \cdot \Delta t + {}^{k}y_j^i \tag{8} $$

where e_m is the mean absolute error of the output variables of the universal approximator of functions used to learn the mean derivative functions. Thus, if the equation (7) is subtracted from the equation (8) and the result of this subtraction is squared, we have:

$$ \left({}^{k+1}\bar{y}_j^i - {}^{k+1}\hat{y}_j^i\right)^2 = \Delta t^2 \cdot e_m^2 \tag{9} $$

The equation (9) states that the local squared error made by E-TUNI can dampen the squared error of training the universal approximator of functions, used in neural training, if $0 < \Delta t < 1$. If $\Delta t > 1$, the local error will be amplified. However, for a more accurate analysis of the global training error, further studies are needed.

The equation (9) is fundamental, as it partially explains why training the E-TUNI with a mean square error e_m^2, greater than that obtained by training the NARMAX model can also yield good results from estimation and potentially surpass those obtained in the NARMAX methodology. However, as stated earlier, this is true only if $\Delta t < 1$.

Therefore, an E-TUNI neural integrator can be used for supervised training of a generic plant. Additionally, this plant can be used in a predictive control framework. Thus, the plant's neural model can be used, as a model of the system's internal response to derive a smooth control policy. This control is then able to track a reference trajectory by minimizing a finite-horizon quadratic functional, given by [16]:

$$ J = \{\sum_{j=1}^{m}[y_r(t_j) - \hat{y}(t_j)]^T \cdot r_y^{-1}(t) \cdot [y_r(t_j) - \hat{y}(t_j)] + \tag{10} $$

$$ \sum_{j=0}^{m-1}[u(t_j) - u(t_{j-1})]^T \cdot r_u^{-1}(t) \cdot [u(t_j) - u(t_{j-1})]\}/2 $$

where, $y_r(t_j)$ is the reference trajectory at the instant t_j; m is the number of horizons ahead; $r_y^{-1}(t_j)$ and $r_u^{-1}(t_j)$ are positive definite weight matrices; $\hat{y}(t_j)$ is the output of the previously trained E-TUNI. Thus, in [16] it is shown that it is necessary to know the partial derivatives $\frac{\partial^{k+q}y^i}{\partial^k u}$ to solve the problem of optimization given by the equation iteratively (10) through the Kalman filter. Also, in [16] these partial derivatives can be calculated as follows:

$$ \frac{\partial^{k+q}y^i}{\partial^k u} = \left(\frac{\partial\, {}^{k+q-1}tan_{\Delta t}\alpha^i}{\partial^{k+q-1}y^i} \cdot \Delta t + I\right) \cdot \frac{\partial^{k+q-1}y^i}{\partial^k u} \tag{11} $$

 doi:10.20944/preprints202508.0719.v1

$$to\ q = 2, 3, ..., m$$

where,

$$\left[\frac{\partial^{k+q-1} y^i}{\partial^k u}\right]_{n_y \times n_u} =$$

$$\Delta t \cdot \begin{bmatrix} \frac{\partial tan_{\Delta t}^{k+q-2} \alpha_1^i}{\partial^k u_1} & \frac{\partial tan_{\Delta t}^{k+q-2} \alpha_1^i}{\partial^k u_2} & \cdots & \frac{\partial tan_{\Delta t}^{k+q-2} \alpha_1^i}{\partial^k u_{n_u}} \\ \frac{\partial tan_{\Delta t}^{k+q-2} \alpha_2^i}{\partial^k u_1} & \frac{\partial tan_{\Delta t}^{k+q-2} \alpha_2^i}{\partial^k u_2} & \cdots & \frac{\partial tan_{\Delta t}^{k+q-2} \alpha_2^i}{\partial^k u_{n_u}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial tan_{\Delta t}^{k+q-2} \alpha_{n_y}^i}{\partial^k u_1} & \frac{\partial tan_{\Delta t}^{k+q-2} \alpha_{n_y}^i}{\partial^k u_2} & \cdots & \frac{\partial tan_{\Delta t}^{k+q-2} \alpha_{n_y}^i}{\partial^k u_{n_u}} \end{bmatrix} \tag{12}$$

$$\left[\frac{\partial tan_{\Delta t}^{k+q-1} \alpha^i}{\partial^{k+q-1} y^i}\right]_{n_y \times n_y} =$$

$$\begin{bmatrix} \frac{\partial tan_{\Delta t}{}^{k+q-1} \alpha_1^i}{\partial^{k+q-1} y_1^i} & \frac{\partial tan_{\Delta t}^{k+q-1} \alpha_1^i}{\partial^{k+q-1} y_2^i} & \cdots & \frac{\partial tan_{\Delta t}^{k+q-1} \alpha_1^i}{\partial^{k+q-1} y_{n_y}^i} \\ \frac{\partial tan_{\Delta t}^{k+q-1} \alpha_2^i}{\partial^{k+q-1} y_1^i} & \frac{\partial tan_{\Delta t}^{k+q-1} \alpha_2^i}{\partial^{k+q-1} y_2^i} & \cdots & \frac{\partial tan_{\Delta t}^{k+q-1} \alpha_2^i}{\partial^{k+q-1} y_{n_y}^i} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial tan_{\Delta t}^{k+q-1} \alpha_{n_y}^i}{\partial^{k+q-1} y_1^i} & \frac{\partial tan_{\Delta t}^{k+q-1} \alpha_{n_y}^i}{\partial^{k+q-1} y_2^i} & \cdots & \frac{\partial tan_{\Delta t}^{k+q-1} \alpha_{n_y}^i}{\partial^{k+q-1} y_{n_y}^i} \end{bmatrix} \tag{13}$$

$$\left[\frac{\partial^{k+1} y^i}{\partial^k u}\right]_{n_y \times n_u} =$$

$$\Delta t \cdot \begin{bmatrix} \frac{\partial\, tan_{\Delta t}{}^k \alpha_1^i}{\partial^k u_1} & \frac{\partial\, tan_{\Delta t}{}^k \alpha_1^i}{\partial^k u_2} & \cdots & \frac{\partial\, tan_{\Delta t}{}^k \alpha_1^i}{\partial^k u_{n_u}} \\ \frac{\partial\, tan_{\Delta t}^k\, \alpha_2^i}{\partial^k u_1} & \frac{\partial\, tan_{\Delta t}{}^k \alpha_2^i}{\partial^k u_2} & \cdots & \frac{\partial\, tan_{\Delta t}{}^k \alpha_2^i}{\partial^k u_{n_u}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial\, tan_{\Delta t}{}^k \alpha_{n_y}^i}{\partial^k u_1} & \frac{\partial\, tan_{\Delta t}{}^k \alpha_{n_y}^i}{\partial^k u_2} & \cdots & \frac{\partial\, tan_{\Delta t}{}^k \alpha_{n_y}^i}{\partial^k u_{n_u}} \end{bmatrix} \tag{14}$$

In the equations (11) to (14), we have that $n_y = n$ is the total number of state variables and n_u is the total number of control variables. Furthermore, to derive these equations, it was assumed that the neural network, which learns the mean derivative functions, was designed with only one late input for the state and control variables, and also only one forward output for the mean derivatives.

So, for example, if a neural network with an MLP architecture is used, the calculation of the partial derivatives, necessary for the use of the gradient training algorithm, can be obtained as follows [16]:

$$\left[tan_{\Delta t}{}^k \alpha_1^i, \quad tan_{\Delta t}{}^k \alpha_2^i, \quad \cdots, \quad tan_{\Delta t}{}^k \alpha_{n_y}^i\right]^T = \tag{15}$$

$$f_{NN}^{MLP} = \left[y_1(t), \cdots, \quad y_{n_y}(t), \quad u_1(t), \quad \cdots, \quad u_{n_u}(t)\right]^T$$

where,

$$\frac{\partial\, y^l}{\partial\, \bar{y}^k} = \frac{\partial\, y^l}{\partial\, \bar{y}^{k+1}} \cdot W^{k+1} \cdot I_{f'(\bar{y}^k)}\ to\ k = l-1, l-2, \cdots, 1$$

$$\frac{\partial\, y^l}{\partial\, \bar{y}^l} = I_{f'(\bar{y}^l)}$$

$$I_{f'(\bar{y}^l)} = \begin{bmatrix} I_{f'(\bar{y}_1^l)} & 0 & \cdots & 0 \\ 0 & I_{f'(\bar{y}_2^l)} & \cdots & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & \cdots & I_{f'(\bar{y}_{n_k}^l)} \end{bmatrix}$$

 doi:10.20944/preprints202508.0719.v1

It is essential to note that l is a generic layer of the MLP network. So y^l are the output values of the l layer, and when l is the last layer, then these outputs necessarily are the mean derivative functions. The $f^{'}(\cdot)$ functions can be any sigmoid function. Furthermore, if another neural architecture is used, for example, RBF networks or Wavelets, it will only change the equation (15). The equations (11), (12), (13) and (14) remain unchanged. The reason for this is that the equations from (11) to (14) refer exclusively to the type of integrator used, which, in this case, is necessarily the E-TUNI.

On the other hand, the equation (15) refers exclusively to the type of feed-forward neural network used, which, in this case, is necessarily the MLP network. Thus, the equation (15) is nothing more than an iterative version of the back-propagation algorithm, which calculates the partial derivatives from the output of the MLP network, concerning its inputs and not concerning the synaptic weights.

It is also worth noticing that, to fully understand the equations from (11) to (14), it is convenient to consult the reference [16], as it is necessary to make a temporal chain of several first-order Euler integrators, which work exclusively with mean derivative functions. This fact is because the horizon m has, in general, temporal advances in an amount greater than the total number of delayed inputs of the E-TUNI used.

3.2. Correct Mathematical Demonstration of the E-TUNI General Expression

In this section, we formally demonstrate the general mathematical expression of E-TUNI, which is nothing more than a first-order Euler-type integrator working with mean derivative functions. This development is done here because, as previously stated, there is a demonstration error in the general E-TUNI expression presented in reference [16]. So, the starting point is Figure 3.

So, for the reader to understand this figure, we explain it, starting from the point ${}^k y_j^i$. The point ${}^k y_j^i$ is an instant of the solution of the considered autonomous dynamic system. The variable j means the jth state variable of the considered dynamic system, where $j = 1, 2, \cdots, n$, that is, there are a total of n state variables. The variable i represents the i-th curve of the family of curves confined in the interval $[y_j^{min}(t_0), y_j^{max}(t_0)]$ and which is the solution of the considered dynamic system. By the continuum hypothesis, one can have infinite curves ($i = 1, 2, \cdots, \infty$). The variable k represents the k-th instant of time, that is, $t^k = t_0 + k \cdot \Delta t$. In the case of the variable k, if the dynamical system solution does not come out of the region of interest $[\, y_j^{min}(t_0), y_j^{max}(t_0) \,]$ and the system is autonomous then the variable k can also go to infinity ($k = 1, 2, \cdots, \infty$). Also, when we write $y_j^i(t_k^x)$ or $y_j^i(t_k^*)$ it means that $t_k < t_k^x < t_{k+1}$ and $t_k < t_k^* < t_{k+1}$.

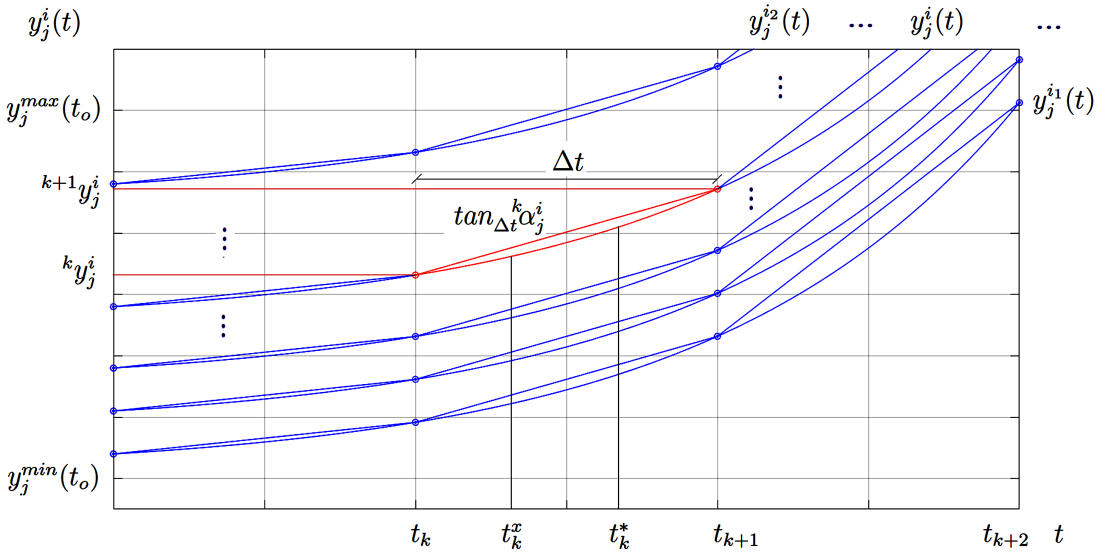

Figure 3. Mapping scheme used to characterize the discretization of the solution obtained through E-TUNI (Source: see [19]).

 doi:10.20944/preprints202508.0719.v1

This notation for ${}^{k}y_j^i$ is sufficient to uniquely map it to its respective region of interest, where its associated secant characteristic $tan_{\Delta t}{}^{k}\alpha_j^i$ is confined. So, in this case, for example, we can say that for the state ${}^{k}y_j^i = y_j^i(t_0 + k \cdot \Delta t)$ we have its associated characteristic secant given by $tan_{\Delta t}{}^{k}\alpha_j^i$.

Thus, the secant $tan_{\Delta t}{}^{k}\alpha_j^i$ is associated with the state variable y_j with a curve i specific to the family of solutions, in this case, by $y_j^i(t)$ and the instant t_k. The fact that the secant is associated with the instant t_k means that it is confined to the closed interval $[t_k, t_{k+1}]$. Thus, the secant $tan_{\Delta t}{}^{k}\alpha_j^i$ starts at t_k and ends at t_{k+1}.

Furthermore, the states $y_j^i(t_k^x)$ and $y_j^i(t_k^*)$ are special states, where both t_k^x and t_k^* are also confined to the closed interval $[t_k, t_{k+1}]$. The instants t_k^x as t_k^* are directly associated, respectively, with the integral mean value theorem and the differential mean value theorem [25,26]. So, having made these preliminary considerations, we begin now with this demonstration in a precise manner. Thus, let the following autonomous system of non-linear differential equations be given by,

$$\dot{y} = f(y) \tag{16}$$

$$y = \begin{bmatrix} y_1 & y_2 & \cdots & y_n \end{bmatrix}^T \tag{17}$$

$$f(y) = \begin{bmatrix} f_1(y) & f_2(y) & \cdots & f_n(y) \end{bmatrix}^T \tag{18}$$

Consider also, by definition, $y_j^i = y_j^i(t)$ for $j = 1, 2, \cdots, n$ a particular trajectory for a family of solutions to the system of differential equations $\dot{y} = f(y)$ passing through $y_j^i(t_0)$ at instant t_0, i.e., initializing from a domain of interest $[y_j^{min}(t_0), y_j^{max}(t_0)]^n$, where $y_j^{min}(t_0)$ and $y_j^{max}(t_0)$ are finite. It is also appropriate to introduce the following vector notation over (16):

$$y_0^i = y^i(t_0) = \begin{bmatrix} y_1^i(t_0) & y_2^i(t_0) & \cdots & y_n^i(t_0) \end{bmatrix}^T \tag{19}$$

$$y^i = y^i(t) = \begin{bmatrix} y_1^i(t) & y_2^i(t) & \cdots & y_n^i(t) \end{bmatrix}^T \tag{20}$$

In [25], by definition, the secant curve between two points ${}^{k}y_j^i$ and ${}^{k+1}y_j^i$ belonging to the curve $y_j^i(t)$ to $j = 1, 2, \cdots, n$ is the line segment joining these two points. Thus, the tangents of the secants between the points ${}^{k}y_1^i$ and ${}^{k+1}y_1^i$, ${}^{k}y_2^i$ and ${}^{k+1}y_2^i$, $\cdots$, ${}^{k}y_n^i$ and ${}^{k+1}y_n^i$ are defined as:

$$tan_{\Delta t}\alpha^i(t + k \cdot \Delta t) = tan_{\Delta t}{}^{k}\alpha^i = \begin{bmatrix} tan_{\Delta t}{}^{k}\alpha_1^i & tan_{\Delta t}{}^{k}\alpha_2^i & \cdots & tan_{\Delta t}{}^{k}\alpha_n^i \end{bmatrix}^T \tag{21}$$

with,

$$tan_{\Delta t}{}^{k}\alpha_j^i = \frac{{}^{k+1}y_j^i - {}^{k}y_j^i}{\Delta t} \tag{22}$$

for $j = 1, 2, \cdots, n$. Thus, we can state now the two fundamental theorems to consolidate our proof. These two theorems are the differential mean value theorem and the integral mean value theorem, which are presented below without proof [25,26].

Theorem 1 (The Differential Mean Value Theorem): If a function $g_j^i(t)$ for $j = 1, 2, \cdots, n$ is a continuous function and defined over the closed interval $[t_k, t_{k+1}]$ is differentiable over the open interval (t_k, t_{k+1}), then there is at least one number t_k^* with $t_k < t_k^* < t_k + \Delta t = t_{k+1}$ such that,

$$\dot{g}_j^i(t_k^*) = \frac{{}^{k+1}g_j^i - {}^{k}g_j^i}{\Delta t} \tag{23}$$

 doi:10.20944/preprints202508.0719.v1

Theorem 2 (The Integral Mean Value Theorem): If a function $g_j^i(t)$ for $j = 1, 2, \cdots, n$ is a continuous function and defined over the closed interval $[t_k, t_{k+1}]$, then there is at least one inner point t_k^x in $[t_k, t_{k+1}]$ such that,

$$g_j^i(t_k^x) \cdot \Delta t = \int_{t_k}^{t_{k+1}} g_j^i(t)dt \tag{24}$$

It is important to note that generally t_k^* is different from t_k^x. Also, the mean value theorems say nothing about how to determine the value of t_k^* and t_k^x. These two theorems simply state that t_k^* and t_k^x are confined to the closed interval $[t_k, t_{k+1}]$.

Property 1: Applying Theorem 2 on the curve $\dot{g}_j^i(t)$ is equivalent to applying Theorem 1 on the curve $g_j^i(t)$ both on the same interval closed $[t_k, t_{k+1}]$, that is, $\dot{g}_j^i(t_k^x) = \dot{g}_j^i(t_k^*)$.

Proof: from Theorem 2 applied to the continuous curve $\dot{g}_j^i(t)$ results in a $\dot{g}_j^i(t_k^x)$ for $t_k < t_k^x < t_{k+1}$ such that $\dot{g}_j^i(t_k^x) \cdot \Delta t = \int_{t_k}^{t_{k+1}} \dot{g}_j^i(t)dt = {}^{k+1}g_j^i - {}^{k}g_j^i$ as a result of the fundamental theorem of calculus. Thus, $\dot{g}_j^i(t_k^x) = \frac{{}^{k+1}g_j^i - {}^{k}g_j^i}{\Delta t} \stackrel{\text{def}}{=} tan_{\Delta t}{}^{k}\alpha_j^i$. On the other hand, the application of Theorem 1 about the continuous and differentiable curve $g_j^i(t)$ implies the existence of a $\dot{g}_j^i(t_k^*)$ for $t_k < t_k^x < t_{k+1}$, such that $\dot{g}_j^i(t_k^*) = \frac{{}^{k+1}g_j^i - {}^{k}g_j^i}{\Delta t} \stackrel{\text{def}}{=} tan_{\Delta t}{}^{k}\alpha_j^i$. Thus, $\dot{g}_j^i(t_k^x) = \dot{g}_j^i(t_k^*)$. However, this proof does not prove that $t_k^x = t_k^*$.□

That said, we prove now the main theorem of this section, namely Theorem 3. But before that, it is worth commenting on some preliminary considerations. In Theorem 3 when we write $\dot{y}^i = f(y^i)$ we are specifying that the general dynamical system $\dot{y} = f(y)$ is harnessed to the particular solution of the curve i belonging to the family of curves that is a solution of $\dot{y} = f(y)$. Thus, $\dot{y}^i = f(y^i)$ is an alternative way of saying that the general dynamical system $\dot{y} = f(y)$ is harnessed to some initial condition $y(t_0) = \eta_0$. Thus, the notation with the index i is more didactic to accept the uniqueness of the mapping explained at the beginning of this sub-section in Figure 3. However, in this article, we did not prove the uniqueness of the mapping discussed in Figure 3, but we assume it is true. To better understand this type of mapping, see a practical example of the same that appears in [19] (see Fig. 7 in this same reference).

Another essential consideration for understanding the proof of Theorem 3 is that it is necessary to know the solution of the dynamic system beforehand, which can then be solved using the Euler integrator that works with mean derivative functions. Also, notice that, since we are working with supervised learning with input/output training patterns, this is not absurd. This fact is easy to understand, as it is necessary to know the references ${}^{k}y_j^i$ and ${}^{k+1}y_j^i$ to accurately estimate the mean derivative $tan_{\Delta t}{}^{k}\alpha_j^i$, between these two points, through a universal approximator of functions with exclusively feed-forward architecture.

Thus, if the solution of the non-linear dynamical system that is wanted to estimate, then it is necessary to treat this problem as n independent equations with only one time variable t each (black-box approach), rather than n vector coupled equations (white-box approach). This last statement can be justified with the help of Principle 1, which is proved, by reduction to absurdity, right after the proof of Theorem 3.

Therefore, if the solution of the considered dynamic system is previously known and given by $y_j^i(t) = g_j^i(t)$ for $j = 1, 2, \cdots, n$ and $i = 1, 2, \cdots, \infty$ then the function $\dot{y}_j^i(t) = \dot{g}_j^i(t)$ are easily obtained by direct differentiation (numerically or analytically). In this way, the original dynamic system $\dot{y}_j^i = f_j(y_1^i, y_2^i, \cdots, y_n^i) = f_j(y^i)$ can be replaced by $\dot{y}_j^i(t) = \dot{g}_j^i(t)$ for $j = 1.2, \cdots, n$ and $i = 1.2, \cdots, \infty$, for a time interval confined in $[t_o, t_f]$ with the help of Principle 1.

However, it should be noted that the math function $\dot{g}_j^i(t)$, in general, has nothing similar to the math function $f_j(y^i)$. Note also that this simplification makes it possible to reduce the proposed proof to a differential calculus of only one variable. Thus, it allows the use of the differential and integral mean value theorems. Note that $\dot{g}_j^i(t)$ must also have the over-index i, as this function must necessarily also depend on the initial condition as well as its equivalent instantaneous derivative function $f_j(y^i)$.

 doi:10.20944/preprints202508.0719.v1

Theorem 3: The discrete and exact general solution for the autonomous system of non-linear ordinary differential equations of the type $\dot{y}^i = f(y^i)$ can be established through the first-order Euler relation of the type ${}^{k+1}y_j^i = tan_{\Delta t}{}^k\alpha_j^i \cdot \Delta t + {}^ky_j^i$, for ${}^ky_j^i$ and Δt fixed; since that the general solution of this dynamical system, given by, $y_j^i(t) = g_j^i(t)$ and $\dot{y}_j^i(t) = \dot{g}_j^i(t)$ are, previously, known for $j = 1, 2, \cdots, n$; $i = 1, 2, \cdots, \infty$ and $t \, \epsilon \, [t_o, t_f]$. Furthermore, the solutions $g_j^i(t)$ for $j = 1, 2, \cdots, n$ must all be continuous and differentiable. However, note that $\dot{g}_j^i(t)$ for $j = 1, 2, \cdots, n$ is suffice to be continuous.

Proof: Let the autonomous non-linear dynamical system of first-order be given by $\dot{y}_j^i(t) = f_j(y_1^i, y_2^i, \cdots, y_n^i) = f_j(y^i)$ for $j = 1, 2, \cdots, n$ and $i = 1, 2, \cdots, \infty$. If the dynamical system solution is known and equals to $y_j^i(t) = g_j^i(t)$ then the function $f_j(y^i)$ can be replaced by $\dot{g}_j^i(t) = h_j^i(t)$, that is, $\dot{y}_j^i(t) = h_j^i(t)$. In this way, we can write that $\int_{k_{y_j^i}}^{k+1_{y_j^i}} dy_j^i(t) = \int_{t_k}^{t_{k+1}} h_j^i(t)dt$. Note that the indices i, j, and k uniquely map the secant that interests us according to Figure 3. This last integral can still be simplified as ${}^{k+1}y_j^i - {}^ky_j^i = \int_{t_k}^{t_{k+1}} h_j^i(t)dt$ as a result of the fundamental theorem of calculus. So, applying the integral mean value theorem, in this last expression, we get ${}^{k+1}y_j^i - {}^ky_i^j = h_j^i(t_k^x) \cdot \Delta t = \dot{g}_j^i(t_k^x) \cdot \Delta t$ where $t_k^x \, \epsilon \, [t_k, t_{k+1}]$ and for $h_j^i(t) = \dot{g}_j^i(t)$. On the other hand, applying the differential mean value theorem on the function $y_j^i(t) = g_j^i(t)$ for $j = 1, 2, \cdots, n$ in the interval $[t_k, t_{k+1}]$ we get that $\dot{g}_j^i(t_k^*) = \frac{{}^{k+1}g_j^i - {}^kg_j^i}{\Delta t} \overset{\text{def}}{=} tan_{\Delta t}{}^k\alpha_j^i$. So, by Property 1 we have that $\dot{g}_j^i(t_k^*) = \dot{g}_j^i(t_k^x) = tan_{\Delta t}{}^k\alpha_j^i$ to $j = 1, 2, \cdots, n$. So, ${}^{k+1}y_j^i = tan_{\Delta t}{}^k\alpha_i^j \cdot \Delta t + {}^ky_j^i$ to $j = 1, 2, \cdots, n$ or, in vector form, it turns out that ${}^{k+1}y^i = tan_{\Delta t}{}^k\alpha^i \cdot \Delta t + {}^ky^i$.□

The reason that the functions $g_j^i(t)$ for $j = 1, 2, \ldots, n$ are all continuous and differentiable is that, in the proof of Theorem 3, we used the differential mean value theorem over $g_j^i(t)$ and this theorem requires these two conditions to be applied. The reason that the functions $\dot{g}_j^i(t)$ for $j = 1, 2, \ldots, n$ are all continuous is that, in the proof of the same Theorem 3, the integral mean value theorem was applied over the function $\dot{g}_j^i(t)$ and this theorem requires this condition. Note that for the application of the integral mean value theorem, the function $\dot{g}_j^i$ is not required to be differentiable.

Principle 1: Given the non-linear autonomous dynamic system $\dot{y}_j^i = f_j(y_1^i, y_2^i, \cdots, y_n^i) = f_j(y^i)$ and their respective general solutions $y_j^i(t) = g_j^i(t)$ for $j = 1, 2, \cdots, n$ and $i = 1, 2, \cdots, \infty$ then, the autonomous instantaneous derivative functions $f_j(y^i)$ can be replaced by the exclusively non-autonomous instantaneous derivative functions given by $\dot{g}_j^i(t)$, that is, $f_j(y_1^i, y_2^i, \cdots, y_n^i) = f_j(y^i) = \dot{g}_j^i(t) = h_j^i(t)$ for $j = 1, 2, \ldots, n$ and $i = 1, 2, \cdots, \infty$.

Proof: Assume, for absurdity, that the referred principle is false. If this is true, then there would be no compromise of veracity between the computational data acquisition system (through sensors) and the real universe, and thus the black-box approach would not work; but this is absurd. So, by exclusion, the principle is true .□

Notice that when a computational data acquisition system is performed, for example, from a particular real-world plant, something interesting happens. Thus, whoever maintains the consistency of the acquired data, through several sensors acting simultaneously and independently of each other, is the omnipresent and immutable manifestation of the natural physical laws, which govern the proper functioning of the universe (e.g., the law of universal gravitation, conservation of energy law, Faraday's law, among others). In this case, little is allowed to human wisdom and technology.

Finally, the E-TUNI universal numerical integrator is suitable for solving only ordinary differential equations. However, it should be noted that the current use of neural networks in solving Partial Differential Equations (PDEs) is quite extensive [27–35]. Additionally, [36] explains how to use the Runge-Kutta numerical integrator to solve partial differential equations (mainly hyperbolic ones). Therefore, using the E-TUNI to solve partial differential equations may be a good option for future work and should be investigated more carefully.

 doi:10.20944/preprints202508.0719.v1

3.3. Mathematical Relationship Between Mean and Instantaneous Derivatives

A very remarkable fact is that it is possible to obtain an equation that relates the instantaneous derivative functions to the mean derivative functions. This equation can be easily obtained using the chain rule for functions on several independent variables [25,26]. In this way, two previously trained neural networks are known to represent the instantaneous and mean derivative functions, respectively, by:

$$\left[\dot{y}_1(t), \quad \dot{y}_2(t), \quad \cdots, \quad \dot{y}_{n_y}(t)\right]^T = \tag{25}$$

$$\left[tan^{\,k}\theta_1^i, \quad tan^{\,k}\theta_2^i, \quad \cdots, \quad tan^{\,k}\theta_{n_y}^i\right]^T =$$

$$f_{NN}^{id}\left[y_1(t), \quad y_2(t), \quad \cdots, \quad y_{n_y}(t), \quad \hat{w}\right]^T$$

and

$$\left[tan_{\Delta t}{}^{k}\alpha_1^i, \quad tan_{\Delta t}{}^{k}\alpha_2^i, \quad \cdots, \quad tan_{\Delta t}{}^{k}\alpha_{n_y}^i\right]^T = \tag{26}$$

$$f_{NN}^{md}\left[y_1(t), \quad y_2(t), \quad \cdots, \quad y_{n_y}(t), \quad \hat{w}\right]^T$$

For reasons of simplification, we do not consider the case using control variables in the equations (25) and (26). So, if we use the chain rule [25,26] in the equation (26) we get:

$$\frac{d}{dt}tan_{\Delta t}{}^{k}\alpha_j^i = \tag{27}$$

$$\frac{\partial\, tan_{\Delta t}{}^{k}\alpha_j^i}{\partial\, {}^{k}y_1^i}\cdot\frac{d\,{}^{k}y_1^i}{dt}+\cdots+\frac{\partial\, tan_{\Delta t}{}^{k}\alpha_j^i}{\partial\, {}^{k}y_{n_y}^i}\cdot\frac{d\,{}^{k}y_{n_y}^i}{dt} =$$

$$\left[\frac{\partial\, tan_{\Delta t}{}^{k}\alpha_j^i}{\partial\, {}^{k}y_1^i}, \quad \cdots, \quad \frac{\partial\, tan_{\Delta t}{}^{k}\alpha_j^i}{\partial\, {}^{k}y_{n_y}^i}\right]\cdot\begin{bmatrix}\frac{d\,{}^{k}y_1^i}{dt}\\ \vdots \\ \frac{d\,{}^{k}y_{n_y}^i}{dt}\end{bmatrix} =$$

$$\left[\frac{\partial\, tan_{\Delta t}{}^{k}\alpha_j^i}{\partial\, {}^{k}y_1^i}, \quad \cdots, \quad \frac{\partial\, tan_{\Delta t}{}^{k}\alpha_j^i}{\partial\, {}^{k}y_{n_y}^i}\right]\cdot\begin{bmatrix}tan^{\,k}\theta_1^i\\ \vdots \\ tan^{\,k}\theta_{n_y}^i\end{bmatrix} =$$

$$\frac{\partial\, tan_{\Delta t}{}^{k}\alpha_j^i}{\partial\, {}^{k}y^i}\circ tan^{\,k}\theta^i \;\; to \;\; j = 1, 2, \cdots, n_y$$

For the sake of simplicity, we can represent the expression (27) as follows:

$$tan_{\Delta t}^{k}\Psi_j^i = \frac{d}{dt}\,tan_{\Delta t}^{k}\alpha_j^i = \frac{\partial\, tan_{\Delta t}{}^{k}\alpha_j^i}{\partial\, {}^{k}y^i}\circ tan^{\,k}\theta^i \tag{28}$$

$$to \;\; j = 1, 2, \cdots, n_y$$

The operator $\circ$ is just the usual dot product applied over a Euclidean space. So, just for completeness, the vector form of the equation (28) can also be expressed by:

$$\frac{d}{dt}\,tan_{\Delta t}^{k}\alpha^i = \frac{\partial\, tan_{\Delta t}{}^{k}\alpha^i}{\partial\, {}^{k}y^i}\cdot tan^{\,k}\theta^i \tag{29}$$

 doi:10.20944/preprints202508.0719.v1

where,

$$\frac{d}{dt}\, tan^{k}_{\Delta t}\alpha^{i} = \begin{Bmatrix} \frac{d}{dt}\, tan^{k}_{\Delta t}\alpha^{i}_{1} \\ \frac{d}{dt}\, tan^{k}_{\Delta t}\alpha^{i}_{2} \\ \vdots \\ \frac{d}{dt}\, tan^{k}_{\Delta t}\alpha^{i}_{n_y} \end{Bmatrix} \tag{30}$$

$$\frac{\partial\, tan^{k}_{\Delta t}\alpha^{i}}{\partial\, {}^{k}y^{i}} = \begin{bmatrix} \frac{\partial\, tan^{k}_{\Delta t}\alpha^{i}_{1}}{\partial\, {}^{k}y^{i}_{1}} & \cdots & \frac{\partial\, tan^{k}_{\Delta t}\alpha^{i}_{1}}{\partial\, {}^{k}y^{i}_{n_y}} \\ \frac{\partial\, tan^{k}_{\Delta t}\alpha^{i}_{2}}{\partial\, {}^{k}y^{i}_{1}} & \cdots & \frac{\partial\, tan^{k}_{\Delta t}\alpha^{i}_{2}}{\partial\, {}^{k}y^{i}_{n_y}} \\ \vdots & \ddots & \vdots \\ \frac{\partial\, tan^{k}_{\Delta t}\alpha^{i}_{n_y}}{\partial\, {}^{k}y^{i}_{1}} & \cdots & \frac{\partial\, tan^{k}_{\Delta t}\alpha^{i}_{n_y}}{\partial\, {}^{k}y^{i}_{n_y}} \end{bmatrix} \tag{31}$$

$$tan\,{}^{k}\theta^{i} = \begin{Bmatrix} tan\,{}^{k}\theta^{i}_{1} \\ tan\,{}^{k}\theta^{i}_{2} \\ \vdots \\ tan\,{}^{k}\theta^{i}_{n_y} \end{Bmatrix} \tag{32}$$

There is an essential fact between instantaneous and mean derivative functions. Using these two kinds of derivative functions, that is, $tan\,{}^{k}\theta^{i}$ and $tan_{\Delta t}\,{}^{k}\alpha^{i}$ it is possible to interpolate a parabola between the discrete interval $[t_k, t_{k+1}]$, which passes at least at two precise points of the exact solution of the considered non-linear dynamical system, in this same interval. Thus, Table 1 considers the points used to perform this parabolic interpolation, and the equations (34), (35), and (36) are the coefficients of the parabola (33).

$${}^{k}y^{i}_{j}(t) = \alpha_k \cdot t^2 + \beta_k \cdot t + \gamma_k \tag{33}$$

$$\alpha_k = \frac{1}{2} \cdot tan_{\Delta t}\,{}^{k}\Psi^{i}_{j} \tag{34}$$

$$\beta_k = tan_{\Delta t}\,{}^{k}\alpha^{i}_{j} - \frac{1}{2} \cdot (t_{k+1} + t_k) \cdot tan_{\Delta t}\,{}^{k}\Psi^{i}_{j} \tag{35}$$

$$\gamma_k = \frac{1}{2} \cdot (t_k \cdot t_{k+1} \cdot tan_{\Delta t}\,{}^{k}\Psi^{i}_{j} - 2 \cdot t_k \cdot tan_{\Delta t}\,{}^{k}\alpha^{i}_{j} + 2 \cdot {}^{k}y^{i}_{j}) \tag{36}$$

where, $t \,\epsilon\, [t_k, t_{k+1}]$.

Table 1. Coordinate points within the interval $[t_k, t_{k+1}]$.

n	Time	$y(t),\ \dot{y}(t)\ and\ \ddot{y}(t)$	Determining Form
1	t_k	${}^{k}y^{i}_{j}$	Initial Instant
2	t_{k+1}	${}^{k+1}y^{i}_{j}$	Given by E-TUNI
3	t^{x}_{k}	$\dot{y}^{i}_{j}(t^{x}_{k}) = tan_{\Delta t}\,{}^{k}\alpha^{i}_{j}$	Output of net
4	t^{x}_{k}	$\ddot{y}^{i}_{j}(t^{x}_{k}) = tan_{\Delta t}\,{}^{k}\Psi^{i}_{j}$	From Equations (28) or (29)

4. Results and Analysis

In this article, we perform a complete computational numerical analysis, studying the training of the neural integration structures discussed in the previous sections, i.e., the E-TUNI. Thus, for all the experiments that are presented below, it was standardized to use MLP neural networks with the traditional back-propagation [4] and Kalman filter [37–39] training algorithms in Example 2; and the Levenberg–Marquardt [40] training algorithm in Example 3. All experiments were trained using only one inner layer in the MLP network.

Example 1. Check the validity of ***Principle 1*** for the dynamical system $\dot{y} = f(y) = y^2$.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 13 August 2025 doi:10.20944/preprints202508.0719.v1

Proof: from $\dot{y} = \frac{dy}{dt} = y^2$ then, $\int_{y(t_0)}^{y(t)} y^{-2} dy = \int_{t_o}^{t} dt$, as a result of the fundamental theorem of calculus. Solving this integral, analytically, results in $y(t) = \frac{y(t_0)}{-y(t_0)t+y(t_0)t_0+1}$. Differentiating the function $y(t)$, with respect to time t, we have that $\dot{y} = \dot{g}(t) = y(t_0)\frac{d}{dt}[-y(t_0)t + y(t_0)t_0 + 1]^{-1} = \frac{y^2(t_0)}{[-y(t_0)t+y(t_0)t_0+1]^2}$. Thus, replacing $y(t)$ in $\dot{y} = y^2$ results in $\dot{y} = f(y) = y^2 = \left[\frac{y(t_0)}{-y(t_0)t+y(t_0)t_0+1}\right]^2 = \dot{g}(t)$.

The graph in Figure 4 illustrates the drawing of the function $y(t)$ (upper part) and the drawing of the function $\dot{y}(t)$ (lower part). For a proper understanding of the graphs in Figure 4, it should be noted that $t_0 = 0$ and there are seven curves in the graphs of Figure 4 representing the solution family of Example 1. For example, in $y(t_0) = y(0) = 1$ part a particular solution of the dynamical system $\dot{y} = y^2$ that have a vertical asymptote at $t = 1$ (upper part of Figure 4). Note that to the left of this asymptote, the solution of the dynamical system has been plotted in blue. On the other hand, to the right of this asymptote, the solution has been plotted in red. This convention is followed to plot the remaining solutions that appear in Figure 4. It is also important to note that the blue curves have distinct vertical asymptotes for each of the initial conditions. On the other hand, all red curves have horizontal asymptotes at $y = 0$, when t tends to infinity.

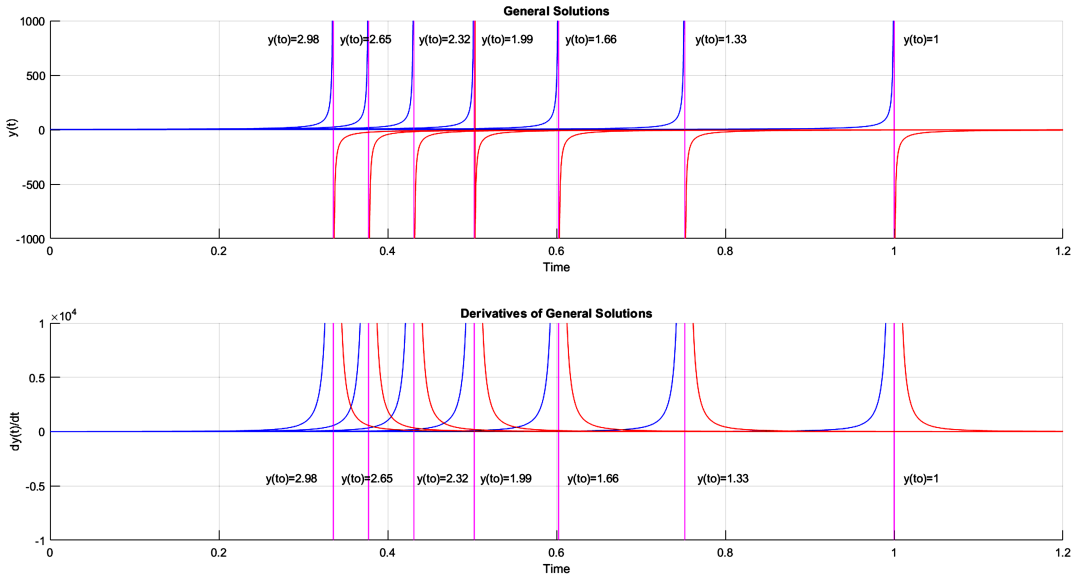

Figure 4. Analytical solution of the dynamical system presented in example 1.

Note that Principle 1 is valid for discontinuous solutions in $y(t)$. However, the use of the general expression of E-TUNI to learn this discontinuous solution is not possible according to the theory presented in this article. The reasons for this impossibility are as follows: (i) the integral and differential mean value theorems require that the E-TUNI solution be continuous and differentiable, and (ii) E-TUNI would have to learn a numerical value equal to infinity, at the points of the vertical asymptotes shown in Figure 4 (upper part), and this is impossible, as conventional computers have finite memory. For the use of E-TUNI, in examples like this, further studies are required.

Example 2. Solve the predictive control problem, necessarily using E-TUNI as a model of the non-linear plant, for the system of non-linear ordinary differential equations, which describes the Earth/Mars orbit transfer dynamics for a rocket of mass m, as shown in Figure 5. Solve this problem for two different horizons, that is, $m = 1$ and $m = 15$, according to equations (10) to (15). The set of ordinary differential equations describing this problem is given below [30]:

 doi:10.20944/preprints202508.0719.v1

$$\begin{aligned} \dot{m} &= -0.0749 \\ \dot{r} &= w \\ \dot{w} &= \frac{v^2}{r} - \frac{\mu}{r^2} + \frac{T \cdot sin\theta}{m} \\ \dot{v} &= \frac{-w \cdot v}{r} + \frac{T \cdot cos\theta}{m} \end{aligned} \tag{37}$$

An illustrative scheme of this dynamic system can be seen in Figure 5. So, the state variables for this predictive control problem are m (rocket mass), r (orbit radius), w (radial velocity), and v (transverse velocity). The only controlling variable is the thrust angle θ of the rocket. The normalized constants of this problem are $\mu = 1.0$ (gravitational constant), $T = 0.1405$ (rocket thrust), $t_o = 0$ (initial instant), and $t_f = 3.3$ (final instant). In this problem, each unit of time equals 58.2 days.

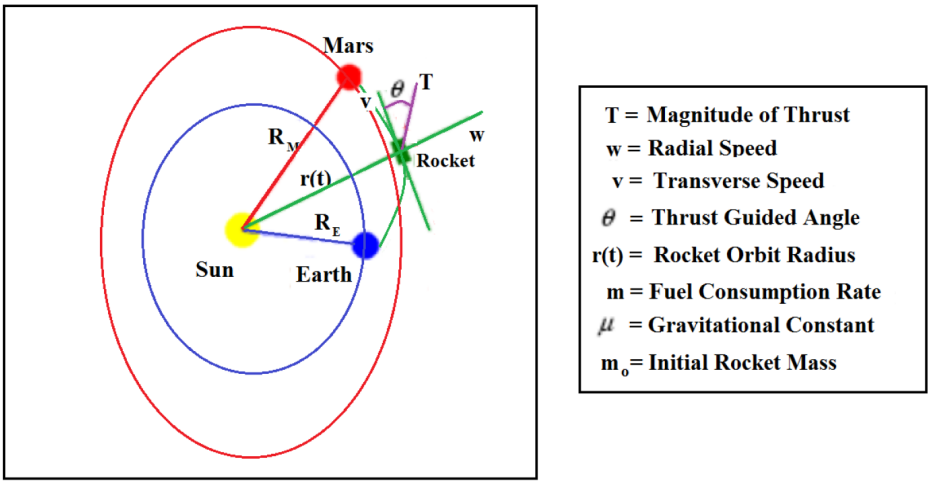

Figure 5. Graphic scheme of the dynamical system associated with example 2 (Sources: [16,20,24,30].

We trained the E-TUNI using an MLP network with only one inner layer. This inner layer contained 41 neurons and one bias neuron. We applied the hyperbolic tangent activation function in the inner layer, while we used a purely linear activation function at the network output. The training and testing yielded mean square errors (MSE) of $9,021x10^{-6}$ and $1,067x10^{-5}$, respectively.

In the neural training of the E-TUNI structure, two standard training algorithms were used: the classic back-propagation [4] and the Kalman filter with recursive and parallel processing [39]. They have been casually changed. In addition, this training took a considerable amount of time, as a shallow architecture network was used instead of a deep architecture network.

Figure 6 presents the trajectory traced by the Kalman filter, for a predictive control structure whose plant model was obtained through E-TUNI. The solid line is the reference trajectory, and the dotted line is the trajectory tracked by the Kalman filter. Figure 7 presents the control estimated by the Kalman filter in a predictive control structure designed according to equations (10) to (15) and associated with the trajectory traced in Figure 6. However, the Kalman filter equations employed were taken from [16]. The integration step used was $\Delta t = 0.1$ and the horizon equal to $m = 1$. As can be seen, the estimated control was not very good for just using one horizon in equation (10).

On the other hand, Figures 8 and 9 present the same E-TUNI structure as in the previous case, but now in a new control estimating with the horizon used equally to $m = 20$. Note that this small change significantly improved the result obtained. However, while the training time for a horizon of $m = 1$ may only take between an hour or two, the training time for a horizon of $m = 20$ can be as long as ten days, if the integration step is changed from $\Delta t = 0.1$ to $\Delta t = 0.01$. For a better understanding of E-TUNI coupled to a predictive control structure see reference [16].

 doi:10.20944/preprints202508.0719.v1

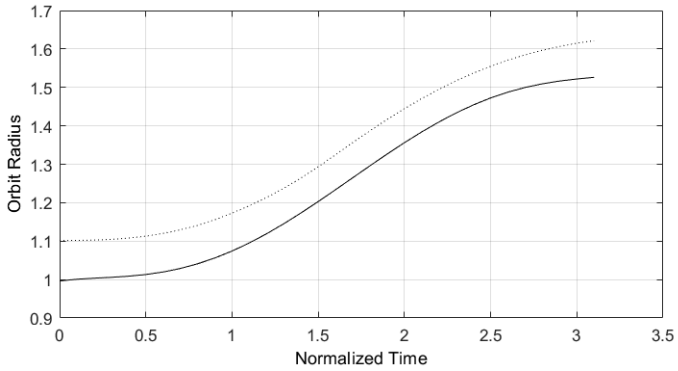

Figure 6. Orbit radius trajectory for $m = 1$ and $\Delta t = 0.1$.

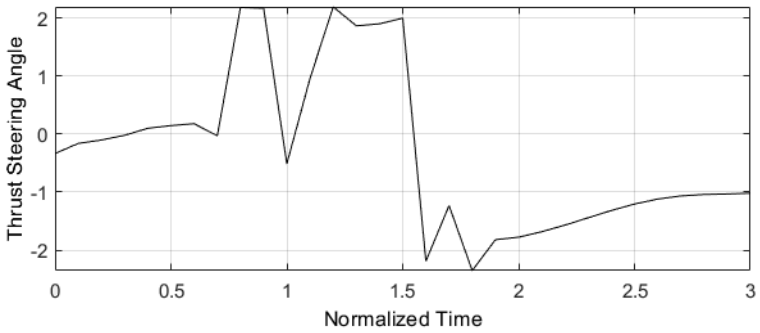

Figure 7. Rocket angle thrust estimated by predictive control structure designed with E-TUNI and referring to Figure 6.

Figures 6–9 are an original extension of what can be found in [16]. In [16], the predictive control structure with E-TUNI for reference trajectories displaced from the initial condition was not explored, as presented here. Similar behavior for the other state variables m, w, and v was observed. Therefore, we have omitted the other graphics due to space constraints.

Figures 10 and 11 were taken from [16]. They represent the application of a predictive control structure on the E-TUNI, for the case where a deviation between the reference trajectories and the initial condition of the dynamic system in question was not imposed. However, note that an integration step of $\Delta t = 0.01$ and a horizon of $m = 10$ was used. Note also that the 10-fold reduction in the integration step, although it makes the algorithm much slower, manages to considerably refine the control policy obtained in Figure 11, compared to Figures 7 and 9.

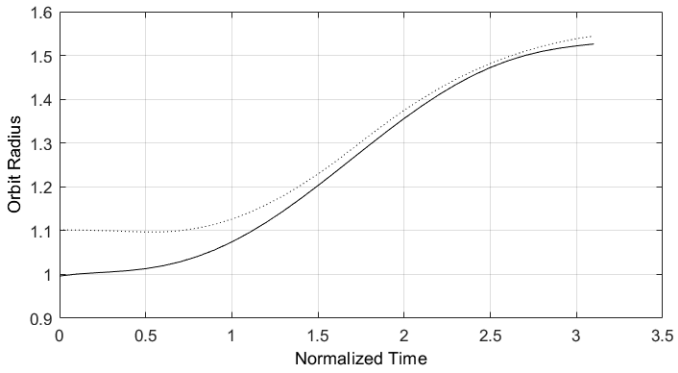

Figure 8. Orbit radius trajectory for $m = 20$ and $\Delta t = 0.1$.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 13 August 2025 doi:10.20944/preprints202508.0719.v1

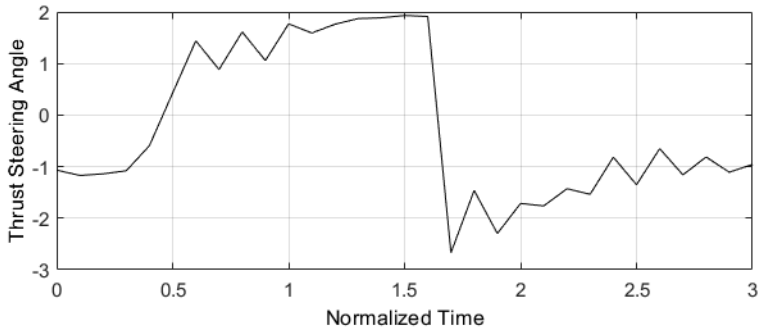

Figure 9. Rocket angle thrust estimated by predictive control structure designed with E-TUNI and referring to Figure 8.

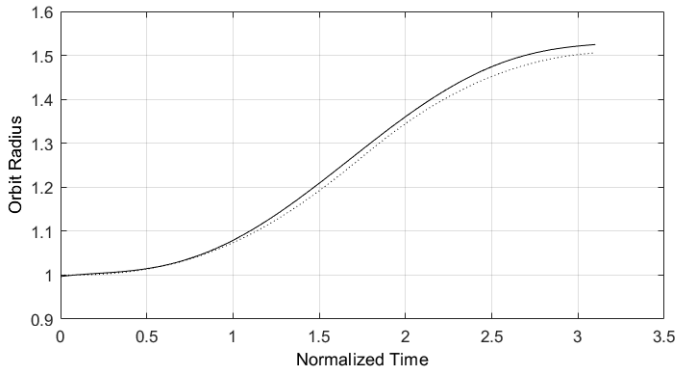

Figure 10. Orbit radius trajectory for $m = 10$ and $\Delta t = 0.01$ (Source: see [16]).

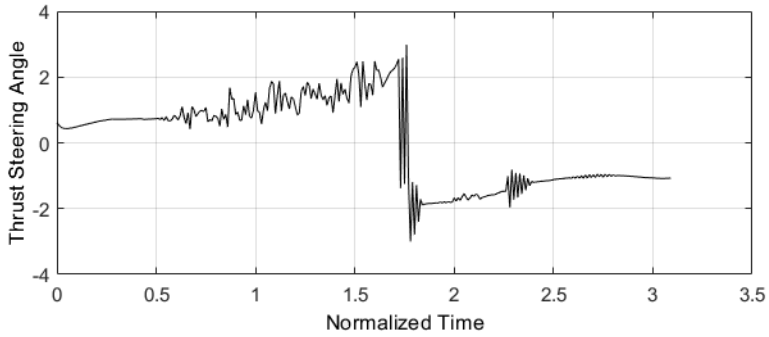

Figure 11. Rocket angle thrust estimated by predictive control structure designed with E-TUNI and referring to Figure 10 (Source: see [16]).

Example 3. The non-linear simple pendulum is a second-order autonomous system given by $l \cdot \ddot{\theta} + g \cdot sin\theta = 0$. Note that l and g are, respectively, the length of the pendulum and the local acceleration due to gravity. Here, we solve this problem by using E-TUNI for several different integration steps. Furthermore, for each of these integration steps, we obtain the interpolating parabolas governed by the equations from (33) to (36).

To precisely understand the resolution of Example 3, we follow the following basic algorithm:

Step 1. Generate the E-TUNI input/output training patterns through the Runge-Kutta 4-5 integrator, applied to the non-linear simple pendulum equation. Note that, as this dynamical system is a non-linear equation, its solution in the phase plane is not a perfect circle.

Step 2. Use the Levenberg-Marquardt algorithm in the direct approach to training two distinct neural networks, namely: (i) a neural network to learn the instantaneous derivative functions and (ii) another neural network to learn the mean derivative functions.

Step 3. Determine the vector $\frac{d}{dt} tan_{\Delta t}{}^{k}\Psi^{i}$ using the equations from (29) to (32). Just pay attention to the fact that the partial derivatives $\frac{\partial}{\partial^{k} y^{i}} tan_{\Delta t}{}^{k}\alpha^{i}$ were obtained numerically and not analytically. For this, a $\Delta^{k} y^{i} = 10^{-6}$ was used.

 doi:10.20944/preprints202508.0719.v1

Step 4. Determine the interpolating parabolas within a horizon of interest $[t_0, t_f]$ and with a step of Δt. For this, the equations from (33) to (36) were used recursively.
Step 5. Compare the results obtained for different Δt integration steps. Also, compare the solution of the interpolating parabolas with the mean derivative equations.
Step 6. A summary of this algorithm is presented in Figures 12 and 13.

To solve this problem appropriately, all graphical simulation results will be displayed using an MLP neural network with only one inner layer (shallow network). In the inner layer, the hyperbolic tangent sigmoid activation function was used, and in the output of the neural network, a purely linear function was used. The training algorithm used was the traditional Levenberg-Marquardt [40], always using the direct approach of E-TUNI and not the indirect approach. For that, the MATLAB neural networks toolbox was used.

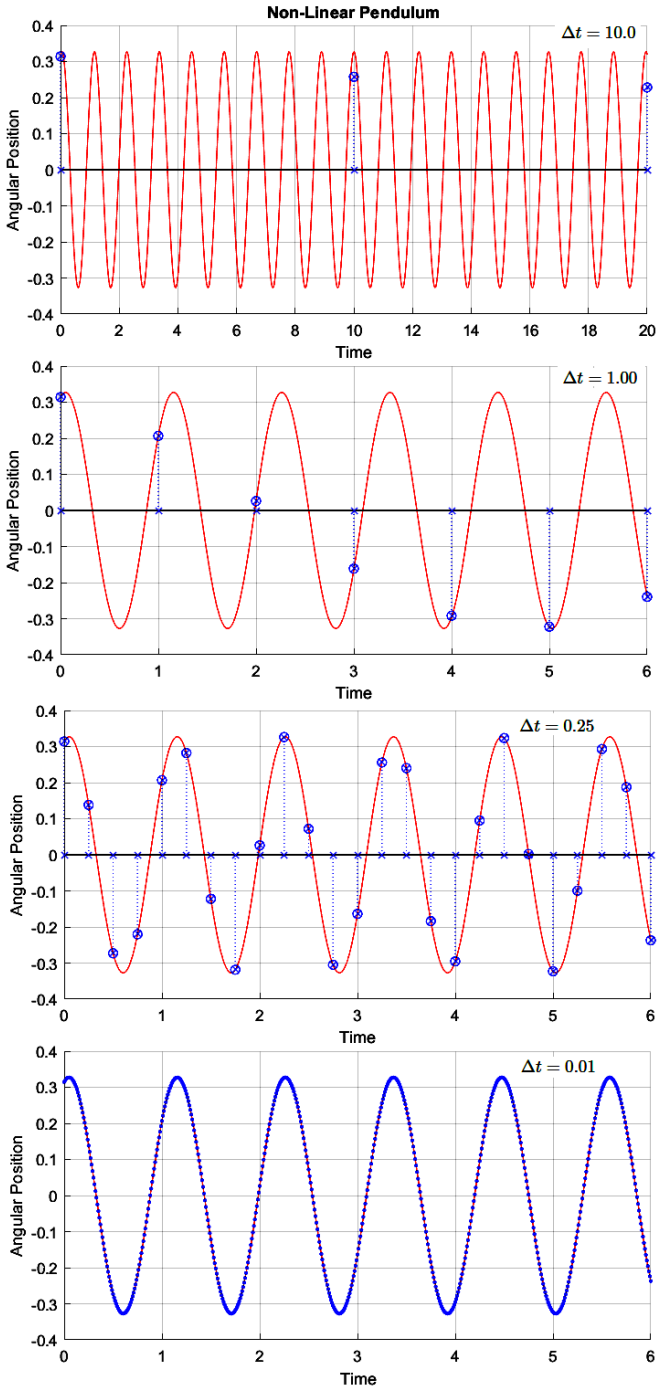

Figure 12. The E-TUNI was designed with several distinct integration steps.

 doi:10.20944/preprints202508.0719.v1

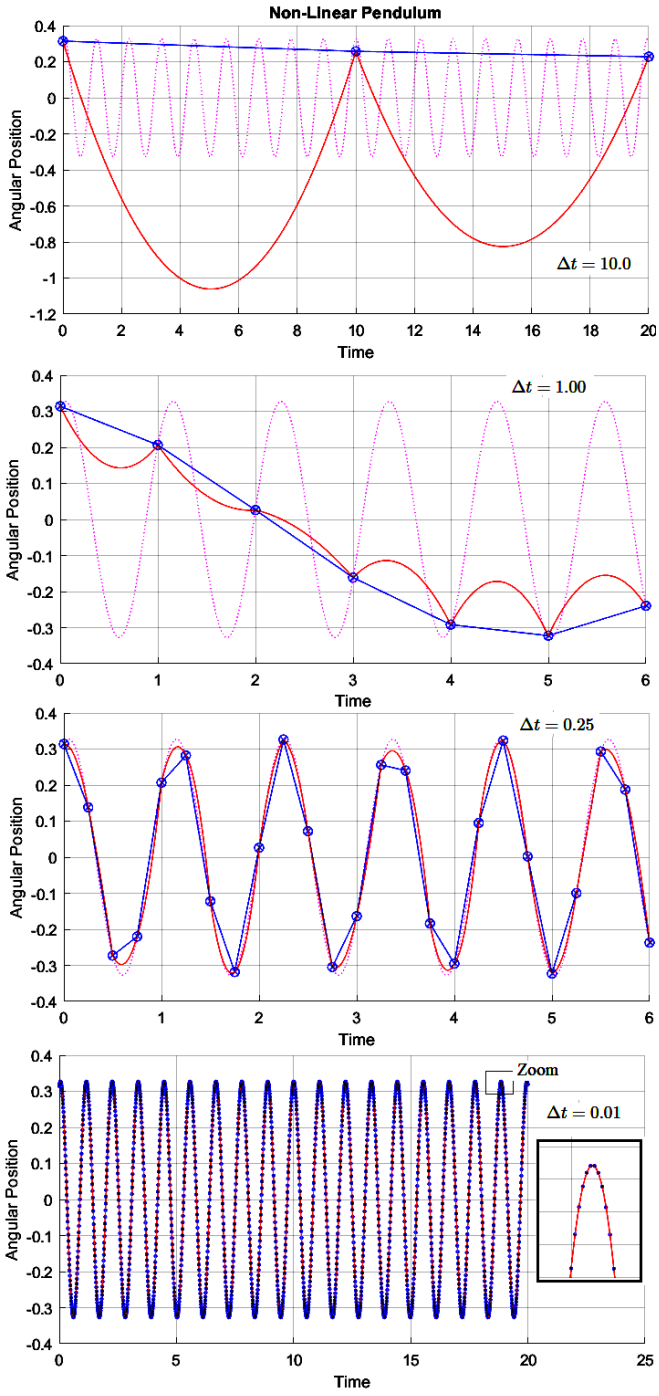

Figure 13. The E-TUNI equivalent to the previous figure, but with the mean derivatives and interpolating parabolas present.

The dynamical system in question was trained within the finite domains $Dom^{\theta} = [-\pi/2, +\pi/2]$ *rad* and $Dom^{\dot{\theta}} = [-2, +2]\ \frac{rad}{s}$. For all neural networks trained, 400 training patterns were used. 75% of them were used for training, 15% for validation, and 15% for testing. All training has been standardized to be trained with exactly 10,000 epochs. Table 2 summarizes the main results achieved in these training sessions.

Analyzing Table 2 it is observed that the greater the integration step Δt, used in the E-TUNI training, the greater the training error for the validation patterns. This fact experimentally confirms (9), which states that the integration step Δt amplifies the Mean Squared Error (MSE) of E-TUNI training when it is greater than 1.00.

 doi:10.20944/preprints202508.0719.v1

However, the most interesting numerical results, concerning Example 3, are shown in Figures 12 and 13. In Figure 12, the integration steps are decreased from top to bottom. Notice that when the integration step Δt is equal to 10.0, E-TUNI could not accurately learn the angular position prediction θ (see all blue points in Figure 12). With the integration steps Δt equal to or less than 1.00, E-TUNI was able to learn them perfectly. The reason for this is that several E-TUNI training was carried out for the integration step Δt equal to 10.0, but the best result achieved was the one presented in Table 2, that is, $mse = 1.7520 \times 10^{-3}$, confirming, again, the validity of the equation (9), as a limiting factor for this method. The E-TUNI training, which involved integration steps Δt equal to or less than 1.00, was easily learned by the MLP neural networks in the first attempts.

Table 2. Summary of training performed for example 3.

Δt	MSE of Validation Patterns	Direct Approach
-	2.5624×10^{-11}	Instantaneous Derivatives
0.01	2.4351×10^{-10}	Mean Derivatives
0.25	3.5268×10^{-8}	Mean Derivatives
1.00	2.1592×10^{-6}	Mean Derivatives
10.0	1.7520×10^{-3}	Mean Derivatives

Figure 13 also presents several simulation graphs with the integration steps Δt, used in the E-TUNI training, also decreasing from top to bottom. The dotted pink lines represent the true values of the dynamical system in question. The solid lines in blue represent the mean derivatives between two consecutive points. Continuous lines in red, interpolating parabolas between two successive points. Note that, for tiny Δt integration steps, the parabolic interpolation using mean and instantaneous derivatives functions, simultaneously, is quite accurate and relatively better than using E-TUNI alone.

Analyzing Figure 13 more closely, observe that the E-TUNI equation is equivalent to consecutive uniform rectilinear movements and with constant velocities equal to the mean derivatives (constant from interval to interval). On the other hand, the equations of the interpolating parabolas are consecutive equations of uniformly varied motions and with consecutive accelerations $\tan_{\Delta t}\ {}^{k}\Psi_{j}^{i}$ (also constant interval to interval).

Note also that uniformly varied motions (interpolating parabolas) allow at most one change of direction in their motion. However, uniform rectilinear motion (E-TUNI) does not. For this reason, it appears that interpolating parabolas are more accurate than mean derivative curves for tiny integration steps. Because neural networks are universal approximators of functions [2,5,6], the physical variables that can be used as input to E-TUNI can be any.

Additionally, the Euler integrator, designed exclusively with instantaneous derivative functions, is never able to achieve the equivalent precision of the Euler integrator, designed with mean derivative functions. This fact happens mainly if substantial integration steps are used.

Finally, it is only possible to use control variables (in the inputs of E-TUNI) to obtain the parabolic interpolations if the control variable remains with its constant value throughout the entire interval $[t_k, t_{k+1}]$. This property is only practical for control if the E-TUNI integration step is tiny and close to zero.

5. Conclusions

In this paper, a detailed mathematical development of the Euler-Type Universal Numerical Integrator (E-TUNI) was presented. Its main mathematical properties were presented, with a direct and quite objective development. A correct demonstration of the general expression of E-TUNI was also presented, since in previous articles, this demonstration was presented in the wrong way. Three practical case studies using E-TUNI were presented, demonstrating the computational versatility of E-TUNI.

The E-TUNI is an alternative methodology to the traditional NARMAX methodology. The E-TUNI can also be an alternative to the conventional Runge-Kutta Neural Network (RKNN) and

 doi:10.20944/preprints202508.0719.v1

Adams-Bashforth Neural Network (ABNN). However, it is essential to note that the NARMAX and mean derivatives (E-TUNI) methodologies are fixed-step, whereas the instantaneous derivatives methodology (RKNN, ABNN, among others) is variable-step, when both are used in the simulation phase. In the training phase, all methods are fixed-step.

To understand how the Euler-Type Universal Numerical Integrator (E-TUNI) works, it is helpful to use Figure 1. This figure illustrates the elementary difference between mean derivative functions and instantaneous derivative functions. As everyone knows, the secant between two points on a curve is the mean derivative, and the tangent line to a point on a curve is the instantaneous derivative function.

Thus, if a Runge-Kutta Neural Network (RKNN) of order four is trained with a tiny training error, then in [20] it is demonstrated that the neural network converges to the instantaneous derivative functions. However, if now this neural network is decoupled from the Runge-Kutta integrator and re-coupled into an E-TUNI, then the output error in the latter integrator will increase because, as illustrated in Figure 1, an Euler integrator designed with an instantaneous derivative function generates a substantial error.

However, if neural training is restarted in E-TUNI, as the neural network is a universal function approximator, then it reduces the training error back to zero. As the solution of autonomous and non-linear dynamical systems is unique, there is only one way to correct this error. So, as Figure 1 illustrates, it is done by making the instantaneous derivative functions converge to the mean derivative functions in the neural architecture. However, as the secant between two points varies with the variation of the horizontal distance between these two points, the E-TUNI works with a fixed integration step. If it is desired to change this step, then new neural training have to be done. Note that the high-order Runge-Kutta Neural Network (RKNN) can vary the integration step within certain limits since the instantaneous derivative function does not depend on the integration step.

Principle 1 can be applied to both continuous and discontinuous solutions of real-world dynamical systems governed by autonomous non-linear ordinary differential equations. However, the general E-TUNI expression using mean derivative functions only applies to continuous solutions, as demonstrated earlier in this article. The integral and differential mean value theorems impose this limitation. A demonstration of the general expression of E-TUNI, for discontinuous solutions, requires further studies.

Note that Principle 1 can provide a rather interesting theoretical framework for explaining why the black-box approach works well in practice. Note also that this principle establishes a precise relationship of equivalence between $\dot{y}_j^i = f_j(y^i)$ (metaphysical property of the universe) and $\dot{y}_j^i = g_j^i(t)$ (the physical property of the universe). Prior knowledge of $\dot{y}_j^i = f_j(y^i)$ is metaphysical, as these equations can be mentally abstracted through mathematics and the laws of physics only. Prior knowledge of $\dot{y}_j^i = f_j(y^i)$ does not require carrying out any laboratory experiments or numerical calculations. On the other hand, knowledge of $\dot{y}_j^i = g_j^i(t)$, in general, requires carrying out a laboratory experiment or analytical (or numerical) resolution of $\dot{y}_j^i = f_j(y^i)$.

Finally, the software used to test the E-TUNI was developed using only shallow neural networks. Furthermore, the predictive control part could not be availed from the Artificial Neural Networks Toolbox of MATLAB, as it required the implementation of the original equations from (10) to (15). In this way, the software was relatively slow in terms of processing time. Thus, modern techniques and algorithms involving Deep Neural Networks (DNNs) or Convolutional Neural Networks (CNNs) can significantly improve the training time and simulation time of E-TUNI in the future.

Author Contributions: P.M.T. designed the proposed model and wrote the main part of the mathematical model. J.C.M., L.A.V.D. and A.M.d.C. supervised the writing of the paper and its technical and grammatical review. P.M.T. and G.S.G. (supervised by J.C.M., L.A.V.D. and A.M.d.C.) developed the software and performed several computer simulations to validate the proposed model. All authors have read and agreed to the published version of the manuscript.

 doi:10.20944/preprints202508.0719.v1

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The original contributions presented in the study are included in the article, further inquiries can be directed to the corresponding author.

Acknowledgments: I would like to thank my great friend Atair Rios Neto for his valuable tips for improving this article. Finally, I would also like to thank the valuable improvement tips given by the good reviewers of this journal. The authors of this article would also like to thank God for making all of this possible.

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

Abbreviations

The following abbreviations are used in this manuscript:

ABNN	Adams-Bashforth Neural Network
CNN	Convolutional Neural Network
DNN	Deep Neural Networks
E-TUNI	Euler-Type Universal Numerical Integrator
MLP	Multi-Layer Perceptron
MSE	Mean Squared Error
NARMAX	Nonlinear Auto Regressive Moving Average with eXogenous input
PCNN	Predictive-Corrector Neural Network
RBF	Radial Basis Function
RKNN	Runge-Kutta Neural Network
SVM	Support Vector Machine
UNI	Universal Numerical Integrator

References

1. McCulloch, W.; Pitts,W. A logical calculus of the ideas immanent in nervous activity. *Bull. Math. Biophys.* **1943**, *5(6)*, 115–133. https://doi.org/10.1007/BF02478259.
2. Kolmogorov, A. N. On the representation of continuous functions of many variables by superposition of continuous functions of one variable and addition. *Doklady Akademii Nauk SSR* **1957**, *114*, 953–956. https://doi.org/10.1090/trans2/028/04.
3. Charpentier, E.; Lesne, A.; Nikolski, N. *Kolmogorov's Heritage in Mathematics*, 1st ed.; Publisher: Spring Berlin, Heidelberg, and New York, USA, 2004.
4. Rumelhart, D. E.; Hinton, G. E.; Williams, R. J. Learning representations by back-propagating errors. *Nature* **1986**, *323*, 533–536. https://doi.org/10.1038/323533a0..
5. Cybenko, G. Approximation by superpositions of a sigmoidal function. *Math. Control Signals Syst.* **1989**, *2(4)*, 303–314. https://doi.org/10.1007/BF02551274.
6. Hornik, K.; Stinchcombe, M.; White, H. Multilayer feedforward networks are universal approximators. *Neural Networks* **1989**, *2(5)*, 359–366. https://doi.org/10.1016/0893-6080(89)90020-8.
7. Haykin, S. *Neural Networks: A Comprehensive Foundation*. Publisher: Prentice-Hall, Inc., New Jersey, USA, 1999.
8. Tasinaffo, P. M.; Gonçalves, G. S.; Cunha, A. M.; Dias, L. A. V. An introduction to universal numerical integrators. *Int. J. Innov. Comput. Inf. Control* **2019**, *15(1)*, 383–406. https://doi.org/10.24507/ijicic.15.01.383.
9. Henrici, P. *Elements of Numerical Analysis*. Publisher: John Wiley and Sons, New York, USA, 1964.
10. Vidyasagar, M. *Nonlinear Systems Analysis*. Publisher: Prentice-Hall, Inc., Electrical Engineering Series, New Jersey, USA, 1978. https://doi.org/10.1137/1.9780898719185.
11. Rama Rao, K. *A Review on Numerical Methods for Initial Value Problems*. Publisher: Internal Report, INPE-3011-RPI/088, São José dos Campos/SP, Brazil, 1984.
12. Chen, S.; Billings, S. A. Representations of nonlinear systems: the NARMAX model. *Int. J. Control* **1989**, *49(3)*, 1013–1032. https://api.semanticscholar.org/CorpusID:122070021.

 doi:10.20944/preprints202508.0719.v1

13. Hunt, K. J.; Sbarbaro, D.; Zbikowski, R.; Gawthrop, P. J. Neural networks for control systems – A survey. *Automatica* **1992**, *28(6)*, 1083–1112. https://api.semanticscholar.org/CorpusID:207511431.
14. Euler, L. P. *Institutiones Calculi Integralis*. Publisher: Impensis Academiae Imperialis Scientiarum, St. Petersburg, 1768.
15. Zhang,Y.; Li, L.; Yang, Y.; Ruan, G. Euler neural network with its weight-direct-determination and structure-automatic-determination algorithms. In Proceedings of the Ninth International Conference on Hybrid Intelligent Systems (IEEE Computer Society), Shenyang, China, 2009, 319–324.
16. Tasinaffo, P. M. Estruturas de Integração Neural Feedforward Testadas em Problemas de Controle Preditivo. Doctoral Thesis, INPE-10475-TDI/945, São José dos Campos/SP, Brazil, 2003.
17. Tasinaffo, P. M.; Dias, L. A. V. ; da Cunha, A. M. A qualitative approach to universal numerical integrators (UNIs) with computational application. *Human-Centric Intelligent Systems* **2024**, *4(4)*, 571—598. https://doi.org/doi.org/10.1007/s44230-024-00087-x.
18. Tasinaffo, P. M.; Dias, L. A. V. ; da Cunha, Ad. M. A quantitative approach to universal numerical integrators (UNIs) with computational application. *Human-Centric Intelligent Systems* **2025**, *5(1)*, 1–20. https://doi.org/10.1007/s44230-024-00084-0.
19. Tasinaffo, P.M.; Gonçalves, G.S.; Marques, J.C.; Dias, L.A.V.; da Cunha, A.M. The Euler-type universal numerical integrator (E-TUNI) with backward integration. *Algorithms* **2025**, *18(153)*, 1–28. https://doi.org/10.3390/a18030153.
20. Wang, Y.-J.; Lin, C.-T. Runge-Kutta neural network for identification of dynamical systems in high accuracy. *IEEE Transactions on Neural Networks* **1998**, *9(2)*, 294–307. https://doi.org/10.1109/72.661124.
21. Uçak, K. A Runge-Kutta neural network-based control method for nonlinear MIMO systems. *Soft Computing* **2019**, *23(17)*, 7769–7803. https://doi.org/10.1007/s00500-018-3405-5.
22. Chen, R. T. Q.; Rubanova, Y.; Bettencourt, J.; Duveand, D. Neural ordinary differential equations. In Proceedings of the 32nd Conference on Neural Information Processing Systems (NeurIPS), Montréal, Canada, 2018, 1–19. https://doi.org/10.48550/arXiv.1806.07366.
23. Uçak, K. A novel model predictive Runge-Kutta neural network controller for nonlinear MIMO systems. *Neural Processing Letters* **2020**, *51(2)*, 1789–1833. https://doi.org/10.1007/s11063-019-10167-w.
24. Sage, A. P. *Optimum Systems Control*. Publisher: Prentice-Hall, Inc., Englewood Cliffs, NJ, 1968.
25. Munem, M. A.; Foulis, D. J. *Calculus with Analytic Geometry (Volumes I and II)*. Publisher: Worth Publishers, Inc., New York, USA, 1978.
26. Wilson, E. *Advanced Calculus*. Publisher: Dover Publications, New York, USA, 1958.
27. Lagaris, I. E.; Likas, A.; Fotiadis, D. I. Artificial neural networks for solving ordinary and partial differential equations. *IEEE Transactions on Neural Networks* **1998**, *9(5)*, 987–1000. https://doi.org/10.1109/72.712178.
28. Hayati, M.; Karami, B. Feedforward neural network for solving partial differential equations. *J. Appl. Sci. (Faisalabad)* **2007**, *7(19)*, 2812–2817. https://doi.org/10.3923/jas.2007.2812.2817.
29. Lagaris, I. E.; Likas, A.; Fotiadis, D. I. Neural-network methods for boundary value problems with irregular boundaries. *IEEE Transactions on Neural Networks* **2000**, *11(5)*, 1041–1049. https://doi.org/10.1109/72.870037.
30. Weinan, E.; Han, J.; Jentzen, A. Deep learning-based numerical methods for high-dimensional parabolic partial differential equations and backward stochastic differential equations. *Commun. Math. Stat.* **2017**, *5*, 349–380. https://doi.org/10.1007/s40304-017-0117-6.
31. Han, J.; Arnulf, J.; Weinan, E. Solving high-dimensional partial differential equations using deep learning. *Proc. Natl. Acad. Sci. U.S.A.* **2018**, *115(34)*, 8505–8510. https://doi.org/10.1073/pnas.1718942115.
32. Sacchetti, A.; Bachmann, B.; Löffel, K.; Künzi, U.-M.; Paoli, B. Neural networks to solve partial differential equations: A comparison With finite elements. *J. IEEE Access* **2022**, *10*, 32271–32279. https://doi.org/10.1109/ACCESS.2022.3160186.
33. Zhu, Q.; Yang, J. A local deep learning method for solving high order partial differential equations. *Numer. Math. Theor. Meth. Appl.* **2022**, *15(1)*, 42–67. https://doi.org/10.48550/arXiv.2103.08915.
34. Lu, L.; Meng, X.; Mao, Z.; Karniadakis, G. E. Deepxde: a deep learning library for solving differential equations. *SIAM Review* **2021**, *63(1)*, 208–228. https://doi.org/10.1137/19M1274067.
35. Han, J.; Nica, M.; Stinchcombe, A. R. A derivative-free method for solving elliptic partial differential equations with deep neural networks. *Journal of Computational Physics* **2020**, *419*, 109672. https://doi.org/10.1016/j.jcp.2020.109672.
36. van der Houven, P. J. The development of Runge-Kutta methods for partial differential equations. *Applied Numerical Mathematics* **1996**, *20*, 261–272. https://doi.org/10.1016/0168-9274(95)00109-3.

 doi:10.20944/preprints202508.0719.v1

37. Kalman, R. E. A new approach to linear filtering and prediction problems. *J. Basic Eng.* **1960**, *82(1)*, 35–45. https://doi.org/10.1115/1.3662552.
38. Singhal, S.; Wu, L. Training multilayer perceptrons with the extended Kalman algorithm. *Adv. Neural Inf. Process. Syst.* **1989**, *1*, 133–140.
39. Rios Neto, A. stochastic optimal linear parameter estimation and neural nets training in systems modeling. In Proceedings of the RBCM - J. of the Braz. Soc. Mechanical Sciences, Brazil, 1997, 138–146.
40. Hagan, M. T.; Menhaj, M. B Training feedforward networks with the Marquardt algorithm. *IEEE Transactions on Neural Networks* **1994**, *5(6)*, 989–993. https://doi.org/10.1109/72.329697.