World Journal of Advanced Engineering Technology and Sciences
eISSN: 2582-8266
Cross Ref DOI: 10.30574/wjaets
Journal homepage: https://wjaets.com/

(RESEARCH ARTICLE)

Check for updates

A systematic approach to extracting multivariate polynomial models from neural networks

Henry Samambgwa * and Thomas Musora

Department of Mathematics and Statistics, School of Natural Sciences and Mathematics, Chinhoyi University of Technology, 78 Magamba Way, Off Chirundu Road, Chinhoyi, Zimbabwe.

World Journal of Advanced Engineering Technology and Sciences, 2025, 17(03), 473-480

Publication history: Received on 12 November 2025; revised on 29 December 2025; accepted on 31 December 2025

Article DOI: https://doi.org/10.30574/wjaets.2025.17.3.1578

Abstract

This study develops a systematic approach to extract multivariate polynomial models from neural networks. A neural network is trained using a random dataset generated using a bivariate polynomial. Numerical values obtained from simulating the neural network are used to estimate the partial derivatives with respect to each input variable with increasing order of partial differentiation until the derivatives become zero. In that way the highest power required in the model is identified for each input variable. A general form of the multivariate polynomial model is written with all possible combinations of powers of the input variables. An appropriate set of corresponding partial differential operators is identified so as to produce a system of partial differential equations from the general polynomial model. The corresponding partial derivatives are estimated numerically and substituted into the equations. On solving the equations, it was found that the method correctly estimates the appropriate parameters for the multivariate polynomial model.

Keywords: Neural Network; Model Extraction; Numerical Partial Differentiation; Multivariate Polynomial

1. Introduction

The future of artificial intelligence necessitates machines that can infer the underlying nature of systems from incomplete information. Trained neural networks replicate the outputs of a system according to specified input conditions. The ability to explicitly infer the intricate structure and operations of the system from a given trained neural network presents priceless capabilities in understanding, tweaking and potentially managing the systems under study. Neural network model extraction has been studied as a system analysis tool and also as a hacking technique for understanding target neural network systems [1]. A recent study [2] developed a technique for neural network multivariate model extraction. Their study was based on variable neutralisation, where terms dependent on absent (values set to zero) variables had no effect, that is zero value in the model. Univariate functional terms were isolated by setting other variables to zero, and then modelling the resulting response on the remaining variable. Neural networks are trained from empirical data [3]. Inputs at each neuron in the hidden or output layer of a neural network are a linear combination of values from preceding neurons. Some choices of activation functions, such as the sigmoid and hyperbolic tangent functions, are made to prevent the overall output of the neural network from becoming a linear combination of the inputs at each neuron [2]. This prevents the neural network from only being able to appropriately model linear functions. The resulting models become black box models (models that replicate the outputs of a system but do not explicitly show the underlying functions governing the system) when systems are modelled based on estimation of weights and biases.

* Corresponding author: Henry Samambgwa.

Some researches choose activation functions that maintain the linearity of the overall neural network model such as the rectified linear unit (ReLu). This results in simplified model extraction where the final weights and biases are used to write down neural network models in analytic form. Ranasinghe et al [4] developed a growing interpretable neural network model for extracting polynomial models as Laurent series. The model succeeded in 87% of their test samples and outperformed symbolic regression alternatives in selected tasks. Their strategy was to train a white box neural network model where the weights were explicit powers in the output. After training the neural network they would then read the final weights and use them to obtain analytic form models.

Kolmogorov-Arnold equations decompose multivariate functions into polynomials with weighted product terms [5] whose parameters can be estimated using multivariate regression analysis. Guo et al [6] built upon this principle and developed a white box linear activation neural network model in which hidden neuron inputs were weighted products instead of the usual weighted sums. The outputs of the model could be expressed analytically using the visible final weights. Liu et al [7] proposed the name Kolmogorov-Arnold Networks (KAN) for this kind of neural network. Danieli and Granot [8] developed a hacking technique for learning ReLu based neural network architectures. Some hack attacks seek exact replicas of target systems, others develop only an approximation that generates similar outputs [9]. Canales-Martinez et al [10] developed a white box neural network modelling technique whose goal was to reduce computer system processing resource usage. The choice of activation function, ReLu, allowed the final weights and biases to be extracted from the neural network so as to express the model in analytic form.

Morala et al [11] developed a method of neural network model extraction, where activation functions were expressed as Taylor series expansions. A single hidden layer was configured. The weights of a trained neural network would then be inserted in the model which was a combination of Taylor series expansions, that was then simplified into a polynomial regression model. Carlini et al [12] compared neural network model extraction to cryptocurrency mining, where computational resources are invested to derive value. They developed a hacking technique that copied target neural network systems. The technique exploited weaknesses in the white box nature of ReLu based neural network models by analysing outputs from strategically selected input conditions.

Eskandarian et al [13] developed a method for extracting correlated time series models from neural networks. Recurrent neural networks are a popular implementation for modelling time series [14], more efficiently than traditional correlation analysis [15]. These are, however, black box models when the choice of activation are sigmoid or hyperbolic tangents. Chrysos et al [16] proposed polynomial neural networks (PNN) where activation functions are polynomials in place of the non-linear sigmoid and hyperbolic tangents. PNNs are more easily interpreted white box models. Tang et al [17] used a PNN to identify parameters for time series models.

Aim

To develop a systematic implementation for extracting multivariate polynomial models from neural networks.

Objectives

- Develop a neural network analysis program that identifies the appropriate order of multivariate polynomial that fits the model,
- Develop a technique for the program to identify appropriate coefficients for the terms of various order in the polynomial,
- Validate and evaluate the performance of the developed program.

2. Methodology

2.1. Differential analysis

In a multivariate polynomial, partial derivatives can be chosen so as to cause specific terms to vanish. By evaluating a set of partial derivatives at strategic input data values, a system of linear equations can be obtained whose variables are the coefficients of the polynomial model. Solving the resulting system of equations then obtains the desired coefficients of the polynomial model.

2.2. Polynomial model extraction

This study develops a technique for extracting multivariate polynomial models from trained neural networks. The technique is outlined in 5 steps:

- Obtain the trained neural network whose polynomial model is to be extracted,
- Identify the appropriate highest power in the model for each variable,
- Obtain the appropriate set of partial differential operations from the identified highest powers,
- Generate the corresponding system of linear equations at strategic input values,
- Solve the system of equations to identify the polynomial model coefficients.
- Obtaining a trained neural network

A feedforward neural network is trained using a data set simulated from a multivariate polynomial. For illustration, the bivariate polynomial model will be used in this study. MATLAB is used to simulate 100 000 data points in the range $[-10,10]$ for both input variables. The neural network is configured with three hidden layers of 10 neurons each and one output neuron, [2 10 10 10 1].

2.3. Identifying highest powers for each variable

Central difference partial differentiation of increasing order is implemented for each input variable until the partial derivative is zero. This is the point after the partial derivative yielded a constant. The number of differential iterations needed to obtain a zero partial derivative is used to identify the highest power of each variable in the polynomial model.

All possible combinations of the input variables are represented in the multivariate polynomial regression model, that is, terms with all possible powers of each variable from zero to the highest. In the bivariate model case where the highest power is two for each variable, the model is represented in Equation (1).

$$R = k_0 + k_1x + k_2y + k_3xy + k_4x^2y + k_5xy^2 + k_6x^2y^2. \quad (1)$$

2.4. Exhaustive partial differential operations

The set of partial differential operators consists of all possible combinations of partial differential operations that result in non-zero partial derivatives. For the bivariate polynomial case, these are:

$$f_x, \quad f_y, \quad f_{xy}, \quad f_{xxy}, \quad f_{xyy}, \quad \text{and } f_{xxyy}.$$

2.5. Generating the system of linear equations

Executing all the partial differential operations on the model results in six additional equations. The operations are executed in MATLAB code [18].

$$k_0 + k_1x + k_2y + k_3xy + k_4x^2y + k_5xy^2 + k_6x^2y^2 = R \quad (2)$$
$$k_1 + k_3y + 2k_4xy + k_5y^2 + 2k_6xy^2 = R_x \quad (3)$$
$$k_2 + k_3x + k_4x^2 + 2k_5xy + 2k_6x^2y = R_y \quad (4)$$
$$k_3 + 2k_4x + 2k_5y + 4k_6xy = R_{xy} \quad (5)$$
$$2k_4 + 4k_6y = R_{xxy} \quad (6)$$
$$2k_5 + 4k_6x = R_{xyy} \quad (7)$$
$$4k_6 = R_{xxyy} \quad (8)$$

Evaluating all equations at $x = 1, y = 1$, the system can be represented in the augmented matrix (9).

$$\left[\begin{array}{ccccccc|c} 1 & 1 & 1 & 1 & 1 & 1 & 1 & R \\ 0 & 1 & 0 & 1 & 2 & 1 & 2 & R_x \\ 0 & 0 & 1 & 1 & 1 & 2 & 2 & R_y \\ 0 & 0 & 0 & 1 & 2 & 2 & 4 & R_{xy} \\ 0 & 0 & 0 & 0 & 2 & 0 & 4 & R_{xxy} \end{array}\right] \quad (9)$$

$$\left[\begin{array}{ccccccc|c} 0 & 0 & 0 & 0 & 0 & 2 & 4 & R_{xyy} \\ 0 & 0 & 0 & 0 & 0 & 0 & 4 & R_{xxyy} \end{array}\right]$$

2.6. Solving the system of equations

The system of linear equations represented by the augmented matrix (9) are solved using elementary row operations. These are executed in programming source code using upper triangular matrix neutralisation in MATLAB [19].

3. Analysis

3.1. Programming environment

The implementation was developed and tested in the MATLAB online [20] programming environment.

Structure of the section

The analysis in this section is outlined in five steps:

A. Generating a trained neural network,
B. Systematically identifying the highest powers for each variable present,
C. Setting up the general form of the polynomial and corresponding partial differential operations,
D. Generating the corresponding system of linear equations,
E. Solving the linear equations to obtain the model coefficients.

3.2. Generating a trained neural network

For testing purposes, a neural network was trained using the bivariate polynomial in equation (10):

$$R = 5 - 2x + x^2 + 3y^2 + 3y - 4xy + 2x^2y + 4xy^2 + 2x^2y^2 + 4y^3 - 2xy^3 + x^2y^3. \quad (10)$$

The source code snippet in **Figure 1** simulated the generation of 10 000 random data points in the range $[-2.5,2.5]$, for each of the two variables, that is: $-2.5 \leq x \leq 2.5$ and $-2.5 \leq y \leq 2.5$.

A feedforward neural network was then trained with two input layer neurons (for the input variables x and y), three hidden layers with ten neurons each, and an output layer with one output neuron (for the response variable R). The $sigmoid$ activation function was used for the hidden layer neurons and the $purelin$ activation function was used at the output layer.

$$\textbf{Sigmoid: } f(x) = \frac{1}{1-e^{-x}}, \quad \textbf{Purelin: } f(x) = x. \quad (11)$$

3.3. Identifying the appropriate highest power for each input variable

The trained neural network was simulated in the neighbourhood of $(x, y) = (1,1)$. Using a step size of $h = 0.25$, the neural network output was numerically partially differentiated w.r.t. each input variable with increasing order of partial differentiation (using the MATLAB code in **Figure 2**) until the result became zero (zero occurs when a constant is differentiated). Thus, the highest power present for each variable was identified.

From **Figure 3**, the partial derivatives w.r.t. x became zero from the third partial derivative. The partial derivatives w.r.t. y became zero from the fourth partial derivative. This indicated that the highest power of x was 2 and the highest power of y was 3.

```
%Setting umber of data points
npoints=10000;

%Generation of random x and y
x = rand(npoints,1) *5.0-2.5;        %-2.5<=x<=2.5
y = rand(npoints,1) *5.0-2.5;        %-2.5<=x<=2.5

%Calculation of response R (x, y)
R=5-2*x+x.^2+3*y.^2+3*y-4*x.*y+2*y.*x.^2+4*x.*y.^2+2*(x.^2). *(y.^2) +4*(y.^3)-2*x.*(y.^3) + (x.^2). *(y.^3);

%Constituting the input vector
in = [x y];

%Setting up the neural network
hidden = [10 10 10];        %hidden layers with 10 neurons each
net = feedforwardnet(hidden);

%Setting sigmoid activation in hidden layers
for i =1: length(hidden)
        net. layers{i}. transferFcn = 'tansig';
end

%Setting activation function for output layer
net. layers{end}. transferFcn = 'purelin';

%Transposing variables
in=in';
response=R';

%Training the neural network
net = train (net, in, response);
```

Figure 1 MATLAB code snippet for simulating 10 000 random data points and training a neural network

3.4. General form of the model and corresponding partial differential equations

The highest powers of x and y were detected to be 2 and 3 respectively. The general form of the polynomial with all possible combinations of powers of x from 0 to 2 and powers of y from 0 to 3 was written down in equation (12).

$$R = k_0 + k_1x + k_2x^2 + k_3y + k_4xy + k_5x^2y + k_6y^2 + k_7xy^2 + k_8x^2y^2 + k_9y^3 + k_{10}xy^3 + k_{11}x^2y^3. \quad (12)$$

The corresponding set of partial differential operations that systematically reduce the polynomial were written in (13).

$$f_x, \quad f_{xx}, \quad f_y, \quad f_{xy}, \quad f_{xxy}, \quad f_{yy}, \quad f_{xyy}, \quad f_{xxyy}, \quad f_{yyy}, \quad f_{xyyy}, \quad f_{xxyyy}. \quad (13)$$

```
%Defining function f to generate outputs by simulating the trained neural network
f = @ (x, y) net ([x y]');

% setting point (x, y) = (1, 1) and step size h = 0.25
x0=1;
y0=1;
h=0.25;

%Calculating first partial derivatives using central difference formulae
fx = (f (x0 + h, y0) - f (x0 - h, y0)) / (2 * h)
fy = (f (x0, y0 + h) - f (x0, y0 - h)) / (2 * h)

%Calculating second partial derivatives
fxx = (f (x0 + 2*h, y0) - 2*f (x0, y0) + f (x0 - 2*h, y0)) / (4*h^2)
fyy = (f (x0, y0 + 2*h) - 2*f (x0, y0) + f (x0, y0 - 2*h)) / (4*h^2)

%Calculating third partial derivatives
fxxx = (f (x0 + 4*h, y0) - 2*f (x0 + 2*h, y0) + 2*f(x0 - 2*h, y0) - f(x0 - 4*h, y0)) /
(16*h^3)
fyyy = (f (x0, y0 + 4*h) - 2*f (x0, y0 + 2*h) + 2*f(x0, y0 - 2*h) - f(x0, y0 - 4*h)) /
(16*h^3)

%Calculating third partial derivatives
fxxxx = (f(x0 + 4*h, y0) - 4*f(x0 + 2*h, y0) + 6*f(x0, y0) - 4*f(x0 - 2*h, y0) + f(x0 - 4*h,
y0)) / (16*h^4)
```

Figure 2 MATLAB code used to calculate partial derivatives from the 1st to 4th order w.r.t. x and y

```
fx =
   7.9996
fy =
  28.1888
fxx =
  11.9957
fyy =
  36.0038
fxxx =
   0.0030
fyyy =
  17.9948
fxxxx =
   0.0743
fyyyy =
  -0.0857
```

Figure 3 MATLAB output showing partial derivatives from the 1st to 4th order w.r.t. x and y.

3.5. Generating the corresponding system of linear equations

The system of equations (14) - (25) was generated by executing the partial derivative operators in (13) on the corresponding equation (12) [21].

$$k_0 + k_1x + k_2x^2 + k_3y + k_4xy + k_5x^2y + k_6y^2 + k_7xy^2 + k_8x^2y^2 + k_9y^3 + k_{10}xy^3 + k_{11}x^2y^3 = R \quad (14)$$

$$k_1 + 2k_2x + k_4y + 2k_5xy + k_7y^2 + 2k_8xy^2 + k_{10}y^3 + 2k_{11}xy^3 = R_x \quad (15)$$

$$2k_2 + 2k_5y + 2k_8y^2 + 2k_{11}y^3 = R_{xx} \quad (16)$$

$$k_3 + k_4x + k_5x^2 + 2k_6y + 2k_7xy + 2k_8x^2y + 2k_9y^2 + 3k_{10}xy^2 + 3k_{11}x^2y^2 = R_y \quad (17)$$

$$k_4 + 2k_5x + 2k_7y + 4k_8xy + 3k_{10}y^2 + 6k_{11}xy^2 = R_{xy} \quad (18)$$

$$
\begin{aligned}
2k_5 \quad\quad\quad\quad + 4k_8y \quad\quad\quad\quad + 6k_{11}y^2 &= R_{xxy} \quad (19)\\
2k_6 + 2k_7x + 2k_8x^2 + 4k_9y + 6k_{10}xy + 6k_{11}x^2y &= R_{yy} \quad (20)\\
2k_7 + 4k_8x + \quad + 6k_{10}y + 12k_{11}xy &= R_{xyy} \quad (21)\\
4k_8 \quad\quad\quad\quad + 12k_{11}y &= R_{xxyy} \quad (22)\\
4k_9 + 6k_{10}x + 6k_{11}x^2 &= R_{yyy} \quad (23)\\
6k_{10} + 12k_{11}x &= R_{xyyy} \quad (24)\\
12k_{11} &= R_{xxyyy} \quad (25)
\end{aligned}
$$

When evaluated at $(x, y) = (1,1)$, the system of equations can be represented in the augmented matrix (26). The value of R and the partial derivatives of R on the right side of the augmented matrix were calculated numerically using MATLAB [18] (**Figure 2**).

$$
\left[\begin{array}{cccccccccccccc|c}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 17.0003\\
0 & 1 & 2 & 0 & 1 & 2 & 0 & 1 & 2 & 0 & 1 & 2 & 0 & 2 & 7.9996\\
0 & 0 & 2 & 0 & 0 & 2 & 0 & 0 & 2 & 0 & 0 & 2 & 0 & 2 & 11.9957\\
0 & 0 & 0 & 1 & 1 & 1 & 2 & 2 & 2 & 2 & 3 & 3 & 3 & 3 & 28.1888\\
0 & 0 & 0 & 0 & 1 & 2 & 0 & 2 & 4 & 0 & 3 & 6 & 6 & 6 & 15.9929\\
0 & 0 & 0 & 0 & 0 & 2 & 0 & 0 & 4 & 0 & 0 & 6 & 6 & 6 & 18.1144\\
0 & 0 & 0 & 0 & 0 & 0 & 2 & 2 & 2 & 4 & 6 & 6 & 6 & 6 & 36.0038\\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 4 & 0 & 6 & 12 & 12 & 12 & 16.0033\\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 4 & 0 & 0 & 12 & 12 & 12 & 19.9633\\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 4 & 6 & 6 & 6 & 6 & 17.9948\\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 6 & 12 & 12 & 12 & -0.0050\\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 12 & 12 & 12 & 12.0754
\end{array}\right] \quad (26)
$$

3.6. Solving the system of equations to get model coefficients

The constant and coefficients for the model, $(k_0 - k_{11})$ where obtained by solving the system represented in augmented matrix (26) by row reduction. The results produced the values of k_0 to k_{11}.

Table 1 Resulting estimates of model constants.

Constant	k_0	k_1	k_2	k_3	k_4	k_5	k_6	k_7	k_8	k_9	k_{10}	k_{11}
Estimated value	4.7303	−1.8412	0.9252	3.2898	−4.2017	2.0944	2.9723	4.0602	1.9720	4.0063	−2.0134	1.0063

4. Discussion

When rounded to one significant figure, the resulting constants in **Table 1** were precisely equal to the constants in equation (10) that were used to generate the random data points. This demonstrated that the model extraction strategy was able to estimate the correct model with high accuracy. The method can be extended to an arbitrary number of input variables with arbitrary polynomial degrees. It can be applied to a wide range of problems including situations were polynomial models need to be retrieved from trained neural networks and scenarios where the model can be estimated using a multivariate polynomial.

5. Conclusion

The study successfully developed a method and MATLAB implementation for extracting multivariate polynomial models from trained neural networks. The analysis demonstrated that the method produces model estimates of high accuracy. This widens the scope of capabilities in neural network research. Extracting analytic models from neural networks allows researchers to study the underlying operations controlling a system. Useful insights can be drawn and in some cases can help identify the nature and identify of the influences governing the system. Further research may be done to develop ways of extracting non-polynomial models from neural networks. Automating extraction methods helps improve the understanding capabilities of neural network systems in the pursuit of artificial general intelligence.

Compliance with ethical standards

Disclosure of conflict of interest

No conflict of interest to be disclosed.

References

[1] Jagielski, M., Carlini, N., Berthelot, D., Kurakin, A., Papernot, N. (2020). High Accuracy and High Fidelity Extraction of Neural Networks. arXiv. Cornell University.

[2] Samambgwa, H., Musora, T., Kamusha, J. (2025). Deriving mathematical models from neural networks: A method for deducing individual effects of factors on a response variable. World Journal for Advanced Engineering Technology and Sciences.

[3] Musora, T. (2023). Application of Panel Data Analysis and Modelling to Economic Data: A Case of Determinants of Economic Growth for SADC and Zimbabwe. Chinhoyi University of Technology.

[4] Ranasinghe, N., Senanayake, D., Seneviratne, S., Premaratne, M., & Halgamuge, S. (2024). GINN-LP: A Growing Interpretable Neural Network for discovering multivariate Laurent Polynomial equations. Proceedings of the AAAI Conference on Artificial Intelligence. Association for the Advancement of Artificial Intelligence (AAAI).

[5] Schmidt-Heiber, J. (2021). The Kolmogorov-Arnold representation theorem revisited. arXiv. Cornell University.

[6] Guo, X., Shi, Z., Li, B. (2024). Multivariate polynomial regression by an explainable sigma-pi neural network. Big Data and Information Analytics. AIMS Press.

[7] Liu, Z., Wang, Y., Vaidya, S., Ruehle, F., Halverson, J., Solja, M., Hou, T. Y., Tegmark, M. (2025). KAN:Kolmogorov–Arnold Networks. arXiv. Cornell University.

[8] Danieli, A., Granot, E. (2023). An Exact Poly-Time Membership-Queries Algorithm For Extracting A Three-Layer Relu Network. arXiv. Cornell University.

[9] Chen, S., Klivans, A. R., Meka, R. (2021). Efficiently Learning Any One Hidden Layer ReLU Network From Queries. In NeurIPS.

[10] Canales-Martínez, I.A., Chávez-Saab, J., Hambitzer, A., Rodríguez-Henríquez, F., Satpute, N., Shamir, A. (2024). Polynomial Time Cryptanalytic Extraction of Neural Network Models. Springer.

[11] Moralaa, P., Cifuentesa, J. A., Lilloa, R. E., Ucara, I. (2021). Towards a mathematical framework to inform Neural Network modelling via Polynomial Regression. arXiv. Cornell University.

[12] Carlini, N., Jagielski, M., Mironov, I. (2020). Cryptanalytic Extraction of Neural Network Models. Lecture Notes in Computer Science. Springer.

[13] Eskandariana, P., Mohasefib, J. B., Pirnejadc, H., Niazkhanie, Z., (2022). A novel artificial neural network improves multivariate feature extraction in predicting correlated multivariate time series. Applied Soft Computing. Elsevier.

[14] Pan, W., Zhang, L., Shen, C. (2021). Data driven time series prediction based on multiplicative neuron model artificial neuron network. Applied Soft Computing.

[15] Samambgwa, H., Musora, T., Mamutse, D. (2025). Modeling and Forecasting the Value of Special Drawing Rights. International Journal of Research and Review.

[16] Chrysos, G. G., Moschoglou, S., Bouritsas, G., Deng, J., Panagakis, Y., Zafeiriou, S. (2021). arXiv. Cornell University.

[17] Tang, Y., Xu, Z., Huang, W. (2025). Design of Self-Optimizing Polynomial Neural Networks with Temporal Feature Enhancement for Time Series Classification. Electronics.

[18] Samambgwa, H., Musora, T. (2025). Automating Numerical Partial Differentiation. GSC Advanced Engineering and Technology.

[19] Samambgwa, H., Musora, T. (2025). Automating upper triangular matrix neutralisation for minimal error. World Journal of Advanced Engineering Sciences and Technology.

[20] MATLAB Online. (2025). Accessed at: matlab.mathworks.com. Mathworks.

[21] Samambgwa, H., Musora, T. (2025). Automating Symbolic Partial Differentiation on Multivariate Polynomials. Global Journal of Engineering and Technology Advances.