A Neural Network Based Approach for Approximating Real Roots of Polynomials

Diogo Freitas
University of Madeira
Master's Programme in Mathematics
Funchal, Portugal
2019214@student.uma.pt

Luiz Guerreiro Lopes
University of Madeira, Funchal,
CIMO/IPB, Bragança, and
ICAAM/UE, Évora, Portugal
lopes@uma.pt

Fernando Morgado-Dias
University of Madeira
Madeira Interactive Technologies Institute
Funchal, Portugal
morgado@uma.pt

***Abstract*—There are many iterative methods for finding all the zeros of a polynomial sequentially or simultaneously. However, the determination of all zeros of a given polynomial by one of the methods that find one zero at a time involves repeated deflations, which leads to the accumulation of rounding errors and inaccurate results. In turn, the simultaneous methods require very good starting approximations for all the zeros in order to converge. In view of these drawbacks, in this work we adopt a different approach based on neural networks for finding the zeros of real polynomials with only real zeros. This approach is tested with random polynomials of different degrees. The results obtained, although preliminary and limited, indicate that this approach seems to be quite robust and promising, and faster when compared with the well known Durand–Kerner method.**

***Index Terms*—Artificial Neural Networks, Polynomials, Roots, Durand–Kerner method.**

I. Introduction

Although there are many iterative methods to calculate one real zero or a pair of complex conjugate zeros of a polynomial, such as the well-known Laguerre's and Jenkins–Traub's methods [1], the determination of all zeros of a given polynomial by one of such methods involves repeated deflations, which can lead to very inaccurate results due to the problem of accumulating rounding errors when using finite accuracy floating-point arithmetic.

Iterative methods for finding all zeros of a polynomial simultaneously, such as the methods of Durand–Kerner and Ehrlich–Aberth (see, e.g., [2]–[4]), appeared in literature only in the 1960s. The simultaneous zero-finding algorithms, in addition to being inherently parallel, have the advantage of avoiding the polynomial deflation steps required by methods that determine only one real root or a pair of complex roots at a time. However, these simultaneous methods need very good initial approximations for all the zeros in order to converge.

Due to the drawbacks mentioned above, and since traditional Artificial Neural Networks (ANN) or shallow neural networks are well known for their capability to model data and to find good approximations for complex problems, in this work we try a different approach for finding real zeros of polynomials based on neural networks, in order to assess their potentiality and limitations in terms of efficiency and accuracy of the approximations when compared with traditional iterative methods for polynomial zero finding.

II. Datasets and methodology

In this section, the steps taken to build a training and a test dataset, and to train the ANNs to produce approximations for the real zeros of polynomials are described.

Even though the neural approach proposed in this paper can be extended to the case of approximating complex zeros of a polynomial, the ANNs are only used here for approximating the real zeros $\alpha_i \, (i = 1, 2, \ldots, n)$ of a degree n real univariate polynomial, $P(x) = a_0 + a_1 x + \ldots + a_n x^n$, with only real zeros, given its coefficients, as already mentioned in Section I. The block diagram of this approach is shown in Fig. 1.

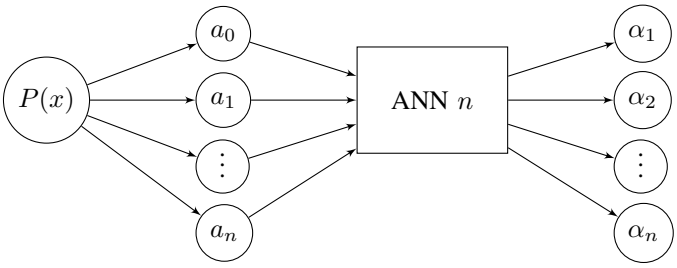

Fig. 1. Flowchart showing the inputs, the processing flow and the outputs of the proposed neural approach to polynomial root finding.

The use of a similar approach based on ANNs for finding all (real and complex) zeros of real polynomials, and also the application of such kind of approach to the most general problem of determining all zeros of complex polynomials are not considered here since they will be the subject of future papers currently in preparation.

Regarding the neural network structure, although there is not an exact method to determine the number of neurons that should be in its hidden layer, there are two common ways to do this: applying the well known Kolmogorov's mapping neural network existence theorem [5] or using a rule of thumb adequate for this purpose [6].

For this work, after several tests according to these two methods, we found that there is little variance resulting from a change in the number of hidden neurons of the neural network. In view of this, in this experimental study we used ten neurons in the hidden layer for all the final tests.

In this preliminary study, we used five neural networks with only three layers (input, hidden, and output layer), that were trained using as input the coefficients of a set of polynomials of degrees 5, 10, 15, 20 and 25, respectively. In Fig. 1, we denote by ANN n the neural network that can output the real zeros of a degree n real polynomial. Tables I and II show the head of the datasets (with 100 000 records) that were used with ANN 5. It is important to note here that, although coefficients and zeros are shown with only four decimal places, double precision values were used to generate the datasets. To generate these datasets, it was used two algorithms to: generates real zeros for any polynomial degree and, given a set of real zeros, compute the respective coefficients.

TABLE I
HEAD OF THE INPUT DATASET FOR TRAINING ANN 5

a_0	a_1	a_2	a_3	a_4	a_5
-4593.5077	3961.1594	-155.2456	-120.6867	3.6453	1
-5351.3845	3272.8352	251.6259	-125.4805	-2.3285	1
643.6638	701.0272	133.1762	-46.9399	-7.0419	1
0.8773	51.3427	28.4148	-15.6812	-2.9906	1
-0.2478	12.5678	55.9301	-66.4759	-2.5088	1
⋮	⋮	⋮	⋮	⋮	⋮

TABLE II
HEAD OF THE OUTPUT DATASET FOR TRAINING ANN 5

α_1	α_2	α_3	α_4	α_5
-9.2110	-8.9445	1.2858	6.0128	7.2117
-9.2925	-5.9211	1.5966	6.3456	9.5999
-4.2366	-1.9342	-1.5788	5.1719	9.6196
-3.1738	-1.2258	-0.0173	2.8990	4.5084
-7.4262	-0.1998	0.0183	1.0033	9.1133
⋮	⋮	⋮	⋮	⋮

Mathematically, an ANN is represented by a weighted, directed graph with nodes. For this study, a multilayer perceptron (MLP) feedforward neural network was chosen. The first layer of the ANN contains the input nodes, which have no incoming links attached to them. The last layer contains the output nodes, and the intermediate hidden layer consists of nodes connected to each of the nodes in the input and output layers [7].

The well-known Levenberg–Marquardt backpropagation algorithm (LMA) was used for ANN training, due to its efficiency and convergence speed, being one of the fastest methods for training feedforward neural networks, especially medium-sized ones. The application of the Levenberg–Marquardt algorithm to neural network training is described, e.g., in [8] and [9]. LMA is a hybrid algorithm that combines the efficiency of the Gauss-Newton method with the robustness of the gradient descent method, making one of these methods more or less dominant at each minimization step by means of a damping factor that is adjusted at each iteration [10].

The hyperbolic tangent sigmoid (tansig) function [11], defined in (1), has been chosen in this study as the activation function for the hidden and output layer nodes, in order to ensure that values stay within a relatively small range and to allow the network to learn nonlinear relationships [12] between coefficients and zeros.

$$\phi(x) = \frac{2}{1 + e^{-2x}} - 1. \tag{1}$$

The use of this antisymmetric (S-shaped) function for the input to output transformation allows the output of each neuron to assume positive and negative values in the interval $[-1, 1]$ (see Fig. 2). A min-max normalization method [13] was used to scale all data into this interval.

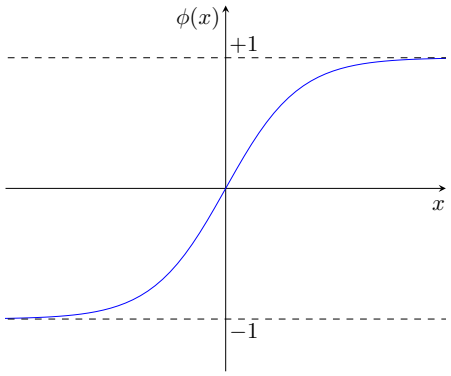

Fig. 2. Hyperbolic tangent sigmoid (tansig) transfer function.

III. DISCUSSION AND RESULTS

In this section, some results obtained with this approach are presented along with comparisons with the numerical approximations provided by the Durand–Kerner method in terms of execution time and accuracy.

The Durand–Kerner (D-K) method, also known as Weierstrass' or Weierstrass–Dochev's method [3], is a well-known iterative method for the simultaneous determination of all zeros of a polynomial that does not require the computation of derivatives, but has the drawback of requiring a good initial approximation to each of the zeros (which must be obtained using another numerical method) in order to converge and produce approximations to these zeros with the required accuracy.

Let $P(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0$ $(a_n \neq 0)$ be a degree n univariate polynomial with real (or complex) coefficients. The D-K method is given by [14]

$$x_i^{(k+1)} = x_i^{(k)} - \frac{P(x_i^{(k)})}{a_n \prod\limits_{\substack{j=1 \\ j \neq i}}^{n} (x_i^{(k)} - x_j^{(k)})}, \tag{2}$$

where $i = 1, \ldots, n$ and $k = 0, 1, \ldots$.

The convergence order of the Durand–Kerner method is quadratic for simple zeros but only linear in case of multiple zeros [15].

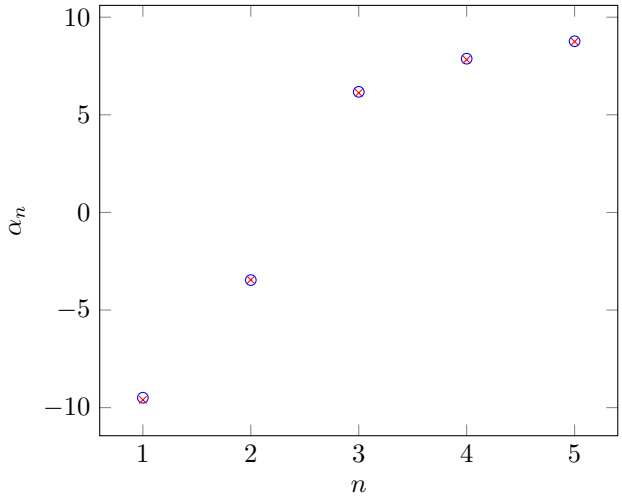

Fig. 3. Comparison between ANN (red) and Durand–Kerner approximations to the real zeros of a random real polynomial of degree 5.

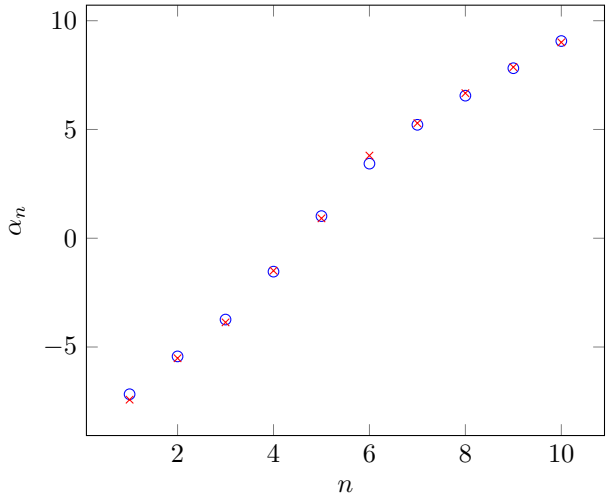

Fig. 4. Comparison between ANN (red) and Durand–Kerner approximations to the real zeros of a random real polynomial of degree 10.

Figs. 3 to 7 show the comparisons, in terms of accuracy, between the approximations to the real zeros of five random real polynomials of degrees 5, 10, 15, 20, and 25 produced by the ANN approach and the Durand–Kerner method.

Analysing the plots, it is clearly noticeable that our approach produces results relatively similar to those obtained with the D-K method. But, as the degree of the polynomial increases, it is possible to observe a slight increase of the differences between the approximations produced by both methods.

Table III shows the Mean Square Error (MSE) for each of the five examples, computed as follows, where D_i denotes the approximation to the i-th zero ($i = 1, \ldots, n$) computed by the D-K method and N_i the corresponding approximation obtained with our approach:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(D_i - N_i)^2. \tag{3}$$

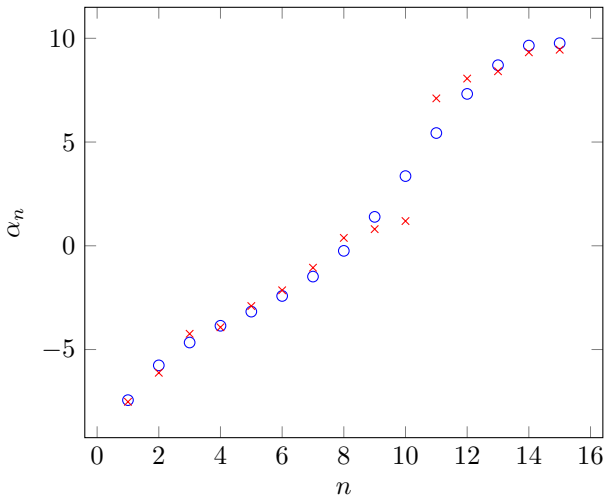

Fig. 5. Comparison between ANN (red) and Durand–Kerner approximations to the real zeros of a random real polynomial of degree 15.

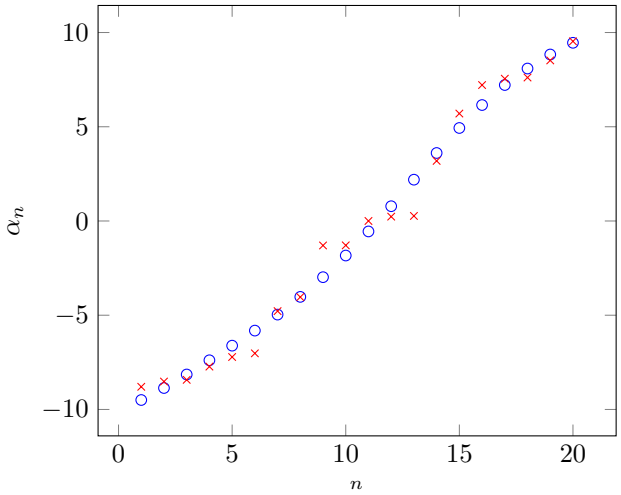

Fig. 6. Comparison between ANN (red) and Durand–Kerner approximations to the real zeros of a random real polynomial of degree 20.

TABLE III
MSE OF THE ANN BASED APPROACH

Degree	MSE
5	0.0036
10	0.0262
15	0.6474
20	0.6213
25	0.4731

The results obtained, although limited, are very encouraging and demonstrate the viability and potentiality of our ANN based approach for approximating real roots of polynomials.

The results on execution time for both methods, showed below, were obtained using a personal computer equipped with a 7th generation Intel Core i7 processor and 16 GB of RAM.

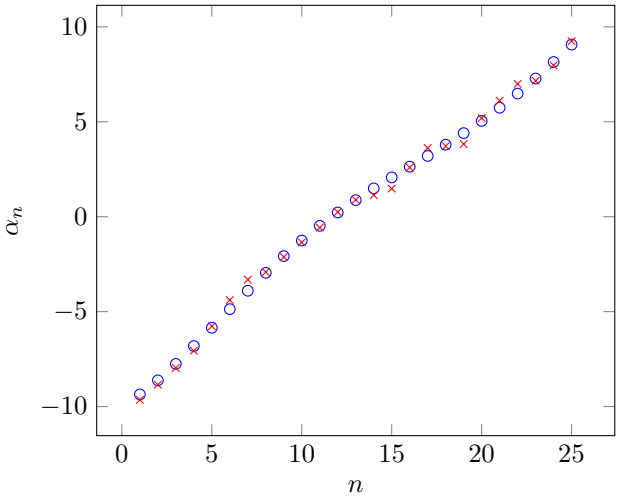

Fig. 7. Comparison between ANN (red) and Durand–Kerner approximations to the real zeros of a random real polynomial of degree 25.

TABLE IV
COMPARISON IN TERMS OF EXECUTION TIME

Degree	ANN approach	D–K method
5	0.004	0.010
10	0.002	0.009
15	0.005	0.011
20	0.003	0.019
25	0.005	0.026

Table IV shows that, when the degree of the polynomial increases, the execution time with ANN remains almost constant. The opposite happens with the Durand–Kerner method, with which an increase in the degree of the polynomial implies an increase of the execution time. Comparing the execution times of both methods, we can observe that the execution time required to compute the approximations to the zeros using ANN is significantly lower than that of the Durand–Kerner method. This result was already expected because computing polynomial zeros using this latter method, unlike ANN, is a pure iterative procedure.

We also assessed the capacity of the networks to generalize the outputs to other spaces of results. For this, new datasets were used with samples that were not employed to train the networks. The computed outputs and targets are compared in Table V.

Since all the MSE values presented in Table V are significantly less than one, we can infer that the networks have a good capacity to generalize the space of results. Thus, with some confidence, we can conclude that the networks can solve any real univariate polynomial of the respective degree with only real zeros.

IV. Conclusion

This short paper is a concise report of ongoing work about the use of artificial neural networks for finding numerical approximations to the zeros of polynomials.

TABLE V
CAPACITY OF THE NETWORKS TO GENERALIZE THE OUTPUTS

Degree	MSE
5	0.4087
10	0.6684
15	0.6666
20	0.4768
25	0.4766

Although the results presented here are preliminary and limited to a particular class of polynomials, namely polynomials with real coefficients and only real zeros, they are very promising and indicate the potential of this neural network based approach for determining the zeros of polynomials.

The proposed approach seems to be quite robust and also shows to be faster than the well known Durand–Kerner iterative method for simultaneous polynomial root finding.

V. Acknowledgments

Acknowledgments to the Portuguese Foundation for Science and Technology (FCT) for their support through the Strategic Project LA 9 – UID/EEA/50009/2013.

References

[1] J. M. McNamee, *Numerical methods for roots of polynomials, Part I*. Amsterdam: Elsevier, 2007.
[2] Bl. Sendov, A. Andreev, and N. Kjurkchiev, "Numerical solution of polynomial equations," in *Handbook of Numerical Analysis, Vol. III*, P. G. Ciarlet and J. L. Lions, Eds. Amsterdam: Elsevier, North-Holand, 1994, pp. 625–776.
[3] M. Petković, *Point estimation of root finding methods*. Berlin: Springer-Verlag, 2008.
[4] O. Cira, *The convergence simultaneous inclusion methods*. Bucareşti: Matrix ROM, 2012.
[5] R. Hecht-Nielsen, "Kolmogorov's mapping neural network existence theorem," in *Proceedings of the IEEE First Annual International Conference on Neural Networks*, M. Caudil and C. Butler, Eds. San Diego, CA: IEEE, 1987, pp. 609–618.
[6] D. Baptista, S. Abreu, C. Travieso-Gonzlez, and F. Morgado-Dias, "Hardware implementation of an artificial neural network model to predict the energy production of a photovoltaic system," *Microprocessors and Microsystems*, vol. 49, pp. 77–86, 2017.
[7] T. L. Fine, *Feedforward Neural Network Methodology*. New York: Springer-Verlag, 1999.
[8] M. T. Hagan and M. B. Menhaj, "Training feedforward networks with the Marquardt algorithm," *IEEE Transactions on Neural Networks*, vol. 5, pp. 989–999, 1994.
[9] M. T. Hagan, H. B. Demuth, M. H. Beale, and O. De Jesús, *Neural Network Design*, 2nd ed. Stillwater, OK: Oklahoma State Univ., 2014.
[10] J. Heaton, *Artificial Intelligence for Humans, Vol. 3: Deep Learning and Neural Networks*. Chesterdfield, MO: Heaton Research, 2015.
[11] P. B. Harrington, "Sigmoid transfer functions in backpropagation neural networks," *Analytical Chemistry*, vol. 65, pp. 2167–2168, 1993.
[12] D. W. Marquardt, "An algorithm for least-squares estimation of nonlinear parameters," *Journal of the Society for Industrial and Applied Mathematics*, vol. 11, pp. 431–441, 1963.
[13] K. L. Priddy and P. E. Keller, *Artificial Neural Networks: An Introduction*. Bellingham, WA: SPIE Press, 2005.
[14] A. Terui and T. Sasaki, "Durand-Kerner method for the real roots," *Japan Journal of Industrial and Applied Mathematics*, vol. 19, pp. 19–38, 2002.
[15] P. Fraigniaud, "The Durand-Kerner polynomials roots-finding method in case of multiple roots," *BIT Numerical Mathematics*, vol. 31, pp. 112–123, 1991.