J. Appl. Numer. Optim. 8 (2026), No. 3, pp. 439-463
Available online at http://jano.biemdas.com
https://doi.org/10.23952/jano.8.2026.3.07

A COMPARISON OF RATIONAL APPROXIMATIONS, NEURAL NETWORKS, AND THEIR COMBINATIONS IN SCIENCE AND ENGINEERING APPLICATIONS

VINESHA PEIRIS[1,*], NADEZDA SUKHORUKOVA[2], REINIER DÍAZ MILLÁN[3], JULIEN UGON[4]

[1]*Centre for Optimisation and Decision Science, Curtin University, WA, Australia*
[2]*Department of Mathematics, Swinburne University of Technology, VIC, Australia*
[3]*School of IT, Deakin University, Waurn Ponds, VIC, Australia*
[4]*School of IT, Deakin University, Burwood, VIC, Australia*

Abstract. Neural networks (NNs) are popular techniques in modern science and engineering applications, where some type of approximation is required. These approaches are able to produce accurate approximations to nonsmooth and non-Lipschitz functions, including multivariate domain functions. Essentially, NNs are approximation tools, inspired by rigorous mathematical approaches and there are still many open problems of purely mathematical nature. Modern computer packages are designed in such way that the construction of approximations by NNs is straightforward for the users and they do not need to know the mathematics behind the scene. This seems to be convenient, but eventually most users would want to open the black-box and improve it. One possibility is to use rational approximation, which combines the high approximation accuracy and the simplicity of the optimisation tools: the algorithms are based on the repeated application of standard linear programming techniques that are part of most modern computer packages. In this paper, we compare the efficiency of function approximation using rational approximation, neural network and their combinations. Our numerical experiments demonstrate the efficiency of rational approximation, even when the number of approximation parameters (that is, the dimension of the corresponding optimisation problems) is small, while for neural networks a higher number of decision variables is required. We demonstrate our findings in numerical examples, one of them is approximating solutions of the KdV (Korteweg-de Vries) equation which appears in fluid dynamics.
Keywords. Chebyshev approximation; Linear optimisation; Neural network approximation; Multivariate approximation; Rational approximation.
2020 Mathematics Subject Classification. 65D15, 65K10, 68T07, 90C90.

1. INTRODUCTION

Neural networks are very powerful and popular techniques used in approximation nowadays, including problems from ordinary and partial differential equations [1], numerical methods for fluid dynamics, etc. [1, 2, 3, 4].

*Corresponding author.
E-mail address: m.peiris@curtin.edu.au (V. Peiris)
Received 27 October 2025; Accepted 22 June 2026; Published online 1 August 2026.

The efficiency and popularity of neural networks was established in the universal approximation theorem [5], which states that neural networks with only one hidden layer can approximate any continuous function to any accuracy, providing that the activation functions are non-polynomials. This powerful theoretical result paves the way for investigating which activation functions work best in practice.

Rational functions are an attractive choice for activation functions because their parameters can themselves be learnt by the network, and it was shown in [2] that good approximation results can be achieved. Approximation techniques in general are another powerful method for function approximation [6, 7], in particular, rational approximation. A comprehensive literature review on uniform approximation by rational function approximation, its geometrical properties can be found in [8].

In this paper, we compare two techniques for function approximations: rational approximation and neural network based approximation. We also consider some combinations of these two approaches in order to improve the performance of the learning system. In particular, we propose a new method for optimising the parameters of a neural network with rational activation functions using an alternating approach where one set of parameters is optimised while the other parameters are fixed and vice versa.

Deep learning and single hidden layer neural network approximation are just a specific type of function approximation, where the function to be approximated is only known by its values in a finite number of points (training data), while the approximation prototype (class of approximations) is a composition of affine mappings and the so called activation functions (special type of univariate function).

In the case of rational approximation, the approximations are simply the ratios of polynomial functions. The choice of rational functions is quite natural: they provide a flexible approximation to a wide range of functions, including nonsmooth and non-Lipschitz function [9, 10, 11]. This flexibility is even comparable with free-knot piecewise polynomial approximation [12]. At the same time, the corresponding optimisation problems are quasiconvex [13, 14, 15] and there are a number of efficient computational methods to tackle them [16, 17, 18] just to name a few.

Our numerical experiments highlight differences between the two methods. The direct rational approximation approach offers greater control over the rational functions and therefore ultimately produces more accurate approximations than neural networks with rational activation functions. All our algorithms are based on the repeated application of linear optimisation techniques. The advantage of this approach is clear. Firstly, most modern computational packages include linear optimisation solvers. Linear optimisation techniques are well-known to modern engineering research community. Finally, linear optimisation enables us to include any linear constraints without without moving outside the field of linear optimisation.

Apart from comparing different approximation techniques, we also implemented an additional improvement of the rational approximation algorithm. The idea is to restrict the condition number of matrices appearing in the auxiliary linear programming problems within rational approximation. This extra restriction is a simple linear constraint, which can be naturally added to the constraints of the linear programs. A similar approach was proposed in [19] for lifting matrix functions. In the current paper, we applied this idea to improve rational approximation algorithms. This approach is especially efficient in the case of multi-dimensional approximation. The results of the numerical experiments are very promising.

Finally, this paper is also an attempt to open the black-box of "standard" neural networks. This research direction is vital, since it leads to a more efficient approach, that customizes the needs of the specific application and, in some cases, enables the researchers to overcome the limitations of neural networks [20, 21].

The paper is organised as follows. In Section 2, we provide the background of the approximation methods we use in this paper: neural network, rational approximation and their combinations. In Section 3, we explain the approximation models we are using in this paper. Then in Sections 4-6, we compare the models. Finally, in Section 7, we draw the conclusions and highlight possible future research directions. Appendix A contains supplementary results, including the results with only two nodes in the hidden layer. The approximations with only two nodes in the hidden layer are not accurate, but we include these results for consistency and comparison.

2. PRELIMINARIES

2.1. **Neural networks.** Neural network approximation is a powerful tool for data and function approximation, which can also be used for data analysis and data classification. The power of neural network approximation is in its structure: the subproblems that the system has to solve are very simple from the point of view of modern optimisation theory and therefore the system can handle problems with hundreds and even thousands of decision variables very efficiently.

Essentially, the objective of neural network approximation is to solve an approximation problem: optimise the weights (parameters) of the network. These weights are the decision variables of certain optimisation problems, whose objective functions represent inaccuracy of approximation. Therefore, it is natural to approach this problem using modern optimisation tools [22, 23, 24].

The solid mathematical background of neural network approximation was established in [25, 26, 27, 28]. These works rely on the results of the celebrated Kolmogorov-Arnold Theorem [29, 30]. The Kolmogorov–Arnold representation theorem states that every multivariate continuous function can be represented as a composition of continuous univariate functions over the binary operation of addition. In general, there is no algorithm for constructing these composition functions. Instead of constructing this representation, modern neural networks techniques approximate this composition function by a composition of affine transformations and the so-called activation functions. The activation functions are univariate non-polynomial functions. The most commonly used activation functions are sigmoid functions and Rectified Linear Unit (ReLU) functions (monotonic piecewise linear function $\varphi(x) = \max\{0, x\}$).

In [2], the authors showed an equivalence between neural networks using rational approximation functions and ReLU networks (see also [31]). Specifically they show that any ReLU network can be approximated to within $\varepsilon > 0$ by a rational network of size $\mathcal{O}(\log(\log(1/\varepsilon))$ and any rational network can be approximated by a ReLU network to within ε of size $\mathcal{O}(\log(1/\varepsilon))^3$. An interesting characteristic of their approach is that the parameters of the activation function can themselves be optimised as part of the learning process. Their work opens the question of how using a direct approach to rational approximation compares with neural network-based approximation.

Most neural network algorithms rely on least squares based measures of inaccuracy (loss function). The goal of optimisation is to optimise the weights by minimising the corresponding loss function. Least squares-based models are minimising a smooth quadratic function, and

therefore fast and simple optimisation techniques (for example, gradient descent) are applicable. In some cases, however, other loss functions are more efficient: uniform based [32], etc. There are also studies that investigate the use of neural networks for uniform approximation of functions [33].

The goal of this paper is to compare the approximation results obtained by neural networks and other approximation techniques. In particular, we are looking at rational approximation due to its approximation power [12]. More details on rational approximation will be provided in the next section. On the other hand, in [31], the author demonstrates that rational functions are as powerful as neural networks with standard ReLU activation functions. This result encouraged many researchers to combine neural networks, rational functions and rational approximation techniques together to enhance the performance of each other [2, 34, 35].

2.2. **Rational approximation.** Rational approximation in Chebyshev (uniform) sense was a very popular research direction in the 1950s-70s [10, 11, 36, 37, 38, 39]. The problem can be formulated as follows:

$$\min_{A,B} \quad \max_{t\in[c,d]} \left| f(t) - \frac{A^T G(t)}{B^T H(t)} \right|, \tag{2.1}$$

subject to

$$B^T H(t) > 0, \tag{2.2}$$

where

- $f(t)$ is the original function,
- $G(t) = (g_0(t), \dots, g_n(t))^T$ and $H(t) = (h_0(t), \dots, h_m(t))^T$ consist of known basis functions,
- $A = (a_0, a_1, \dots, a_n)^T \in \mathrm{R}^{n+1}$, $B = (b_0, b_1, \dots, b_m)^T \in \mathrm{R}^{m+1}$ are our decision variables.

There are two main groups of methods to approach rational approximation. The first group [40] is dedicated to "nearly optimal" solutions. This approach (also known as the AAA approach) is very efficient and therefore very popular, but it is "nearly optimal". The extension of AAA to multivariate cases is still open. Moreover, this method is only designed for unconstraint optimisation and, as we will see later in this paper, this will limit our ability to work with ill-conditioned matrices. The second group of methods is based on modern optimisation techniques: the corresponding optimisation problems are quasiconvex and can be solved using a general quasiconvex optimisation method.

There are many methods for rational approximation [11, 41, 42, 43] (just to name a few), but the implementation of most of them relies on solving linear programming problems, which can be efficiently solved by most modern computational packages. Therefore, additional linear constraints can be easily added to these implementations without making the problems complex. Currently, the most popular optimisation methods for rational approximation are the bisection method for quasiconvex functions and the differential correction method. Some other types of methods can be found in [41, 44, 45], etc.

The popular version of the differential correction method is due to [46]. It calculates A_{k+1}^T and B_{k+1}^T by minimising the expression

$$\max_{t\in[c,d]} \{|f(t)B^T H(t) - A^T G(t)| - \Delta_k B^T H(t)\}, \tag{2.3}$$

where

$$\Delta_k = \max_{t\in[c,d]} \left| f(t) - \frac{A_k^T G(t)}{B_k^T H(t)} \right|. \tag{2.4}$$

Note that equation (2.3) can be reformulated as a linear programming problem.

Bisection method takes a different approach by looking at the problem from the point of view of quasiconvexity. The objective function in equation (2.1) quasiconvex [19] and problem (2.1) - (2.2) can be reformulated as follows:

$$\min \quad z \tag{2.5}$$

subject to

$$f(t) - \frac{A^T G(t)}{B^T H(t)} \leq z, \tag{2.6}$$

$$\frac{A^T G(t)}{B^T H(t)} - f(t) \leq z, \tag{2.7}$$

$$B^T H(t) > 0. \tag{2.8}$$

One can now define a bisecting interval with upper (u) and lower (l) bounds and fix the z at the mid-point of the interval. Then, check if the set of constraints (2.6)-(2.8) has a feasible solution. If there is a feasible point, update the upper bound $u = z$, otherwise update the lower bound $l = z$. This procedure terminates when $u - l < \varepsilon$ where ε is a small number.

The advantage of these two methods is that these methods can be easily extended to the case of non-monomial cases (generalised rational approximation) [15, 38] and to multivariate settings [47]. The theory of univariate rational approximation is well-established, whereas multivariate rational approximation is still developing. This is due to the complexity of handling multivariate rational functions, though some studies have made progress with improved results [48, 49]. Overall, the differential correction method has a quadratic convergence [41], while the bisection method converges linearly [13]. Therefore, in our experiments with univariate rational approximation, we use the differential correction method. At the same time, the bisection method is still a good choice due to its simplicity. Hence, we use the bisection method in our multivariate rational approximation experiments.

3. Models

We compare six different approaches for approximating a given continuous function. Four of them are based on neural networks (NN) and the remaining two are purely rational approximation based approaches. The approaches are as follows:

(1) NN with ReLU activation function,
(2) NN with rational approximation to ReLU activation (we apply the differential correction method to approximate ReLU),
(3) NN with rational activation function, the coefficients of the rational activation function are learnt from NN,
(4) NN with rational activation, the coefficients of the rational activation function and the parameters of the network are learnt with the split method, where the training process is done in three steps: between the input layer and the hidden layer and then between the hidden layer and the output layer, and finally, the coefficients of the rational activation,

(5) Rational approximation (Differential correction), this is the direct rational approximation between the input and output layer,
(6) Rational approximation (AAA), this is the direct "near-optimal" rational approximation between the input and output layer.

We use neural networks with three layers: input layer, hidden layer and output layer. We have the following options for the number of nodes in the hidden layer:

- a network with 2 nodes in the hidden layer (see Appendix A),
- a network with 10 nodes in the hidden layer.

In our experiments, we use different numbers of epochs: 50, 100 and 200. We record the training time per epoch for each network training procedure where batch size is the same as the number of data points in the domain.

For NN methods, we use the MSE-based loss function (MSE stands for Mean Squared Error and simply means "least squares") and uniform loss. For the MSE-based loss, we use ADAM optimiser, while for the uniform loss, we use ADAMAX optimiser (a version of ADAM specially designed for uniform loss). All the rational based approximation models are designed for uniform loss, which is a stronger criterion than least squares and clearly more appropriate if the goal is to approximate a function.

In our experiments, we consider two different settings.

- The domain is a segment (univariate function). We approximate the nonsmooth function

$$f(x) = \sqrt{|x-0.25|}, \quad x \in [-1,1], \tag{3.1}$$

  by using the above approaches with different conditions (number of nodes in the hidden layer, number of epochs, etc.). The choice of the function $f(x)$ is due to its difficulty for most approximation techniques: this function is nonsmooth and non-Lipschitz at $x = 0.25$.
- The domain is bi-variate. It is also possible to extend the results to higher dimensions, but all these extensions are out of the scope of this paper.

In the next section, we only present the important results of the numerical experiments. In particular, we report the results related to neural networks with 10 nodes in the hidden layer where the loss function is in the form of uniform norm. The Appendix A contains a very thorough discussion of the remaining results.

4. Results: Neural Network-based Approximation

The findings in this section are from a neural network with 10 nodes in the hidden layer. The loss function is based on the uniform norm and we use ADAMAX as the optimiser.

4.1. **Neural network with ReLU activation.**

We start our experiments with the standard ReLU activation function. Table 1 demonstrates the loss function value and computational time for different epochs.

One can see that the value of the loss function slightly improves when the number of epochs is increasing. The last column of this table reports the running time per epoch and in this case, computational time increases with the increasing epochs.

Figure 1 shows the approximations computed by the network with ReLU activation. It appeares that the approximation around the "difficult point" $x = 0.25$ is better in the case of the

TABLE 1. Results: experiments of NN with ReLU activation

Epoch	Final loss	Run time (per epoch)
50	0.384359	1.14s ± 113ms
100	0.289953	1.27s ± 161ms
200	0.150630	1.44s ± 203ms

uniform loss than it is in the case of MSE (Figures for MSE can be found in Appendix A). Therefore, the uniform loss may be a better measure when it is essential to obtain accurate approximations around such points.

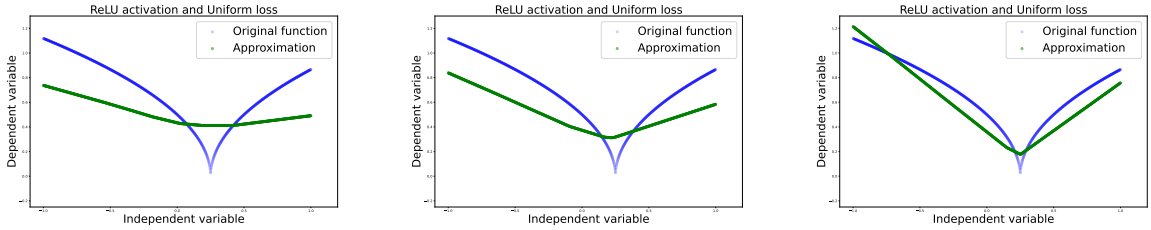

FIGURE 1. Approximation is computed by a neural network with ReLU activation, 50, 100 and 200 epochs.

4.2. **Neural network with rational approximation to ReLU activation.** In this section, we present the results of numerical experiments, where the experiment settings are similar to those in Section 4.1, but the activation function is a rational approximation to the ReLU function. Hence, our activation function is a rational function of degree $(3,2)$: the rational approximation is the ratio of two polynomials, the degree of the numerator is 3 and the degree of the denominator is 2.

The coefficients of the rational activation function come from the best rational $(3,2)$ approximation to the ReLU function computed by the differential correction method. These coefficients are fixed throughout the whole training procedure.

Table 2 shows the improvement in the accuracy compared to the standard ReLU activation in Section 4.1. This observation is especially prominent when the number of epochs is 50 or 100.

TABLE 2. Results: experiments of NN with rational approximation to ReLU

Epoch	Final loss	Run time (per epoch)
50	0.335994	1.53s ± 267ms
100	0.247002	1.78s ± 336ms
200	0.186674	1.61s ± 346ms

Figure 2 shows that the flexibility of the approximation is improving, even for the "difficult point".

Overall conclusion: the approximation results do not vary much when the rational approximation to ReLU is used as the activation function rather than ReLU itself. This observation is valid for both MSE and uniform loss. Results related to MSE can be found in Appendix A.

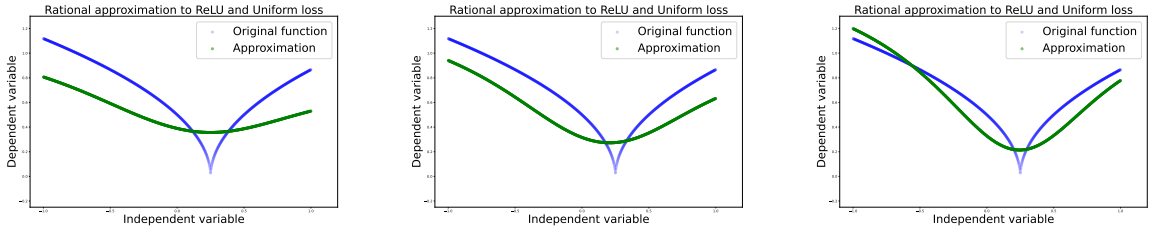

FIGURE 2. Approximation is computed by a neural network with rational approximation to ReLU, 50, 100 and 200 epochs.

4.3. **Neural network with rational activation.** In this case, our activation function is a rational function of degree $(3,2)$. The coefficients of the rational activation function are now a part of the parameter set. We learn these coefficients as we learn other parameters during the training procedure. This type of network with rational activation function whose coefficients are learnable parameters are called 'rational neural networks' and more details can be found in [2]. The Python code for this section is also from [2].

Table 3 shows that when the number of nodes in the hidden layer is 10, the optimal loss function values are very close to the previous two cases where the activation function is ReLU or the activation function is the best rational approximation to ReLU. Figure 3 confirms these results.

TABLE 3. Results: experiments with rational activation function

Epoch	Final loss	Run time (per epoch)
50	0.356822	1.44s $\pm$ 123ms
100	0.201677	2.29s $\pm$ 1.39ms
200	0.165149	1.51s $\pm$ 179ms

The approximation appears to be accurate even around the "difficult point" when the number of epochs is 200 (similar to MSE).

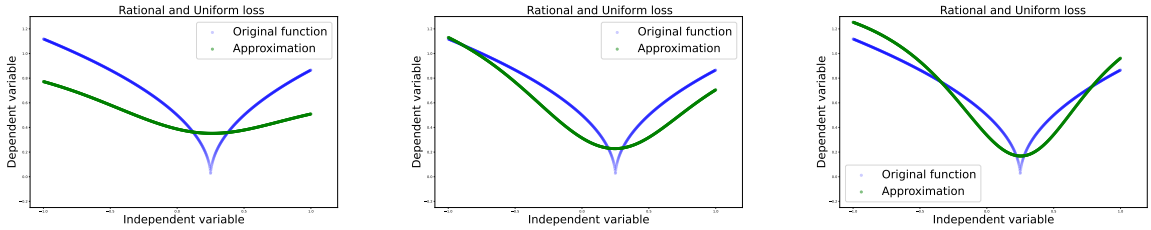

FIGURE 3. Approximation is computed by the usual training process, epoch is 50, 100, 200.

4.4. **Neural network with rational activation, learnt with split training method.** Similar to the network in Section 4.4, our activation function is a rational function of degree $(3,2)$ and the coefficients are a part of the parameter set. We learn these coefficients as we learn other parameters during the training procedure. However, the training process is done in three steps:

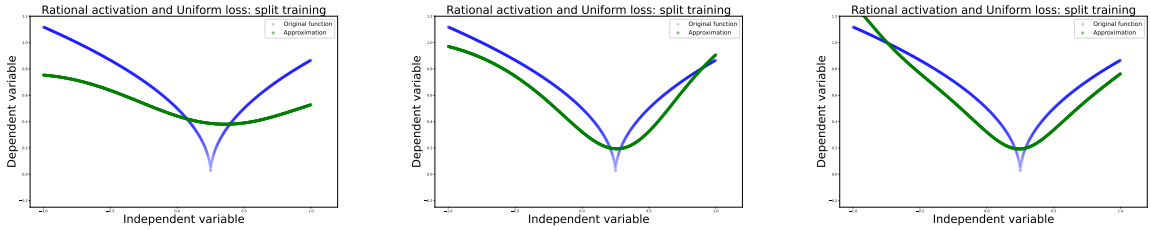

FIGURE 4. Approximation is computed by the split training process, 50, 100, 200 epochs.

(1) optimise the weights between the input and hidden layer,
(2) optimise the weights between the hidden layer and the output;
(3) optimise the rational coefficients.

This approach is based on a block coordinate-wise optimisation; therefore, even when it produces a sequence that decreases the functional values, the limit point is not guaranteed to be globally optimal.

Table 4 shows that the results are comparable for this method and for the cases of rational activation and rational approximation to the ReLU activation function (in terms of the optimal values of the loss function).

TABLE 4. Results: experiments with split method

Epoch	Final loss	Run time (per epoch)
50	0.365163	1.55s $\pm$ 268ms
100	0.182327	1.41s $\pm$ 156ms
200	0.182637	1.65s $\pm$ 319ms

Figure 4 shows with the increase of the number of epochs, does not improve the approximation.

Overall conclusion: the approximation results are more accurate when the rational activation function is used with the split training method rather than the usual training approach in section 4.3. These results are comparable with the results obtained in section 4.1 and 4.2. In particular, when the number of epochs is between 50 and 100. This section only includes the results with 10 nodes in the hidden layer. We refer readers to Appendix A for the results related to different number of hidden nodes.

5. RESULTS: DIRECT RATIONAL APPROXIMATION APPROACH

In this section, we produce the results for the Differential Correction and AAA Methods. All the experiments correspond to uniform loss. The decision variables are the coefficients of the rational function. We compare two main methods for constructing rational approximations: the differential correction method (converges to a global optimal solution) and the AAA method (practical method, converges to a "near optimal" solution). The codes for both methods are implemented in Python.

5.1. **The differential correction method.**

5.1.1. *Rational approximation of degree (21,20), the differential correction method.* Consider a neural network with three layers (input, hidden and output) where the hidden layer consists of 10 nodes and the activation function defined on the hidden layer is a rational function of degree $(3,2)$.**The overall output of this neural network is a rational function of degree (21,20).** Therefore, we can approximate the function from (3.1) by a rational function of degree $(21,20)$ using the differential correction method and compare this approximation with the results obtained by the above neural network with uniform loss function and 10 nodes in the hidden layer. In theory, the direct approximation by a rational function of degree $(21,20)$ is more flexible. Results related to the case where the number of nodes on the hidden layer is 2, can be found in Appendix B.

We also compute the rational approximations of degree $(20,20)$ and use this additional result to compare with the AAA method, which is expecting the same degree in the numerator and the denominator. Table 5 summarises the computational time and the optimal loss. The results demonstrate that the computational time is very small for all three settings and corresponding errors are very similar. Interestingly, the increase in the degree does not always improve the optimal loss function value. This can be explained as a result of computational instability when the dimension of the corresponding optimisation problems is increasing.

TABLE 5. Direct rational approximation for degrees $(21,20)$, $(20,20)$ and $(21,21)$ by the differential correction method.

Degrees	Error (uniform norm)	Run time
(21,20)	0.003123	46.7 s
(20,20)	0.003244	47.2 s
(21,21)	0.002815	47.5 s

Figure 5 depicts the approximations of degree (21,20), (20,20) and (21,21). The approximation is accurate, but there is a strong oscillation of the approximation, which is not surprising for high degree rational functions.

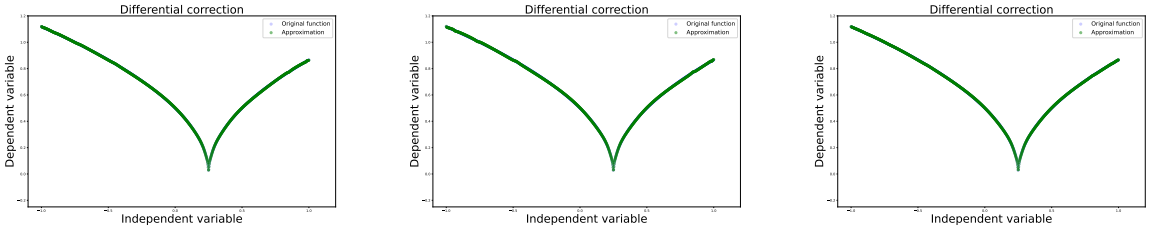

FIGURE 5. Approximation by the differential correction method, rational approximation degree is $(21,20)$, $(20,20)$ and $(21,21)$ respectively.

5.2. **The AAA method.**

5.2.1. *Rational approximation of degree (20,20) with the AAA method.* In order to compare the differential correction method with AAA, we apply the AAA method in the same experimental settings as they are in the case of the differential correction method. Due to the limitations of the

TABLE 6. Results: AAA method, degree (20,20) and (21,21)

Degrees	Error type	Error	Run time
(20,20)	Uniform error MSE	0.000261 0.00055	450ms
(21,21)	Uniform error MSE	3.679617e-06 4.84345e-05	496ms

AAA method, we cannot use different degrees for the numerator and the denominator. Hence, for these experiments, we use the rational approximation of degree (20,20) and the rational approximation of degree (21,21). Python codes that we utilised in this section were developed by C. Hofreither [50] and they are available online.

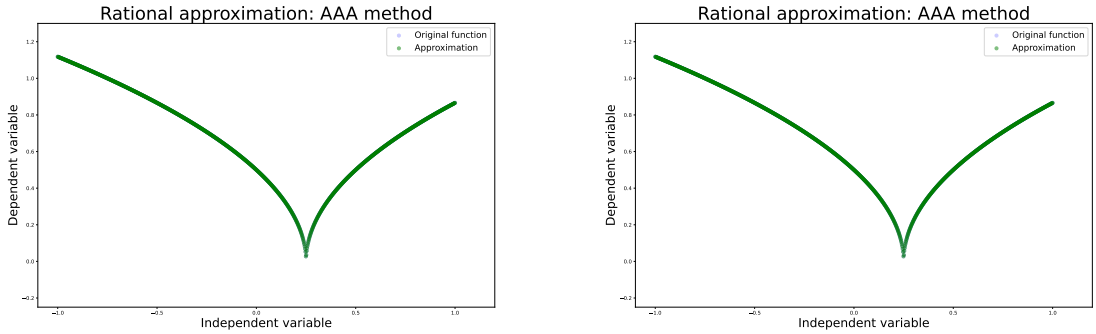

FIGURE 6. Approximation is computed by the AAA method, degree (20,20) and (21,21).

Table 6 and Figures 6 shows that the approximations are very accurate and the computational time is very low. These results suggest that the AAA method is very fast and accurate, even when the degree is high. The function and the approximation are indistinguishable from the picture.

Remark 5.1. Table 6 demonstrates that AAA produces more accurate approximations than the differential correction, while, unlike the differential correction, AAA does not claim to terminate at an optimal solution. This inconsistency can be explained by in a number of ways.

- The core of the differential correction method is in solving large linear programming problems. If the linear programming solvers encounter instability while trying to solve the corresponding linear programming problems, the differential correction method may not convert to the global solution.
- The differential correction requires the denominator of the corresponding rational approximation to be strictly positive in the domain, while AAA produces a number of poles close to the domain. This may be seen as an advantage, but at the same time it should be taken with care, especially when working on practical applications.

6. Results: Multivariate Domains

In this section, we approximate a solution to the KdV (Korteweg-de Vries) equation. The expressive power of neural networks have sparked an increasing interest in applying physics informed neural nets to solve partial differential equations such as the KdV equation [3, 2]. This equation appears in many applications, including fluid dynamics, waves and symmetry:

$$u_t = -uu_x - u_{xxx}, \quad u(x,0) = -\sin(\pi x/20)$$

by using four neural networks, which contain ReLU, sinusoidal, rational, and polynomial activation functions. In this section, our goal is to approximate u_i on (x_i,t_i) by using the multivariate bisection method and the algorithm was implemented in Matlab. The specifications of the data set are as follows:

- $(u_i)_{1\leq i\leq N}$ - observation data (known function values) of size 512×201.
- $(x_i,t_i)_{1\leq i\leq N}$ - spatio tempural points (known domain values).
- $x_i \in [0,40]$ with the step size of 1.
- $t_i \in [-20,19.84375]$ with the step size of 0.390625.

In [2], authors used Euclidean norm to compute the errors but here, we compute the error of the approximation in the uniform norm. Therefore, there are not many possibilities for comparison with the findings in [2]. All of our experiments are constructed by using a certain set of domain points (using every k^{th} point of each dimension and the corresponding function values). The approximation and the uniform error term are computed on the same reduced domain. This extra step has been taken to ensure that big data volume does not lead MATLAB programs to crash.

In our numerical experiments, we use direct rational approximations obtained by the bisection method. There are a number of reasons for this choice.

(1) The AAA method can not be used in multivariate settings.
(2) The bisection method demonstrated its efficiency in multivariate domains.
(3) The bisection method can handle constraint problems and this important feature is used to improve the computational stability.
(4) Theoretical studies of (generalised) rational approximations for multivariate domains demonstrate that the corresponding optimisation problems are pseudoconvex [49] and there are efficient methods to solve them.

6.1. **The original data set.** The following pictures (Figure 7, Figure 8, Figure 9) are of the original data set. Some images are constructed by using a certain set of domain points. In particular, we use the pairs of every 20^{th} point of the domain (along each of the two dimensions) and pairs of every 10^{th} point of the domain in order to reduce the dimension of the original problem. For simplicity, we sometimes refer to these pairs as "Every 10^{th} point of the domain" or "Every 20^{th} point of the domain".

Now, we compute rational approximations of different degrees and report the uniform approximation error and the computational time for each degree when $k = 20$ and $k = 10$. We start our experiments with the rational approximation of degree $(2,2)$ and keep increasing the degree up to $(20,20)$. The classical monomials in the rational function were replaced by Chebyshev monomials for better approximations.

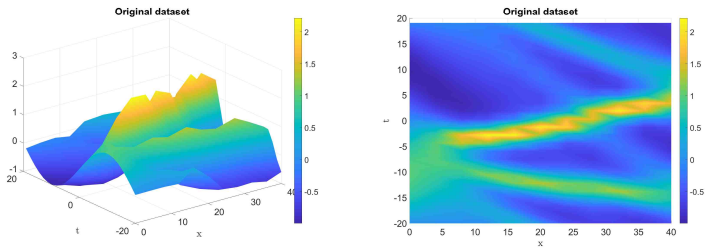

FIGURE 7. Pictures constructed for the original data set taking the pairs of every 20^{th} point of the domain: 3D and 2D view of the original data set

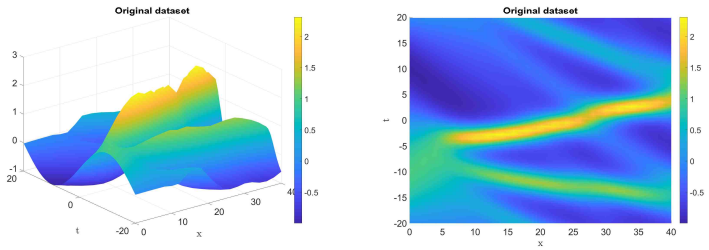

FIGURE 8. Pictures constructed for the original data set taking the pairs of every 10^{th} point of the domain: 3D and 2D view

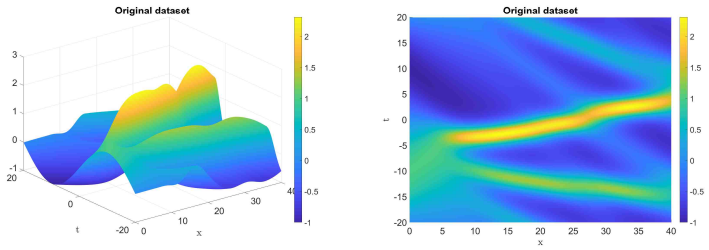

FIGURE 9. Pictures constructed for the original data set taking all the points of the domain: 3D and 2D view

We approximate the original function by rational approximations of different degrees and the results are reported in Table

For visulaisation purpose, we only report the 3D and 2D pictures of the rational approximation of degree (2,2) and degree (20,20) along with its error curve.

From Table 7, one can clearly see that the algorithm runs faster when the number of discretised points in the domain is smaller.

In the case of approximation by a rational function of degree (20,20), even though the degree of the rational function is higher, the number of parameters (the coefficients of the polynomials in the numerator and the denominator) is smaller (≈ 400) compared to the number of parameters in the neural networks that the authors of [2] used in their experiments (≈ 8035).

As the degree of the rational function increases, the condition number of the constraint matrices appearing in the auxiliary linear programming problem of the bisection method increases and in some cases, the MATLAB code tends to complain and crash. This issue can be solved

TABLE 7. Uniform error and computational time for different degrees and sampling densities of rational approximation.

Degree	Sampling density	Uniform error	Time (sec.)
(2,2)	Every 20^{th} point	1.294497	2.049840
	Every 10^{th} point	1.369915	2.504411
(5,5)	Every 20^{th} point	0.583136	7.823996
	Every 10^{th} point	0.609975	10.455391
(10,10)	Every 20^{th} point	0.172560	22.208227
	Every 10^{th} point	0.209718	42.384391
(18,18)	Every 20^{th} point	0.014062	66.460222
	Every 10^{th} point	0.034389	167.995076
(20,20)	Every 20^{th} point	0.010817	158.933609
	Every 10^{th} point	0.027058	438.548139

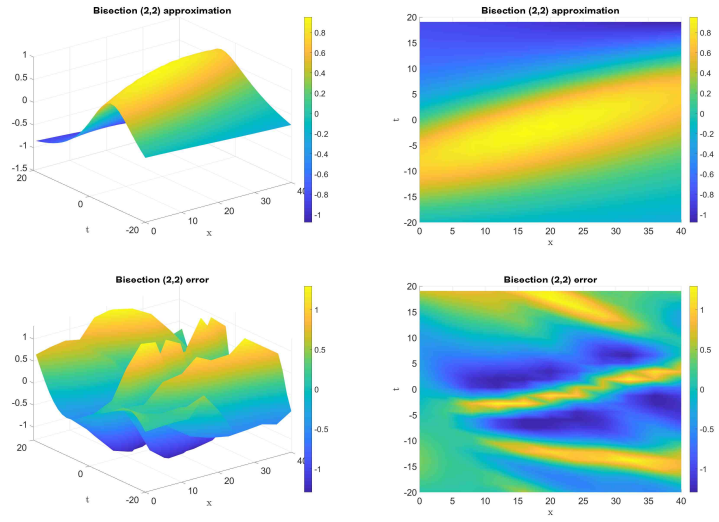

FIGURE 10. Approximations (above) and the error curves (below) are constructed on the domain which consists of pairs of every 20th point of the original domain: 3D and 2D view

by adding an extra linear constraint to the auxiliary problem of the bisection method which restricts the denominator polynomial from above. The bisection method has the flexibility of adding constraints and this additional adjustment allows us to compute higher degree rational approximations. Similar observation can be found in [19] where authors use this property in matrix function evaluations for univariate approximation (lifting matrix functions). The experiments in this section where the degree of the approximation is $(20, 20)$ were conducted with an additional constraint that restricts the denominator from above. The numerical experiments demonstrate that this extra improvement leads to more accurate approximations since the program can handle high degree polynomials for both the numerator and the denominator.

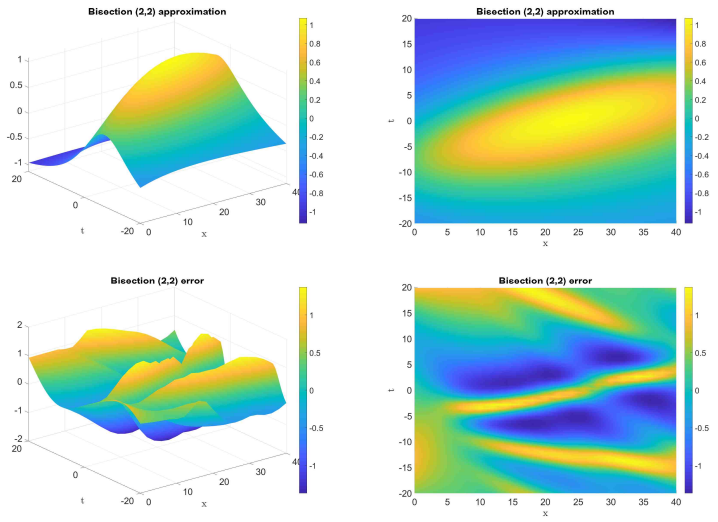

FIGURE 11. Approximations (above) and the error curves (below) are constructed on the domain which consists of pairs of every 10th point of the original domain: 3D and 2D view

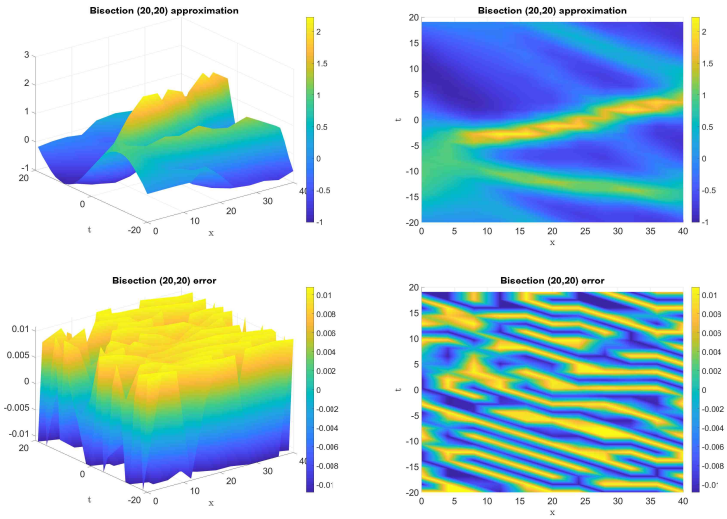

FIGURE 12. Approximations (above) and the error curves (below) are constructed on the domain which consists of pairs of every 20^{th} point of the original domain: 2D and 3D view.

Note that our results cannot be compared with the results in [2] since the errors are computed in different norms. However, we demonstrate that the rational approximation in a multivariate domain is as powerful as neural networks and it is capable of approximating complicated functions by using much fewer parameters than a neural network.

For each degree, we also computed the uniform error on the whole domain once the approximation is computed on a selected reduced domain. The results can be found in Table 8. Note that in the case of degree (20,20), the denominator is bounded from above.

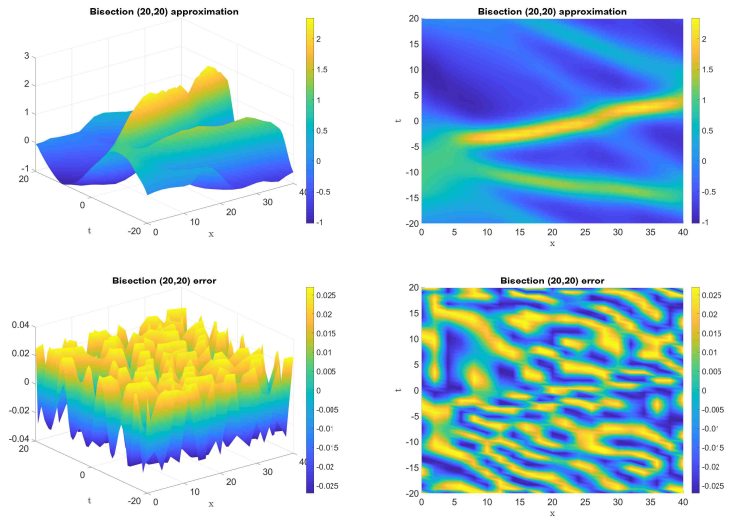

FIGURE 13. Approximations (above) and the error curves (below) are constructed on the domain which consists of pairs of every 10^{th} point of the original domain: 3D and 2D view.

TABLE 8. Uniform error computed on the whole domain

	Degree	Uniform error
Pairs of every 20^{th} point of the domain	(2,2)	1.490621
	(5,5)	1.575434
	(10,10)	245.346794
	(18,18)	11404.128402
	(20,20)	88023.910932
Pairs of every 10^{th} point of the domain	(2,2)	1.408030
	(5,5)	0.702071
	(10,10)	0.334155
	(18,18)	15.397669
	(20,20)	9.191410

One can clearly see that the uniform error computed on the whole domain is very similar to the uniform error computed on the domain with fewer selected points (reduced domain) when the degree of the approximation is small. This observation is very prominent in the case where the domain consists of pairs of every 10^{th} point of the domain. For higher degrees, the difference between the errors computed on the whole domain and the reduced domain gets much larger when the domain has much fewer points (when the domain has pairs of every 10^{th} point of the whole domain).

7. DISCUSSION, CONCLUSIONS AND FUTURE RESEARCH DIRECTIONS

In this paper, we compared several approaches to approximating functions, each being a slight modification of the other one. Starting from a classical neural network with a ReLU activation function, we gradually changed our set up, first by replacing the activation function

with a rational function approximating ReLU, then including the coefficients of the rational activation function in the learning parameters, and finally replacing the neural network with a best rational approximation of the same number of parameters using traditional optimisation approaches.

Our finding is that as each step increases the flexibility and size of the feasible space, we observe an improvement in the approximation. In particular, the optimisation-based approximation achieves better approximation than the neural network parts.

The training time is also in favour of the optimisation-based approximation, especially because it is often possible to achieve a good approximation with fewer parameters. As the dimension of the function to approximate increases, it is not clear whether this advantage remains.

Rational approximation (direct or in combination with other techniques) showed its efficiency. The AAA method is accurate for unconstraint approximation in univariate settings. For multivariate settings, the direct rational approximation showed its efficiency.

Our study did not consider other aspects of the approximation, such as generalisability. It is well known in general that neural network approximation has good generalisability properties and avoids overfitting. However, it is not clear whether adding extra flexibility (for example including the coefficients of the activation function in the learning parameters) retains those generalisability properties. Further work needs to be done to gain better insights about this.

Acknowledgements

The authors are grateful to the reviewers for useful suggestions which improved the contents of this paper. We are grateful to the Australian Research Council for supporting this work via Discovery Project DP180100602.

REFERENCES

[1] R. Lopez, E. Balsa-Canto, E. O˜nate, Neural networks for variational problems in engineering, Int. J. Numer. Meth. Eng. 75 (2008), 1341–1360.

[2] N. Boull'e, Y. Nakatsukasa, A. Townsend, Rational neural networks, Advances in Neural Information Processing Systems 33 (2020), 14243–14253.

[3] M. Raissi, P. Perdikaris, G.E. Karniadakis, Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations, J. Comput. Phy. 378 (2019), 686–707.

[4] M. Raissi, A. Yazdani, G.E. Karniadakis, Hidden fluid mechanics - Learning velocity and pressure fields from flow visualizations, Science, 367 (2000), 1026–1030.

[5] M. Leshno, Y.L. Vladimir, A. Pinkus, S. Schocken, Multilayer feedforward networks with a nonpolynomial activation function can approximate any function, Neural Netw. 6 (1993), 861–867.

[6] N.L. Trefethen, Approximation Theory and Approximation Practice, extended edition, Society for Industrial and Applied Mathematics, 2019.

[7] Y. Bai, X. Lu, L. Zhang, Function approximations via $l_1 - l_2$ optimization, J. Appl. Numer. Optim. 6 (2024), 371-389.

[8] A. Alimov, I. Tsar'kov, Geometric Approximation Theory, Springer, Switzerland, 2021.

[9] H.L. Loeb, On rational fraction approximations at discrete points, convair astronautics, math, The Computer Journal 9, 1957.

[10] T. J. Rivlin, Polynomials of best uniform approximation to certain rational functions, Numer. Math. 4 (1962), 345-349.

[11] A. Ralston, Rational Chebyshev Approximation by Remes' Algorithms, Numer. Math. 7 (1965), 322-330.

[12] P. Petrushev, V. Popov, Rational Approximation of Real Functions, Cambridge University Press, Cambridge, 1987.
[13] S. Boyd, L. Vandenberghe, Convex Optimization, Cambridge University Press, New York, 2010.
[14] H.L. Loeb, Algorithms for chebyshev approximations using the ratio of linear forms, J. Soc. Ind. Appl. Math. 8 (1960), 458–465.
[15] V. Peiris, N. Sharon, N. Sukhorukova, J. Ugon, Generalised rational approximation and its application to improve deep learning classifiers, Appl. Math. Comput. 389 (2021), 125560.
[16] J.P. Crouzeix, Conditions for convexity of quasiconvex functions, Math. Oper. Res. 5 (1980), 120–125.
[17] J.X.D.C. Neto, J.O. Lopes, M.V. Travaglia, Algorithms for quasiconvex minimization, Optimization 60 (2011), 1105–1117.
[18] V. Peiris, N. Sukhorukova, The extension of the linear inequality method for generalized rational chebyshev approximation to approximation by general quasilinear functions, Optimization 71 (2011), 999–1019.
[19] N. Sharon, V. Peiris, N. Sukhorukova, J. Ugon, Flexible rational approximation and its application for matrix functions, arXiv preprint arXiv:2108.09357, 2023.
[20] R. Guidotti, A. Monreale, S. Ruggieri, F. Turini, F. Giannotti, D. Pedreschi, A survey of methods for explaining black box models, ACM Comput. Surv. 51 (2018), 93.
[21] S. Gratton, V. Mercier, E. Riccietti, P.L. Toint, A block-coordinate approach of multi-level optimization with an application to physics-informed neural networks, Comput. Optim. Appl. 89 (2024), 385–417.
[22] D.H. Benjamin, R. Vidal, Global optimality in neural network training, In: 2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, pp. 4390–4398, IEEE, Honolulu, 2017.
[23] I. Goodfellow, Y. Bengio, A. Courville, Deep Learning, MIT Press, Massachusetts, 2016
[24] R.-Y. Sun, Optimization for deep learning: An overview, J. Oper. Res. Soc. Chin. 8 (2020), 249–294.
[25] G. Cybenko, Approximation by superpositions of a sigmoidal function, Math. Control Signals Sys. 2 (1989), 303–314.
[26] K. Hornik, Approximation capabilities of multilayer feedforward networks, Neural Netw. 4 (1991), 251–257.
[27] A. Pinkus, Approximation theory of the MLP model in neural networks, Acta Numer. 8 (1999), 143–195.
[28] A. Pinkus, Approximation theory of the MLP model in neural networks, Acta Numer. 8 (19999), 143–195.
[29] V. Arnold, On functions of three variables, Dokl. Akad. Nauk SSSR 114 (1957), 679–681.
[30] A.N.Kolmogorov, On the representation of continuous functions of many variables by superposition of continuous functions of one variable and addition, Dokl. Akad. Nauk SSSR 114 (1957), 953–956.
[31] M. Telgarsky, Neural networks and rational functions, In: International Conference on Machine Learning, pp. 3387–3393, PMLR, 2017,
[32] V. Peiris, V. Roshchina, N. Sukhorukova, Artificial neural networks with uniform norm-based loss functions, Adv. Comput. Math. 50 (2004), 31.
[33] Y. Makovoz, Uniform approximation by neural networks, J. Approx. Theory 95 (1998), 215–228.
[34] A. Molina, P. Schramowski, K. Kersting, Padè activation units: End-to-end learning of flexible activation functions in deep net, In: International Conference on Learning Representations (ICLR), 2019.
[35] V. Peiris, Rational activation functions in neural network with uniform norm based loss function and its application in classification, Commun. Optim. Theory 2022 (2022), 3.
[36] N.I. Achieser, Theory of Approximation, Frederick Ungar, New York, 1965.
[37] B. Boehm, Functions whose best rational chebyshev approximation are polynomials, Numer. Math. 6 (1964), 235–242.
[38] E.W. Cheney, H.L. Loeb, Generalized rational approximation, J. Soc. Ind. Appl. Math. B 1 (1964), 11–25.
[39] G. Meinardus, GApproximation of Functions: Theory and Numerical Methods, Springer, Heidelberg, 1967.
[40] Y. Nakatsukasa, O. Sete, L. N. Trefethen, The rational algorithm for rational approximation, SIAM J. Sci. Comput. 40 (2018), A1494-A1522.
[41] I. Barrodale, M. Powell, F. Roberts, The differential correction algorithm for rational l^∞-approximation, SIAM J. Numer. Anal. 9 (1972), 493–504.
[42] C.M. Lee, F.D.K. Roberts, A comparison of algorithms for rational l^∞ approximation, Math. Comput. 27 (1973), 111–121.
[43] M.R. Osborne, G.A. Watson, An algorithm for minimax approximation in the nonlinear case, The Computer Journal 12 (1969), 63–68.

[44] T.H. Pham, On approximate efficient solutions of nonsmooth minimax programming problems involving inequality and equality constraints with applications, J. Appl. Numer. Optim. 7 (2025), 233-252.
[45] M. Mabrouk, M. Laghdir, A. Ed-dahdah, A. Rikouane, Sequential approximate solutions for robust fractional optimization problems, J. Appl. Numer. Optim. 7 (2025), 215-231.
[46] E.W. Cheney, T. Southard, A survey of methods for rational approximation, with particular reference to a new method based on a forumla of darboux, SIAM Rev. 5 (1963), 219–231.
[47] A. Aghili, N. Sukhorukova, J. Ugon, Bivariate rational approximations of the general temperature integral, J. Math. Chemistry 59 (2021), 2049–2062.
[48] K. Park, Bivariate n-term rational approximation, J. Approx. Theory 136 (2005), 60–83.
[49] R. Díaz Millán, V. Peiris, N. Sukhorukova, J. Ugon, Multivariate approximation by polynomial and generalized rational functions, Optimization 71 (2022), 1171–1187.
[50] C. Hofreither, An algorithm for best rational approximation based on barycentric rational interpolation, Numer. Algo. 88 (2021), 365–388.

A. Results: Neural Network-based Approximation.

In this section, we discuss results of neural network based approximations where the network has either 2 or 10 nodes in the hidden layer. We use MSE loss and also uniform norm-based loss function with corresponding optimiser: ADAM or ADAMAX.

A.1. **Neural Network with ReLU Activation.** Table 9 summarises the results for the case of 2 nodes in the hidden layer. One can see that the value of the loss function does not improve significantly when the number of epochs is increasing. Overall, the computational time (per epoch) does not increase significantly when the number of epochs increases from 50 to 200. The last column of this table reports the running time per epoch and therefore the computational time is almost proportional to the number of epochs. Therefore, the large number of epochs increases the overall computational time proportionally without achieving significantly better accuracy.

TABLE 9. Performance of a neural network with ReLU activation.

No. of Nodes	Loss function	Optimiser	Epochs	Final loss	Run time (per epoch)
2	MSE	ADAM	50	0.048732	$1.20\,\text{s} \pm 0.124\,\text{s}$
			100	0.045840	$1.39\,\text{s} \pm 0.197\,\text{s}$
			200	0.045839	$1.46\,\text{s} \pm 0.159\,\text{s}$
2	Uniform	ADAMAX	50	0.474607	$1.42\,\text{s} \pm 0.128\,\text{s}$
			100	0.469579	$1.69\,\text{s} \pm 0.592\,\text{s}$
			200	0.470377	$1.64\,\text{s} \pm 0.329\,\text{s}$
10	MSE	ADAM	50	0.021143	$1.73\,\text{s} \pm 0.234\,\text{s}$
			100	0.007018	$1.82\,\text{s} \pm 0.262\,\text{s}$
			200	0.002857	$1.62\,\text{s} \pm 0.261\,\text{s}$

For the sake of completeness, we report the figures related to NN approximations where the loss function is based on MSE. Figure 14 clearly suggests that the increase in the number of nodes in the hidden layer improved the approximation accuracy and this accuracy is much better for 100 or 200 epochs. The inaccuracy near the "difficult point" $x = 0.25$, where $f(x)$ is nonsmooth and non-Lipschitz is still significant, even for a larger number of epochs.

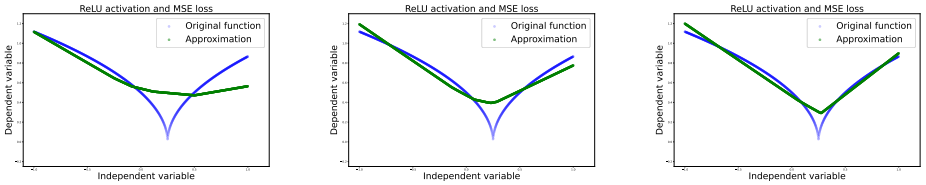

FIGURE 14. Approximation is computed by a neural network with 10 nodes in the hidden layer, ReLU activation, MSE loss and 50, 100 and 200 epochs.

A.2. **Neural Network with Rational Approximation to ReLU Activation.** In this section, we present the results of numerical experiments, where the experiment settings are similar to those in section A.1, but the activation function is the rational approximation of degree (3,2) to ReLU. The results reported in Table 10 clearly indicate that this approach is almost the same as the direct use of ReLU as the activation function with 2 nodes in the hidden layer. However, further improvement in the accuracy when the number of nodes in the hidden layer is increased from 2 to 10. This observation is especially prominent when the number of epochs is 200.

TABLE 10. Performance of a neural network with Rational approximation to ReLU activation.

No. of Nodes	Loss function	Optimiser	Epochs	Final loss	Run time (per epoch)
2	MSE	ADAM	50	0.053184	$1.47\,\text{s} \pm 0.204\,\text{s}$
			100	0.050132	$1.38\,\text{s} \pm 0.149\,\text{s}$
			200	0.048847	$1.56\,\text{s} \pm 0.218\,\text{s}$
2	Uniform	ADAMAX	50	0.488380	$1.42\,\text{s} \pm 0.131\,\text{s}$
			100	0.474617	$1.37\,\text{s} \pm 0.133\,\text{s}$
			200	0.472513	$1.48\,\text{s} \pm 0.201\,\text{s}$
10	MSE	ADAM	50	0.017438	$2.15\,\text{s} \pm 0.547\,\text{s}$
			100	0.005941	$2.07\,\text{s} \pm 0.412\,\text{s}$
			200	0.003010	$1.75\,\text{s} \pm 0.415\,\text{s}$

Figure 15 illustrates that the accuracy of approximation is improving, even for the "difficult point", especially when the number of epochs is 200.

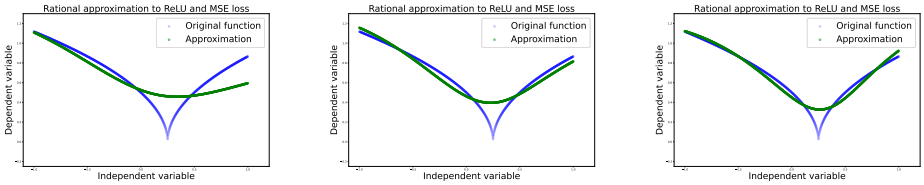

FIGURE 15. Approximation is computed by a neural network with 10 nodes in the hidden layer, rational approximation to ReLU activation, MSE loss and 50, 100 and 200 epochs.

A.3. **Neural Network with Rational Activation.** In this case, our activation function is a rational function of degree $(3,2)$. The coefficients of the rational activation function are now a part of the parameter set. We learn these coefficients as we learn other parameters during the training procedure. Table 11 shows that the values of the loss function are similar to the case of ReLU and comparable with the results in the case of rational approximation to ReLU. The optimal loss function value decreases with the increasing epoch. Figure 16 confirms this observation.

It is important to note that the increase in the number of nodes in the hidden layer does not increase significantly the computational time.

TABLE 11. Performance of rational neural network.

No. of Nodes	Loss function	Optimiser	Epochs	Final loss	Run time (per epoch)
2	MSE	ADAM	50	0.049027	$1.7\,\text{s} \pm 0.252\,\text{s}$
			100	0.045914	$1.56\,\text{s} \pm 0.170\,\text{s}$
			200	0.030181	$1.54\,\text{s} \pm 0.158\,\text{s}$
2	Uniform	ADAMAX	50	0.485671	$1.57\,\text{s} \pm 0.208\,\text{s}$
			100	0.473893	$1.53\,\text{s} \pm 0.236\,\text{s}$
			200	0.466458	$1.58\,\text{s} \pm 0.195\,\text{s}$
10	MSE	ADAM	50	0.022051	$1.45\,\text{s} \pm 0.176\,\text{s}$
			100	0.006272	$1.58\,\text{s} \pm 0.462\,\text{s}$
			200	0.002737	$1.57\,\text{s} \pm 0.347\,\text{s}$

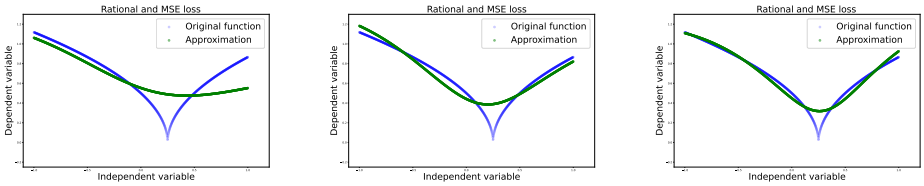

FIGURE 16. Approximation is computed by rational neural network with 10 nodes in the hidden layer, the usual training process, MSE loss and 50, 100 and 200 epochs.

A.4. **Neural Network with Rational Activation, Learn with Split Training Method.** In this section, our activation function is a rational function of degree $(3,2)$. The coefficients are now a part of the parameter set. We learn these coefficients as we learn other parameters during the training procedure using a split method.

The results in Table 12 shows that 2 nodes in the hidden layer are slightly better for the split method than they are in the case of rational activation function or rational approximation to ReLU activation in terms of the optimal loss function value in the case of 200 epochs. The computational time is slightly lower for the split method in many instances, but the difference is very insignificant.

When the number of nodes in the hidden layer increases to 10, the approximation accuracy in terms of the optimal loss function value is improving and much better than rational neural networks.

Figure 17 shows that the approximation inaccuracy mostly appears at the "difficult point" and there are some oscillations when the number of epochs is 200.

Overall conclusion: rational activation functions with split training computes better approximations in most cases, even when the loss function is MSE.

TABLE 12. Performance of rational neural network with split training.

No. of Nodes	Loss function	Optimiser	Epochs	Final loss	Run time (per epoch)
2	MSE	ADAM	50	0.049547	$1.41\,\text{s} \pm 0.160\,\text{s}$
			100	0.045311	$1.51\,\text{s} \pm 0.200\,\text{s}$
			200	0.039817	$2.06\,\text{s} \pm 0.438\,\text{s}$
2	Uniform	ADAMAX	50	0.499596	$1.7\,\text{s} \pm 0.294\,\text{s}$
			100	0.481489	$1.56\,\text{s} \pm 0.174\,\text{s}$
			200	0.426179	$1.56\,\text{s} \pm 0.237\,\text{s}$
10	MSE	ADAM	50	0.007860	$1.41\,\text{s} \pm 0.196\,\text{s}$
			100	0.002467	$1.57\,\text{s} \pm 0.345\,\text{s}$
			200	0.001783	$1.51\,\text{s} \pm 0.228\,\text{s}$

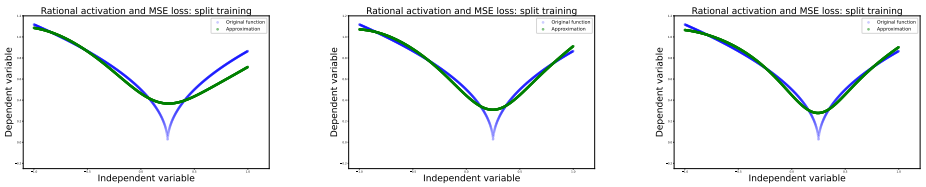

FIGURE 17. Approximation is computed by rational neural network with 10 nodes in the hidden layer, split training process, MSE loss and 50, 100 and 200 epochs.

B. DIRECT RATIONAL APPROXIMATION APPROACH: THE DIFFERENTIAL CORRECTION AND AAA METHOD.

In this section, all the experiments correspond to uniform loss. We compare two main methods for constructing rational approximations: the differential correction method and the AAA method.

B.1. **The Differential Correction Method.**

B.1.1. **Rational approximation of degree (5,4), the differential correction method.** When one considers a neural network with three layers and just 2 nodes in the hidden layer and the rational activation function of degree $(3,2)$, the output is a rational function of degree $(5,4)$. Therefore, we can approximate the function from (3.1) by a rational function of degree $(5,4)$ using the differential correction method and compare this approximation with the results obtained for uniform loss function and 2 nodes in the hidden layer. In theory, the direct approximation by a rational function of degree $(5,4)$ is more flexible. We use Python in our experiments. We also compute the rational approximations of degree $(4,4)$ and $(5,5)$ for comparison. We will also use these additional results to compare with the AAA method, which is expecting the same degree in the numerator and the denominator. Table 13 summarises the computational time and the optimal loss. The results demonstrate that the computational time is very small for all three settings and corresponding errors are very similar.

TABLE 13. Results: direct rational approximation for degrees $(5,4)$, $(4,4)$ and $(5,5)$ by differential correction method

Degrees	Error (uniform norm)	Run time
(5,4)	0.023819	465 ms
(4,4)	0.028614	1.15 s
(5,5)	0.021469	1.99 s

Figure 18 depicts the function and the corresponding rational approximation of degree (4,4), (5,4) and (5,5).

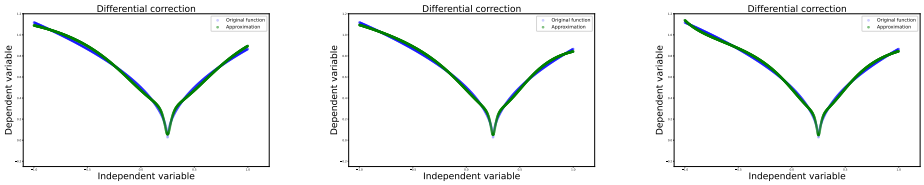

FIGURE 18. Approximation by the differential correction method, rational approximation degree is $(4,4)$, $(5,4)$ and $(5,5)$ respectively.

B.2. **The AAA Method.**

B.2.1. **Rational approximation of degree (5,4) with the AAA method**. In order to compare the differential correction method with AAA, we apply the AAA method in the same experimental settings as they are in the case of the differential correction method. Due to the limitations of the AAA method, we cannot use different degrees for the numerator and the denominator. Hence, for these experiments, we use the rational approximation of degree (4,4) and the rational approximation of degree (5,5). The codes are developed in Python by C. Hofreither [50].

Table 14 summarises the results for the AAA method. The AAA method is designed for uniform loss, but we report the corresponding MSE loss as well for comparison. One can see that the optimal loss is high. This is due to instability. Unfortunately, this instability cannot be fixed by limiting the condition number (differential correction).

TABLE 14. Results: AAA method, degrees (4,4) and (5,5)

Degrees	Error type	Error	Run time
(4,4)	Uniform error	0.142291	45.9 ms
	MSE	1.111253	
(5,5)	Uniform error	0.898891	60.2ms
	MSE	1.228715	

Figure 19 highlights the difficulties of approximations. These difficulties are especially prominent in the case of degree (4,4).

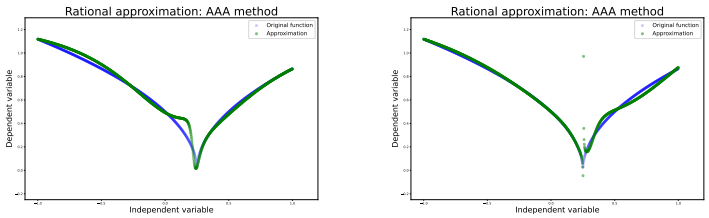

FIGURE 19. Approximation is computed by the AAA method, degree (4,4) and (5,5).