Proceedings of ALGORITMY 2020
pp. 21–30

SOLVING ORDINARY DIFFERENTIAL EQUATIONS USING ARTIFICIAL NEURAL NETWORKS - A STUDY ON THE SOLUTION VARIANCE

TONI SCHNEIDEREIT* AND MICHAEL BREUß

Abstract. Solving differential equations can be realised with simple artificial neural network architectures. Several methods make use of trial solutions with different construction approaches and can provide reliable results. However, many parameters, different optimisation methods and random weight initialisation result in a non constant variance to the exact solution. To our knowledge, this variance has not been studied yet. We investigate several parameters and constant versus random weight initialisation for two solution methods to determine their reliability with the use of backpropagation and ADAM optimisation.

Key words. ordinary differential equations, artificial neural networks, trial solution, backpropagation, ADAM optimisation

AMS subject classifications. 62M45, 65L05, 90C90

1. Introduction. The link between mathematics and physical phenomena is often formulated by differential equations (DEs), which combine functions and their derivatives to produce a mathematical model of real world behaviour. They often do not have an analytical solution, but can be approximated by numerical methods [1] like finite differences, finite elements or Runge-Kutta methods, and also with artificial neural networks (ANNs) [2]. Focusing on ANNs, their structure is based on models of biological neural networks with different layers containing neurons as processing units and connecting weights inbetween, representing synapses.

In order to solve ODEs with ANNs, several methods make use of a trial solution (TS) for optimisation. The TS structure itself varies depending on the model. An approach we simply call trial solution method (TSM) [4], introduced a TS as a sum of two parts, which satisfies initial/boundary conditions (ICs/BCs) by construction. Furthermore, it contains the ANN output and is an essential part of the cost funtion. Same TS structure is used for example in the Legendre neural network [3], which features an expansion to Legendre polynomials. A different TS, proposed in [5], we call modified trial solution method (mTSM). This approach declares the TS to be the ANN output and ICs/BCs appear as additional terms in the cost function.

Our goal is to perform fundamental research on solving ODEs using ANNs in order to obtain further knowledge about this topic and to perhaps improve the competitiveness. Released in 2019, we want to find out, if [5] can provide new impact on this topic.

Optimisation methods we use in this paper are backpropagation (BP) [7] and ADAM [9]. Both make use of the (negative) cost function gradient, which provides information about the direction of steepest descent. While BP can have a constant or adaptive learning rate, ADAM was developed to be an adaptive optimisation method. The learning rate in general is a scaling factor for the gradient. An advantage of ADAM is the potential of rapid training speed, but perhaps it gets outperformed by

*Applied Mathematics Group, BTU Cottbus-Senftenberg, Platz der Deutschen Einheit 1, 03046 Cottbus, Germany, {Toni.Schneidereit,breuss}@b-tu.de

non adaptive methods due to their out-of-sample behaviour and sometimes it may even fail to converge [10]. On the other hand, a disadvantage of BP is the uniform scaling of the gradient in all directions.

The amount of parameters for ODE, ANN and optimisation is numerous. Our contribution in the main part of this paper is a study of differences in weight initialisation with zeros and small random values, as well as number of the ANN training cycles, number of training data, the domain size for ODE solution, an ODE parameter variation and finally a direct comparison between the optimisation methods. Prior to this we point out the ANN architecture, solution approaches and how the optimisation works. We finish the paper with a conclusion and remarks on future work.

2. ANN architecture and optimisation. Our ANN features three layers, one input layer (IL) to provide data into the ANN, one hidden layer (HL) for processing and one output layer (OL) to generate an output of the processed data. While IL and OL usually consist of linear processing units, the HL has nonlinear units. These units are called neurons. Fig. 2.1 displays the architecture of this ANN with one IL bias neuron, which can be considered as an offset, and H as the number of HL neurons.

Each neuron is connected through weights to every single neuron in the next layer. That is, w_j from IL to HL, u_j from IL bias to HL and v_j from HL to OL with $j{=}1,\dots,H$. All weights are stored in the vector $\vec{p}$. The ANN is fed by input data $x{\in}D{=}[a,b]{\subset}\mathbb{R}$, which is passed to the HL by neuron σ_{11}. Every hidden layer neuron $\sigma_{j2},\ j{=}1,\dots,H$ receives a weighted sum as input $z_j{=}w_j\sigma_{11}{+}u_j,\ j{=}1,\dots,H$ which is then processed by an activation function.

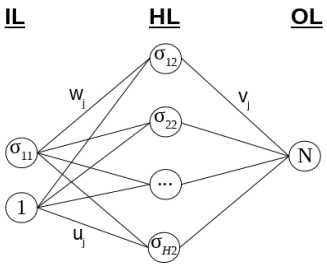


Fig. 2.1: ANN architecture featuring one input layer, one hidden layer with sigmoid activation functions and one output layer

There are different activiation functions in the field of ANNs, like the sigmoid function $\sigma(x){=}1/(1+e^{-x})$, a continous and arbitrarily often differentiable function with values between 0 and 1. The universal approximation theorem [11] states, that one HL with a finite number of sigmoidal activation functions is able to approximate every continous function on a subset of $\mathbb{R}$. Other useful activation functions are e.g. rectified linear unit (ReLU) or hyperbolic tangent (tanH). The ANN output is then generated by a linear activation function and the weighted sum from the HL

$$N(x,\vec{p}) = \sum_{j=1}^{H} v_j \sigma(z_j).$$

In general, it is common to initialise the weights as small random values [6], therefore the first computation of $N(x,\vec{p})$ returns an arbitrary value. This value is used to compute the *cost* or *loss* function $E[\vec{p}]$. For supervised learning, where both input data $x_i,\ i{=}1,\dots,n$ and correct output data $d_i,\ i{=}1,\dots,n$ are known, the cost function may be chosen as the squared l_2-Norm

$$E[\vec{p}] = \frac{1}{2n}\sum_{i=1}^{n} \left\| d_i - N(x,\vec{p})\big|_{x=x_i} \right\|_2^2,$$

while in case of unsupervised learning, where no correct output data is known, the cost function is part of the modeling process.

For cost function minimisation, the adjustable weights are subject to optimisation. A commonly used optimisation technique, based on gradient descent, is backpropagation, which uses the cost function gradient with respect to ANN weights to determine their influence on $N(x,\vec{p})$ and to update them. The updating process is also called training and is usually done several times with all input data. After one complete iteration through all input data, one epoch of training is done and for efficient training (finding a minimum in the weight space), several epochs of training are required. For k-th epoch, backpropagation with momentum update rule [8] reads

$$\vec{p}(k+1) = \vec{p}(k) \underbrace{-\alpha \frac{\partial E[\vec{p}(k)]}{\partial \vec{p}(k)} + \beta \Delta \vec{p}(k-1)}_{\Delta \vec{p}(k)} .$$

Learning rate α can be constant (cBP) or variable (vBP), which may prevent the optimiser from oscillating around a minimum. The momentum term, with momentum parameter β, uses impact from last epoch to not get stuck in a local minimum or saddle point. Different approaches for learning rate control exist [12], we employ the linear decreasing model

$$\alpha(k) = \begin{cases} \alpha_0 - \frac{\alpha_0 - \alpha_e}{k_c} k, & k \leq k_c \\ \alpha_e, & k > k_c \end{cases} \tag{2.1}$$

with an initial learning rate α_0, a final learning rate α_e and an epoch cap k_c.

ADAM (adaptive moment estimation) is an adaptive optimisation method for ANNs based on cost function gradient as well. In order to compute learning rates for each weight, it uses estimations of first (mean) and second (uncentered variance) moments of the gradient, see [9] for details.

3. Solutions Methods. In this section, we will describe the trial solution construction for the already mentioned methods more in detail, as well as the approaches on how to make use of ANNs in order to solve ODEs.

3.1. Trial Solution Method. TSM is suitable to solve both ODEs and PDEs as well as systems of ODEs and PDEs. In order to satisfy ICs/BCs, TS is constructed to satisfy these conditions. Since we will only take a first order ODE into account, following [4], suppose

$$G\left(x, u(x), \frac{d}{dx}u(x)\right) = 0, \quad x \in D \subset \mathbb{R}, \tag{3.1}$$

with given ICs/BCs. In Eq. (3.1), $u(x)$ denotes the exact solution function with x as independent variable. As previous mentioned, TS $u_t(x,\vec{p})$ is constructed to satisfy ICs/BCs, so it can be written as a sum of two parts

$$u_t(x,\vec{p}) = A(x) + B(x)N(x,\vec{p}). \tag{3.2}$$

In Eq. (3.2), $A(x)$ is supposed to satisfy ICs/BCs, while $B(x)$ is constructed to become zero at these points to eliminate the impact of $N(x,\vec{p})$. Now, TS transforms Eq. (3.1) into

$$G\left(x, u_t(x,\vec{p}), \frac{\partial}{\partial x}u_t(x,\vec{p})\right) = 0, \tag{3.3}$$

which requires the partial derivative of TS with respect to input x.

In order to generate training data for the ANN, the collocation method is used to discretize the domain D into an uniform grid with n gridpoints. Eq. (3.3) with the discrete domain is now subject to unconstraint optimisation problem $\min_{\vec{p}}$ G, which results in the cost function

$$E[\vec{p}] = \frac{1}{2n}\sum_{i=1}^{n}\left\| G\left(x_i, u_t(x,\vec{p})\big|_{x=x_i}, \frac{\partial}{\partial x}u_t(x,\vec{p})\big|_{x=x_i}\right)\right\|_2^2.$$

3.2. Modified Trial Solution Method. A variation of TSM, proposed in [5], introduces

$$u_t(x,\vec{p}) = N(x,\vec{p}), \tag{3.4}$$

an uniform TS for all DEs. Therefore it does not satisfy ICs/BCs by construction, they rather appear in the cost function

$$\begin{aligned} E[\vec{p}] = &\frac{1}{2n}\sum_{i=1}^{n}\left\| G\left(x_i, u_t(x,\vec{p})\big|_{x=x_i}, \frac{\partial}{\partial x}u_t(x,\vec{p})\big|_{x=x_i}\right)\right\|_2^2 \\ &+ \frac{1}{2m}\sum_{l=1}^{m}\left\| \left(u_t(x,\vec{p})\big|_{x=x_l} - K(x)\big|_{x=x_l}\right)\right\|_2^2, \end{aligned}$$

as additional terms with $K(x_l),\ l{=}1,\ldots,m$ as ICs/BCs.

4. Experiments and Results. Experiments on ANN solution with TSM and mTSM are based on the difference to the exact solution $u_{exact}(x) = e^{\lambda x}$ of

$$\frac{d}{dx}u(x) = \lambda u(x), \quad u(0) = 1, \tag{4.1}$$

a homogenous first order ODE with $\lambda{\in}\mathbb{R}_-$, which is well studied for stability [13].

The numeric error shown in subsequent diagrams is defined as

$$\Delta u = \frac{1}{n}\sum_{i=1}^{n}\left| u_{exact}(x)\big|_{x=x_i} - u_t(x,\vec{p})\big|_{x=x_i}\right|.$$

For TSM the TS reads $u_t(x,\vec{p}) = 1 + xN(x,\vec{p})$ and the related cost function

$$E[\vec{p}] = \frac{1}{2n}\sum_{i=1}^{n}\left\| N(x,\vec{p})\big|_{x=x_i} + x_i\frac{\partial}{\partial x}N(x,\vec{p})\big|_{x=x_i} - \lambda\left(1 + x_iN(x,\vec{p})\big|_{x=x_i}\right)\right\|_2^2.$$

For mTSM the TS $u_t(x,\vec{p}) = N(x,\vec{p})$ results in the cost function

$$E[\vec{p}] = \frac{1}{2n}\sum_{i=1}^{n}\left\| \frac{\partial}{\partial x}N(x,\vec{p})\big|_{x=x_i} - \lambda N(x,\vec{p})\big|_{x=x_i}\right\|_2^2 + \frac{1}{2}\left\| N(x,\vec{p})\big|_{x=x_1} - 1\right\|_2^2.$$

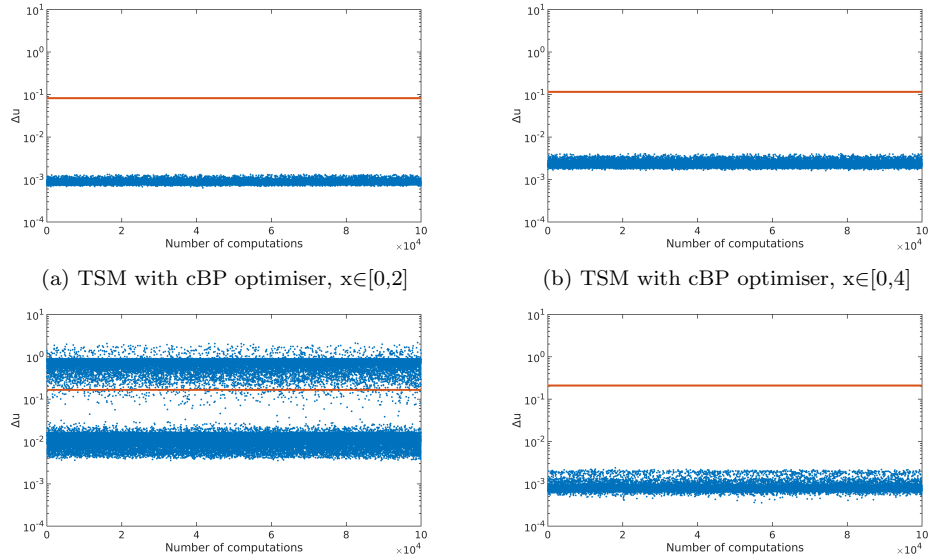
In subsequent experiments we study Δu with respect to domain size, k_{max}, ntD, different λ and optimisation methods.

In some diagrams we show the contrast between computations with $\vec{p}_{rnd}^{\,init}$ (in range of 0 to 1e-2) and $\vec{p}_{zero}^{\,init}$, which returns constant results, while $\vec{p}_{rnd}^{\,init}$ causes fluctuations and variations in every computation.

Previous experiments, not documented here, pointed out that one HL, with one IL bias neuron and five HL neurons, returns sufficient results for $\vec{p}_{rnd}^{\,init}$

Weights initialised to small random values	$\vec{p}_{rnd}^{\,init}$
Weights initialised to zero	$\vec{p}_{zero}^{\,init}$
Number of maximal epochs	k_{max}
Number of training data	ntD
Mean value of numeric error for $\vec{p}_{rnd}^{\,init}$	$\overline{\Delta u}$

Table 4.1: Important abbreviations used in the experiment section



(a) TSM with cBP optimiser, x∈[0,2]

(b) TSM with cBP optimiser, x∈[0,4]

(c) TSM with cBP optimiser, x∈[0,8]

(d) mTSM with cBP optimiser, x∈[0,8]

Fig. 4.1: **Experiment 4.1**. Domain variation with k_{max}=1e5, ntD=10 and λ=-5, (orange/solid) $\vec{p}_{zero}^{\,init}$, (blue/dotted) $\vec{p}_{rnd}^{\,init}$

versus the computational effort. In addition, the universal approximation theorem justifies the use of one HL with sigmoid activation function $\sigma(x)=1/(1+e^{-x})$, too.

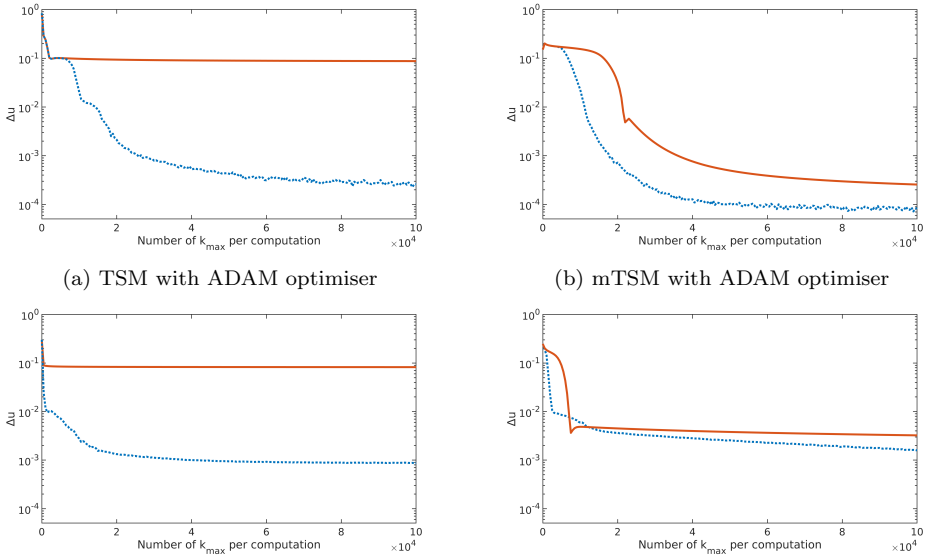
The ANN, the solution methods and the optimiser were implemented in Fortran 03/08, by following the proposed methods in the corresponding papers, without the use of deep learning libraries. Therefore we are able to perform investigations related to every aspect of the methods and the code.

In most current experiments we used cBP instead of vBP, to reduce the amount of parameters. The learning rate for cBP is α=1e-3 with β=9e-1. Only in optimisation comparison, vBP appears with α_0=1e-2, α_e=1e-3, k_c=1e4 and β=9e-1 as well. ADAM parameters are, as employed in [9], α=1e-3, β_1=9e-1, β_2=9.99e-1 and ϵ=1e-8. In addition, some experiments show averaged graphs to see the general trend with a reduced influence of fluctuations.

4.1. Experiment 1: Domain variation. In Fig. 4.1a with x∈[0,2] we find a mean value of $\overline{\Delta u}$=8.8e-4 and for a doubled domain size of x∈[0,4] in Fig. 4.1b, $\overline{\Delta u}$=2.3e-3. Another increase to twice the domain size x∈[0,8] in Fig. 4.1c reveals more minima far from the first, one batch with $\overline{\Delta u}$=1.1e-2 and another one around $\overline{\Delta u}$=7.9e-1.

In comparison, mTSM in Fig. 4.1d with the same domain size shows a mean value of $\overline{\Delta u}$=7.9e-4, which is the lowest of them all. But initialisation with $\vec{p}_{zero}^{\,init}$ does not provide good approximations. ADAM provides better results with less variation and lower $\overline{\Delta u}$ for all four settings, but is not documented here.

Concluding, we choose x∈[0,2] in further experiments since for TSM and mTSM, with both ADAM and cBP, it is a reliable choice for fixing this parameter to eliminate domain variation influence. Furthermore, an increase of the domain size, where the exact solution can get closer to zero, with TSM and cBP makes the approximation less accurate. However, increasing ntD may provide even better results, especially for Fig. 4.1c.



(a) TSM with ADAM optimiser (b) mTSM with ADAM optimiser

(c) TSM with cBP optimiser (d) mTSM with cBP optimiser

Fig. 4.2: **Experiment 4.2**. Number of maximal epochs variation with k_{max}=1...1e5 with x∈[0,2], ntD=10 and λ=-5, (orange/solid) $\vec{p}_{zero}^{\,init}$, (blue/dotted) $\vec{p}_{rnd}^{\,init}$

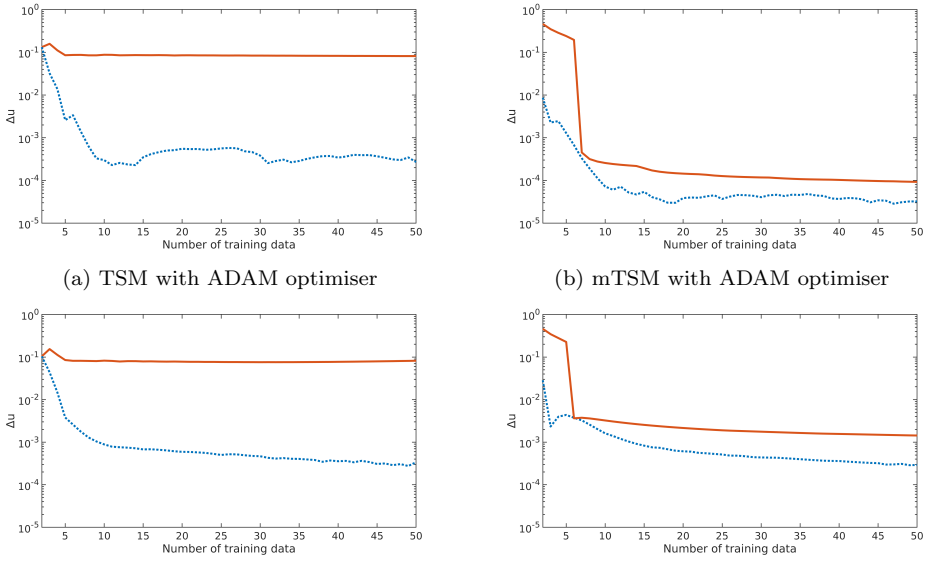
4.2. Experiment 2: Number of epochs per computation. In order to reduce the computational effort, we increased k_{max} by 500 in every iteration and averaged additional 1e2 iterations for the same k_{max}. An obvious difference between TSM and mTSM is the behaviour after setting $\vec{p}_{zero}^{\,init}$. With nearly the same Δu for k_{max}=1e5 in Fig. 4.2a and Fig. 4.2c, the difference to Fig. 4.2b and Fig. 4.2d is significant. While $\vec{p}_{zero}^{\,init}$ in Fig. 4.2b shows an even better result for k_{max}=1e5 than $\vec{p}_{rnd}^{\,init}$ for cBP in Fig. 4.2c and Fig. 4.2d, the accuracy for $\vec{p}_{rnd}^{\,init}$ is still better. The local mimimum at k_{max}=7500 in Fig. 4.2d for $\vec{p}_{zero}^{\,init}$ performs better than $\vec{p}_{rnd}^{\,init}$ in this k_{max} range. In all diagrams, $\vec{p}_{rnd}^{\,init}$ provides better results for higher k_{max} than $\vec{p}_{zero}^{\,init}$. Although ADAM shows a better approximation in the end over cBP, we expected a faster decrease of Δu for smaller numbers of k_{max}.

Therefore, the solid line behaviour justifies the use of $\vec{p}_{rnd}^{\,init}$ and k_{max}=1e5 in other experiments.

4.3. Experiment 3: Number of training data variation. We averaged 1e2 iterations for the same ntD. The use of $\vec{p}_{zero}^{\,init}$ in this experiment does not provide good approximations for TSM with both ADAM (Fig. 4.3a) and cBP (Fig. 4.3c).

For mTSM and $\vec{p}_{zero}^{\,init}$ there is a major decreasing at ntD=7 with ADAM in Fig. 4.3b and a decreasing shape afterwards, as well as for cBP in Fig. 4.3d at ntD=6 with the decrease to continue for further ntD. However, most reliable results provide $\vec{p}_{rnd}^{\,init}$ in the studied ntD range for all methods, but again with significant differences. While TSM and mTSM with cBP show an overall decreasing behaviour, ADAM seems to have a global minimum at ntD=11 for TSM in Fig. 4.3a. For mTSM and ADAM in Fig. 4.3b, $\vec{p}_{rnd}^{\,init}$ shows a local minimum at ntD=19.

Since these results only show averaged values within a certain ntD range, additional computations with the same parameter settings can provide local differences as well as different behaviours for further ntD. Fig. 4.4 shows a non averaged version of Fig. 4.3b, with a larger ntD range. Both $\vec{p}_{rnd}^{\,init}$ and $\vec{p}_{zero}^{\,init}$ provide better accuracy



(a) TSM with ADAM optimiser

(b) mTSM with ADAM optimiser

(c) TSM with cBP optimiser

(d) mTSM with cBP optimiser

Fig. 4.3: **Experiment 4.3**. Number of training data variation with ntD=2...50 with k_{max}=1e5, x∈[0,2] and λ=-5, (orange/solid) $\vec{p}_{zero}^{\,init}$, (blue/dotted) $\vec{p}_{rnd}^{\,init}$

with more ntD.

The decreasing behaviour for small ntD in Fig. 4.4 as well as in Fig. 4.3a to Fig. 4.3d justifies the use of ntD=10 for the other experiments, compared to the computational effort.

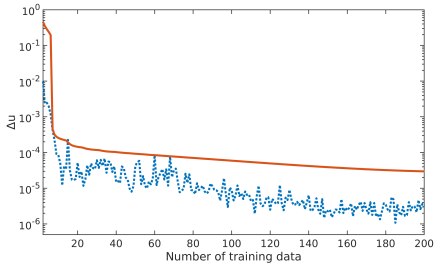
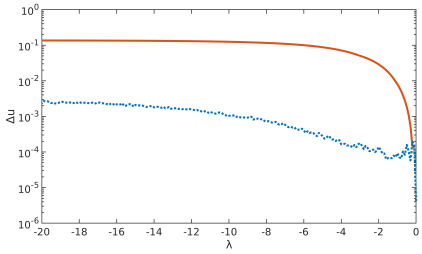


Fig. 4.4: mTSM with ADAM optimiser, ntD=2...200, k_{max}=1e5, x∈[0,2] and λ=-5, (orange/solid) $\vec{p}_{zero}^{\,init}$, (blue/dotted) $\vec{p}_{rnd}^{\,init}$

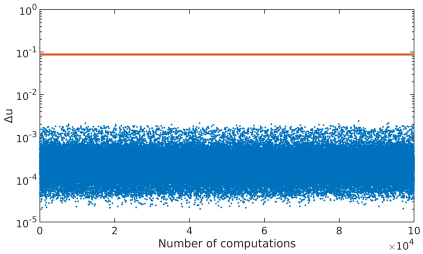
4.4. Experiment 4: λ variation. For experiments on λ variation we use ADAM and TSM. Fig. 4.5a shows averaged Δu with 1e2 iterations for each λ and compares $\vec{p}_{rnd}^{\,init}$ (blue/solid) with $\vec{p}_{zero}^{\,init}$ (orange/dashed). We find a local minimum around λ≈-1.6 for $\vec{p}_{rnd}^{\,init}$. The curve seems to converge for both weight initialisations, the smaller λ gets, while λ=-5, used in other experiments, lays between the most and the less reliable results for our parameter setting. However, Fig. 4.5b displays the solution variance with a $\overline{\Delta u}$=2.7e-4 and values in range of 2.1e-5≤Δu≤2.4e-3. For even smaller negative λ in Fig. 4.5c and Fig. 4.5d, the solution variance seems to decrease but the approximation becomes less accurate.

To mention the experiments on the other methods, results show different behaviour regarding the solution method. Within, the $\vec{p}_{zero}^{\,init}$ initialisation provide similar results for both cBP and ADAM. But for $\vec{p}_{rnd}^{\,init}$ there are significant differences.

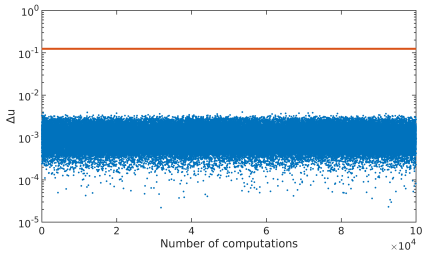
In conclusion, as λ influences the descend of the exponential exact solution, the faster it turns close to zero in the current domain, the less accurate the approximation becomes.



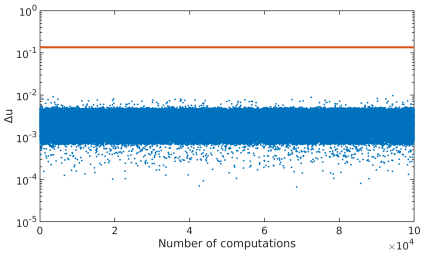
(a) TSM with ADAM optimiser, $\lambda \in$[-20,0]



(b) TSM with ADAM optimiser, λ=-5



(c) TSM with ADAM optimiser, λ=-10



(d) TSM with ADAM optimiser, λ=-20

Fig. 4.5: **Experiment 4.4**. ODE parameter (λ) variation with k_{max}=1e5, x$\in$[0,2] and ntD=10, (orange/solid) $\vec{p}_{zero}^{\,init}$, (blue/dotted) $\vec{p}_{rnd}^{\,init}$

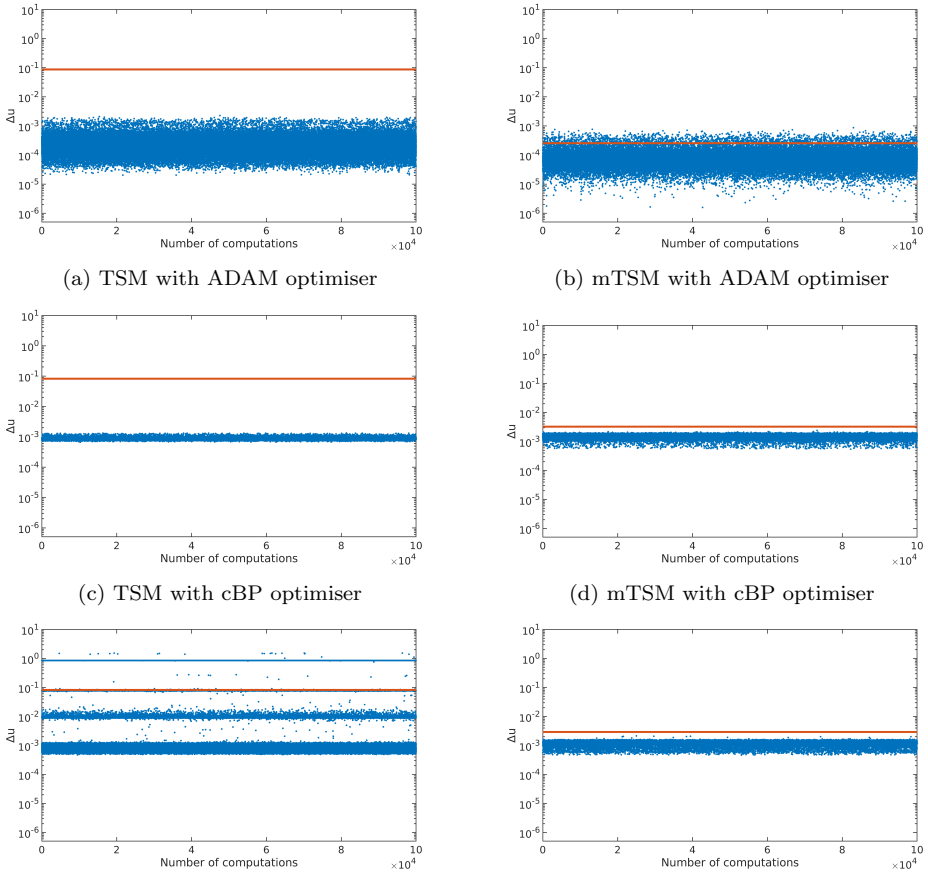
4.5. Experiment 5: optimisation method. We find for TSM with ADAM in Fig. 4.6a, results fluctuating around $\overline{\Delta u}$=2.7e-4 with values in range of 2.0e-5$\leq\Delta$u$\leq$2.3e-3. In comparison, for mTSM with ADAM in Fig. 4.6b we have an average result of $\overline{\Delta u}$=8.2e-5 with 1.6e-6$\leq\Delta$u$\leq$8.8e-4. A different behaviour in accuracy shows TSM with cBP, where the fluctuations in Fig. 4.6c are between 6.7e-4$\leq\Delta$u$\leq$1.3e-3 with $\overline{\Delta u}$=8.8e-4, and in Fig. 4.6d for mTSM with cBP, 5.5e-4$\leq\Delta$u$\leq$2.4e-3 with $\overline{\Delta u}$=1.6e-3. Fig. 4.6e reveals several minima far from reliable approximations for TSM with vBP, while mTSM with vBP in Fig. 4.6f provides 4.7e-4$\leq\Delta$u$\leq$2.1e-3 with $\overline{\Delta u}$=1.3e-3.

Best approximations provide ADAM with mTSM (Fig. 4.6b) for both $\vec{p}_{zero}^{\,init}$ and $\vec{p}_{rnd}^{\,init}$, but the difference between best and worst Δu is almost 5e2. The least variance, but also less good approximations, returns TSM with cBP in Fig. 4.6c.

5. Conclusion and future work. Weight initialisation with zeros provides constant results and works well for mTSM, especially with ADAM. Whereas small random values for initial weights provide reliable results in current experiments, except for TSM and vBP, which depends on adaptive stepsize parameters, which have not been studied yet. However, the adaptive learning rate for vBP seems to work for mTSM. The experiments show a significant influence on stability and reliability when changing parameters.

However, we found out, that an increase of the ODE parameter makes the approximation become less accurate with TSM and ADAM. Same holds for an increase of the domain size with TSM and cBP. That is, the closer the exact solution comes to zero, the less accurate TSM seems to be.

Furthermore, it is not entirely clear yet, whether to favour one method and optimisation combination or not. We see evidence for mTSM to be more reliable in general, especially with zero initialised weights and the ADAM optimiser. On the other hand, mTSM with ADAM seems to struggle using an arbitrary amount of training data



(a) TSM with ADAM optimiser

(b) mTSM with ADAM optimiser

(c) TSM with cBP optimiser

(d) mTSM with cBP optimiser

(e) TSM with vBP optimiser

(f) mTSM with vBP optimiser

Fig. 4.6: **Experiment 4.5**. Optimiser comparison with k_{max}=1e5, x∈[0,2], ntD=10 and λ=-5, (orange/solid) $\vec{p}_{zero}^{\,init}$, (blue/dotted) $\vec{p}_{rnd}^{\,init}$

although the general accuracy trend gets better, while mTSM with cBP seems to require more training data in order to provide equal results.

Future research will include more experiments with studies on ANN weight initialisation and in addition we will take non-dimensional DEs into account. Our main goal is to increase the solution accuracy for constant initialised weights as well as to find a reliable predicition for the initialisation. Studies will continue on more complex equations, like Bernoulli equation and Riccati equation.

Acknowledgments. This publication was funded by the Graduate Research School (GRS) of the Brandenburg University of Technology Cottbus-Senftenberg. This work is part of the Research Cluster Cognitive Dependable Cyber Physical Systems.

REFERENCES

[1] M. Hanke-Bourgeois, *Grundlagen der Numerischen Mathematik und des Wissenschaftlichen Rechnens*, B.G. Teubner Verlag / GWV Fachverlage GmbH, (2006).

[2] K. Kumar, and G.S.M. Thakur, *Advanced Applications of Neural Networks and Artifical Intelligence: A Review*, I.J Information Technology and Computer Science, 6 (2012), pp. 57–68.

[3] S. Mall, and S. Chakraverty, *Application of Legendre Neural Network for solving ordinary differential equations*, Applied Soft Computing, 43 (2016), pp. 347–356.

[4] I.E. Lagaris, A. Likas, and D.I. Fotiadis, *Artificial neural networks for solving ordinary and partial differential equations*, IEEE Transactions on Neural Networks, 9.5 (1998), pp. 987–1000.

[5] M.L. Piscopo, M. Spannowsky, and P. Waite, *Solving differential equations with neural networks: Applications to the calculation of cosmological phase transitions*, Physical Review D, 100.1 (2019), pp. 016002-1–016002-12.

[6] D. Nguyen, and B. Widrow, *Improving the learning speed of 2-layer neural networks by choosing initial values of the adaptive weights*, 1990 IJCNN International Joint Conference on Neural Networks, 3 (1990), pp. 21–26.

[7] R. Hecht-Nielsen, *Theory of the Backpropagation Neural Network*, Academic Press, 1992, pp. 65–93.

[8] V.V. Phansalkar, and P.S. Sastry, *Analysis of the Back-Propagation Algorithm with Momentum*, IEEE Transactions on Neural Networks, 5.3 (1994), pp. 505–506.

[9] D.P. Kingma, and J. Ba, *ADAM: A Method for Stochastic Optimization*, arXiv preprint:1412.6980v9, (2014).

[10] L. Luo, Y. Xiong, Y. Liu, and X. Sun, *Adaptive Gradient Methods with Dynamic Bound of Learning Rate*, arXiv preprint:1902.09843v1, (2019).

[11] G. Cybenko, *Approximation by Superpositions of a Sigmoidal Function*, Mathematics of Control, Signals, and Systems, 2.4 (1989), pp. 303–314.

[12] Y. Kaneda, Q. Zhao, Y. Liu, and Y. Pei, *Strategies for determining effective step size of the backpropagation algorithm for on-line learning*, 7th International Conference of Soft Computing and Pattern Recognition (SoCPaR), (2015), pp. 155–160.

[13] G.G. Dahlquist, *G-stability is equivalent to A-stability*, BIT Numerical Mathematics, 18.4 (1978), pp. 384–401.