Computer Aided Geometric Design 85 (2021) 101977

Contents lists available at ScienceDirect

Computer Aided Geometric Design

www.elsevier.com/locate/cagd

Parameterization for polynomial curve approximation via residual deep neural networks

Felix Scholz [a,*], Bert Jüttler [a,b]

[a] Radon Institute for Computational and Applied Mathematics (RICAM), Austrian Academy of Sciences, Linz, Austria
[b] Institute of Applied Geometry, Johannes Kepler University Linz, Austria

ARTICLE INFO

Article history:
Available online 10 March 2021

Keywords:
Parameterization
Curve fitting
Deep learning

ABSTRACT

Finding the optimal parameterization for fitting a given sequence of data points with a parametric curve is a challenging problem that is equivalent to solving a highly non-linear system of equations. In this work, we propose the use of a residual neural network to approximate the function that assigns to a sequence of data points a suitable parameterization for fitting a polynomial curve of a fixed degree. Our model takes as an input a small fixed number of data points and the generalization to arbitrary data sequences is obtained by performing multiple evaluations. We show that the approach compares favorably to classical methods in a number of numerical experiments that include the parameterization of polynomial as well as non-polynomial data.

1. Introduction

Approximating a sequence of data points with a polynomial curve is an important problem in geometric modeling. For standard techniques such as least squares fitting of Bézier curves, a parameterization of the data points is needed as a pre-requisite. However, in many practical applications, only the positions of the data points are given and the parameterization is unknown. This makes finding an optimal approximation of the given data with a polynomial curve a challenging problem, even if the data was originally sampled from a polynomial curve. It is well known that the parameterization has a large impact on the approximation and that a bad choice of parameterization results in a sub-optimal approximation error (Floater and Surazhsky, 2006; Floater, 2008).

Over the years, many methods have been proposed for the solution of this problem. They can be grouped into two categories:

Methods in the *first category* start with an initial guess for the parameterization and optimize it in order to achieve good approximation results. Typically, this results in a constrained optimization problem that is solved iteratively. One of the first methods that use this idea is the *intrinsic parameterization* (Hoschek, 1988) that has later been improved by Saux and Daniel (2003). Speer et al. (1998) presented an iterative method based on a global optimization procedure. An iterative reparameterization with the aim of achieving uniform angular speed along the curve was proposed by Yang et al. (2013).

Another approach that falls into this category is to approximate the solution of the system of equations that define the optimal parameterization. Since this system is highly non-linear, advanced optimizing methods such as the bat algorithm

* Corresponding author.
E-mail addresses: felix.scholz@ricam.oeaw.ac.at (F. Scholz), bert.juettler@jku.at (B. Jüttler).

https://doi.org/10.1016/j.cagd.2021.101977
0167-8396/

have been applied to this problem (Iglesias et al., 2015, 2016). The potential of an improved approximation power when optimizing the parameterization and the control points simultaneously was studied by Rababah (1995).

In the *squared distance minimization* method of Yang et al. (2004) and Wang et al. (2006), the problem of parameterization for fitting B-spline curves is avoided by finding an initial guess of the approximation and updating its control points iteratively. Gao et al. (2019) combined this process with results from deep learning in the context of reconstructing spline curves from image data. Several recurrent neural networks, that allow for variable input and output sizes, are used to produce good initial guesses to be used in an optimization method.

The *second category* includes methods for choosing a fixed parameterization a priori from the given data points. The most common approaches in this category include the estimation of the parameterization using uniform parameters or parameters that correspond to the chord lengths of the sequence of data points. Another commonly applied technique is the *centripetal parameterization* introduced by Lee (1989) that has been a foundation for several improvements aiming at avoiding unwanted wiggles, such as the *refined centripetal parameterization* (Fang and Hung, 2013) and the *dynamic centripetal parameterization* (Balta et al., 2020).

An extension of the centripetal parameterization to the surface case, the *augmented parameterization*, has been presented by Antonelli et al. (2016) in the context of approximating surface data by composite surfaces. Furthermore, the *universal parameterization* was proposed by Lim (1999, 2002) with the aim of creating natural looking curves for geometric modeling.

In the present paper we investigate the use of a deep neural network to predict the optimal parameters for given sequences of data points. Our approach therefore falls into the second category of methods that estimate a parameterization a priori. We note that – like all methods from this category – our method has the potential to be used in conjunction with methods from the *first category* by performing further optimization steps on the resulting parameterization.

We train the neural network on a large data set of sequences that consist of a fixed small number of points that were sampled randomly from polynomial curves. We propose a method for applying the resulting model to the parameterization of a point sequence of arbitrary length. Moreover, we show that the model generalizes to point sequences that were sampled from non-polynomial curves.

A related approach has been proposed by Laube et al. (2018), where a neural network is used for the parameterization of data points in the context of fitting with spline curves of degree 3. In that paper, the authors use a standard multi-layer perceptron with a fixed input size of 100 data points and three hidden layers with 1,000 nodes each. The overall number of parameters of the neural network is therefore more than $2.2e6$. Consequently, the process of training and evaluating the network is quite costly. Moreover, if the given number of data points is not equal to 100, one has to either increase it (by interpolation) or to decrease it (by sub-sampling).

In our approach, instead of parameterizing 100 data points from a single curve at a time, we construct a network for the parameterization of a minimum number of $2d+1$ points for the approximation with a curve of degree d. This is motivated by results about geometric interpolation with polynomial curves (Höllig and Koch, 1996; Jaklič et al., 2007; Vavpetič, 2020). Although these contributions focus mostly on the construction of curve segments from Hermite-type boundary data, they show how to unleash the full approximation power of polynomial curves – and similarly of Pythagorean-hodograph curves (Farouki et al., 2021) – by a suitable reparameterization. This approach, however, leads to non-linear computational problems. In our contribution, we use a deep neural network to implicitly capture the solutions of the non-linear problems, in order to reduce the computational challenges.

Since no suitable real-world dataset for this problem is publicly available, we rely on randomly generated training data. In contrast to the training data in (Laube et al., 2018), which is also randomly generated, we do not exclude self-intersecting curves, and we generate data sets for different polynomial degrees.

After constructing the network, we evaluate the model multiple times in order to parameterize an arbitrary number ($> 2d+1$) of data points. This extension makes our method flexible with respect to the number of input data points. For representing the parameterization, we employ a residual neural network architecture (He et al., 2016), which enables us to use a very deep network with a relatively small number of nodes in each layer. Our method results in approx. $5.5e4$ degrees of freedom, which is equivalent to ca. 2.5% of the data required by the network of Laube et al. (2018). This leads to advantages regarding memory storage, training time and complexity of the evaluation. Summing up the comparison with that previous work, our proposed method results in a much sparser network that is easy to train and evaluate, and the method is moreover directly applicable to any polynomial degree and number of input data points.

The remainder of the paper is organized as follows: We state the problem in Section 2 and analyze its complexity in Section 3. In Section 4 we describe the deep neural network and its hyperparameters as well as the training data. We show our numerical results in Section 5 and finalize the paper with a conclusion and some possible directions for future work.

2. Problem statement

In this paper, we consider the problem of parameterizing a given sequence

$$(\mathbf{p}_i)_{i=1\dots N} \subset \mathbb{R}^2$$

of data points in $\mathbb{R}^2$. This means that we want to obtain a sequence

$$t_1 < t_2 < \dots < t_N$$

of real parameters that corresponds to the sequence of data points. In particular, it is our aim to find a parameterization that is suitable for the approximation of the given data points with a polynomial curve

$$\mathbf{c} : [0, 1] \to \mathbb{R}^2$$

of a fixed maximum degree d. In order to quantify the approximation power of $\mathbf{c}$, we use the mean squared error between the parameterized data points and the evaluations of the polynomial curve:

$$\frac{1}{N} \sum_{i=1}^{N} \|\mathbf{c}(t_i) - \mathbf{p}_i\|^2. \tag{1}$$

If a parameterization (t_i) of the data points $\mathbf{p}_i$ is available, finding the polynomial curve $\mathbf{c}$ of degree d that minimizes (1) is a linear problem that can be solved, e.g., by representing the polynomial curves as Bézier curves

$$\mathbf{c}(t) = \sum_{j=0}^{d} \mathbf{c}_j B_{j,d}(t), \tag{2}$$

where $\mathbf{c}_j \in \mathbb{R}^2$ and $B_{j,d}$ are the Bernstein polynomials

$$B_{j,d}(t) = \binom{d}{j} t^j (1-t)^{d-j}.$$

The control points $\mathbf{c}_j$, which define the Bézier curve (2), are obtained by solving the least-squares problem

$$\underset{\mathbf{c}_0,\dots,\mathbf{c}_d \in \mathbb{R}^2}{\arg\min} \sum_{i=1}^{N} \left| \mathbf{p}_j - \sum_{j=0}^{d} \mathbf{c}_j B_{j,d}(t_i) \right|^2. \tag{3}$$

However, while the problem of finding the best approximating polynomial of degree d to a sequence of parameterized data points is well understood, the task of finding the parameterization t_i that leads to smallest residual in (3) is a complicated problem with no known solution. Standard methods include the estimation of the parameterization by either the uniform parameterization

$$t_i = (i-1)\frac{1}{N-1},$$

the parameterization by chord length

$$\begin{aligned} t_1 &= 0 \\ t_i &= \frac{\sum_{k=2}^{i} \|\mathbf{p}_k - \mathbf{p}_{k-1}\|}{\sum_{k=2}^{N} \|\mathbf{p}_k - \mathbf{p}_{k-1}\|} \quad \text{for } i = 2 \dots (N-1) \\ t_N &= 1 \end{aligned}$$

and the centripetal parameterization

$$\begin{aligned} t_1 &= 0 \\ t_i &= \frac{\sum_{k=2}^{i} \|\mathbf{p}_k - \mathbf{p}_{k-1}\|^a}{\sum_{k=2}^{N} \|\mathbf{p}_k - \mathbf{p}_{k-1}\|^a} \quad \text{for } i = 2 \dots (N-1) \\ t_N &= 1, \end{aligned}$$

where $0 < a < 1$ (typically $a = \frac{1}{2}$). In most cases, the approximating polynomial curves resulting from these methods do not reach the high accuracy that geometric approximation methods have been shown to achieve in the context of Hermite interpolation by optimizing both parameters and coefficients.

3. Analysis of the non-linear function to be learned

Let

$$\mathbf{a}(t) = \sum_{j=0}^{d} \mathbf{a}_j t^j \tag{4}$$

be a polynomial curve with coefficients $\mathbf{a}_j \in \mathbb{R}^2$. We evaluate this curve at N parameters

$$t_1 < t_2 < \ldots < t_N, \tag{5}$$

thus obtaining N data points

$$\mathbf{p}_i = \sum_{j=0}^{d} \mathbf{a}_j t_i^j. \tag{6}$$

If the parameters t_i are known, the coefficients of the curve can be recovered from (6) by solving the least squares problem

$$\underset{\mathbf{a}_0,\ldots,\mathbf{a}_d \in \mathbb{R}^2}{\arg\min} \sum_{i=1}^{N} \left| \mathbf{p}_i - \sum_{j=0}^{d} \mathbf{a}_j t_i^j \right|^2, \tag{7}$$

whose solution has zero residual, since the points were all sampled from the curve. We rewrite this problem in matrix form by defining the matrix $T \in \mathbb{R}^{N\times d}$ as

$$T_{ij} = t_i^j.$$

Then, (7) is equivalent to

$$\underset{\mathbf{a}\in\mathbb{R}^{d+1}}{\arg\min} \|\mathbf{p} - T\mathbf{a}\|^2$$

and the solution is given by

$$\mathbf{a} = \left(T^T T\right)^{-1} T^T \mathbf{p}. \tag{8}$$

By inserting (8) into (6) we arrive at

$$T\left(T^T T\right)^{-1} T^T \mathbf{p} - \mathbf{p} = 0. \tag{9}$$

This is a system of N rational equations in the parameters $(t_1, \ldots, t_N)$. If the parameterization (5) of the data points (6) is not given, the problem of parameterizing $\mathbf{p}_j$ in such a way that the sequence of points can be represented exactly by a polynomial curve of degree d is equivalent to solving (9) for the parameters t_j. Note that we used the monomial basis for representing polynomials in this section. For the theoretical analysis of the problem this is equivalent to representing polynomials with respect to the Bernstein polynomials.

While (9) defines implicitly the parameterization function of $(\mathbf{p}_1, \ldots, \mathbf{p}_N)$, there is no obvious path to an efficient evaluation of this function. Moreover, (9) does not have a solution for given data points that cannot be represented by a polynomial curve. Instead of attempting to solve this system of rational equations, we approximate the function using an artificial neural network.

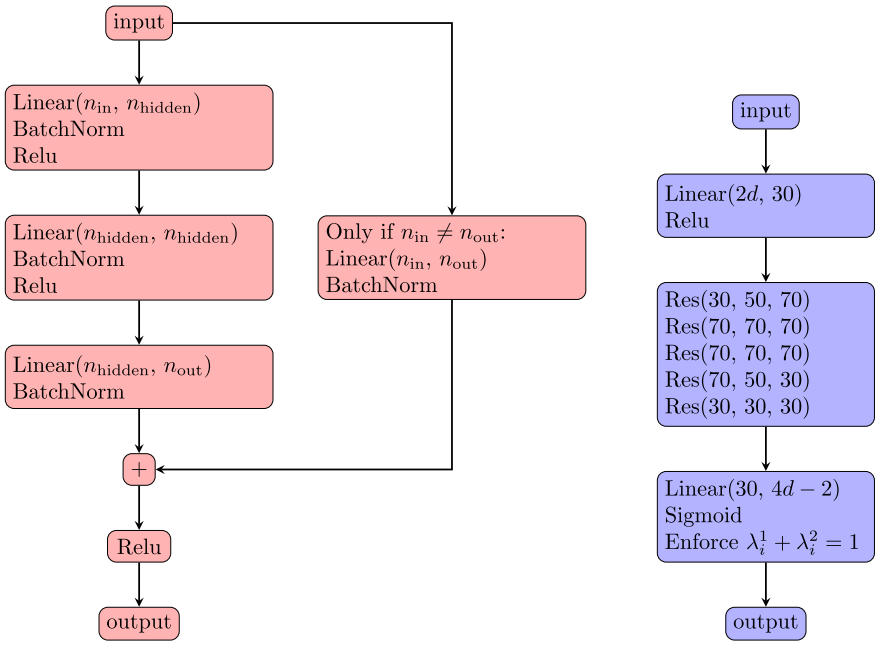
4. Description of the deep neural network

4.1. Overview

We use a deep neural network to find a parameterization of $(2d + 1)$ data points belonging to a polynomial curve of degree d. Due to the high complexity of the problem and in order to mitigate the vanishing gradient problem that occurs when employing neural networks that consist of many layers, we use a residual neural network.

First introduced by He et al. (2016) in the context of convolutional neural networks for image recognition, residual neural networks make it possible to efficiently train neural networks with many hidden layers. A residual neural network consists of stacked building blocks, where each building block has a shortcut connection between its input and its output. Whenever the input and output have the same dimension, this shortcut connection is simply an identity, when the dimensions differ an additional layer is introduced to project the input into the space of the output. Due to this construction, each block (which itself consists of a small number of layers), instead of approximating a function $H(\mathbf{x})$, approximates a residual function $F(\mathbf{x}) = H(\mathbf{x}) - \mathbf{x}$.

The input of our network consists of an ordered list of the $2d$ edges between neighboring data points. The use of an edge representation of the curve instead of the vertex representation makes the trained network translation-invariant. As an output, our network provides two weights λ_i^1, λ_i^2 with $\lambda_i^1 + \lambda_i^2 = 1$ for each vertex except for the two boundary vertices, whose parameters are set to 0 and 1, respectively. The weights define the relationship between each vertex and their neighbors. Afterwards, the resulting parameterization is obtained by solving the linear system



(a) Residual block Res(n_{in}, n_{hidden},n_{out}) (b) Neural network

Fig. 1. The residual neural network for curve parameterization. The left diagram shows the layout of a building block, while the right diagram shows the layout of the whole network.

$$\begin{aligned} t_i &= \lambda_i^1 t_{i-1} + \lambda_i^2 t_{i+1} \quad \forall i \in (2, \dots, 2d), \\ t_1 &= 0, \\ t_{2p+1} &= 1. \end{aligned} \tag{10}$$

The extension to a higher number of samples of the curve is treated in Section 5.1.2.

4.2. Layout and degrees of freedom

We adapt the residual neural network for nonlinear regression presented by Chen et al. (2020). Our neural network consists of a fully connected input layer, five residual building blocks and a fully connected output layer. Each residual building block itself contains three fully connected layers and an additional layer in the shortcut connection if the input and output dimensions differ. Batch normalization (Ioffe and Szegedy, 2015) is applied after each layer and after the shortcut connection in residual blocks with differing input and output dimension. For all hidden layers the rectified linear unit (ReLu) is used as activation function. In the output layer we use a sigmoid activation function in order to obtain output values between 0 and 1.

We denote a residual building block by Res($n_{\text{in}}, n_{\text{hidden}}, n_{\text{out}}$). The left diagram in Fig. 1 shows the layout of a single residual building block.

The right diagram in Fig. 1 shows the complete layout of the neural network for curve parameterization. Since each residual building block consists of three fully connected layers, the network has a total depth of 17 layers. Together with the two shortcut layers and the batch normalization, the network has $53998 + 244d$ learnable degrees of freedom. In our numerical examples, we used the same network layout for all polynomial degrees d. Since the problem of finding a suitable parameterization is more complicated for higher polynomial degrees, it can be necessary to increase the number of layers and nodes as d increases.

4.3. Loss function

There are several possibilities to measure the loss of the neural network.

1. In a straightforward way, we can use the weights μ_i^1, μ_i^2 that correspond to the original parameterization of the input curve as labels and optimize the mean squared error

$$\frac{1}{4d}\sum_{i=2}^{2d}(\lambda_i^1-\mu_i^1)^2+(\lambda_i^2-\mu_i^2)^2 \tag{11}$$

of the network's output.

2. We can solve the system (10) to obtain the parameterization $(s_i)_{i=1\dots(2d+1)}$ and use the original parameterization $(t_i)_{i=1\dots(2d+1)}$ of the curve as a label in order to measure the mean squared error

$$\frac{1}{2d+1}\sum_{i=1}^{2d+1}(t_i-s_i)^2. \tag{12}$$

3. We can use the parameterization to perform a least-squares approximation of the data points with a Bézier curve $\mathbf{c}$ of degree d and evaluate the curve at the parameters. We then use the data points themselves as labels and compute the loss as

$$\frac{1}{2d+1}\sum_{i=1}^{2d+1}(\mathbf{c}(s_i)-\mathbf{p}_i)^2. \tag{13}$$

The advantage of the first approach is, that it is computationally very efficient, since only the trainable layers of the network are used in the forward and backward pass. In the second and third approach, additional computational steps are performed that do not directly correspond to trained parameters of the network. However, measuring the error with respect to the parameters or the fitted data leads to a smaller number of epochs that is necessary for reaching a minimum validation error when training the network. In the third approach the input is equal to the label, and the resulting method can therefore be classified as an unsupervised machine learning method.

In the numerical experiments that we present in Section 5, we used the third approach.

4.4. Generation of the training data

In order to train the network for any given degree, we need a data set that contains a large number of curves of that degree. Since there is no such data set publicly available, we opt for generating the training set randomly. To this end, we represent the polynomial curves as Bézier curves and we generate a set of curves by randomly choosing the coefficients $\mathbf{c}_j$ according to the standard normal distribution. In order to arrive at a general model for the parameterization of curves, we do not further reduce the feature space. In particular, we do not exclude self-intersecting curves from the data set.

For each curve $\mathbf{c}(t)$, we define parameters $0=t_1<t_2<\ldots<t_{2d}<t_{2d+1}=1$ by choosing the parameters $t_2\dots t_{2d}$ randomly according to the uniform distribution. We evaluate the curve at the parameters to obtain the data points

$$\mathbf{p}_i=\mathbf{c}(t_i)\quad \text{for } i=1\dots 2d+1.$$

We normalize the data points by scaling them in such a way that they all lie inside the square $[0,1]^2$. More precisely, we transform the points into

$$\tilde{\mathbf{p}}_i=\frac{1}{\max_{\xi\in\{x,y\}}\left(\max_{j=1\dots N}\mathbf{p}_j^\xi-\min_{j=1\dots N}\mathbf{p}_j^\xi\right)}\left(\mathbf{p}_i-\left(\min_{j=1\dots N}\mathbf{p}_j^x,\min_{j=1\dots N}\mathbf{p}_j^y\right)\right).$$

Finally, we obtain the edges

$$\mathbf{e}_i=\tilde{\mathbf{p}}_{i+1}-\tilde{\mathbf{p}}_i\quad \text{for } i=1\dots 2d$$

that are used as input for the neural network.

We note that the normalization of the input data does not affect the generality of the neural network's results, since the same normalization can be applied to any given input data. After fitting the curve to the normalized data points, the resulting Bézier curve is then transformed by applying the inverse transformation to its control points.

4.5. Hyperparameters

We implemented the network using the PyTorch library and trained it in parallel on three 3 NVIDIA Tesla K40 GPUs. The batch size was set to 16 on each GPU. As an optimizer, we used the Adam optimizer (Kingma and Ba, 2014) with a learning rate of 10^{-3}. For each polynomial degree d, we trained a model on a data set containing 100000 polynomial curves that were generated as described in Section 4.4. The standard technique of splitting the data set into a training and a validation set was used to find the model with the best generalization properties.

5. Application of the curve parameterizing residual neural network

While the training of the neural network can take some time (on our computing node, the training usually took one night to arrive at the optimal generalization error), the evaluation of the trained network is very efficient and can be performed on a standard CPU. In this section, we apply our network to a number of sets of curves, both polynomial and non-polynomial in order to analyze the generalization of the network to unseen data. We compare the results with the classical methods of uniform parameterization and parameterization by chord length.

In our experiments, we used the same network architecture for all the polynomial degrees. Since the parameterization problem gets more difficult with increasing polynomial degrees, we expect the resulting approximation error to be dependent on the polynomial degree. In practice, this should be dealt with by increasing the number of layers and nodes with the increasing polynomial degree.

5.1. Parameterizing polynomial curves

In the first set of examples, we apply our network to the parameterization of discrete data that was sampled from polynomial curves.

5.1.1. Minimum number of data points per curve

In this experiment, we use our networks for the task of parameterizing $(2d+1)$ data points that were sampled from a polynomial curve of degree d, where d is the same degree that was used to train the network. We evaluate the network on a test set of 1000 curves that were generated in the same way as the training data set described in Section 4.4. Note that this data is completely unseen by the neural network and is also not part of the validation set that was used to optimize the hyperparameters.

We record the mean-squared error and the maximum error of the parameters and the fitted curve at the parameter values and compare those errors with the errors resulting from the uniform parameterization, the chord-length parameterization as well as the centripetal parameterization with exponent $a = \frac{1}{2}$. Moreover, we record the one-sided Hausdorff error

$$\max_i \min_{t\in[0,1]} \|\mathbf{p}_i - \mathbf{c}(t)\|.$$

The results of this experiment are shown in Table 1. We observe that the parameterization that is predicted by the neural network leads to an improvement of the approximation error compared to the parameterization by chord length by a factor of approximately 30 for $d = 2$, approximately 13 for $d = 3$, approximately 6 for $d = 4$ and a factor of approximately 4 for $d = 5$.

5.1.2. Arbitrary number of data points per curve

In our next experiment, we apply the network to the parameterization of a number of $L > 2d+1$ data points $(\mathbf{p}_i)_{i=1,\dots,L}$ that were sampled from a polynomial curve of degree d. Since our network expects a constant number of $(2d+1)$ data points as input, there are $\binom{L}{2d+1}$ possible combinations of the sampled data that can be used as input for our network. Because for large values of L it is clearly computationally infeasible to evaluate the network on all combinations, we choose a subset of the possible evaluations q_i in the following way:

We always set $\mathbf{q}_1 = \mathbf{p}_1$ and $\mathbf{q}_{2d+1} = \mathbf{p}_L$. This is to ensure that each evaluation covers the whole curve and is not restricted to a region where the curve is almost flat. We then generate for each point $\mathbf{p}_i \in (\mathbf{p}_i)_{i=d+1,\dots,L-d}$ a fixed number of K evaluation sets by setting $\mathbf{q}_{d+1} = \mathbf{q}_i$ and choosing

$$(\mathbf{q}_j)_{j=2,\dots,d} \subseteq \{\mathbf{p}_2, \dots, \mathbf{p}_{i-1}\}$$

and

$$(\mathbf{q}_j)_{j=d+2,\dots,2d} \subseteq \{\mathbf{p}_{i+1}, \dots, \mathbf{p}_{L-1}\}$$

randomly without allowing duplicate evaluations. Evaluating the neural network on each of the $(L-2d)K$ combinations of input data points, we obtain $(L-2d)K(2d-1)$ equations of the form

$$t_i = \lambda_a t_a + \lambda_b t_b, \tag{14}$$

where λ_a and λ_b are the result of an evaluation of the neural network on an input that contains the sequence $(\mathbf{p}_a, \mathbf{p}_i, \mathbf{p}_b)$. Finally, we solve the least squares problem corresponding to the equations (14) with the additional conditions

$$t_1 = 0$$
$$t_L = 1$$

Table 1
Comparison of the errors with respect to the test data set when fitting $2d+1$ data points samples from a polynomial curve of degree d for uniform (U), centripetal (C), chord length (L) and neural network-based (N) parameterization.

	parameterization		curve		
$d=2$	MSE	maximum error	MSE	maximum error	Hausdorff error
U	2.5e−2 (260%)	2.6e−1 (186%)	1.6e−2 (696%)	1.7e−1 (333%)	8.5×10^{-2} (189%)
C	9.7e−3 (101%)	1.5e−1 (107%)	4.6e−3 (200%)	8.1e−2 (159%)	5.4×10^{-2} (120%)
L	9.8e−3 (100%)	1.4e−1 (100%)	2.3e−3 (100%)	5.1e−2 (100%)	4.5×10^{-2} (100%)
N	**2.7e−3** (28%)	**6.6e−2** (47%)	**7.8e−5** (3%)	**1.0e−2** (20%)	$\mathbf{8.2\times10^{-3}}$ (**18**%)
	parameterization		curve		
$d=3$	MSE	maximum error	MSE	maximum error	Hausdorff error
Um	1.9e−2 (136%)	2.4e−1 (133%)	1.0e−2 (588%)	1.5e−1 (300%)	7.9×10^{-2} (172%)
C	1.0e−2 (71%)	1.6e−1 (89%)	3.1e−3 (182%)	7.2e−1 (144%)	5.3×10^{-2} (115%)
L	1.4e−2 (100%)	1.8e−1 (100%)	1.7e−3 (100%)	5.0e−2 (100%)	4.6×10^{-2} (100%)
N	**4.7e−3** (36%)	**1.0e−1** (56%)	**1.3e−4** (8%)	**1.5e−2** (30%)	$\mathbf{1.2\times10^{-2}}$ (**26**%)
	parameterization		curve		
$d=4$	MSE	maximum error	MSE	maximum error	Hausdorff error
U	1.6e−2 (106%)	2.2e−1 (116%)	6.7e−3 (761%)	1.2e−1 (293%)	6.5×10^{-2} (171%)
C	1.0e−2 (66%)	1.6e−1 (84%)	1.5e−3 (170%)	5.7e−2 (139%)	4.2×10^{-2} (111%)
L	1.5e−2 (100%)	1.9e−1 (100%)	8.8e−4 (100%)	4.1e−2 (100%)	3.8×10^{-2} (100%)
N	**5.5e−3** (36%)	**1.2e−1** (63%)	**1.4e−4** (16%)	**1.7e−2** (41%)	$\mathbf{1.4\times10^{-2}}$ (**37**%)
	parameterization		curve		
$d=5$	MSE	maximum error	MSE	maximum error	Hausdorff error
U	1.3e−2 (81%)	2.0e−1 (100%)	4.8e−3 (842%)	1.1e−1 (306%)	5.6×10^{-2} (165%)
C	9.0e−3 (60%)	1.5e−1 (75%)	1.0e−3 (175%)	4.9e−2 (136%)	3.7×10^{-2} (109%)
L	1.6e−2 (100%)	2.0e−1 (100%)	5.7e−4 (100%)	3.6e−2 (100%)	3.4×10^{-2} (100%)
N	**1.2e−2** (75%)	**1.6e−1** (80%)	**1.5e−4** (26%)	**1.9e−2** (53%)	$\mathbf{1.6\times10^{-2}}$ (**47**%)

to obtain the parameterization.

In our numerical example, we set $K=10$ and generate test sets for different values of L, each containing 100 curves. The results are shown in Table 2. We observe that even for large numbers of samples on a given curve, the approximation error is improved by using the parameterization obtained by the neural network.

However, while the neural network approach always leads to an improvement with respect to the chord length parameterization, the centripetal parameterization also performs well and, for high values of p and L, where it outperforms the neural network approach. One way that the neural network approach can be improved here is by increasing the size of the network architecture with the degree that it is trained on. As described in the beginning of this section, in our examples we employ the *same* network design for *all* polynomial degrees. Moreover, the method of generalizing the result of the neural network to larger numbers of input data points should be improved in the future.

The computational complexity of evaluating the neural network on all chosen combinations of data points naturally increases with d, L and K. However, the computation time stays manageable with average total evaluation times that vary between 2 ms for $d=2, L=5, K=1$ and 1.5 s for $d=5, L=80, K=10$ on a standard CPU.

5.2. Parameterizing non-polynomial curves

In this section, we apply our method to the problem of fitting non-polynomial data with polynomial curves of degree d. To this end, we create a test set of 100 complex trigonometric polynomials

$$\mathbf{c}(t)=\mathbf{a}_0+\sum_{j=1}^{r}\mathbf{a}_j\cos(jt)+i\sum_{j=1}^{r}\mathbf{b}_j\sin(jt)$$

of degree r, where $\mathbf{a}_j,\mathbf{b}_j\in\mathbb{R}$. We sample $2d+1$ random points from each curve and use the network to parameterize these points before computing the best-approximation in the least-squares sense.

Table 3 shows the mean squared error as well as the one-sided Hausdorff error when fitting trigonometric polynomials of various degrees r with polynomial curves of various degrees d. The parameterizations that are predicted by the neural network lead to an improvement of the approximation error also in the case of non-polynomial data. These results verify that the neural network can generalize the information learned from the data set of polynomial curves to non-polynomial curves, even when these curves are chosen from spaces of higher dimensions. This property is especially important for

Table 2
Mean squared error of the fitted data points for different sample sizes L, averaged over 100 curves for uniform (U), centripetal (C), chord length (L) and neural network-based (N) parameterization.

$d=2$	$L=20$		$L=50$		$L=80$	
	MSE	Hausdorff	MSE	Hausdorff	MSE	Hausdorff
U	3.7e−3 (132%)	5.0e−2 (57%)	2.1e−3 (91%)	8.4e−2 (111%)	1.5e−3 (50%)	7.1e−2 (85%)
C	2.1e−3 (75%)	7.1e−2 (82%)	1.5e−3 (65%)	6.6e−2 (87%)	1.4e−3 (47%)	6.3e−2 (76%)
L	2.8e−3 (100%)	8.7e−2 (100%)	2.3e−3 (100%)	7.6e−2 (100%)	3.0e−3 (100%)	8.4e−2 (100%)
N	**5.3e−4** (19%)	**3.5e−2** (**40**%)	**7.3e−4** (32%)	**4.2e−2** (**55**%)	**7.9e−4** (26%)	**4.4e−2** (**53**%)

$d=3$	$L=20$		$L=50$		$L=80$	
	MSE	Hausdorff	MSE	Hausdorff	MSE	Hausdorff
U	3.5e−3 (194%)	1.2e−1 (144%)	1.7e−3 (77%)	9.3e−2 (108%)	1.1e−3 (55%)	7.5e−2 (91%)
C	1.7e−3 (94%)	7.9e−2 (95%)	1.2e−3 (55%)	6.8e−2 (79%)	1.1e−3 (55%)	6.3e−2 (77%)
L	1.8e−3 (100%)	8.3e−2 (100%)	2.2e−3 (100%)	8.6e−2 (100%)	2.0e−3 (100%)	8.2e−2 (100%)
N	**6.7e−4** (37%)	**5.5e−2** (66%)	**8.7e−4** (40%)	**5.7e−4** (66%)	**8.5e−4** (43%)	**5.3e−2** (65%)

$d=4$	$L=20$		$L=50$		$L=80$	
	MSE	Hausdorff	MSE	Hausdorff	MSE	Hausdorff
U	2.3e−3 (261%)	1.0e−1 (167%)	1.3e−3 (118%)	8.4e−2 (118%)	7.6e−4 (76%)	6.7e−2 (98%)
C	8.5e−4 (97%)	6.0e−2 (94%)	7.1e−4 (65%)	5.6e−2 (79%)	5.0e−4 (50%)	4.6e−2 (68%)
L	8.8e−4 (100%)	6.4e−2 (100%)	1.1e−3 (100%)	7.1e−2 (100%)	1.0e−3 (100%)	6.8e−2 (100%)
N	**3.3e−4** (38%)	**4.1e−2** (64%)	**4.3e−4** (39%)	**4.4e−2** (62%)	**4.1e−4** (41%)	**4.3e−2** (63%)

$d=5$	$L=20$		$L=50$		$L=80$	
	MSE	Hausdorff	MSE	Hausdorff	MSE	Hausdorff
U	2.1e−3 (333%)	9.4e−2 (168%)	1.1e−3 (141%)	8.6e−2 (134%)	6.7e−4 (94%)	6.7e−2 (114%)
C	6.4e−4 (102%)	5.1e−2 (91%)	**5.1e−4** (**65**%)	**5.2e−2** (**81**%)	**3.7e−4** (53%)	**4.3e−2** (**73**%)
L	6.3e−4 (100%)	5.6e−2 (100%)	7.8e−4 (100%)	6.4e−2 (100%)	7.1e−4 (100%)	5.9e−2 (100%)
N	**4.8e−4** (76%)	**4.9e−2** (**88**%)	5.8e−4 (74%)	6.2e−2 (97%)	5.6e−4 (79%)	6.3e−2 (106%)

Table 3
Mean squared error when approximating complex trigonometric polynomials of degree r with polynomial curves of degree d. Fitted over $2d+1$ data points using uniform (U), centripetal (C), chord length (L) and neural network-based (N) parameterization.

$d=2$	$r=2$		$r=3$		$r=4$		$r=5$	
	MSE	Hausdorff	MSE	Hausdorff	MSE	Hausdorff	MSE	Hausdorff
U	6.6e−3 (1065%)	1.2e−1 (300%)	1.1e−2 (846%)	1.6e−1 (302%)	1.3e−2 (722%)	1.8e−1 (250%)	1.9e−2 (404%)	2.4e−1 (200%)
C	1.6e−3 (258%)	3.2e−2 (155%)	3.0e−3 (231%)	8.3e−2 (157%)	4.2e−3 (233%)	1.0e−1 (139%)	8.3e−3 (177%)	1.6e−1 (133%)
L	6.2e−4 (100%)	4.0e−2 (100%)	1.3e−3 (100%)	5.3e−2 (100%)	1.8e−3 (100%)	7.2e−2 (100%)	4.7e−3 (100%)	1.2e−1 (100%)
N	**7.7e−5** (12%)	**1.5e−2** (38%)	**1.9e−4** (15%)	**2.2e−2** (42%)	**3.7e−4** (21%)	**3.1e−2** (43%)	**1.4e−3** (30%)	**6.4e−2** (53%)

$d=3$	$r=2$		$r=3$		$r=4$		$r=5$	
	MSE	Hausdorff	MSE	Hausdorff	MSE	Hausdorff	MSE	Hausdorff
U	3.6e−3 (3600%)	9.2e−2 (708%)	4.1e−3 (1952%)	2.0e−1 (167%)	6.1e−3 (1298%)	2.4e−1 (150%)	8.9e−3 (1113%)	2.9e−1 (153%)
C	5.1e−4 (510%)	3.4e−2 (262%)	7.0e−4 (333%)	1.4e−1 (117%)	1.4e−3 (298%)	1.8e−1 (113%)	2.8e−3 (350%)	2.2e−1 (116%)
L	1.0e−4 (100%)	1.3e−2 (100%)	2.1e−4 (100%)	1.2e−1 (100%)	4.7e−4 (100%)	1.6e−1 (100%)	8.0e−4 (100%)	1.9e−1 (100%)
N	**4.1e−5** (41%)	**1.1e−2** (92%)	**4.4e−5** (21%)	**9.3e−2** (76%)	**6.0e−5** (13%)	**1.1e−1** (67%)	**1.5e−4** (19%)	**1.3e−1** (69%)

$d=4$	$r=3$		$r=4$		$r=5$		$r=6$	
	MSE	Hausdorff	MSE	Hausdorff	MSE	Hausdorff	MSE	Hausdorff
U	2.9e−3 (3222%)	1.8e−2 (194%)	3.4e−3 (1619%)	2.2e−1 (157%)	4.0e−3 (1333%)	2.4e−1 (141%)	5.2e−3 (1283%)	2.8e−1 (165%)
C	4.2e−4 (466%)	1.3e−1 (140%)	6.1e−4 (290%)	1.7e−1 (121%)	7.9e−4 (263%)	1.9e−1 (112%)	1.2e−3 (286%)	2.1e−1 (124%)
L	9.0e−5 (100%)	9.3e−2 (100%)	2.1e−4 (100%)	1.4e−1 (100%)	3.0e−4 (100%)	1.7e−1 (100%)	4.2e−4 (100%)	1.7e−1 (100%)
N	**3.3e−5** (37%)	**7.9e−2** (85%)	**5.2e−5** (25%)	**9.7e−2** (69%)	**7.0e−5** (23%)	**1.1e−1** (65%)	**9.6e−5** (23%)	**1.2e−1** (71%)

$d=5$	$r=4$		$r=5$		$r=6$		$r=7$	
	MSE	Hausdorff	MSE	Hausdorff	MSE	Hausdorff	MSE	Hausdorff
U	2.0e−3 (2353%)	1.8e−1 (150%)	2.7e−3 (1800%)	2.1e−1 (131%)	3.4e−3 (1700%)	2.2e−1 (147%)	4.3e−3 (1344%)	2.6e−1 (144%)
C	2.8e−4 (329%)	1.4e−1 (117%)	4.4e−4 (293%)	1.6e−1 (81%)	5.3e−4 (265%)	1.7e−1 (113%)	7.8e−4 (244%)	2.0e−1 (111%)
L	8.5e−5 (100%)	1.2e−1 (100%)	1.5e−4 (100%)	1.3e−1 (100%)	2.0e−4 (100%)	1.5e−1 (100%)	3.2e−4 (100%)	1.8e−1 (100%)
N	**4.2e−5** (49%)	**9.2e−2** (77%)	**5.7e−5** (38%)	**1.1e−1** (69%)	**6.2e−5** (31%)	**1.1e−1** (73%)	**1.1e−4** (34%)	**1.3e−1** (72%)

practical applications, where it is usually not known that the measured data can be exactly represented by a polynomial curve of a fixed degree.

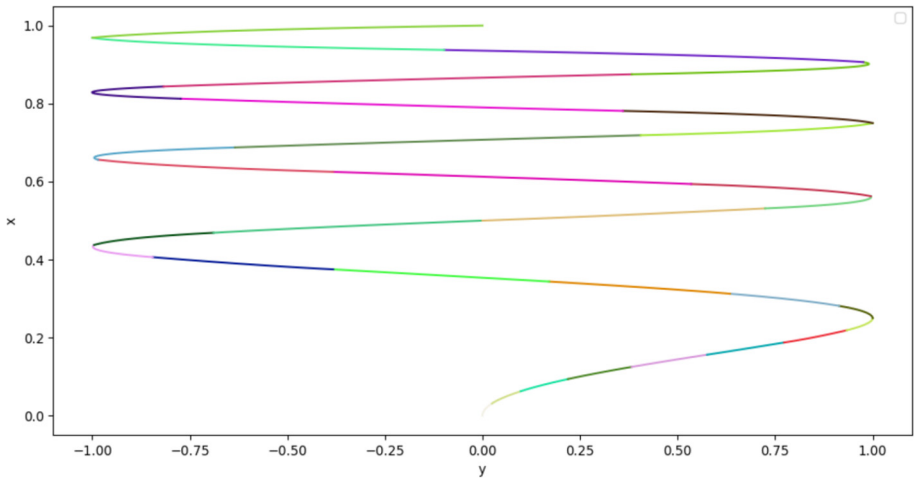


Fig. 2. Approximation of the curve (15) with 64 quadratic curve segments.

5.3. Approximating data with polynomial curve segments

In our last experiment, we apply the neural network to the task of approximating data points from a non-polynomial curve with continuous piece-wise polynomials. In order to test the asymptotic behavior of the approximation error, we sample points from the curve

$$\mathbf{c}(t) = \left(t, \sin(8\pi t^2)\right) \tag{15}$$

by splitting the interval $[0, 1]$ into 2^n segments and evaluate the curve at $2d + 1$ uniformly distributed parameters in each segment. In each segment, we apply the normalization described in Section 4.4 and use the parameterization produced by the neural network to fit a polynomial curve of degree d to the normalized data points. Finally, the control points of the resulting curve segment are transformed with the inverse of the normalization. An approximation of the curve with 64 quadratic curve segments is shown in Fig. 2.

In Fig. 3, we show the mean squared error at the parameter values when approximating (15) with continuous piece-wise polynomials of various degrees. We compare the approximation resulting from the neural network with the approximation resulting from a chord length parameterization. The parameterization predicted by the neural network results in an improved approximation compared to a chord length parameterization as long as the level of refinement is not to high. This is due to the fact that after some refinement steps, the curve becomes almost flat in most segments. It is then very hard for the neural network to make valid predictions for the parameterization. Should this case arise, one can simply keep track of the mean-squared error during the fitting process and adaptively fall back to the chord-length parameterization wherever necessary.

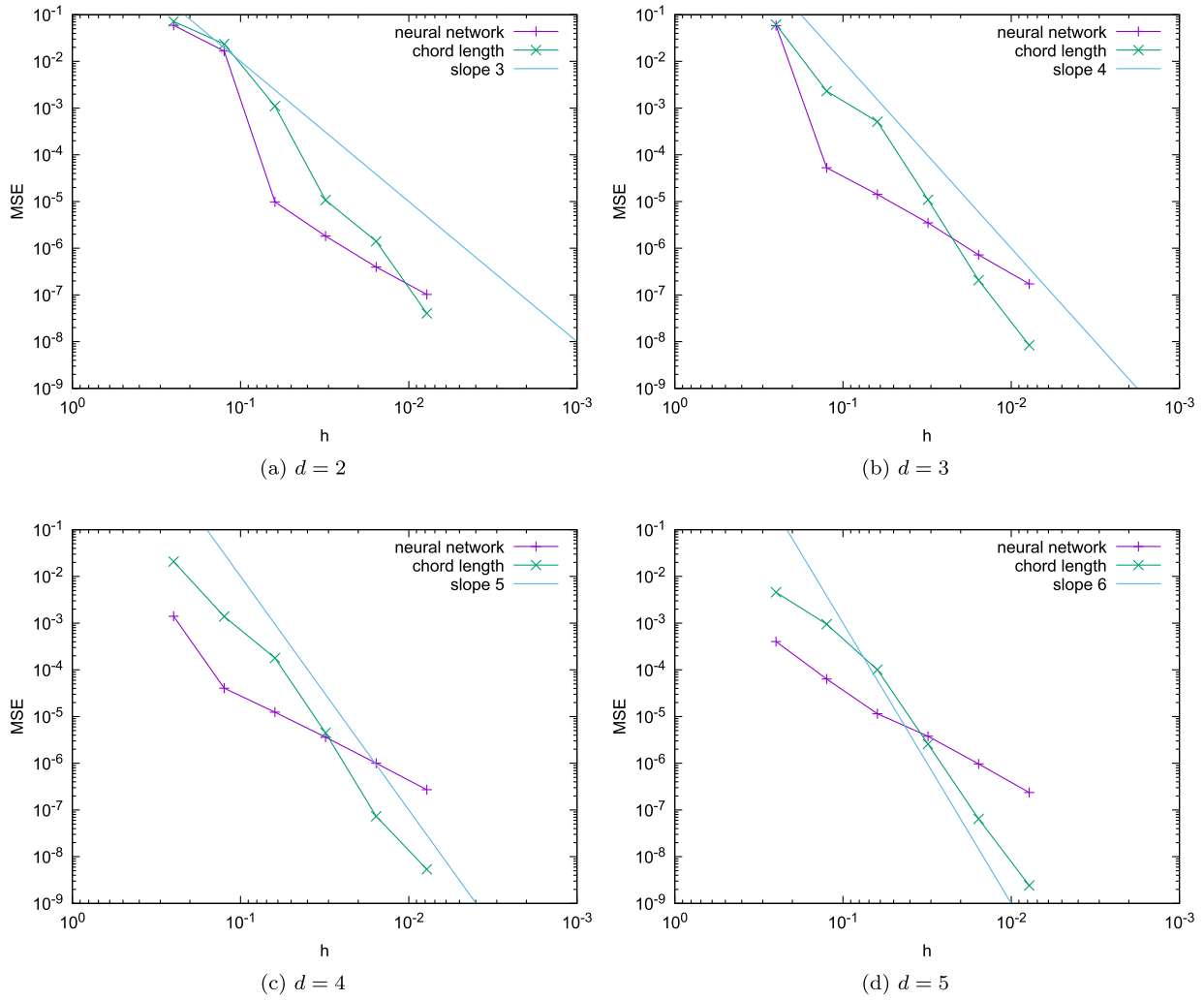
6. Conclusion

In the present paper we proposed an efficient method for parameterizing discrete curve data using a residual neural network, with the aim of using the parameterization for the approximation with polynomial curves. Due to the architecture of our network and the choice to use only a small number of data points as an input, we arrived at an efficient and easy to implement method, that can be applied to arbitrary polynomial degrees and arbitrary numbers of input data points. An important challenge for future research is the construction of a neural network for the parameterization of discrete surface data, which is needed for fitting polynomial surfaces. In the surface case, the parameterization can be represented by weights on the edges in similar way as we did in (10), see Floater and Hormann (2005). However, the varying number of neighbors makes it more difficult to handle this type of data with a neural network and methods from geometric deep learning (Bronstein et al., 2017) will be employed.

In the present work we have focused on optimizing the accuracy (measured the mean square error) of the approximating polynomial curves with respect to the input data. Other targets are possible and should be explored in the future. Besides other error metrics, these include measures for the shape of the curve such as the bending energy. In future work, these targets can be optimized for by adapting the error function used in the training of the network.

Furthermore, we will study the necessary increase in network size for higher polynomial degrees and investigate possibilities to further improve the method of generalizing the results of the neural network to larger numbers of sample points.

Another direction for future research concerns the normalization process presented in Section 4.4. In general, normalization of the input data is an important tool to decrease the training error and to help the neural network learn faster. In



(a) $d = 2$ (b) $d = 3$

(c) $d = 4$ (d) $d = 5$

Fig. 3. Mean squared error when approximating (15) with polynomial segments.

order to further decrease the range of possible configurations of the $(2d + 1)$ input data points that need to be handled by the network, a transformation that maps the first, d-th and last data point to predefined positions will be investigated.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgement

Supported by the European Research Council through grant GA no. 694515 “CHANGE”.

References

Antonelli, M., Beccari, C.V., Casciola, G., 2016. High quality local interpolation by composite parametric surfaces. Comput. Aided Geom. Des. 46, 103–124.
Balta, C., Öztürk, S., Kuncan, M., Kandilli, I., 2020. Dynamic centripetal parameterization method for B-spline curve interpolation. IEEE Access 8, 589–598.
Bronstein, M.M., Bruna, J., LeCun, Y., Szlam, A., Vandergheynst, P., 2017. Geometric deep learning: going beyond Euclidean data. IEEE Signal Process. Mag. 34, 18–42.
Chen, D., Hu, F., Nian, G., Yang, T., 2020. Deep residual learning for nonlinear regression. Entropy 22, 193.
Fang, J.J., Hung, C.L., 2013. An improved parameterization method for B-spline curve and surface interpolation. Comput. Aided Des. 45, 1005–1028.
Farouki, R.T., Pelosi, F., Sampoli, M.L., 2021. Approximation of monotone clothoid segments by degree 7 Pythagorean-hodograph curves. J. Comput. Appl. Math. 382, 16.
Floater, M.S., 2008. On the deviation of a parametric cubic spline interpolant from its data polygon. Comput. Aided Geom. Des. 25, 148–156.

Floater, M.S., Hormann, K., 2005. Surface parameterization: a tutorial and survey. In: Dogson, N., Floater, M.S., Sabin, M. (Eds.), Advances in Multiresolution for Geometric Modelling. In: Mathematics and Visualization. Springer, Berlin, Heidelberg, pp. 157–186.
Floater, M.S., Surazhsky, T., 2006. Parameterization for curve interpolation. In: Jetter, K., Buhmann, M.D., Haussmann, W., Schaback, R., Stöckler, J. (Eds.), Topics in Multivariate Approximation and Interpolation. In: Studies in Computational Mathematics, vol. 12. Elsevier, pp. 39–54.
Gao, J., Tang, C., Ganapathi-Subramanian, V., Huang, J., Su, H., Guibas, L.J., 2019. Deepspline: data-driven reconstruction of parametric curves and surfaces. arXiv:1901.03781.
He, K., Zhang, X., Ren, S., Sun, J., 2016. Deep residual learning for image recognition. In: 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 770–778.
Hoschek, J., 1988. Intrinsic parametrization for approximation. Comput. Aided Geom. Des. 5, 27–31.
Höllig, K., Koch, J., 1996. Geometric Hermite interpolation with maximal order and smoothness. Comput. Aided Geom. Des. 13, 681–695.
Iglesias, A., Gálvez, A., Collantes, M., 2015. Bat algorithm for curve parameterization in data fitting with polynomial Bézier curves. In: 2015 International Conference on Cyberworlds (CW), pp. 107–114.
Iglesias, A., Gálvez, A., Collantes, M., 2016. Four adaptive memetic bat algorithm schemes for Bézier curve parameterization. In: Gavrilova, M., Tan, C., Sourin, A. (Eds.), Transactions on Computational Science XXVIII. In: Lecture Notes in Computer Science, vol. 9590. Springer, Berlin, Heidelberg, pp. 127–145.
Ioffe, S., Szegedy, C., 2015. Batch normalization: accelerating deep network training by reducing internal covariate shift. In: Bach, F., Blei, D. (Eds.), International Conference on Machine Learning. PMLR, Lille, France, pp. 448–456.
Jaklič, G., Kozak, J., Krajnc, M., Žagar, E., 2007. On geometric interpolation by planar parametric polynomial curves. Math. Comput. 76, 1981–1993.
Kingma, D.P., Ba, J., 2014. Adam: a method for stochastic optimization. arXiv:1412.6980.
Laube, P., Franz, M.O., Umlauf, G., 2018. Deep learning parametrization for B-spline curve approximation. In: 2018 International Conference on 3D Vision (3DV). IEEE, pp. 691–699.
Lee, E., 1989. Choosing nodes in parametric curve interpolation. Comput. Aided Des. 21, 363–370.
Lim, C.G., 1999. A universal parametrization in B-spline curve and surface interpolation. Comput. Aided Geom. Des. 16, 407–422.
Lim, C.G., 2002. Universal parametrization in constructing smoothly-connected B-spline surfaces. Comput. Aided Geom. Des. 19, 465–478.
Rababah, A., 1995. High order approximation method for curves. Comput. Aided Geom. Des. 12, 89–102.
Saux, E., Daniel, M., 2003. An improved Hoschek intrinsic parametrization. Comput. Aided Geom. Des. 20, 513–521.
Speer, T., Kuppe, M., Hoschek, J., 1998. Global reparametrization for curve approximation. Comput. Aided Geom. Des. 15, 869–877.
Vavpetič, A., 2020. Optimal parametric interpolants of circular arcs. Comput. Aided Geom. Des. 80, 8.
Wang, W., Pottmann, H., Liu, Y., 2006. Fitting B-spline curves to point clouds by curvature-based squared distance minimization. ACM Trans. Graph. 25, 214–238.
Yang, H., Wang, W., Sun, J., 2004. Control point adjustment for B-spline curve approximation. Comput. Aided Des. 36, 639–652.
Yang, J., Wang, D., Hong, H., 2013. Improving angular speed uniformity by reparameterization. Comput. Aided Geom. Des. 30, 636–652.