Acta Mech
https://doi.org/10.1007/s00707-025-04385-8

ORIGINAL PAPER

Martin Kovanda · **René Marek** · **Petr Tichavský**

Parameter estimation in cyclic plastic loading

Received: 6 December 2024 / Revised: 7 May 2025 / Accepted: 14 May 2025

Abstract The increasing complexity of modern constitutive models of cyclic metal plasticity requires more efficient ways to achieve their optimal calibration. Traditional approaches, such as random search combined with Nelder–Mead optimization, are computationally expensive. In addition, they struggle with highly non-convex functions that have numerous local minima and complex behavior, making these methods highly sensitive to initial conditions. While numerical refinement is key, a better prediction for its initial point directly saves costs. In this work, we focus only on the uniaxial cyclic loading, as it is the dominant part of a general calibration process for such a model and can also utilize a closed-form solution, further speeding up the procedure. We propose a neural network framework with a loss function that combines the loss on both the predicted parameters and the generated stress responses. This network is then used to predict an initial point for Nelder–Mead optimization. Our method was also compared to the non-gradient Tensor Train Optimization method on both synthetic data and measured experiments.

1 Introduction

In this era of increasing demands for safety, durability, and material efficiency, increasing the accuracy of predicting the behavior of solid materials under various forms of loading is of great importance, as it also allows for more reliable maintenance planning. To this end, very advanced constitutive models are being developed.

Following numerous bridge and railway failures, cyclic plastic loading has become an important topic in materials research. Under such loading, metals can manifest a strong Bauschinger effect as well as several other recognizable phenomena. A constitutive model that accounts for some of these features is the main focus of this paper.

Before being used, such a material model must first be calibrated with data from a real uniaxial cyclic loading experiment. A detailed description of such a procedure is in Suchocki and Kowalewski [1]. Regardless of the form of its control, the experiment represents a link between time t, tensile stress $S(\epsilon_t, \boldsymbol{q})$ and total axial deformation $\epsilon_t(t)$, which is a sum of the elastic and plastic deformation $\epsilon_e(t)$ and $\epsilon(t)$, respectively. The set of evolving internal variables $\boldsymbol{q}$ reflects the loading history via evolution rules that are functions of the associated plastic strain. Since the elastic deformation can be easily subtracted, it is more convenient for the following analysis to work with the plastic deformation only, provided that stress points within the active elastic region

M. Kovanda (✉)· R. Marek
Institute of Thermomechanics of the Czech Academy of Sciences, Dolejškova 1402/5, Prague 8 18200, Czech Republic
E-mail: kovanda@it.cas.cz

M. Kovanda · P. Tichavský
Institute of Information Theory and Automation of the Czech Academy of Sciences, Pod Vodárenskou věží 1143/4, Prague 8 18200, Czech Republic E-mail: tichavsk@utia.cas.cz

Published online: 09 June 2025

are skipped. The relevant elastic property of the material, defined by the Young's modulus E, is determined analytically at the beginning of the analysis.

Many constitutive models have been developed to deal with various identified phenomena. A leading example of a single yield surface model with nonlinear kinematic hardening rule is the one introduced by Frederick and Armstrong [2]. Chaboche and Rousselier [3] then introduced a superposition of several of these hardening rules and obtained the multicomponent Armstrong–Frederick.

1.1 The constitutive model

The original constitutive model "MAFTr" chosen as the testing subject for this paper is described by Dafalias and Feigenbaum [4] as featuring a multicomponent Armstrong–Frederick kinematic hardening rule with a stress threshold and r-modification. In tensor form, the model uses von Mises theory for its yield function as

$$f = \tfrac{3}{2}\,(\boldsymbol{s}-\boldsymbol{\alpha}):(\boldsymbol{s}-\boldsymbol{\alpha}) - k^2, \tag{1}$$

where (:) denotes the double dot product, $\boldsymbol{s}$ is the deviatoric stress tensor obtained from the Cauchy stress tensor $\boldsymbol{\sigma}$, $\boldsymbol{\alpha}$ is the sum of four evolving deviatoric backstress components as $\boldsymbol{\alpha} = \sum_{i=1}^{4} \boldsymbol{\alpha}_i$, and k stands for the evolving size of the yield surface, initiated as the yield stress k_0. A normalized associative flow rule is $\dot{\boldsymbol{\epsilon}}_\mathrm{p} = \lambda \boldsymbol{n}$, where the dot denotes the time derivation, λ is the plastic multiplier, and $\boldsymbol{n}$ is the unit outer normal to the yield surface,

$$\boldsymbol{n} = \frac{\partial f}{\partial \boldsymbol{\sigma}} \cdot \left\| \frac{\partial f}{\partial \boldsymbol{\sigma}} \right\|^{-1} = \frac{\boldsymbol{s}-\boldsymbol{\alpha}}{\|\boldsymbol{s}-\boldsymbol{\alpha}\|}. \tag{2}$$

The first three components of backstress evolve according to

$$\dot{\boldsymbol{\alpha}}_i = \lambda \sqrt{\tfrac{2}{3}} c_i \left(\sqrt{\tfrac{2}{3}} a_i \boldsymbol{n} - (r_i \boldsymbol{\alpha}_i + (1-r_i)\,(\boldsymbol{\alpha}_i : \boldsymbol{n})) \right) \tag{3}$$

with $i = 1, 2, 3$. The initial hardening rates depend on parameters c_i and saturation levels a_i. This kinematic hardening rule employs a BCD modification using the ratio $r_i = \sqrt{3/2}\,\|\boldsymbol{\alpha}_i\| / a_i$ that alters the behavior in multiaxial loading by applying a weighted average between two recovery terms. No dedicated parameter is used to further control this property, so the model calibration process only requires uniaxial cyclic loading.

The evolution of the fourth component of backstress

$$\dot{\boldsymbol{\alpha}}_4 = \lambda \sqrt{\frac{2}{3}} c_4 \left(\sqrt{\frac{2}{3}} a_4 \boldsymbol{n} - (r_4 \boldsymbol{\alpha}_4 + (1-r_4)\,(\boldsymbol{\alpha}_4 : \boldsymbol{n})) \left\langle 1 - \frac{\bar{a}}{\|\boldsymbol{\alpha}_4\|} \right\rangle \right) \tag{4}$$

uses Macaulay brackets that deactivate the recovery terms, thereby injecting a region of linear hardening within the threshold marked by the parameter $\bar{a}$. A modified ratio $r_4 = \sqrt{3/2}\,\|\boldsymbol{\alpha}_4\| \,/\, (a_4 + \bar{a})$ reflects the added region of linear hardening. For all components, $r_i \in [0, 1]$.

To take into account the phenomena of isotropic hardening, a simple nonlinear hardening rule is added to the model after Chaboche and Rousselier [3] as

$$\dot{k} = \lambda \kappa_1 \,(1 - \kappa_2 k) \tag{5}$$

where κ_2 represents the inverted saturation level of the isotropic variable k. Parameter κ_1 further multiplies the rate of hardening.

1.2 Model's closed-form solution

For the uniaxial case, the axial stress response of the model can be reduced to a scalar function of plastic strain only. Orientation of loading is denoted by the variable $D = 1$ for tension and $D = -1$ for compression. Given the initial values of oriented backstress component norms α_i^0 and the value of accumulated equivalent plastic

Table 1 Parameters of analytical model developed by Marek et al. and their descriptions

Parameter	Unit	Description
k_0	MPa	Initial yield strength
κ_1	MPa	Adjustment of the rate of isotropic hardening
κ_2	MPa^{-1}	Inverted saturation limit of isotropic hardening
c_i	-	Adjustment of the evolution rates of the backstress components
a_i	MPa	Saturation limits of the backstress components
$\bar{a}$	MPa	The threshold of the last component to nonlinear behavior

strain ϵ^0, the next increment of accumulated plastic strain $\Delta\epsilon > 0$ in the direction D drives the evolution of model's internal variables according to

$$k = \kappa_2^{-1}\left[1-(1-\kappa_2 k_0)\exp\left(-\sqrt{\tfrac{3}{2}}\kappa_1\kappa_2\left(\epsilon^0+\Delta\epsilon\right)\right)\right] \tag{6}$$

and

$$\alpha_i = \sqrt{\tfrac{2}{3}}Da_i - \left(\sqrt{\tfrac{2}{3}}Da_i - \alpha_i^0\right)\exp\left(-c_i\Delta\epsilon\right) \tag{7}$$

for $i = 1, 2, 3$. If $\bar{a} > 0$, the evolution of the fourth component of backstress requires calculating thresholds of plastic strain increments required to enter and exit the region of linear hardening. See Marek et al. [5] for the details of the procedure. Finally, the axial stress response of the model reads

$$M_\theta = \sqrt{\frac{3}{2}}\sum_{i=1}^{4}\alpha_i + Dk. \tag{8}$$

The vector parameter $\boldsymbol{\theta}$ of the model includes the initial yield strength k_0, the adjustment of the rate of isotropic hardening κ_1, the inverted saturation limit of isotropic hardening κ_2, coefficients c_i of the evolution rates of the backstress components, the components' saturation limits a_i, and the linear threshold of the last component, see Table 1.

The vectors $\mathbf{c}$ and $\mathbf{a}$ have the same arbitrary length representing the number of backstress components. With a higher number, the model may provide better accuracy, but at the cost of increased complexity and memory requirements. As in the original paper by Dafalias and Feigenbaum [4], four components are used, which gives $\boldsymbol{\theta} \in \mathbb{R}_+^{12}$.

The task is to propose reliable methods for estimating an optimal $\boldsymbol{\theta}^*$ for the model to describe the material behavior in a given experiment as accurately as possible. Formally, the goal is to obtain

$$\boldsymbol{\theta}^* := \underset{\boldsymbol{\theta}\in\mathbb{R}_+^{12}}{\arg\min}\, L_2\left(\left|\mathbf{S}_m - M_{\boldsymbol{\theta}}(\epsilon)\right|\right) \tag{9}$$

for a known ϵ corresponding to the measured experiment, where $\mathbf{S}_m$ refers to the measured stress response and L_2 stands for the L_2 norm in the space of the stress responses.

1.3 Organization of the paper

The remainder of the paper is structured as follows. Section 2 summarizes the optimization techniques. Section 3 describes how the real experimental data are processed to be used for neural networks. It also proposes the a priori distribution of the estimated parameters of the material model. This distribution is then used in the data generation procedure needed to train neural networks. Two synthetic datasets are generated, one for a fixed plastic deformation sequence and the second for multiple setups. In Section 4, four different neural network architectures are described along with the training strategy and the used loss function. Section 5 presents the results of these approaches and then compares the best trained network with the control procedure and the randomly initialized Nelder–Mead method. Finally, in Section 6, the accuracy of the found parameters is analyzed using CRLB and Monte Carlo simulations. Concluding remarks are given in Section 7.

2 Optimization techniques

Estimating $\boldsymbol{\theta}^*$ is a non-trivial problem since $L_2(|\mathbf{S}_m - M_{\boldsymbol{\theta}}(\epsilon)|)$ is generally not a convex function. Current approaches often use a random search to determine the initial point of the Nelder and Mead [6] method. This method is a non-gradient optimization technique that iteratively refines potential solutions until an optimal result is achieved. However, this approach is time-consuming and usually finds a suboptimal solution that is far from the global minimum.

In this section, we review the optimization techniques that we propose to use. First, there are neural network techniques, see, e.g., a comprehensive review of deep learning methods applied to computational mechanics of Vu-Quoc and Humer [20]. Second, it is the tensor-based optimization TTOpt [7].

2.1 Neural networks

Neural networks started to dominate over other methods in more and more domains because of their ability to learn non-trivial features from the training dataset using a gradient-based back-propagation method. For the calibration of material models (i.e., the estimation of model parameters), the early neural network models were based on feed-forward networks (FFNs), e.g., in Mareš et al. [8]. FFNs consist of layers of neurons where information flows in one direction, from input to output, without loops, making them relatively simple and efficient for many tasks. Other approaches included physically informed neural networks (PINNs), introduced by Raissi et al. [9]. These networks integrate physical laws into the neural network training process, often by incorporating differential equations as part of the loss function. PINNs also face challenges, such as the complexity of incorporating complex physical laws into neural architectures. This is treated in Haghighat et al. [10].

Recently, Schulte et al. [11] proposed using a neural network to predict an initial point for a subsequent Nelder–Mead optimization method. This approach significantly reduces the time complexity compared to a random search that yields similar stress estimates. They showed that the integration of ANNs with classical parameter identification significantly improves the initial guess accuracy for material model calibration, thus increasing the overall efficiency.

Early convolutional neural networks (CNNs) were designed as a stack of multiple convolutional and pooling layers. This linear scheme was soon abandoned after the introduction of GoogLeNet (also known as Inception v1) by Szegedy et al.[12]. Soon this scheme began to be used for signal processing as well. For example, InceptionTime, developed by Fawaz et al. [13], outperformed the best previous methods in a variety of time series classification contests.

Alternatively, recurrent neural networks (RNNs) can also be applied to signal data. However, these networks often suffered from vanishing or exploding gradients. To solve this problem, Hochreiter and Schmidhuber developed a module called the long short-term memory (LSTM) [14], which introduces a complex gating mechanism to regulate the flow of information, allowing the network to retain important long-term dependencies while avoiding the vanishing gradient problem. This architecture effectively maintains a balance between remembering and forgetting information through its unique design of input, output, and forgetting gates. The main drawback is its computational complexity. This problem was partially solved after many years in the gated recurrent units (GRUs) module created by Cho et al. [15]. GRU simplified the LSTM design by combining the input and forget gates into a single update gate, and merging the cell state and the hidden state. This step significantly reduced the complexity of the module and made all training more efficient.

Overall, a challenge that seems to be consistent among various neural network based papers is to obtain a good training dataset. This problem is usually solved by generating synthetic data using a chosen material model. In this paper, we propose two different ways of generating a training dataset. The first one contains only one plastic deformation sequence. However, subjecting a new specimen to a prescribed sequence of plastic deformations requires monitoring of stress levels and a dynamic control of loading amplitudes, which can be difficult to achieve. The second approach involves a variety of plastic deformation sequences and is therefore more general, but also more complex and challenging for neural networks.

Finally, in this paper we also propose a loss function based on stress prediction, which significantly improves the network capabilities. However, there are still challenges in proposing an appropriate architecture and finding optimal hyperparameters. This often involves training multiple architectures on the training dataset and selecting the best performing one on the validation dataset.

2.2 Tensor train optimization

The Tensor Train Optimization method, TTOpt, was introduced by Sozykin et al. [7]. It is a general non-gradient method for minimization of multivariate functions. This method uses sampling of the function on a rectangular grid to obtain a high-order tensor. The tensor is then approximated by the Tensor Train (TT) model [16] with a moderate number of parameters. Although the number of tensor elements can be astronomically large, only a fraction of them is used to build the model, as is done in the so-called TT-cross algorithm introduced by Oseledets and Tyrtyshnikov [17].

The minimum tensor element is found as a by-product of the TT model construction. It does not use any database, and it is just a special optimization procedure. It only requires the utility function to be callable and a discrete grid over which the function is to be minimized. A python implementation of the procedure is available online [21]. The optimization result lies in the chosen grid, so it is only an approximation of the function minimum. Similar to neural networks, it can also be used as an initialization point for the simplex refinement procedure.

3 Data preparation

Experimental measurements of two 304-L stainless steel specimens were performed to test the newly developed tools.

Sequences of generic uniaxial strain-controlled loading segments were performed on individual specimens, while the stress response was recorded at a sampling rate of 10 Hz for a total of 4 h. A moving average noise reduction technique was chosen for further processing. A mild smoothing process was applied first, followed by the identification and reconstruction of reversal points. These appear partially rounded due to inherent viscoplastic and other hysteretic phenomena. As the model in question does not capture this type of behavior, a controlled elimination of this minute detail was chosen. A final round of smoothing was then applied, which worked between these reversal points without affecting them. In addition, where necessary, subtle corrections were made to ensure that the plastic strain data points within each loading segment represented monotonic functions to ensure an existing solution for interpolation. Overall, this method effectively reduces noise while maintaining the integrity of the data.

Since the MAFTr model does not depend on the speed of the experiment, only the plastic modulus and its change indicate the density of samples required to retain most of the contained information. Downsampling the data is desired as it reduces computational cost. However, this process needs to preserve points of reversals as they in fact define the experiment. Rather than simply choosing the sampling rate according to the plastic modulus, we have decided to help the neural networks by preparing the data with a preset period of $N = 17$ increments for each segment of plastic loading, so as not to confuse them with randomized occurrence of reversals. In extreme cases of very short segments of loading, this can produce oversampling.

Let $\epsilon_0 = 0$ stand for the plastic deformation at the beginning of the experiment, and let $\epsilon_1^{(\mathrm{rev})}, \ldots, \epsilon_K^{(\mathrm{rev})}$ represent the plastic deformation at the end of each segment, where K is the number of all loading segments. In this paper, $K = 40$ is chosen for both measured and synthetic data. Then, the k-th segment is divided into N increments of different lengths $\delta_k^{(n)}$ such that

$$\epsilon_k^{(\mathrm{rev})} = \epsilon_{k-1}^{(\mathrm{rev})} + \sum_{n=1}^{N} \delta_k^{(n)}. \tag{10}$$

This setup gives a total of $Q = 1 + NK = 681$ points.

More data points are needed at the beginning of each loading segment, where the stress level changes more rapidly. For this purpose, the increments of plastic deformation are generated with a bias using a geometric sequence. Let $R = \delta_k^{(N)}/\delta_k^{(1)}$ be the ratio between the first and last increment. In this paper, $R = 20$ was chosen. The geometric sequence of the increments $\delta_k^{(1)}, \ldots, \delta_k^{(N)}$ is generated as

$$\delta_k^{(n+1)} := \sqrt[N-1]{R}\, \delta_k^{(n)}, \ \forall n \in \{1, \ldots, N-1\}, \tag{11}$$

where

$$\delta_k^{(1)} := \frac{\left(\epsilon_k^{(\mathrm{rev})} - \epsilon_{k-1}^{(\mathrm{rev})}\right)\left(\sqrt[N-1]{R} - 1\right)}{R^{N/(N-1)} - 1}. \tag{12}$$

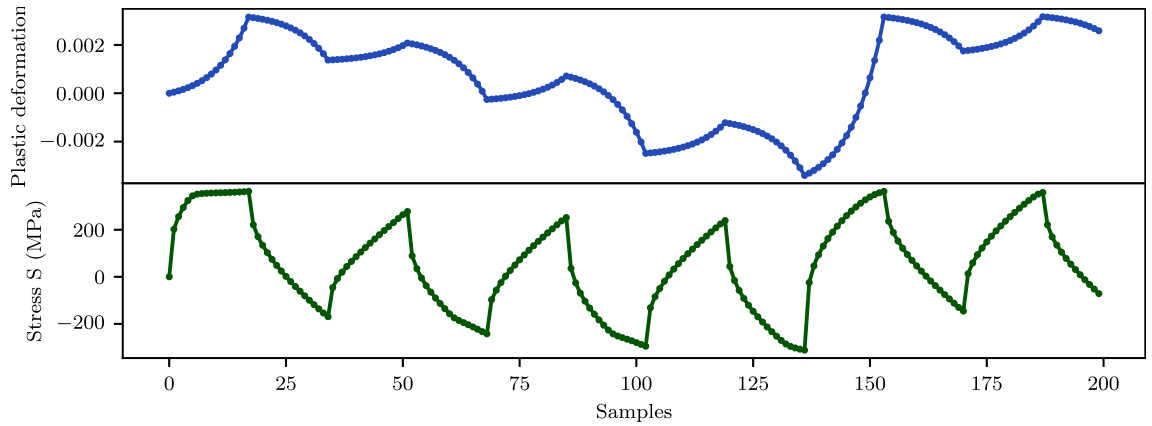

Fig. 1 Top: Example of the first 200 points of plastic deformation from the first measured experiment generated using Eq. (14) to cover each loading segment by 16 inner points in a geometric sequence of increments. Bottom: Interpolated stress from the measured data corresponding to the interpolated deformation points

Table 2 Range of the a priori uniform distribution for each (transformed) parameter, given the conditions $\sum a_i \in [100, 400]$ and $c_1 \geq c_2 \geq c_3$. The symbol log stands for the natural logarithm

	k_0	κ_1	κ_2^{-1}	$\log(c_1)$	$\log(c_2)$	$\log(c_{3,4})$	$a_{1,2,3,4}$	$\overline{a}$
min	15	100	30	$\log(1,000)$	$\log(10)$	$\log(10)$	0	0
max	250	10,000	150	$\log(50,000)$	$\log(5,000)$	$\log(2,000)$	200	500

The interpolated data points are then calculated as

$$\epsilon_i^{(n)} = \epsilon_{i-1}^{(\mathrm{rev})} + \sum_{k=1}^{n} \delta_k^{(i)}, \tag{13}$$

$\forall k \in \{1, \ldots, K\},\ \forall n \in \{1, \ldots, N-1\}$.

The resulting plastic deformations interpolated from the first measured experiment are therefore defined as

$$\boldsymbol{\epsilon}^{(\exp_1)} := \left(0, \epsilon_1^{(1)}, \ldots, \epsilon_1^{(N)}, \ldots, \epsilon_K^{(1)}, \ldots, \epsilon_K^{(N)}\right), \tag{14}$$

where $\epsilon_k^{(N)} := \epsilon_k^{(\mathrm{rev})}$ stands for the k-th reversal point, see Figure 1.

3.1 A priori $\boldsymbol{\theta}$ distribution

After preparing the sequence of plastic deformations and their associated stress responses, it is possible to estimate the parameter $\boldsymbol{\theta}$. For both neural networks and TTOpt, it is necessary to first select an a priori distribution for $\boldsymbol{\theta}$. In the case of neural networks, this allows the creation of a training set of parameters and corresponding stress responses (see Sect. 3.2). TTOpt, on the other hand, requires a region defined by the Cartesian product of intervals $[\theta_{i,\min}, \theta_{i,\max}]$ for each parameter θ_i in which the optimal $\boldsymbol{\theta}$ is to be found, and an appropriately chosen sampling for each of these intervals.

To cover the entire hand-picked interval, a uniform distribution is chosen separately for each parameter θ_i, see Table 2. All suggested ranges are more than adequate for most structural steels and can be easily adjusted as needed. There are two additional constraints. The parameters a_i are generated so that their sum would be in the range $[100, 400]$. Second, since the pairs $(c_i, a_i),\ i \in \{1, 2, 3\}$ are commutative, they are generated with the condition $c_1 \geq c_2 \geq c_3$ to make the training objective unique. Unsorted parameters would make it harder for neural networks to predict correct values because their order would be inconsistent. For example, an optimal solution with permuted (c_i, a_i) pairs could be considered incorrect.

In practice, the first condition is realized by first generating $\widetilde{a} \sim U[100, 400]$ and $a'_1, \ldots, a'_4 \sim U[0, 1]$. Then, the desired parameters $a_1, \ldots, a_4$ are calculated as

$$a_i := \frac{a'_i}{\sum_{j=1}^{4} a'_j}\widetilde{a}, \qquad \forall i \in \{1, \ldots, 4\}. \tag{15}$$

This process is repeated until the a_i satisfy the condition $a_i \in [0, 200]$. The second condition is solved by simply sorting the first three c_i parameters.

3.2 Dataset

Training neural networks requires large amounts of data. Running real experiments on such a scale is not feasible. However, the analytical model $M_{\boldsymbol{\theta}}$ provides a fast method to obtain an approximate stress response in a cyclic plastic loading experiment, given the parameter $\boldsymbol{\theta}$ and the plastic deformation ϵ. For this reason, the dataset is created by generating $\boldsymbol{\theta}$ from a designed random distribution and then obtaining stress responses using the analytical model $M_{\boldsymbol{\theta}}$.

In this paper, two datasets are created. The first dataset is generated to train neural networks specifically for the plastic deformation measured in the first real experiment $\boldsymbol{\epsilon}^{(\mathrm{exp}_1)}$. The second dataset is then generalized so that the neural networks would be able to perform parameter estimation for experiments with any sequence of plastic deformation with at least 40 segments.

Let $\mathbf{P}_{12}$ represent the a priori distribution for $\boldsymbol{\theta}$ described in Section 3. The first dataset $\mathbf{D}_1$ consists of pairs $(S_i, \boldsymbol{\theta}_i)$ and is created as

$$\mathbf{D}_1 := \left\{ \left(\mathbf{M}_{\boldsymbol{\theta}_i}\left(\boldsymbol{\epsilon}^{(\mathrm{exp}_1)}\right),\ \boldsymbol{\theta}_i \right),\ i \in \{1, \ldots, I\} \right\}, \tag{16}$$

where $\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_I \overset{\mathrm{iid}}{\sim} \mathbf{P}_{12}$ and $I = 10^6$ is the chosen length of the dataset. Our experiments showed that taking more data does not significantly improve the quality of trained models, see Sect. 5. The plastic deformation does not need to be a part of the dataset, because it is identical for all loading trajectories in the dataset.

The second dataset is generated to develop models capable of parameter estimation independent of the experimental deformation setting. In this case, the newly generated dataset $\mathbf{D}_2$ must also include the plastic deformation, since it is different in all generated data. These deformations need to be generated to mimic real experiments. Since the material model depends only on the deformation and the speed of the experiment is arbitrary, it is convenient to first generate the plastic deformation only at the end of each loading segment $\epsilon_1^{(\mathrm{rev})}, \ldots, \epsilon_K^{(\mathrm{rev})}$, where $K = 40$ stands for the number of segments.

Let $[-B, B]$, $B = 0.007$, represent the boundary interval of $\epsilon_k^{(\mathrm{rev})}$, $\kappa = 0.0002$ stand for the minimum allowed absolute value of the increment and $\epsilon_0^{(\mathrm{rev})} = 0$ represent the starting point. Then, each new reversal point is generated recursively as

$$\epsilon_k^{(\mathrm{rev})} = a_k + \beta_k\left((-1)^{k+1}B - a_k\right), \tag{17}$$

where a_k is defined as

$$a_k := \epsilon_{k-1}^{(\mathrm{rev})} + (-1)^{k+1}\kappa \tag{18}$$

and $\beta_k \sim \mathrm{Beta}(2, 6)$ is a randomly generated parameter that satisfies the condition $0 \leq \beta_k \leq 1$. An additional rule is set so that for $k > 2$ there is a 20% chance that $\epsilon_k^{(\mathrm{rev})} = \epsilon_{k-2}^{(\mathrm{rev})}$. This simulates repeating the same cycles multiple times, which is often present in real applications, albeit in the form of total strain or stress control. An example of randomly generated plastic deformations is shown in Fig. 2. The data points within each segment are then generated using the Eq. (13). An example of a randomly generated set of data points is shown in Fig. 1.

Let $\mathbf{U}_\epsilon$ represent the combined distribution of all data samples generated using the procedure described in Eq. (14), i.e.,

$$\boldsymbol{\epsilon} := \left(0, \epsilon_1^{(1)}, \ldots, \epsilon_1^{(N)}, \ldots, \epsilon_K^{(1)}, \ldots, \epsilon_K^{(N)}\right), \tag{19}$$

$\forall \boldsymbol{\epsilon} \sim \mathbf{U}_\epsilon$, where $\epsilon_k^{(N)} := \epsilon_k^{(\mathrm{rev})}$ stands for the k-th reversal point, $N = 17$ is the number of deformation points in each segment, and $K = 40$ stands for the number of segments. Then, the second dataset $\mathbf{D}_2$ consists of the triplets $(S_i, \boldsymbol{\epsilon}_i, \boldsymbol{\theta}_i)$ and is created as

$$\mathbf{D}_2 := \left\{ \left(\mathbf{M}_{\boldsymbol{\theta}_i}\left(\boldsymbol{\epsilon}_i\right),\ \boldsymbol{\epsilon}_i,\ \boldsymbol{\theta}_i \right),\ i \in \{1, \ldots, I\} \right\}, \tag{20}$$

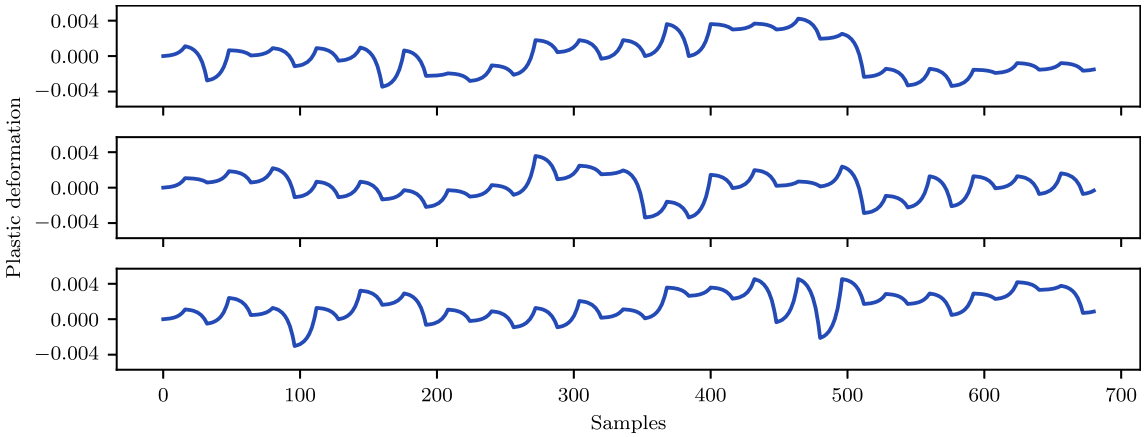

Fig. 2 Example of 3 different randomly generated sequences of plastic deformation in the dataset $\mathbf{D}_2$

where $\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_I \overset{\text{iid}}{\sim} \mathbf{P}_{12}$, $\boldsymbol{\epsilon}_1, \ldots, \boldsymbol{\epsilon}_I \overset{\text{iid}}{\sim} \mathbf{U}_\epsilon$ and $I = 10^6$ is the chosen length of the dataset. The overall generation of both dataset $\mathbf{D}_1$ and $\mathbf{D}_2$ takes about 5 min on AMD Ryzen 9 7900 processor.

Similar to the training datasets $\mathbf{D}_1$ and $\mathbf{D}_2$, corresponding validation datasets $\mathbf{D}_1^V$, $\mathbf{D}_2^V$ with 20,000 tuples and test datasets $\mathbf{D}_1^T$, $\mathbf{D}_2^T$ with 1024 tuples are created for evaluation purposes.

4 Optimization details

In this paper, feed-forward networks (FFNs), convolutional neural networks (CNNs), gated recurrent unit (GRU), and long short-term memory (LSTM) are evaluated. Each architecture takes as an input a matrix of shape 1×681 for the dataset $\mathbf{D}_1$ and 2×681 for $\mathbf{D}_2$ because the second dataset also contains plastic deformation. To bring the variance closer to 1, each architecture first transforms the signal using the batch normalization, which converts the input data separately for each channel (for both stress and plastic deformation, if applied).

The hyperparameters of the architecture (number and type of layers, number of neurons in each feed-forward layer, etc.) are first found by a manual directed search based on their performance on the validation datasets $\mathbf{D}_1^V$ or $\mathbf{D}_2^V$. Networks are trained on the training set in two epochs using the AdamW optimizer created by Loshchilov and Hutter [18], with a learning rate of about 10^{-3} and a weight decay of 10^{-2}. Our experiments showed that the best performing networks were trained with a learning rate in the range of $[0.0005, 0.002]$. The batch size is always set to 128. Finally, each network is evaluated on a test dataset $\mathbf{D}_1^T$ or $\mathbf{D}_2^T$. All neural network implementations were done using the PyTorch framework on the NVIDIA GeForce RTX 4090 graphics card.

4.1 Loss functions

Each θ_i parameter has a different order of magnitude. For this reason, it would be difficult to train neural networks to predict them directly, since the L_2 loss function used would most likely favor some parameters over others. To avoid this potential problem, each $\boldsymbol{\theta}$ is normalized element-wise based on means and variances determined by the distribution $\mathbf{P}_{12}$. In practice, these means and variances are computed based on 10^6 realizations of the $\mathbf{P}_{12}$ distribution.

Let $\boldsymbol{\theta}^{(N)}$ represent the parameter $\boldsymbol{\theta}$ normalized element-wise using the mean and variance estimates. The output of each model is a vector of length 12 representing the parameter $\boldsymbol{\theta}^{(N)}$. After scaling, there is no easy way to prevent the networks from predicting negative values for individual parameters. For this reason, the post-processing should include replacing any negative estimated parameter with a small number $\eta = 10^{-9}$ chosen for numerical stability. On the other hand, since negative values were not present in the training dataset, a prediction with negative parameters may indicate a poorly designed network.

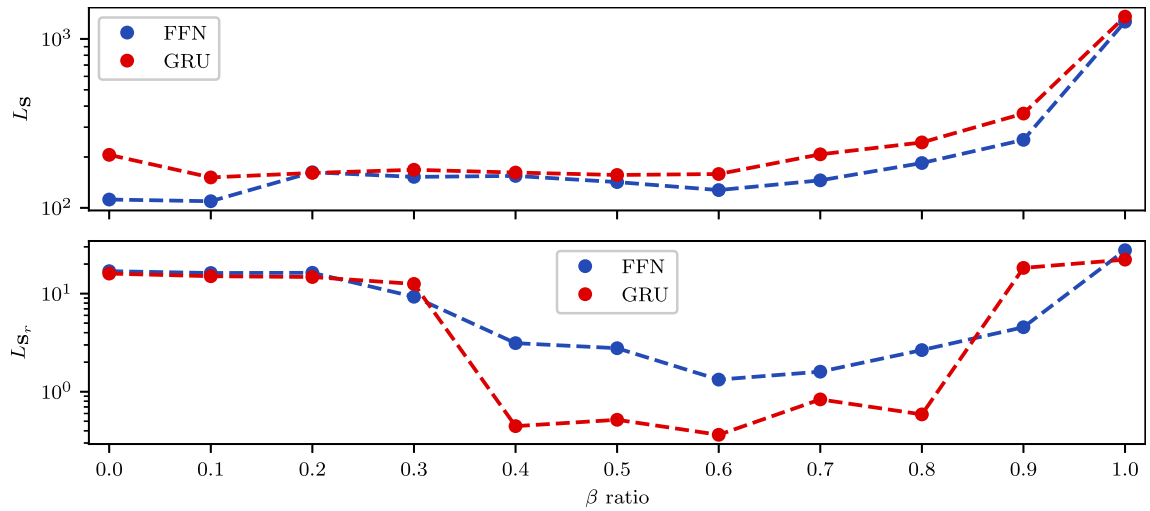

Fig. 3 $L_{\mathbf{S}}$ and $L_{\mathbf{S}_r}$ metrics for randomly configured FFN and GRU trained with $L^{\beta}_{\boldsymbol{\theta},\mathbf{S}}$ for different β ratios, see Eq. (23)

The performance of the developed architectures is measured by 2 metrics. The first one is the $L_{\boldsymbol{\theta}}$, defined as

$$L_{\boldsymbol{\theta}} := \frac{1}{12}\sum_{i=1}^{12}\left(\theta_i^{(N)} - \widehat{\theta}_i^{(N)}\right)^2. \tag{21}$$

The normalized parameter $\boldsymbol{\theta}^{(N)}$ is used to deal with different scales between individual parameters. The second metric is the $L_{\mathbf{S}}$, which measures the average quadratic difference between the reference stress $\mathbf{S}$ and the predicted stress, i.e.,

$$L_{\mathbf{S}} := \frac{1}{Q}\sum_{i=1}^{Q}\left(S_i - \widehat{S}_i\right)^2, \quad \widehat{\mathbf{S}} := M_{\widehat{\boldsymbol{\theta}}}(\boldsymbol{\epsilon}), \tag{22}$$

where $Q = 681$ is the number of plastic deformation samples. This metric indicates how far the predicted stress is from the reference stress. Since the neural network works with the stress responses only, the latter criterion seems more natural.

Note that unlike the former criterion, $L_{\mathbf{S}}$ requires additional implementation, since the gradient must flow through the analytic model $M_{\widehat{\boldsymbol{\theta}}}$ here. This was achieved by using the PyTorch autograd method [19], which automatically calculates gradients for the computational graph defined by $M_{\widehat{\boldsymbol{\theta}}}$.

Although $L_{\mathbf{S}}$ is more difficult to evaluate, it is easier to learn by the network. Typically, only two epochs of training appear to be needed compared to ten epochs when $L_{\boldsymbol{\theta}}$ is used.

It was observed that the training can be further improved by using a linear combination between the parameter loss and the stress loss, i.e.,

$$L^{\beta}_{\boldsymbol{\theta},\mathbf{S}} := \beta L_{\boldsymbol{\theta}} + \alpha(1-\beta)L_{\mathbf{S}}, \tag{23}$$

$\beta \in [0, 1]$, $\alpha \in \mathbb{R}_+$. The parameter $\alpha = 30$ is chosen for convenience to minimize the difference between $L_{\boldsymbol{\theta}}$ and $L_{\mathbf{S}}$ at the beginning of training.

Numerical experiments show that training the networks with $L^{\beta}_{\boldsymbol{\theta},\mathbf{S}}$ improves the performance of the models, as they are now trained to make closer stress estimates while trying to predict similar $\boldsymbol{\theta}$, see Fig. 3. The optimal β seems to be $\beta \in [0.4, 0.8]$ for randomly generated FFN and GRU. This observation seems to be consistent across our experiments. In this paper, $\beta = 0.5$ was chosen for all architectures. Note, however, that the improvement in performance becomes apparent only after the Nelder–Mead refinement is applied.

4.2 Feed-forward networks (FFN)

One of the simplest architectures is a neural network based solely on feed-forward layers. Each hidden layer is followed by a ReLU activation function. Since $\boldsymbol{\theta}^{(N)}$ can be negative, no activation function is used after

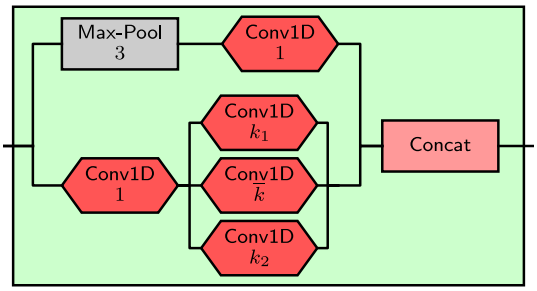

Fig. 4 Diagram of InceptionBlock used in this paper

the last layer. ReLU is used because it is the simplest nonlinear activation function commonly used for neural networks. The advantage of using this architecture is its low computational complexity while achieving sufficient performance. The disadvantage is the need to have a fixed input size. In this paper, this is not a problem since the input always has a fixed length of $Q = 681$. In practice, however, the measured stress would have to be clipped to this length. The set of hyperparameters consists of a number of layers with a decreasing number of neurons in each successive layer.

4.3 Convolutional neural networks (CNN)

Convolutional neural networks have gained popularity for tasks such as image recognition and object detection. Soon they began to be tested also on 1D signals. In 2020, an architecture called InceptionTime, developed by Fawaz et al. [13], surpassed the previous best non-neural model, HIVE-COTE, in terms of time series classification performance. For this reason, a modification of InceptionTime is tested for parameter estimation. In this paper, we use a sequence of N InceptionBlocks followed by a Conv1D layer to produce only 1 signal channel in the output. This result vector is then processed in some cases by M stacked FFN layers and finally by a feed-forward layer to get 12 outputs.

Each InceptionBlock consists of 2 parallel threads. The first uses maximum pooling with a size of 3, followed by a Conv1D layer with 32 filters of size 1. In the second thread, the signal is first processed with 64 Conv1D filters of size 1, creating a so-called bottleneck. This bottleneck is then processed by 3 groups of 32 Conv1D filters with sizes k_1, k_2 and $\overline{k} = \frac{1}{2}(k_1 + k_2)$, where k_1 and k_2 are hyperparameters to be found. These 4×32 output channels are then concatenated along the channel axis into a final output matrix. A diagram of the InceptionBlock is shown in Fig. 4.

4.4 Recurrent neural networks (RNN)

In recent years, recurrent neural networks have been widely used for time series analysis. Their great advantage is that they can take an input of any length and produce a constant output shape. This makes them potentially more suitable for this task, since in practice the experiment may always vary in length.

The main drawback is that recurrent networks must be able to remember all the necessary information from the entire signal. For this reason, the simple recurrent unit tends to perform poorly for longer inputs. In this paper, we instead use bidirectional versions of both the gated recurrent unit (GRU) [15] and the long short-term memory (LSTM) units [14]. These commonly used units are designed to overcome the problem of vanishing gradients when processing long sequences by implementing a form of memory called hidden state. This allows the networks to better retain information from the beginning of the signal to the end.

LSTM and GRU both have the number of layers and the size of the hidden state as hyperparameters. The number and size indicate the dimensionality of the hidden state and directly affect the capacity and complexity of the model. Our experiments showed that adding fully connected layers does not generally lead to better performance. For this reason, all considered networks did not include feed-forward layers.

Table 3 Metrics of 12 chosen FFN networks on test dataset $\mathbf{D}_1^T$

id	FFN layers	$L_{\mathbf{S}}$	$L_{\mathbf{S}_r}$
1	[231, 210]	155.16	2.874
2	[231, 225]	186.29	3.223
3	[250, 183]	231.84	4.486
4	[302, 127]	269.11	5.217
5	[167, 130, 64]	247.27	6.579
6	[324, 205, 123]	200.53	4.603
7	[393, 208, 98]	151.73	3.149
8	[182, 171, 119, 101]	**84.02**	2.492
9	[218, 208, 142, 53]	167.37	3.778
10	[236, 140, 117, 113]	124.15	2.644
11	[393, 208, 98, 52]	106.90	**2.314**
12	[393, 208, 113, 52]	170.36	4.039

Table 4 Metrics of 11 chosen InceptionTime networks on test dataset $\mathbf{D}_1^T$

id	k_1	k_2	Blocks	FFN layers	$L_{\mathbf{S}}$	$L_{\mathbf{S}_r}$
1	3	11	2	–	94.34	1.169
2	9	21	2	–	144.57	0.343
3	21	45	1	–	177.14	3.691
4	9	37	1	[310]	228.68	1.241
5	9	39	1	[310]	104.15	0.366
6	7	15	2	[295, 143]	238.78	1.593
7	17	17	3	[172, 138]	139.93	1.552
8	15	19	1	[359, 155]	**89.16**	**0.245**
9	7	23	2	[232, 155]	143.85	1.205
10	21	27	2	[232, 223]	119.52	0.435
11	9	31	2	[357, 85]	267.66	3.218

5 Results

The performance of 12 selected FFN networks on dataset $\mathbf{D}_1^T$ is shown in Table 3. These results show that it can be difficult to find the best performing FFN network, as even a small change in the network structure has a large effect on the overall network performance, e.g., networks #11 and #12. In addition, performance improves dramatically after a Nelder–Mead simplex refinement is used. It is also worth noting that the best performing network does not necessarily perform well after refinement, indicating that the prediction may be closer to a worse local minimum.

Table 4 shows that InceptionTime has a potential to produce better predictions than FFN. However, similar to FFN, it is very difficult to find appropriate hyperparameters, see, e.g., networks #4 and #5. Furthermore, since the output is generated by a fully connected layer, the trained networks must also have the same input shape.

The results of 14 LSTM networks are shown in Table 5. These metrics show that using a too small hidden size value leads to poorly performing networks, even with a high number of layers. These results also show that while the best performing InceptionTime network seems to give better results than the best LSTM network, the LSTM seems to be more stable with respect to the choice of hyperparameters.

The final architecture that has been tested is based on GRU. The results shown in Table 6 indicate that this architecture is both stable and very powerful. The refined predictions are better compared to all previously tested methods. For large hidden size value, this architecture seems to overfit the training data and therefore performs poorly on the test data.

5.1 Results for dataset $\mathbf{D}_2$

All previous results were associated with the $\mathbf{D}_1$ dataset. The second dataset consists of both stress response and plastic deformation, as it is designed to train networks capable of estimating parameters for various deformation sequences. This makes the second dataset more challenging.

Table 5 Metrics of 14 chosen LSTM networks on test dataset $\mathbf{D}_1^T$

id	Hidden size	Layers	$L_{\mathbf{S}}$	$L_{\mathbf{S}_r}$
1	**38**	5	397.51	13.669
2	41	**2**	362.42	13.132
3	97	6	283.98	4.207
4	124	5	278.95	4.178
5	136	**2**	235.66	2.263
6	152	5	149.91	**0.510**
7	163	**2**	267.55	2.949
8	169	**2**	228.27	1.654
9	172	7	172.97	0.585
10	213	6	**146.08**	0.637
11	217	4	195.70	0.645
12	220	5	174.14	0.774
13	267	**2**	162.68	0.661
14	274	**2**	206.03	0.771

Table 6 Metrics of 12 chosen GRU networks on test dataset $\mathbf{D}_1^T$

id	Hidden size	Layers	$L_{\mathbf{S}}$	$L_{\mathbf{S}_r}$
1	**38**	9	152.93	0.657
2	45	8	135.45	0.273
3	59	8	111.64	**0.137**
4	84	4	143.48	0.262
5	96	**3**	135.01	0.269
6	110	5	148.99	0.164
7	154	**3**	173.91	0.993
8	193	5	133.06	0.154
9	197	7	154.69	0.234
10	225	**3**	**103.47**	0.167
11	298	**3**	135.64	0.165
12	385	**3**	266.96	8.971

Table 7 Metrics of 12 chosen TTOpt models on test dataset $\mathbf{D}_2^T$. F describes the fineness of the grid, i.e., the number of points along each dimension

id	Rank	F	$L_{\mathbf{S}}$	$L_{\mathbf{S}_r}$
1	1	2^{10}	1461.41	326.960
2	1	2^{15}	410.85	16.785
3	2	2^{8}	90.03	1.960
4	2	2^{10}	165.84	5.089
5	3	2^{8}	**48.93**	3.638
6	3	2^{10}	63.30	**0.961**
7	3	2^{12}	91.16	4.463
8	5	2^{20}	149.65	4.569
9	7	2^{20}	295.06	4.826
10	10	2^{20}	617.58	38.652
11	15	2^{20}	1021.95	133.278
12	20	2^{20}	1747.98	289.652

Since the TTOpt method does not require a training dataset, all variants were tested on the $\mathbf{D}_2$ dataset only. The results for 12 different models are shown in Table 7. Our experiments show that the best performing TTOpt #6 has a rank of 3 and the used grid has 1024 points along each dimension.

To compare the performance of neural networks on the $\mathbf{D}_2$ dataset, the 3 best performing networks from each architecture tested were modified to take 2 input channels and then trained on the $\mathbf{D}_2$ dataset. The results are shown in Table 8. As expected, the results are generally worse than for the $\mathbf{D}_1$ dataset. However, only the GRU networks seem to keep the performance at a similar level.

Table 8 Metrics of TOP 3 networks of each tried architecture on test dataset $\mathbf{D}_2^T$ compared to TOP3 TTOpt models

Network	$L_{\mathbf{S}}$	$L_{\mathbf{S}_r}$
FFN #08	350.02	7.058
FFN #10	336.91	6.378
FFN #11	377.13	8.957
GRU #03	207.18	1.015
GRU #06	242.43	**0.500**
GRU #08	182.71	0.762
InceptionTime #02	450.62	4.383
InceptionTime #05	580.00	4.137
InceptionTime #08	526.86	4.578
LSTM #06	619.05	14.187
LSTM #09	558.68	18.547
LSTM #10	528.03	7.523
TTOpt #03	90.03	1.960
TTOpt #05	**48.93**	3.638
TTOpt #06	63.30	0.961

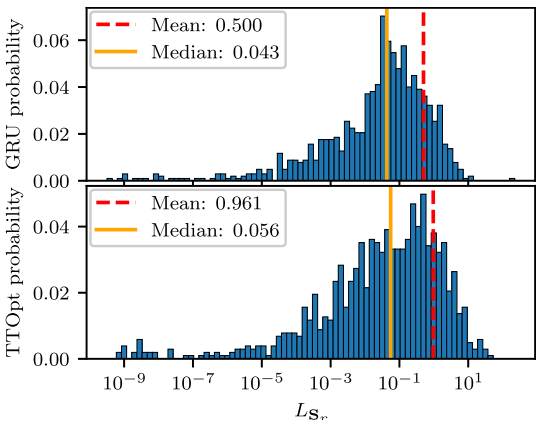

Fig. 5 Histogram of refined predictions of both GRU #6 and TTOpt #6 on test dataset $\mathbf{D}^T$

A histogram of predictions of both GRU #6 and TTOpt #6 on test dataset $\mathbf{D}^T$ is depicted in Fig. 5. These graphs show that the medians of predictions on the synthetic dataset are lower than the mean values, meaning that most of the data are predicted better than the average.

The predictions of GRU #6 and TTOpt #6 are shown in Figs. 6 and 7. The refined GRU prediction is slightly better than the refined TTOpt prediction, but both approaches produce high-quality results. As expected, the model's performance tends to be worse during randomized loading due to the limitations of its underlying constitutive theory.

Both methods also perform well against random search with Nelder–Mead simplex optimization. Figures 8 and 9 show a histogram of $L_{\mathbf{S}_r}$ values based on 400,000 randomly generated $\boldsymbol{\theta}$ from the a priori distribution described in Section 3.1 and a histogram of 4,000 TTOpt predictions after their refinement using Nelder–Mead simplex optimization. For comparison, a single GRU refined prediction is depicted by a dashed line. These figures show that the use of both GRU and TTOpt provides a high-quality starting point for the Nelder–Mead optimization, as they outperform the vast majority of randomly generated estimates after refinement. This proves that pure random search with Nelder–Mead is very inefficient.

All trained neural networks were trained on 1 million data samples. Figure 10 shows how different amounts of training data affect the GRU #6 architecture. Intuitively, the more data the network is trained on, the better its performance on synthetic data. At the same time, however, the quality of the prediction does not improve or even deteriorates for real data. This could be explained by the limitations of the used MAFTr model and also by the noise contained in the measured data. For this reason, 1 million data samples seem sufficient for all tested neural networks for this material model.

Each approach has a different time complexity. As shown in Table 9, training the GRU network takes about 4,000 s, while the other approaches do not require any training. On the other hand, the evaluation of

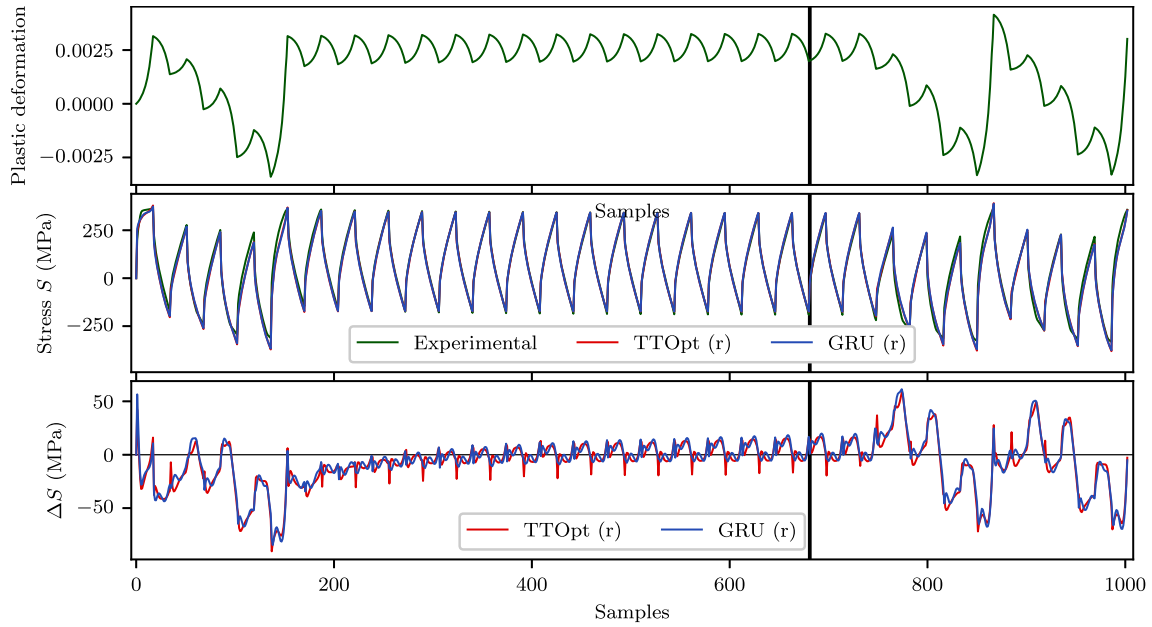

Fig. 6 Top: Plastic deformation in the measured experiment #1. Middle: Predicted stress using the refined estimated parameters $\boldsymbol{\theta}$ of both GRU #6 and TTOpt #6. Bottom: Stress prediction error. The vertical line indicates the end of the signal on which the $\boldsymbol{\theta}$ parameter was estimated and refined

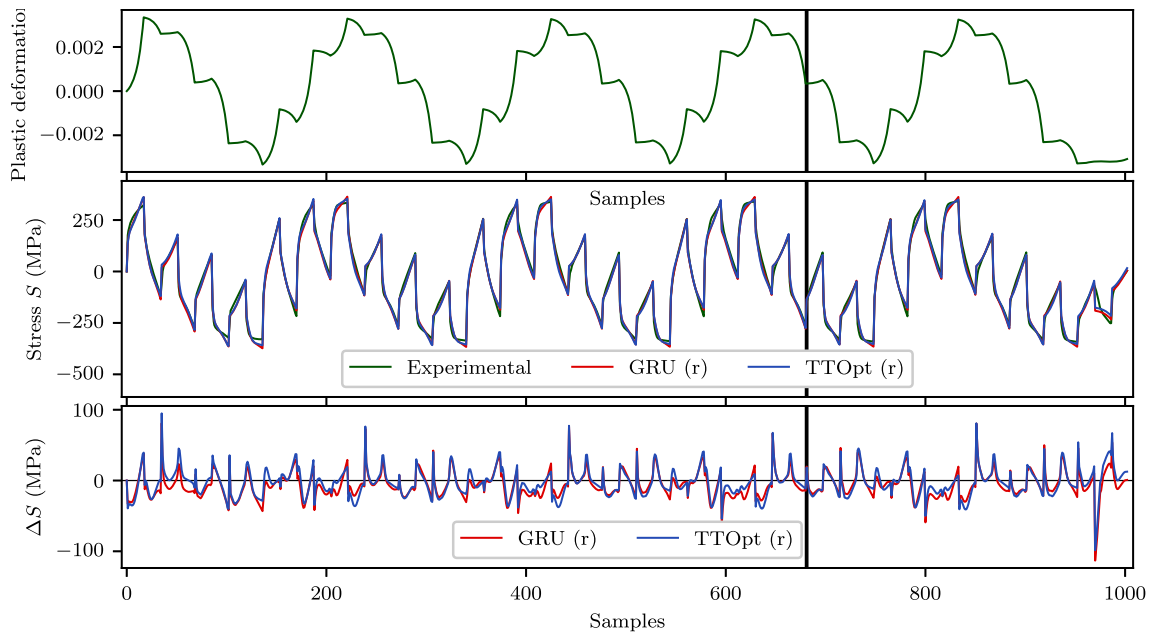

Fig. 7 Top: Plastic deformation in the measured experiment #2. Middle: Predicted stress using the refined estimated parameters $\boldsymbol{\theta}$ of both GRU #6 and TTOpt #6. Bottom: Stress prediction error. The vertical line indicates the end of the signal on which the $\boldsymbol{\theta}$ parameter was estimated and refined

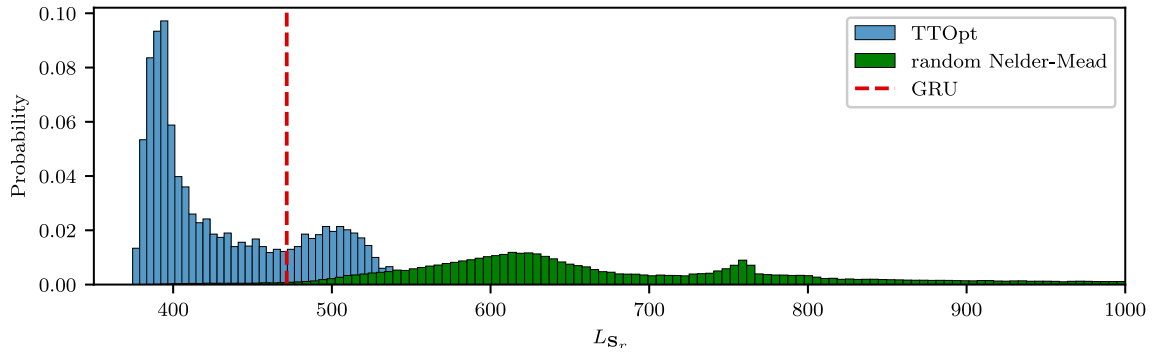

Fig. 8 Histograms of randomly generated $\boldsymbol{\theta}$ using the a priori distribution described in Section 3.1 and the TTOpt #6 predictions after their refinement using the Nelder–Mead simplex optimization on experiment #1. For comparison, the refined prediction of GRU #6 is depicted by a dashed line

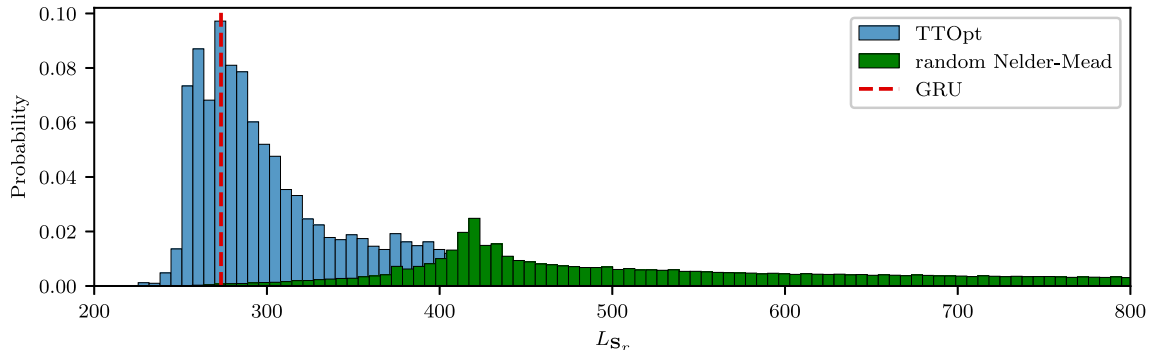

Fig. 9 Histograms of randomly generated $\boldsymbol{\theta}$ using the a priori distribution described in Section 3.1 and the TTOpt #6 predictions after their refinement using the Nelder–Mead simplex optimization on experiment #2. For comparison, the refined prediction of GRU #6 is depicted by a dashed line

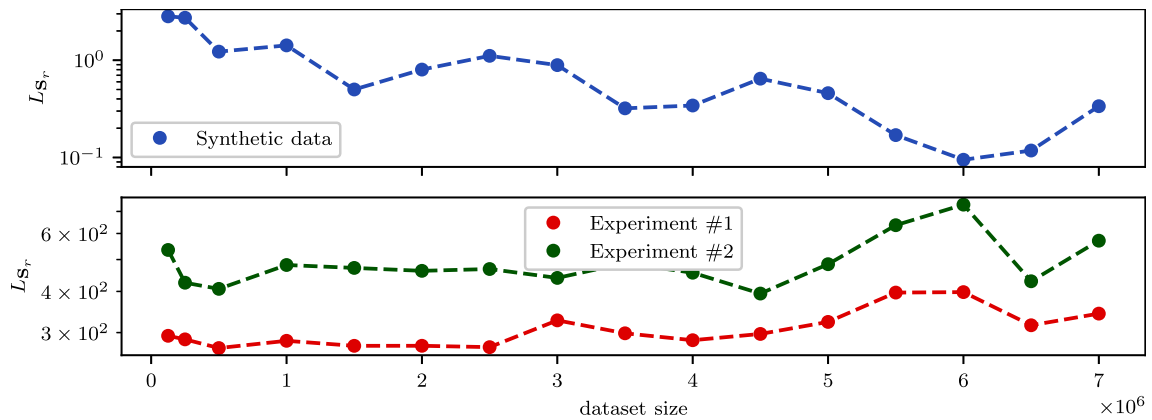

Fig. 10 Refined loss on synthetic data (top) and on 2 measured experiments (bottom) given by GRU #6 trained on various amount of data

the trained GRU takes only about 40 ms, which makes it very useful for scenarios where time is limited. For a more accurate prediction, the use of TTOpt seems to be a better choice, because although each evaluation takes about 10 s, the obtained estimates are of better quality than a simple random search. In all cases, the Nelder–Mead refinement takes about 0.2 s, which corresponds to about 2,000 steps. Our experiments show that after this time the improvement in prediction is negligible.

Table 9 Comparison of time complexity for different approaches

Approach	t_{train} (s)	t_{pred} (s)	t_{ref} (s)
GRU #06	4,000	0.04	0.2
TTOpt #06	–	10	0.2
Rsandom Nelder–Mead	–	–	0.2

6 Sensitivity analysis

In addition to the refined prediction itself, it is also useful to estimate its standard deviation. Since the analytic model is differentiable in parts (locally differentiable), one way to estimate the standard deviation is to use the Cramér–Rao lower bound (CRLB) computed at the point of current parameter estimation. The CRLB is a theoretical bound that provides a lower bound on the variance of unbiased estimators of a parameter and indicates the best accuracy achievable by such estimators. Since the neural networks may not provide unbiased estimates, this bound is only a rough approximation of the variance.

For simplicity, let us assume that the measurement noise is Gaussian distributed with variance σ^2,

$$\mathbf{S} \sim \mathcal{N}\Big(\mathbf{M}(\boldsymbol{\theta}), \sigma^2 \mathbf{I}_L\Big), \tag{24}$$

where $\mathbf{M}(\boldsymbol{\theta})$ stands for the closed-form solution of the MAFTr material model seen as a function of $\boldsymbol{\theta}$, and $\mathbf{I}_L$ represents the identity matrix. In this paper, $Q = 681$ is used in all experiments. Then, it can be shown that the Fisher information matrix is given by

$$\big[\mathbf{J}(\boldsymbol{\theta})\big]_{ij} = \frac{1}{\sigma^2}\Big[\frac{\partial \mathbf{M}(\boldsymbol{\theta})}{\partial \theta_i}\Big]^T\Big[\frac{\partial \mathbf{M}(\boldsymbol{\theta})}{\partial \theta_j}\Big]. \tag{25}$$

Here, the derivatives are calculated using the PyTorch autograd function. Assume that $\mathbf{J}(\boldsymbol{\theta})$ is a regular matrix. Then, every unbiased estimator $\widehat{\boldsymbol{\theta}}(\mathbf{S})$ satisfies a condition

$$\operatorname{cov}(\widehat{\boldsymbol{\theta}}) - \mathbf{J}^{-1}(\boldsymbol{\theta}) \geq \mathbf{0} \tag{26}$$

and therefore also

$$\operatorname{var}(\widehat{\theta}_i) \geq \text{CRLB}_i = [\mathbf{J}^{-1}(\boldsymbol{\theta})]_{ii}. \tag{27}$$

These variances depend on $\boldsymbol{\theta}$. For this reason, the refined $\widehat{\boldsymbol{\theta}}$ provided by GRU #6 is used as an approximation of the optimal $\boldsymbol{\theta}^*$.

To further analyze the error, the STD can also be estimated using the estimation method itself. The desired estimated STD (ESTD) of the $i-$th parameter can be calculated as

$$\text{ESTD}_i^2 = \frac{1}{N}\sum_{n=1}^{N}\big[\text{MET}\big(\mathbf{M}(\widehat{\boldsymbol{\theta}}) + \boldsymbol{\epsilon}_n\big)_i - \widehat{\boldsymbol{\theta}}_i\big]^2, \tag{28}$$

where $\boldsymbol{\epsilon}_n \sim \mathcal{N}(\mathbf{0}, \mathbf{I}_Q),\ \forall n \in \{1, ..., N\}$ is a randomly generated standard Gaussian noise and MET stands for the estimation method (e.g., GRU or TTOpt). In our experiment, $Q = 681$ and $N = 1,000$. The model prediction along with the ESTD, CRLB with $\sigma^2 = 1$, and the ratio of ESTD and the parameter value for GRU is shown in Table 10. For some parameters, the empirical STD of the error is observed to be greater than the CRLB. This may indicate that the GRU was trapped in a suboptimal local minimum. On the other hand, the ESTD of κ_2 was smaller than the corresponding CRLB, probably because the estimator is not unbiased.

7 Conclusion

Parameter estimation for a constitutive model of cyclic plasticity is a challenging task because it leads to the minimization of a non-convex function with many local minima. One possible solution is to train a neural network on a synthetic dataset consisting of stress–parameter pairs ($\mathbf{D}_1$ dataset) or stress–parameter–deformation tuples ($\mathbf{D}_2$ dataset). Our experiments show that training can be significantly improved using a loss function based on a combination of MSE losses from both parameter estimation and the corresponding stress response

Table 10 Refined parameters of experiment #1 predicted using GRU#6 trained on dataset $\mathbf{D}_2$ with their estimated STD (ESTD), Cramér–Rao lower bound (CRLB) and the ratio of ESTD and the parameter value

	$\widehat{\boldsymbol{\theta}}_i$	ESTD_i	$\sqrt{\mathrm{CRLB}_i}$	$\mathrm{ESTD}_i/\widehat{\boldsymbol{\theta}}_i$
k_0	195.3	2.534	0.7936	0.01297
κ_1	23,207	190.5	16.78	0.008210
κ_2	0.02872	0.000891	0.01682	0.03103
c_1	31,844	360.4	16.08	0.01132
c_2	2,539	142.1	3.221	0.05596
c_3	424.1	62.86	5.868	0.1482
c_4	410.3	108.9	7.932	0.2655
a_1	67.57	6.086	0.5326	0.09006
a_2	138.9	5.721	0.6352	0.04117
a_3	16.27	8.293	0.8546	0.5094
a_4	84.07	3.890	3.560	0.04627
$\overline{a}$	100.4	23.57	0.7223	0.2348

prediction. This technique makes training slower as the gradient must propagate through the analytical constitutive model, but it reduces the number of training steps required and results in better quality networks. The improvement in performance is significant, namely after the Nelder–Mead refinement is applied.

Recurrent networks based on GRU and LSTM proved to be a good choice for the simpler $\mathbf{D}_1$ dataset, as they both gave similar results for different hyperparameters. On the other hand, CNNs and FFNs seemed to be unstable with respect to the hyperparameter search. Only the GRU networks showed a good performance on the more advanced $\mathbf{D}_2$ dataset, which aims to generalize over different plastic deformation sequences.

The reference non-neural method based on Tensor Trains (TTOpt) also proved to provide high-quality results. Compared to GRU, TTOpt needs about 250 times more time for evaluation (50 times including Nelder–Mead optimization); however, since it is a non-deterministic method, it can be run multiple times and eventually achieve better results. This makes GRU a better choice for tasks where speed plays a major role, while TTOpt may be a better choice where the priority is to get the optimal prediction.

The advantage of TTOpt also is that it does not require any training and therefore does not depend on the structure of the training data. Both GRU and TTOpt performed better than random generation of estimates after their refinement using the Nelder–Mead method.

The entire approach used in this paper is not model specific and can therefore be adapted relatively easily to other constitutive models of rate-independent plasticity.

Funding Open access publishing supported by the institutions participating in the CzechELib Transformative Agreement. This work was supported by the Czech Science Foundation through the project No. 22-11101S. Martin Kovanda acknowledges the support of the Ministry of Education, Youth and Sports of the Czech Republic under the project Metamaterials for extremely thermally stressed machine components (METEX), project No. CZ.02.01.01/00/23_020/0008501, co-funded by the European Union. René Marek acknowledges support from the Czech Science Foundation under grant No. GA23-05338S and from the Institute of Thermomechanics of the Czech Academy of Sciences within grant project No. RVO:61388998.

Declarations

Conflict of interest The authors have no conflict of interest to declare that is relevant to the content of this article.

Ethics approval and consent to participate Not applicable.

Consent for publication All authors have reviewed the final version of the manuscript and approve its publication.

Materials availability Not applicable.

References

1. Suchocki, C., Kowalewski, Z.: A new method for identification of cyclic plasticity model parameters. Arch. Civ. Mech. Eng. **22**(2), 69 (2022). https://doi.org/10.1007/s43452-022-00388-7
2. Frederick, C.O., Armstrong, P.J.: A mathematical representation of the multiaxial bauschinger effect. Materials at High Temperatures 24, 1–26 (2007) https://doi.org/10.1179/096034007X207589 . Originally published as CEGB Report RD/B/N 731, Central Electricity Generating Board, 1966
3. Chaboche, J.L., Rousselier, G.: On the Plastic and Viscoplastic Constitutive Equations-Part I: Rules Developed With Internal Variable Concept. Journal of Pressure Vessel Technology **105**(2), 153–158 (1983). https://doi.org/10.1115/1.3264257
4. Dafalias, Y., Feigenbaum, H.: Biaxial ratchetting with novel variations of kinematic hardening. International Journal of Plasticity **27**, 479–491 (2011). https://doi.org/10.1016/j.ijplas.2010.06.002
5. Marek, R., Parma, S., Klepac, V., Feigenbaum, H.P.: A quick calibration tool for cyclic plasticity using analytical solution. Engineering Mechanics **27**(28), 249–252 (2022). https://doi.org/10.21495/512249
6. Nelder, J.A., Mead, R.: A simplex method for function minimization. The Computer Journal **7**(4), 308–313 (1965). https://doi.org/10.1093/comjnl/7.4.308
7. Sozykin, K., Chertkov, A., Schutski, R., PHAN, A.-H., Cichocki, A., Oseledets, I.: TTOpt: A maximum volume quantized tensor train-based optimization and its application to reinforcement learning. In: Oh, A.H., Agarwal, A., Belgrave, D., Cho, K. (eds.) Advances in Neural Information Processing Systems (2022). https://openreview.net/forum?id=Kf8sfv0RckB
8. Mareš, T., Janouchová, E., Kucerová, A.: Artificial neural networks in the calibration of nonlinear mechanical models. Advances in Engineering Software **95**, 68–81 (2016). https://doi.org/10.1016/j.advengsoft.2016.01.017
9. Raissi, M., Perdikaris, P., Karniadakis, G.E.: Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. Journal of Computational Physics **378**, 686–707 (2019). https://doi.org/10.1016/j.jcp.2018.10.045
10. Haghighat, E., Abouali, S., Vaziri, R.: Constitutive model characterization and discovery using physics-informed deep learning. Engineering Applications of Artificial Intelligence **120**, 105828 (2023). https://doi.org/10.1016/j.engappai.2023.105828
11. Schulte, R., Karca, C., Ostwald, R., Menzel, A.: Machine learning-assisted parameter identification for constitutive models based on concatenated loading path sequences. European Journal of Mechanics, A/Solids **98** (2023) https://doi.org/10.1016/j.euromechsol.2022.104854
12. Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., Rabinovich, A.: Going deeper with convolutions. In: 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 1–9. IEEE Computer Society, Los Alamitos, CA, USA (2015). https://doi.org/10.1109/CVPR.2015.7298594 . https://doi.ieeecomputersociety.org/10.1109/CVPR.2015.7298594
13. Ismail Fawaz, H., Lucas, B., Forestier, G., Pelletier, C., Schmidt, D.F., Weber, J., Webb, G.I., Idoumghar, L., Muller, P.-A., Petitjean, F.: Inceptiontime: Finding alexnet for time series classification. Data Min. Knowl. Discov. **34**(6), 1936–1962 (2020). https://doi.org/10.1007/s10618-020-00710-y
14. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural computation **9**, 1735–80 (1997). https://doi.org/10.1162/neco.1997.9.8.1735
15. Cho, K., Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., Bengio, Y.: Learning phrase representations using RNN encoder–decoder for statistical machine translation. In: Moschitti, A., Pang, B., Daelemans, W. (eds.) Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), pp. 1724–1734. Association for Computational Linguistics, Doha, Qatar (2014). https://doi.org/10.3115/v1/D14-1179 . https://aclanthology.org/D14-1179
16. Oseledets, I.V.: Tensor-train decomposition. SIAM Journal on Scientific Computing **33**(5), 2295–2317 (2011). https://doi.org/10.1137/090752286
17. Oseledets, I., Tyrtyshnikov, E.: TT-cross approximation for multidimensional arrays. Linear Algebra and its Applications **432**(1), 70–88 (2010). https://doi.org/10.1016/j.laa.2009.07.024
18. Loshchilov, I., Hutter, F.: Decoupled weight decay regularization. In: International Conference on Learning Representations (2019). https://doi.org/10.48550/arXiv.1711.05101
19. Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., Chintala, S.: Pytorch: An imperative style, high-performance deep learning library. In: Advances in Neural Information Processing Systems 32, pp. 8024–8035. Curran Associates, Inc., Red Hook, NY (2019)
20. Loc Vu-Quoc, A.H.: Deep learning applied to computational mechanics: A comprehensive review, state of the art, and the classics. Computer Modeling in Engineering & Sciences **137**(2), 1069–1343 (2023). https://doi.org/10.32604/cmes.2023.028130
21. Chertkov, A.: ttopt: Topology optimization in Python [GitHub repository]. GitHub. (2023). https://github.com/AndreiChertkov/ttopt