NorSand4AI: A Comprehensive Triaxial Test Simulation Database for NorSand Constitutive Model Materials

Luan Carlos de Sena Monteiro Ozelim[1], Michéle Dal Toé Casagrande[1], and André Luís Brasil Cavalcante[1]

[1]Department of Civil and Environmental Engineering, University of Brasilia,Campus Universitário Darcy Ribeiro, SG12, Asa Norte. 70910-900, Brasilia-DF, Brazil

Correspondence: Luan Carlos de Sena Monteiro Ozelim (luanoz@gmail.com, ozelim@unb.br)

Abstract. In soil sciences, parametric models known as constitutive models (e.g., the Modified Cam Clay and the NorSand) are used to represent the behavior of natural and artificial materials. In contexts where liquefaction may occur, the NorSand constitutive model has been extensively applied by both industry and academia due to its relatively simple critical state formulation and low number of input parameters. Despite its suitability as a good modelling framework to assess static liquefaction, the NorSand model still is based upon premises which may not perfectly represent the behavior of all soil types. In this context, the creation of data-driven and physically-informed metamodels emerges. Literature suggests that data-driven models should initially be developed using synthetic datasets to establish a general framework, which can later be applied to experimental datasets to enhance the model's robustness and aid in discovering potential mechanisms of soil behavior. Therefore, creating large and reliable synthetic datasets is a crucial step in constructing data-driven constitutive models. In this context, the NorSand model comes handy: by using NorSand simulations as the training dataset, data-driven constitutive metamodels can then be fine-tuned using real test results. The models created that way will combine the power of NorSand with the flexibility provided by data-driven approaches, enhancing the modelling capabilities for liquefaction. Therefore, for a material following the NorSand model, the present paper presents a first-of-its-kind database that addresses the size and complexity issues of creating synthetic datasets for nonlinear constitutive modeling of soils by simulating both drained and undrained triaxial tests. Two datasets are provided: the first one considers a nested Latin Hypercube Sampling of input parameters encompassing 2000 soil types, each subjected to 40 initial test configurations, resulting in a total of 160000 triaxial test results. The second one considers nested quasi-Monte Carlos sampling techniques (Sobol and Halton) of input parameters encompassing 2048 soil types, each subjected to 42 initial test configurations, resulting in a total of 172032 triaxial test results. By using the quasi-Monte Carlo dataset and 49 of its subsamples, it is shown that the dataset of 2000 soil types and 40 initial test configurations is sufficient to represent the general behavior of the NorSand model. In this process, four Machine Learning algorithms (Ridge Regressor, KNeighbors Regressor and two variants of the Ridge Regressor which incorporate nonlinear Nystroem kernels mappings of the input and output values) were trained to predict the constitutive and test parameters based solely on the triaxial test results. These algorithms achieved 13.91 % and 16.18 % mean absolute percentage errors among all the fourteen predicted parameters for undrained and drained triaxial tests inputs, respectively. As a secondary outcome, this work introduces a Python script that

links the established VBA implementation of NorSand to the Python environment. This enables researchers to leverage the comprehensive capabilities of Python packages in their analyses related to this constitutive model.

1 Introduction

In situations where liquefaction is a potential concern, geotechnical engineers and soil scientists seek suitable modeling frameworks to accurately evaluate and mitigate associated risks. One specific scenario highlighting this need is the case of filtered tailings piles. These piles pose significant geotechnical risks related to liquefaction, requiring thorough assessment through appropriate constitutive modeling. Factors such as the height and speed of stacking play crucial roles in creating vulnerable regions within the pile susceptible to liquefaction. The existence of a liquefaction trigger, particularly in undrained loading conditions, has the potential to result in the structural collapse of the pile.

In this scenario, the NorSand constitutive model emerges as a suitable alternative to liquefaction modelling due to its relatively simple critical state formulation and low number of input parameters. This model is a generalized critical state model based on the state parameter ψ, as defined by Jefferies (1993):

$$\psi = e - e_c \tag{1}$$

where e is the current void ration and e_c is the void ratio at the critical state. The NorSand model emulates natural soil behavior by incorporating associated plasticity and limited hardening, which enables dilation similar to that observed in real soils. This limited hardening causes yielding during unloading conditions and provides second-order detail in replicating observed soil behavior (Silva et al., 2022; Jefferies and Been, 2015).

Despite its suitability as a good modelling framework to assess static liquefaction (Sternik, 2015), the NorSand model still is based upon premises which may not perfectly represent the behavior of all soil types. Also, only recently the NorSand method has been implemented in commercial Finite Element softwares (Rocscience, 2022; Itasca Consulting Group, 2023; Bentley, 2022). Besides, regarding open-source distributions, only the Visual Basic (VBA) implementation presented by Jefferies and Been (2015) is available. It is precisely in this context that the creation of data-driven and physically-informed metamodels emerges. These metamodels, when based on artificial intelligence techniques, especially machine learning (ML) and deep learning (DL), may be able to provide accurate and computationally cheap models, allowing them to be a perfect link between complex computational models and real-time data collection and monitoring. Such methods need to be trained on large-scale datasets and this is where the NorSand model comes handy: by using NorSand simulations as the training dataset, data-driven constitutive metamodels can then be fine-tuned using real test results. These models will combine the power of NorSand with the flexibility provided by data-driven approaches, enhancing the modelling capabilities for liquefaction.

Thus, the current paper aims to address three main issues: the quantity and complexity of synthetic datasets for nonlinear constitutive modeling of soils and the availability of open-source implementations of the NorSand constitutive model. The first two aspects are addressed by simulating both drained and undrained triaxial tests. Two datasets are provided: the first one

will be used to study how large a given dataset must be in order to accurately capture the behavior of a NorSand material; while the second one, completely different from the first dataset, will be a perfect out-of-sample testing dataset used to perform the sample size validations mentioned. A byproduct of such sample size validation will be the training of different machine learning algorithms to perform the following learning task: obtain the input parameters of the NorSand model solely from the results of triaxial tests. Different sampling techniques will be used to produce the datasets mentioned, such as nested Latin Hypercube and quasi-Monte Carlo Sampling of input parameters. Then, the third aspect is considered by presenting an implementation which connects the well-known VBA implementation to the Python environment. We will use the VBA code as the "processing kernel" of our Python implementation, taking advantage of the years of tests and validation of the algorithm provided by Jefferies and Been (2015). This new Python code allows other researchers to use the full power of Python packages during their analyses involving NorSand.

The paper is structured as follows: Section 2 presents the general concepts of data-driven metamodels, with special emphasis to soil constitutive modelling. Then, Section 3 introduces the Norsand model. Section 4, on the other hand, presents the Methods considered in the present paper. Section 5 describes the Data Records associated with the present paper, while Section 6 presents the Technical Validation of the results. Section 7 presents some Usage Notes and Codes considered in the present paper. Finally, Section 8 presents the conclusions.

2 Data-driven metamodels

Montáns et al. (2019) emphasize that human learning involves observing and experiencing the world, collecting data, and identifying patterns through repeated experiments. Scientific discovery involves formalizing these patterns and relationships into laws and equations, transforming data into properties and variables, and converting observations into events. Although laws and equations aid learning, the classical learning process in science is often slow and expensive, requiring extensive observation and experimentation to understand the main variables and their impact on the phenomenon. Data-driven procedures, on the other hand, seek, if possible, an implicitly unbiased approach to our learning experience based on raw data from actual or synthetic observations. These procedures have the added advantage of testing correlations between different variables and observations, learning unanticipated patterns in nature, and allowing us to discover new scientific laws or even make predictions without the availability of such laws.

The recent rapid increase in the availability of measurement data from physical systems as well as from massive numerical simulations has stimulated the development of many data-driven methods for modeling and predicting dynamics. At the forefront of data-driven methods are Deep Neural Networks (DNNs). DNNs not only achieve superior performance for tasks such as image classification, but have also proven effective for future state prediction of dynamical systems (Haghighat et al., 2021). A key limitation of DNNs, and similar data-based methods, is the lack of interpretability of the resulting model: they are focused on prediction and do not provide governing equations or clearly interpretable models in terms of the original set of variables. An alternative data-based approach uses symbolic regression to directly identify the structure of a nonlinear dynam-

ical system from data (Schmidt and Lipson, 2009). This works remarkably well for discovering interpretable physical models, but symbolic regression is computationally expensive and can be difficult to scale to large problems (Montáns et al., 2019).

2.1 Data-driven constitutive modelling

In order to create metamodels from Neural Networks (NN), this type of approach generally requires a priori calibration of the algorithms from data considered to be representative of material behavior (He et al., 2021). For example, NNs have been applied to model a variety of materials, including concrete materials (Ghaboussi et al., 1991), hyperelastic materials (Shen et al., 2005), viscoplastic steel material (Furukawa and Yagawa, 1998), and homogenized properties of mixed structures (Lefik and Schrefler, 2003). Once calibrated, NN-based constitutive models have been integrated into finite element codes to predict path- or rate-dependent material behaviors (Lefik and Schrefler, 2003; Hashash et al., 2004; Jung and Ghaboussi, 2006; Stoffel et al., 2019).

Recently, DNNs with special mechanistic architectures, such as Recurrent Neural Networks (RNNs), have been applied to path-dependent materials (Wang and Sun, 2018; Mozaffar et al., 2019; Heider et al., 2020). It is clear that this type of approach has found significant applications in a wide range of engineering fields, as reinforced by He et al. (2021), when they argue that data-driven computation with physical constraints is an emerging computational paradigm that allows the simulation of complex materials directly based on the materials database and disregards the classical constitutive model construction.

To develop a data-driven constitutive model, a substantial and reliable dataset is necessary. However, obtaining a sufficiently large dataset for soil science can be challenging since experimental data is often limited and inadequate for training ML and DL algorithms. Generating synthetic data using a theoretical function can be a useful alternative, as it allows for the creation of an unlimited supply of data (Zhang et al., 2021a).

The literature suggests that data-driven models should initially be developed using synthetic datasets to establish a general framework, which can later be applied to experimental datasets to enhance the model's robustness and aid in discovering potential mechanisms of soil behavior (Zhang et al., 2021a). By calibrating constitutive models on synthetic datasets, the impact of experimental and measurement errors on the mapping ability of machine learning algorithms can be eliminated (Zhang et al., 2020). Therefore, creating large and reliable synthetic datasets is a crucial step in constructing data-driven constitutive models.

2.2 Data-driven soil constitutive models

Currently, there is a lack of robust and high-volume datasets in the literature for soil modeling tasks. One effective method to generate synthetic datasets is through numerical simulations performed on digital soil models. Typically, these simulations involve selecting a parametric constitutive model, sampling some parameters, and running simulations that mimic real-world test setups. In soil modeling, triaxial tests are commonly simulated using conventional physics-driven constitutive models, such as simple monotonic Konder's expression (Basheer, 2000) or more advanced models like the Modified Cam Clay (MCC) (Fu et al., 2007; Zhang et al., 2023).

In particular, a simple sand shear constitutive model was used to generate synthetic datasets in the work of Zhang et al. (2021b). A total of fourteen curves were generated to develop the ML-based constitutive model (nine curves for training and five curves for testing).

On the other hand, the MCC constitutive model was utilized to produce a benchmark stress–strain dataset of a virtual soil in the work of Zhang et al. (2023). In that study, a total of 250 soil types were considered, with 125 being part of the training dataset and the remaining 125 in the testing dataset. Considering all the initial states in the paper by Zhang et al. (2023), 1125 sets of stress–strain samples were employed as the training dataset, while 1250 sets of stress–strain samples constituted the testing dataset.

The MCC model has been a fundamental element in numerous complex models developed in recent times (Yao et al., 2008). However, this model and its variations are not well-suited for depicting the behavior of actual sands due to their insufficient representation of key features such as yielding and dilation. This is because these models assume that soils denser than the critical state line are over-consolidated, resulting in an unrealistically high stiffness and excessively exaggerated strength (Woudstra, 2021). As indicated in the Introduction section, the NorSand constitutive model presents clear advantages over MCC model and, therefore, shall be described in detail in the next Section.

3 NorSand

The NorSand constitutive model is a comprehensive critical state model that effectively accounts for the impact of void ratio on soil behavior, providing a robust framework for modeling static liquefaction in engineering applications. A distinctive characteristic of soils is that their void ratios or relative densities influence their mechanical properties. In this regard, NorSand, as a constitutive model, aptly elucidates changes in soil behavior resulting from variations in void ratio (Jefferies and Been, 2015).

Within the Critical State Soil Mechanics (CSSM) framework, NorSand aligns with widely used models like the Original Cam Clay (OCC, Schofield and Wroth (1968)) and the MCC (Roscoe and Burland, 1968). In fact, the NorSand and OCC yield surfaces have the same shapes and the same flow rules. CSSM is founded on two principles: 1) the presence of a unique failure locus known as the Critical State Locus (CSL), and 2) the assertion that shear strain guides soil toward the CSL.

The primary limitation of MCC, especially when applied to sands, lies in its inability to capture the dilation behavior observed in dense sands. Moreover, it proves inadequate in predicting the behavior of loose sands and is unsuitable for addressing liquefaction-related issues. NorSand's key advantage lies in its incorporation of a state parameter, representing the difference between the current void ratio of the soil and its critical state. This approach uniquely relates soil dilation or compaction to the state parameter (Rocscience, 2022).

NorSand stands out for its ease of use, particularly for practical geotechnical engineers. It relies on a minimal set of material properties, conveniently measurable through standard laboratory tests. The model effectively captures a wide range of soil behaviors influenced by varying density and confining stress. The key additional parameter, beyond what is necessary for

defining a MCC model, is the state parameter. In situations where precision in representing volume change is crucial, the added effort required for parameter determination is more than justified.

Developed initially for sands based on observations in large-scale hydraulic fills such as tailing dams, NorSand applicability extends beyond, encompassing any soil where particle-to-particle interactions are controlled by contact forces and slips, rather than cohesive bonds. Present applications of NorSand span a range from well-graded tills to sands and clayey silts (Jefferies and Been, 2015).

The input parameters of the NorSand model are presented in Table 1, where the meaning of each parameter is also presented in the column "Description". The sampling ranges presented will be discussed in the next section, as they are not intrinsic to the NorSand model.

Table 1. Input values for NorSand model also used as inputs for the NorSandTXL VBA routine (Jefferies and Been, 2015).

Soil properties				
Parameter Class	Parameter	Sampling range	Units	Description
CSL parameters	$\Gamma\|_{p'=1kPa}$	[0.9,1.4]	-	CSL mean effective stress at $p' = 1kPa$
	λ	[0.01,0.07]	$(\ln \text{kPa})^{-1}$	Slope of CSL defined on base e
Plasticity	M_{tc}	[1.2,1.5]	-	Critical friction ratio, with triaxial compression as a reference condition
	N	[0.2,0.5]	-	Volumetric coupling parameter
	χ_{tc}	[2,5]	-	Relates minimum dilatancy to corresponding ψ , with triaxial as a reference condition
	H_0	[75,500]	-	H is the loading plastic hardening modulus, such that:
	H_ψ	[200,500]	-	$H = H_0 + H_\psi \psi$
Elasticity	$G_{max}\|_{p'_0}$	[30,100]	*MPa*	Shear modulus at $p' = p'_0$
	G_{exp}	[0.1,0.6]	-	Exponent of nonlinear shear modulus change with stress, $G_{max} = G_{max}\|_{p'_0} (p'/p'_0)^{G_{exp}}$
	ν	[0.1,0.3]	-	Poisson's ratio
	Initial Soil State			
Parameter Class	Parameter	Sampling range	Units	Description
Stress and Deformability	ψ_0	[-0.2,$\psi_{max}/5$]	-	Initial critical state parameter, where $\psi_{max} = M_{tc}/(\chi(1+N))$
	p'_0	[50,1000]	kPa	Initial mean effective stress
	K_0	[0.8,1.2]	-	Geostatic stress ratio
	OCR ("R")	[0.5,3]	-	Overconsolidation ratio

4 Methods

4.1 Data Generation

The NorSandTXL program is an Excel spreadsheet with all coding in the VBA environment and can be downloaded at http://www.crcpress.com/product/isbn/9781482213683, as indicated in the book by Jefferies and Been (2015). This particular spreadsheet simulates drained and undrained triaxial tests of materials governed by the NorSand constitutive model. The input features available in NorSandTXL were presented in Table 1, as well as their sampling ranges. The sampling ranges adopted come from literature results on the behavior of real granular materials. An initial version of such ranges was first presented by Jefferies and Shuttle (2002) and has been updated ever since. The ranges presented in Table 1 reflect the latest compilation available and reported by Jefferies and Been (2015). This way, practitioners will especially benefit from the datasets generated, since the parameters involved have been chosen as to represent real granular materials.

In order to massively simulate triaxial test conditions for materials following the NorSand constitutive model, a Python routine has been developed. This routine performs two main steps: sampling and simulation. For the sampling process, all 14 input parameters are sampled in a nested manner, as there are two levels of hierarchy in the parameters: the higher level deals with the soil properties, which are unique for a given material, while the lower level considers the initial soil state during the triaxial tests. As a result, the sampling process needs to: a) account for different types of materials and b) for each type of material, consider several testing conditions. Two datasets will be produced, as the next subsection will describe.

Thus, the following sampling procedure is considered to account for n_{soils} types of soils under $n_{conditions}$ initial testing conditions:

- Sample the soil properties (the first ten parameters in Table 1), obtaining a vector of properties sp_i, $i = 1, ..., n_{soils}$, such that $sp_i \in \mathbb{R}^{10}$. The sampling is performed using the centered Latin hypercube sampling algorithm implemented in the *chaospy* package (Feinberg and Langtangen, 2015) with a maximin criterion (first dataset) or using a Sobol (Sobol, 1967) quasi-Monte Carlo sampling technique implemented in *SciPy* (Virtanen et al., 2020) (second dataset).

- For each sp_i, the initial testing conditions (the last four parameters in Table 1) are sampled using the standard Latin hypercube sampling algorithm implemented in the *chaospy* package (Feinberg and Langtangen, 2015) with a ratio criterion (first dataset) or a Halton (Halton, 1960) quasi-Monte Carlo sampling scheme (second dataset) implemented in *SciPy* (Virtanen et al., 2020). This way, the vectors $ic_{i,j} \in \mathbb{R}^4$, $j = 1, ..., n_{conditions}$ are obtained for each sp_i. The maximum value of ψ_0 is set to $\psi_{max}/5$ (as indicated in Table 1) for numerical stability. Additionally, to make the $ic_{i,j}$ different for each sp_i, the random seed of the sampling algorithm is changed for each i.

From the procedure above, the matrix In of input parameters is obtained, whose rows are NorSandTXL input vectors obtained by concatenating each sp_i with all the $ic_{i,j}$, i.e., $[concat(sp_1, ic_{1,1}), concat(sp_1, ic_{1,2}), ..., concat(sp_{n_{soils}}, ic_{n_{soils}, n_{conditions}})]$, where $concat$ denotes a concatenation operation between vectors. This implies that In is a $(n_{soils} n_{conditions})$ by 14 matrix. The filling capabilities of the sampling schemes considered can be seen in Figure 1.

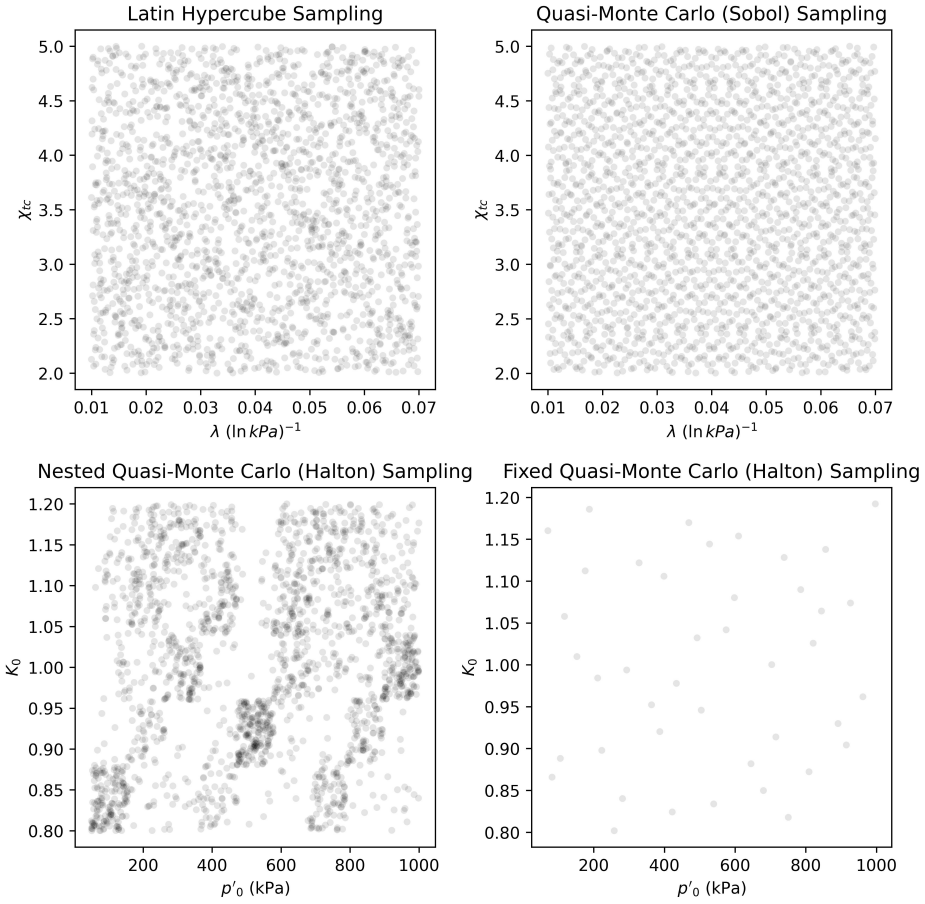

Figure 1. Scatter plot illustrating how each space filling technique works for particular pairs of constitutive and test-related parameters

Figure 1 reveals that the Latin Hypercube sampling presents an apparent randomness on how the points are spread in the space. Quasi-Monte Carlo techniques, on the other hand, have a high predictability (as they are deterministic), but also fill in the input space adequately. The difference between the lower plots on Figure 1 is that the lower-left one presents the sampled pairs of values for a total of 2048 materials, while the lower-right one presents the sampled pairs for a single material. The nested quasi-Monte Carlo sampling suffers from its deterministic nature, but shuffling the values help to provide a better spread, as shall be discussed.

One may notice that besides ψ_0, which is restricted by a fraction of $\psi_{\max}$, an independent sampling of input parameters was conducted. This was considered to explore the behavior of the NorSand model across all conceivable regions of the input parameter space. The objective was to enhance understanding of the analytical characteristics of the transfer function, which accepts these parameters as inputs and produces triaxial test results as outputs. This strategy ensures that the learning process remains unbiased, thereby preventing the algorithm from solely learning the transfer function within a specific area of interest.

Broadening the scope of learning task beyond such confines can positively influence the overall learning process. For specific applications where the correlation among input parameters holds greater significance, adjusting loss weights for points within and outside the region of interest could be beneficial. This adjustment represents a choice that can be made. In future works, especially in the development of constitutive models tailored for specific purposes, it is advisable to consider this correlation structure.

The simulation step, on the other hand, involves opening the Excel spreadsheet provided in the book by Jefferies and Been (2015), inputting the sampled parameters, running both drained and undrained simulations for the input parameters and collecting their respective results. By design, the NorSandTXL Excel spreadsheet considers 4000 strain steps to go from zero to approximately 20% nominal axial strain at the end of the simulated test. The authors of the spreadsheet indicate that this amount is both convenient and sufficient (Jefferies and Been, 2015). On the other hand, for a triaxial effective stress state with vertical stress σ'_a (kPa) and confining stress σ'_r (kPa), a total of 10 entities are reported from the tests, which are: ϵ_1 (axial strain); ϵ_v (volumetric strain); $p' = (\sigma'_a + 2\sigma'_r)/3$ (mean effective stress in kPa); $q = \sigma'_a - \sigma'_r$ (deviatoric stress in kPa); e (void ratio); p_i/p' (stress ratio); $(p_i/p')_{max}$ (maximum stress ratio); ψ (state parameter); D_p (dilation) and $\eta = q/p'$. Thus, the dataset is a 4000×10 array, as presented in Table 2.

Table 2. Example of the dataset collected from the 'NorSandTXL' spreadsheet.

ϵ_1	ϵ_v	p'	q	e	p_i/p'	$(p_i/p')_{max}$	ψ	D_p	η
0	0	200	0	0.9021	0.42306	1	0	0.92603	0
0.06097	0.04314	209.561	28.2795	0.90128	0.40376	1	0	0.92603	0.13495
0.07544	0.05481	210.703	31.7059	0.90106	0.40811	0.99319	0.001981083	1.31505	0.15048
0.0897	0.06628	211.821	35.0611	0.90084	0.41236	0.99284	0.002085266	1.29952	0.16552
					...				
19.3293	2.10004	387.564	562.29	0.86216	1.00101	1.00087	-0.000251146	-0.00146	1.45083
19.3334	2.10003	387.564	562.29	0.86216	1.00101	1.00087	-0.000251018	-0.00146	1.45083
19.3374	2.10002	387.564	562.29	0.86216	1.00101	1.00087	-0.000250889	-0.00146	1.45083

After the simulation is run, the results are saved in *.h5* format files for posterior processing. The file extension *.h5* is associated with the Hierarchical Data Format (HDF5) (The HDF Group, 1997-2023), which is a type of high-performance distributed file system. It is specifically designed to manage large and complex data sets efficiently and flexibly. Additionally, it enables a self-describing file format that is portable and supports parallel I/O for data compression (Lee et al., 2022), and has shown superior performance with high-dimensional and highly structured data (Nti-Addae et al., 2019). Literature indicates that the HDF5 has been popular in scientific communities since the late 1990s (Lee et al., 2022), which is evident by the large number of open-source and commercial software packages for data visualization and analysis that can read and write HDF5 (Group, Accessed on April 24, 2023). As a result, this is the data format chosen for the present paper.

4.2 Sample size validation

The samples generated using the methods in the last subsection need to be sufficiently large in order to represent the general behavior of the NorSand model. The best way to show that the sample size is sufficient is to study how a model calibrated (or trained) on a given dataset performs. So, we chose the most direct (and actually most important) learning task one could face while working with the datasets generated: back-calculation of the constitutive parameters of the model based solely on the triaxial test results. In short, from the triaxial tests we will learn the values of the parameters which govern the behavior of the material.

This way, it is possible to recall that a total of 14 parameters (10 constitutive and 4 related to test conditions) are used to generate the triaxial test results (4000×10 array where 4000 denotes the number of time steps of the loading process and 10 is the number of quantities monitored during the test), as presented in Table 2. From last subsection's notation, Let In_i (shape 1×14) be the i-th row of the In matrix, which contains the constitutive parameters, and let ttu_i and ttd_i be the results of the triaxial test under undrained and drained conditions, respectively (4000×10 arrays, each) obtained by using these parameters on the NorSandTXL routine.

We will consider the following learning problem: From a sample of input parameters $In = In_{n,m}$, which considers n different types of soil and m different test configuration (therefore with nm rows), we will use the ttu_i (or ttd_i), for $i = 1, ..., nm$, to learn the vectors of parameters In_i, for $i = 1, ..., nm$. We wish to investigate what are the values of n and m that suffice to produce an accurate representation of the model. In order to do so, following standard learning tasks in a Machine Learning context, we need training, validation and testing data. It is worth noticing that our methodology needs to be robust, so we indeed need the validation dataset because hyperparameter tuning will be performed.

The first dataset obtained by following the methods of subsection 4.1 was generated by a Latin Hypercube Sampling (LHS) algorithm, which is known to provide low-discrepancy sequences of values (i.e., the samples are spread in the domain of the sampled variables). Despite being a really powerful technique, LHS does not have an interesting property: sequences obtained by LHS are not extensible. To put it simply, being extensible means that a sample of size j contains the values of the sample of size k, $j > k$. This way, it would not be possible to sub-sample from our original sample In in order to build smaller datasets without loosing the space-filling capability of the dataset. This way, we needed to consider another sampling scheme to perform our investigation.

We chose to combine two quasi-Monte Carlo low discrepancy sequence generation techniques (Sobol (Sobol, 1967) and Halton (Halton, 1960)), which are also extensible, to perform our tests. In that case, we generated a dataset with $n = 2048$ and $m = 42$ using Sobol sampling for the constitutive parameters (10 parameters) and Halton sampling for the experimental test condition variables (4 variables) using the *SciPy* Python package (Virtanen et al., 2020). Both sequences have been scrambled (Owen and Rudolf, 2021) to improve their robustness for space filling. By using these parameters, we ran the NorSandTXL routine in the same manner as described in subsection 4.1 and obtained the corresponding triaxial test results for both drained and undrained cases. Let us call this new dataset and $qIn_{2048,42}$.

By using the extensibility property of the sequences considered, 49 sub-samples were taken: $qIn_{n,m}$ for n in [32, 64, 128, 256, 512, 1024, 2048] and m in [6, 12, 18, 24, 30, 36, 42]. One may see that powers of 2 were used as sample sizes for the Sobol sampling scheme, which is standard and derives from its implementation in *scipy.stats*. It is worth noticing that, in general, none of the entries of $In_{n,m}$ will be in $qIn_{n,m}$, which indicates that using $qIn_{n,m}$ for training and validation and $In_{n,m}$ for testing does not allow for any data "leakage". Besides, there is a clear benefit in using $In_{n,m}$ as a test set: all the models will be tested on the same dataset.

For the learning task considered, we used the *scikit-learn* Python package (Pedregosa et al., 2011) and chose 4 algorithms: Ridge Regressor, KNeighbors Regressor and two variants of the Ridge Regressor which incorporate nonlinear mappings of the input and output values. The first two algorithms mentioned belong to two different classes: linear and neighbors-based regressors. They were chosen to illustrate how different types of algorithms learn our chosen task. The variants of the Ridge Regressor were chosen to account for nonlinearities by using the kernel trick. Considering the high dimensionality of the input datasets, using traditional kernels is not computationally feasible, so we used Nystroem kernels (Yang et al., 2012), which approximate a kernel map using a subset of the training data. By combining Nystroem kernels and Ridge Regressors, we can map the inputs to a nonlinear feature space and then consider a linear regression on these features. This is a similar approach as the one considered to build Support Vector Machine Regressors, but with a slightly different regularization for the decision boundary.

We also considered mapping the output values (14 parameters, in our case) to the [0,1] range by combining the *scikit-learn* implementations of TransformedTargetRegressor and QuantileTransformer, which transforms the target values (outputs of the pipeline) to follow a uniform distribution. Therefore, for a given component, this transformation tends to spread out the most frequent values. It also reduces the impact of (marginal) outliers (Pedregosa et al., 2011). For all the algorithms considered, we also used a QuantileTransformer to preprocess the input values.

This way, Figure 2 presents the methodology proposed and applied to assess the quality of the sample size. In the present paper, the LHS-generated dataset with $n_{soils} = 2000$ and $n_{conditions} = 40$, whose input parameter matrix is $In_{2000,40}$, will have its sufficiency assessed.

It is possible to describe the workflow in Figure 2 as:

For n in [32,64,128,256,512,1024,2048]:

For m in [6,12,18,24,30,36,42]:

- For each simulated triaxial test corresponding to the parameters matrix $qIn_{n,m}$, select only the columns corresponding to ϵ_1, p', q and e (axial strain, mean effective stress, deviatoric stress and void ratio, respectively), which are the variables commonly measured and reported. The other 7 columns are manipulations of these three (D_p or η, for example) and could be used as alternative regression variables, but such selection is not the focus of the present paper. This reduced simulation dataset is of shape 4000×4.

- Each triaxial test simulation may have different start/end values for ϵ_1, so it is important to "align" all the test considered. By alignment we mean that all the tests will have measurements for the same values of ϵ_1. This will enable us to use

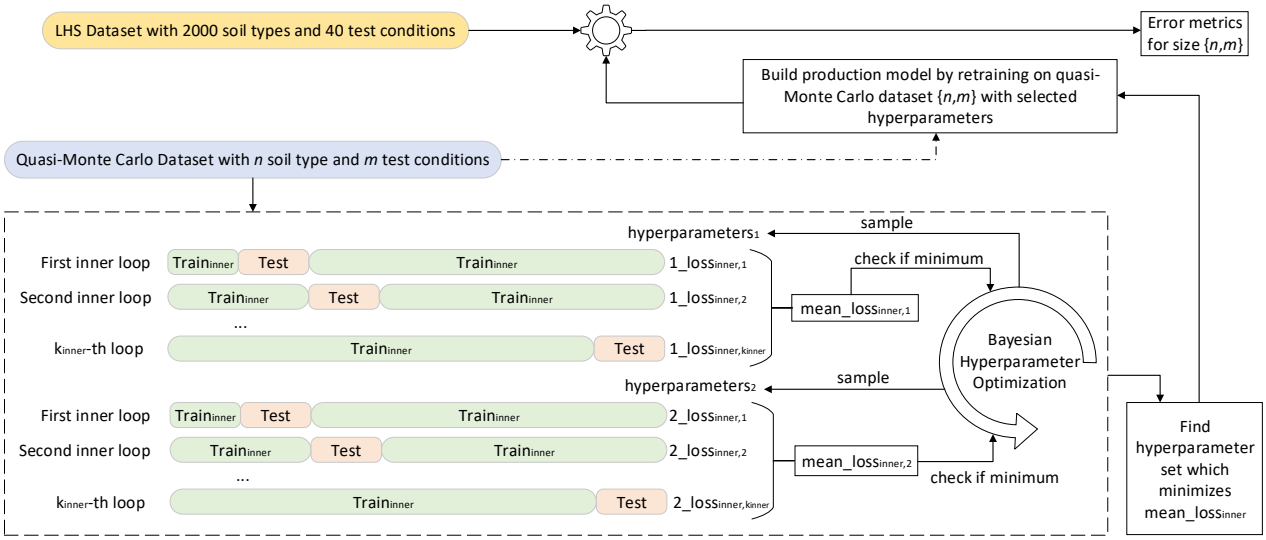

Figure 2. Methodology used to assess the sufficiency of the dataset containing 2000 soil types and 40 test conditions to represent the general behavior of the NorSand model

this variable as an index and, therefore, decrease the dimensionality of each triaxial test simulation from 4000×4 to 4000×3 (each line will correspond to a single value of ϵ_1). We must select the smallest maximum value of ϵ_1 across all simulations (which was found to be around 15.74% for the datasets considered and is represented as the vertical line in Figures 3 and 4).

- Downsample the 4000 timesteps to 40, by using evenly spaced values on a logarithmic scale (function *logspace* from Python package *numpy*: more values in the beginning of the time steps, where more changes are observed. This process is illustrated on Figures 3 and 4, where the downsampling is performed for 40 points logarithmically spaced between $\epsilon_1 = 10^{-3}\%$ and 15.78 %). This reduces each simulated triaxial test corresponding to the parameters matrix $qIn_{n,m}$ from 4000×10 to 40×3. The concatenation of all triaxial test results corresponding to the parameters matrix $qIn_{n,m}$ shall be named $qInN_{n,m}$ and is of size $(nm, 40, 3)$.

- Perform a GroupKFold cross-validation scheme to find the best hyperparameters of an algorithm A using $qInN_{n,m}$ and inputs and $qIn_{n,m}$ as outputs. The loss function considered during the GroupKFold cross-validation is the mean absolute percentage error across all folds;

- Retrain the algorithm A using all $qInN_{n,m}$ and $qIn_{n,m}$ after fixing the hyperparameters as the optimal ones obtained during the cross-validation scheme;

- Test the trained algorithm A_t on In_{n_h,m_h}, where n_h and m_h are the hypothesized sufficient number of materials and test conditions, respectively;

– Obtain the mean absolute percentage error in the predictions of all the 14 input parameters corresponding to In_{n_h,m_h};

– Get the overall mean error, corresponding to all the input parameters.

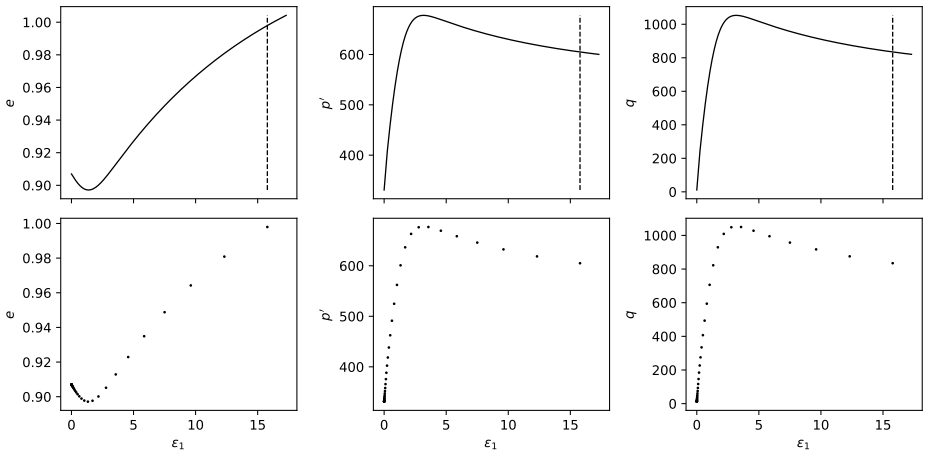

Figure 3. Downsampling process from 4000 to 40 points in the logarithmic scale for drained tests

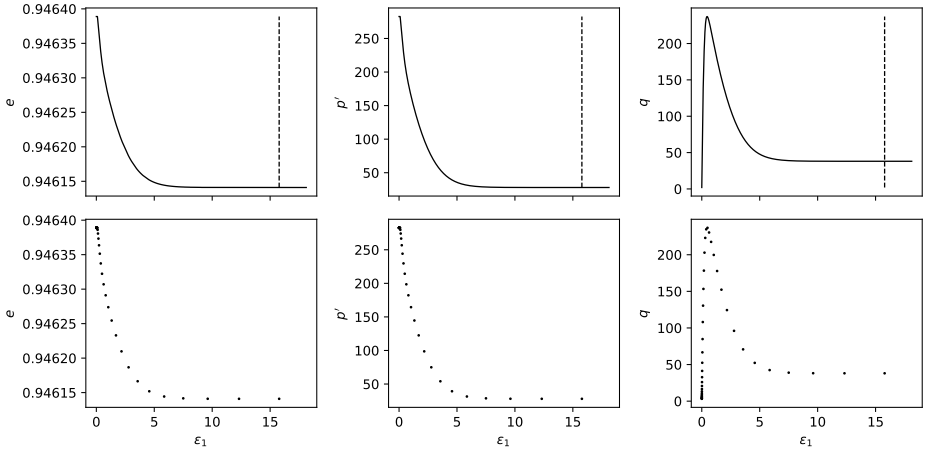

Figure 4. Downsampling process from 4000 to 40 points in the logarithmic scale for undrained tests

As described, for training and validation, we considered a GroupKFold cross validation technique, which is a K-fold iterator variant with non-overlapping groups (Pedregosa et al., 2011). This approach makes sure no material (group) is present both in train and validation set, which would lead to data "leakage".

A Bayesian optimization was performed to look for the best hyperparameters using the cross-validation folds generated. This process was carried out using the *Hyperopt* Python package (Bergstra et al., 2015), which considers Tree-structured

Parzen Estimators. The search space for the Ridge and KNeighbors Regressors are the ones considered in the *Hyperopt-Sklearn* Python package (Komer et al., 2014). For the Nystroem kernel, a custom search space was defined and consisted of: 'gamma' parameter uniformly on [0,1]; 'n_components' parameter as a random equi-probable choice among [600,1200,1800]; 'kernel' parameter as a random equi-probable choice among ["additive_chi2", "chi2", "cosine", "linear", "poly", "polynomial", "rbf","laplacian", "sigmoid"]; 'degree' parameter as the integer value truncation of an uniform random variable on [1, 10] and 'coef0' parameter uniformly on [0,1].

Finally, after the best hyperparameters are found, they are fixed and the algorithm A is retrained with the full dataset $qInN_{n,m}$. This calibrated version is then used to test the quality of the model on the triaxial test results corresponding to the dataset In_{n_h,m_h}. Then, the errors obtained for each model are plotted and analyzed. The reader may find the complete codes used to implement the steps above in (Ozelim et al., 2023b).

5 Data Records

In the present paper, it is shown that the LHS-generated dataset with $n_{soils} = 2000$ and $n_{conditions} = 40$ is a sufficient dataset. Thus, the folder containing such dataset can be found in Ozelim et al. (2023a) and has the following structure:

NorSandTXL_H5 \Simus**TT**\Par_**X**_**Y**.h5

where **TT** stands for the test type (Drained or Undrained), **X** is the material index (from 0 to 1999) and **Y** is the sequential index for the input parameters (from 0 to 79999).

Each *Par_**X**_**Y**.h5* file contains a dataset titled 'NorSandTXL' which includes the simulation results as presented in Table 2. It is worth noticing that the values stored are of the type *float32*, which is sufficient for the applications envisioned for the dataset. In addition to the simulation results, the dataset also contains the attributes shown in Table 3. The correspondence between the attributes, whose data type is either *float32* or *<U7* (fixed-length character string of 7 Unicode characters), and NorSandTXL input parameters is also presented in Table 3. It is easy to see that the dataset attributes in each file allow for a complete reproduction of the results, if desired. The units of the parameters are consistent with NorSandTXL, as presented in Table 1.

In order to prove the sufficiency of $In_{2000,40}$, we generated the dataset $qIn_{2048,42}$ following the methods previously presented. This latter dataset is also available at Ozelim et al. (2023a) with a similar folder structure. In that case, the upper-level folder is named $NorSand\_2048\_42$. It is worth noticing that, due to upload difficulties, $NorSand\_2048\_42$ was split as $NorSand\_2048\_42\_Drained$ and $NorSand\_2048\_42\_Undrained$, where each file contains the simulations for drained and undrained scenarios, respectively.

6 Technical Validation

Considering that the engine running the triaxial test simulations is the Excel spreadsheet presented in the book by Jefferies and Been (2015) and that such spreadsheet has been extensively validated by both academia and industry, there is no need

Table 3. Attributes of the 'NorSandTXL' dataset present in each *Par_X_Y.h5* file.

Attribute	Parameter/Value
'Gamma'	$\Gamma\|_{p'=1kPa}$
'lambda'	λ
'Mtc'	M_{tc}
'N'	N
'Xtc'	χ_{tc}
'H0'	H_0
'Hy'	H_ψ
'Gmax_p0'	$G_{max}\|_{p'_0}$
'G_exp'	G_{exp}
'n'	ν
'Psi_0'	ψ_0
'p0'	p'_0
'K0'	K_0
'OCR'	OCR ("R")
'Type'	Drained or Undrained

to discuss the technical quality of the dataset. On the other hand, it is necessary to show that $In_{2000,40}$ suffices to cover the general behavior of the NorSand models.

By following the methods previously described and plotting the mean absolute percentage error (MAPE) result of the 49 models (each trained and validated with samples of different sizes subsampled from $qIn_{2048,42}$) Figure 5 and 6 were obtained for drained and undrained conditions, respectively. The 4 algorithms considered were Ridge, KNeighbors, Ridge-K (with nonlinear kernel on inputs) and Ridge-KT (with nonlinear kernel on inputs and also QuantileTransformer on the outputs). It is clear in the figures that, for contours of 0.5% gains in MAPE, the sample size of 2000×40 is actually more than enough for the learning task considered. This can be stated by noticing that the contours with lower error encompass samples with an exponential range of sizes (the x-axis is in log scale). This indicates a really small gradient on the error in the $n \times m$ space, implying a good sample size. This happens for all 4 algorithms, indicating that not only linear and neighbors-based regressors have reached their maximum ability to learn, but also the nonlinear variants considered. It can be seen that the two nonlinear transformations applied (to inputs and to both inputs and outputs) present a similar behavior, although with considerably smaller MAPEs.

The analysis of Figures 5 and 6 indicate that for the learning task hereby considered, undrained tests generally presented a better performance while compared to drained tests. A possible cause for such behavior is that during undrained tests the void ratio is kept constant. Thus, for the learning task considered, the algorithm does not need to perform any nonlinear operations on one third of the input dataset (which consists of e, p and q for 40 values of ϵ_1). So, with the same number of training samples

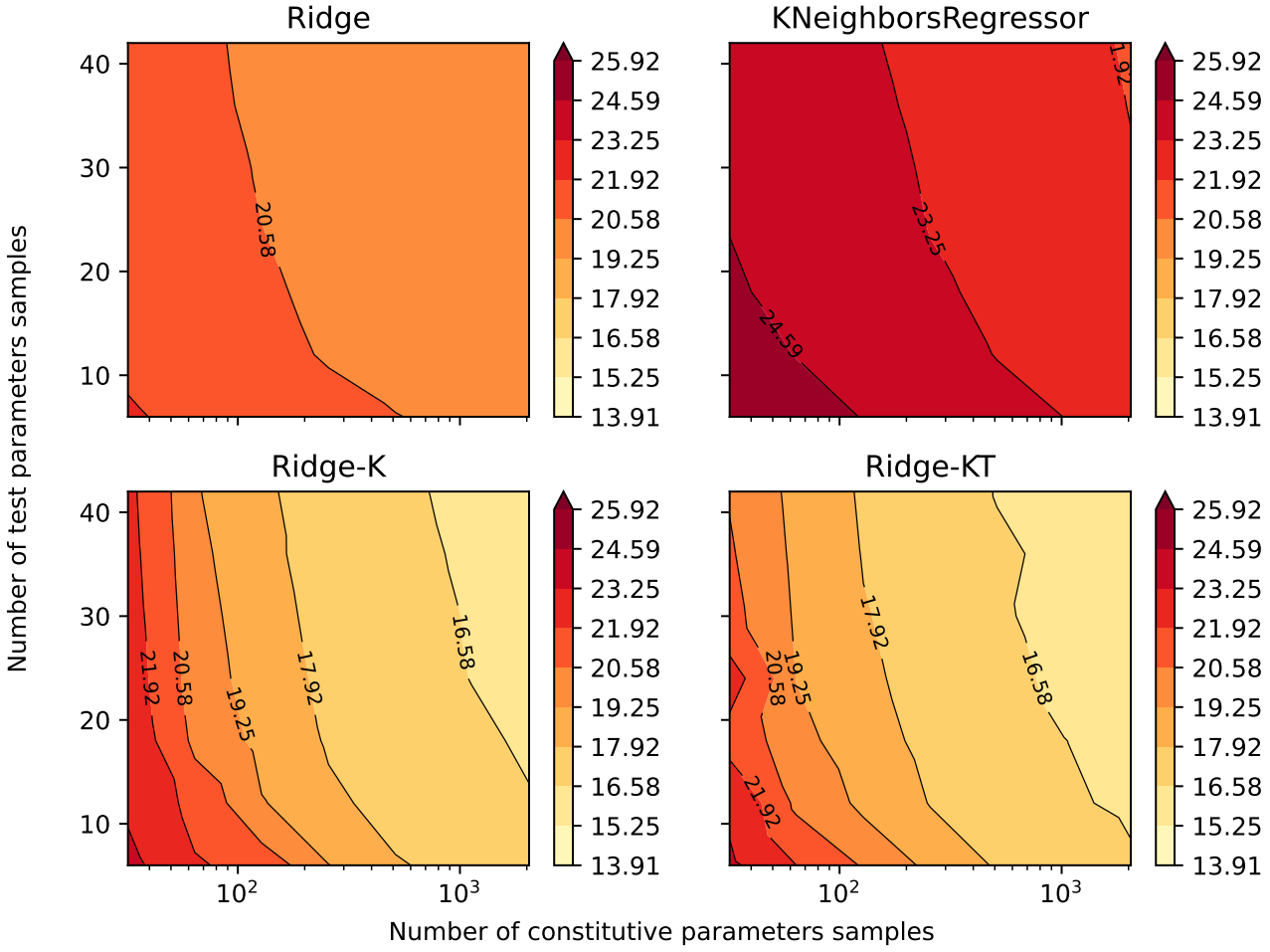

Figure 5. Mean absolute percentage error for all the 14 parameters after being back-calculated solely from drained triaxial test results.

and analytical structure of the learning algorithm, it is expected that less nonlinearities in the inputs would result in a better performance (smaller errors) of the predicted outputs.

Due to the space-filling qualities of both $In_{2000,40}$ and $qIn_{2048,42}$, $qIn_{2048,42}$ can also be considered a sufficient dataset to represent the NorSand model.

6.1 Understanding the learning task

6.1.1 Drained versus undrained tests performance

Figure 7 presents the MAPE for each of the predicted parameters by the best performing algorithm (Ridge-KT trained and validated on the 2048×42 dataset and tested on the 2000×40 one.

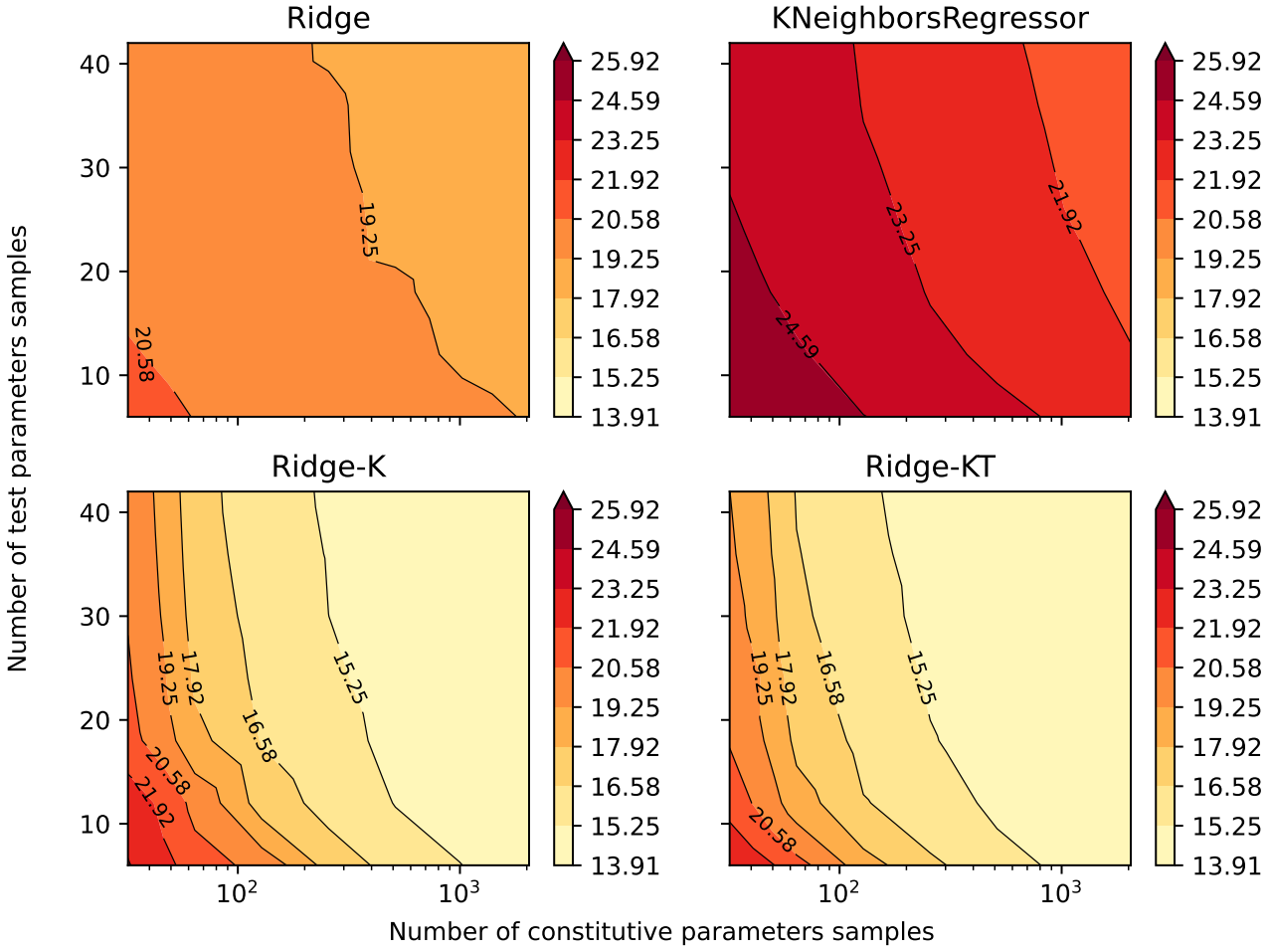

Figure 6. Mean absolute percentage error for all the 14 parameters after being back-calculated solely from undrained triaxial test results.

At first glance, Figure 7 suggests that using single tests to back-calculate parameters is not the best alternative, as the combination of both drained and undrained tests can potentially lead to better results. This will be the topic of future studies, especially on how many drained and undrained tests lead to optimal results. Jefferies and Been Jefferies and Been (2015) have discussed this situation, suggesting the minimal combination would be of two undrained and one drained tests.

Also from Figure 7, it can be seen that, in general, the models trained on either drained or undrained datasets achieved a similar prediction performance for parameters K_0, p'_0, e_0, ν, H_ψ, χ_{tc}, N, M_{tc} and Γ. For the parameters linked to the test setup, namely K_0, p'_0 and e_0, this is somewhat expected as there are no nonlinearities involved in finding such values from triaxial test results (it is matter of simply checking the initial values of stresses and void ratios).

For ν, what can be observed from Figure 8 is that the ML algorithm did not fully succeeded in its learning task, as there is a great spreading of the points along the identity line. In special, most of the points are located in the central vertical region,

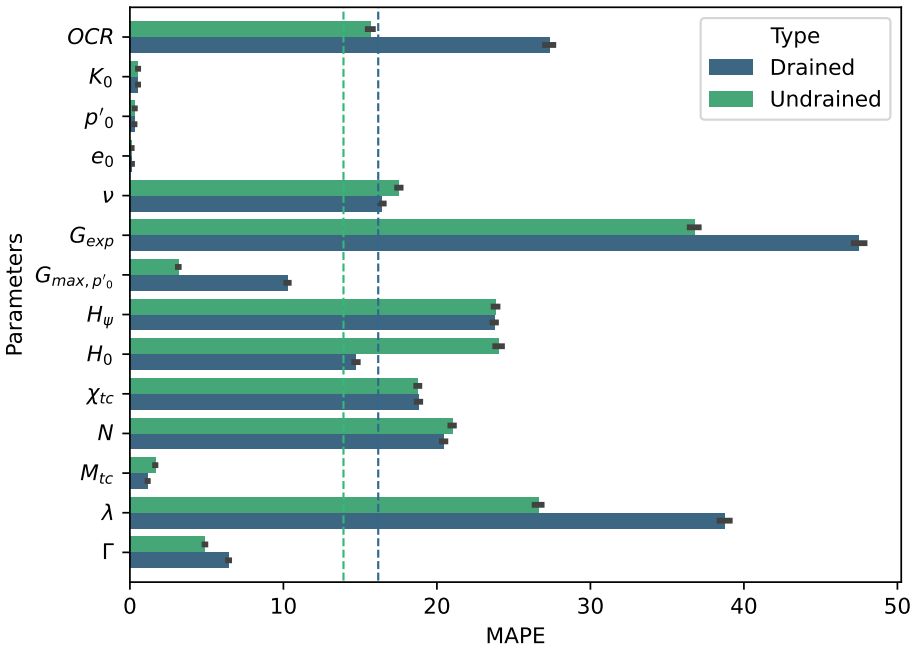

Figure 7. Drained and undrained mean absolute percentage errors for each parameter obtained by the best performing algorithm (Ridge-KT) with the 2048×42 training dataset. Vertical lines represent the mean MAPE for all parameters according to the colors in the plot (drained or undrained models)

indicating that most of the time the predicted values were the close to the mid-point of the interval (0.2), which is a naive-approximator known as Dummy Regressor (outputs the mean of the training dataset). This result may also be caused by the apparent low impact that ν has on the final result of the triaxial test.

The same Dummy Regressor behavior was observed for H_ψ, χ_{tc}, N and M_{tc}, as illustrated in Figure 9. In such figure, it can be seen that the spreading of the points is still considerable around the identity line. Also, the most extreme mean-outputting behavior was observed for H_ψ, as Figure 10 illustrates.

For Γ, Figure 11 reveals that the mean-outputting behavior is not prominent anymore, revealing a good learning capability of the ML algorithm. Even tough the MAPE is about the same for algorithms trained on either drained or undrained tests, for the undrained cases there is a more symmetrical distribution of points around the identity line, which indicates less bias in the predictions. In this context, less bias and equivalent MAPE would suggest the ML algorithm trained on undrained tests is a better choice for estimating Γ.

On the other hand, OCR, G_{exp}, G_{max,p'_0} and λ had smaller MAPEs when predicted by algorithms trained on undrained tests. For the first three parameters, this is consistent with calibration procedures indicated in literature (validation of elastic properties using undrained tests as suggested by Jefferies and Been (2015)). The performance of the Ridge-KT algorithm for these parameters can be seen in Figures 12 to 14.

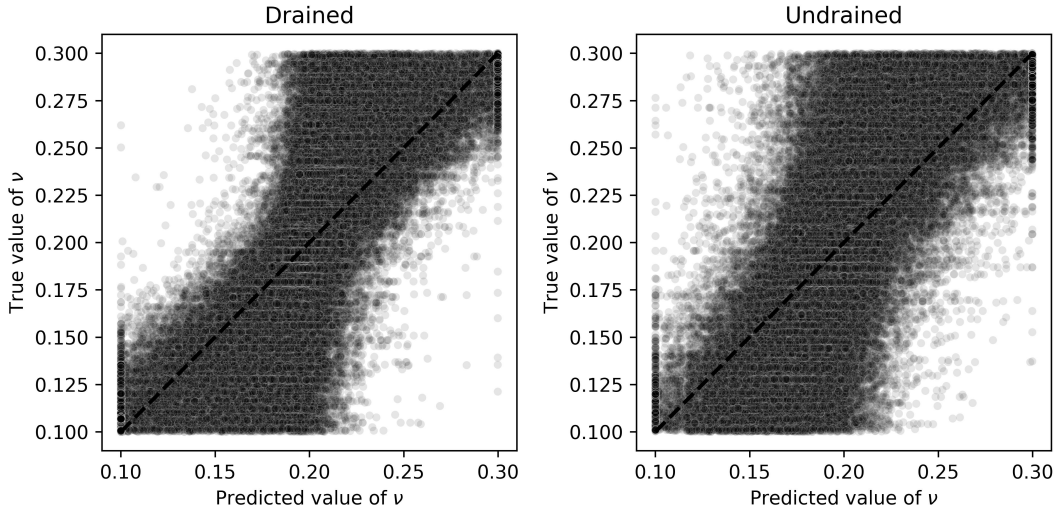

Figure 8. Scatter plots of true and predicted values for ν obtained by the best performing algorithm (Ridge-KT) with the 2048×42 training dataset for both drained and undrained tests.

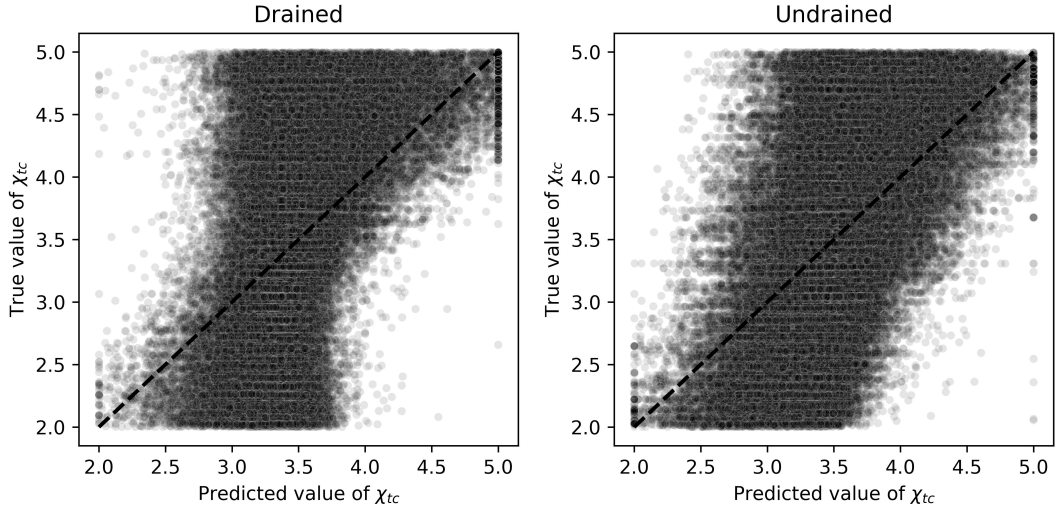

Figure 9. Scatter plots of true and predicted values for χ_{tc} obtained by the best performing algorithm (Ridge-KT) with the 2048×42 training dataset for both drained and undrained tests.

For the OCR values, it is clear from Figure 12 that when drained tests are used to calibrate the ML algorithm, there is no clear trend in the plot. It is closer to a Z pattern, which indicates a slight mid-point prediction behavior, which pulls the values closer to the mean training value. When undrained tests are used in the training and validation process, there is a much clearer prediction pattern.

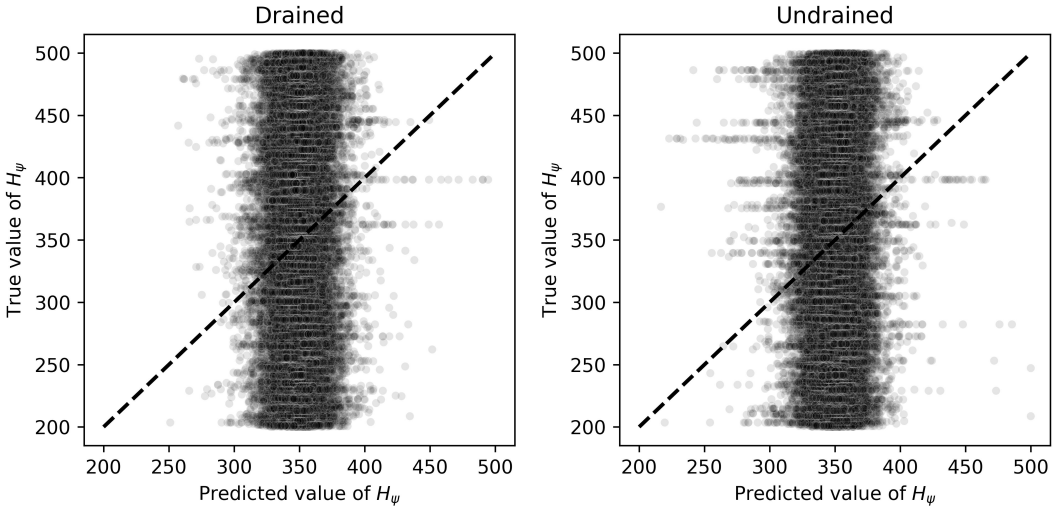

Figure 10. Scatter plots of true and predicted values for H_ψ obtained by the best performing algorithm (Ridge-KT) with the 2048×42 training dataset for both drained and undrained tests.

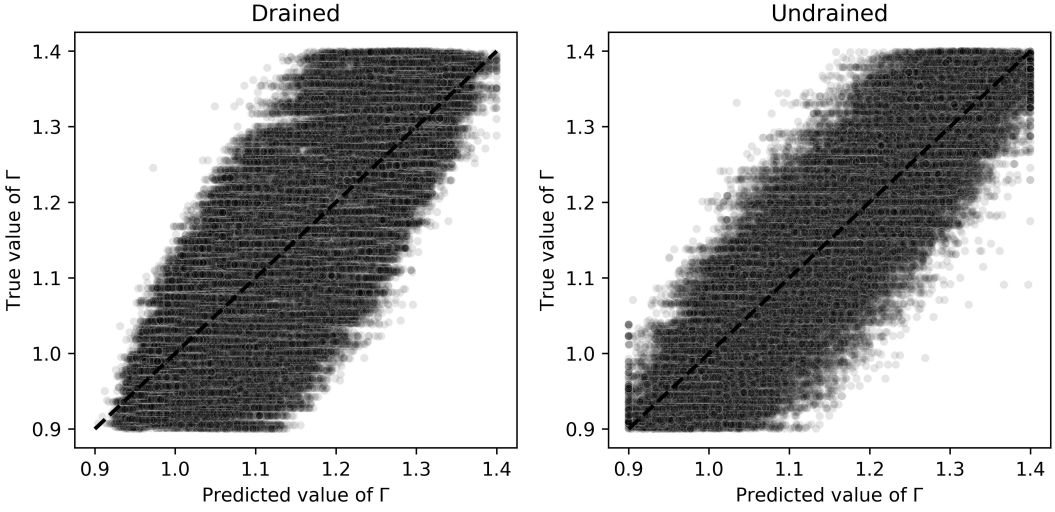

Figure 11. Scatter plots of true and predicted values for Γ obtained by the best performing algorithm (Ridge-KT) with the 2048×42 training dataset for both drained and undrained tests.

For the elastic properties G_{exp} and G_{max,p'_0}, Figures 13 and 14 indicate clear superior performances for algorithms trained and validated using undrained results. For G_{exp}, the relatively low impact of this parameter on the general outputs of the triaxial tests (within the range considered) could impair the learning tasks. A better performance is seen when undrained tests are used, but there is still room for improvement. This is not the case of G_{max,p'_0}, which has a clear sharp trend as seen in Figure 14.

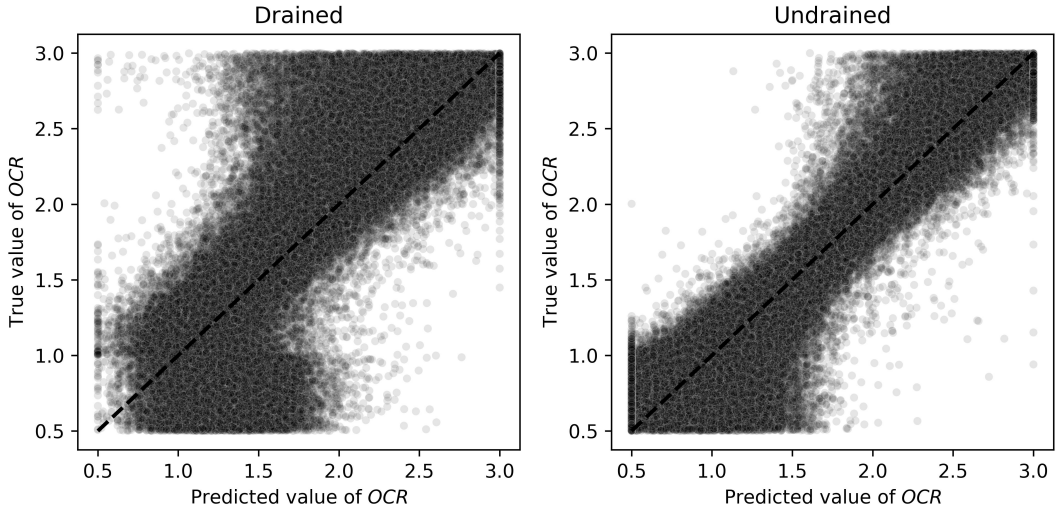

Figure 12. Scatter plots of true and predicted values for OCR obtained by the best performing algorithm (Ridge-KT) with the 2048×42 training dataset for both drained and undrained tests.

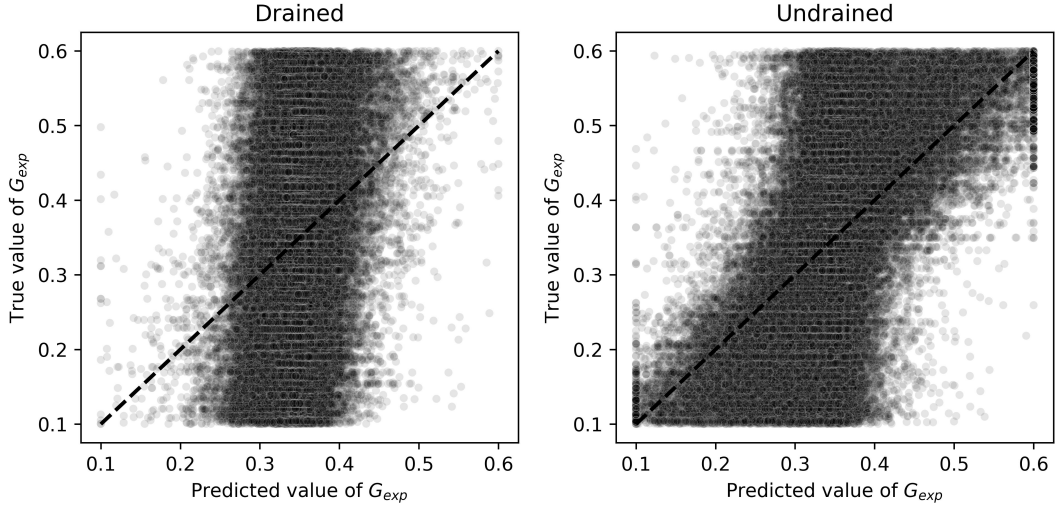

Figure 13. Scatter plots of true and predicted values for G_{exp} obtained by the best performing algorithm (Ridge-KT) with the 2048×42 training dataset for both drained and undrained tests.

For λ, a similar behavior to Γ is observed regarding prediction biases, as seen in Figure 15. The ML algorithm trained and validated using undrained tests provides a more balanced and symmetric prediction scenario, illustrating why it outperforms the algorithm calibrated using drained tests.

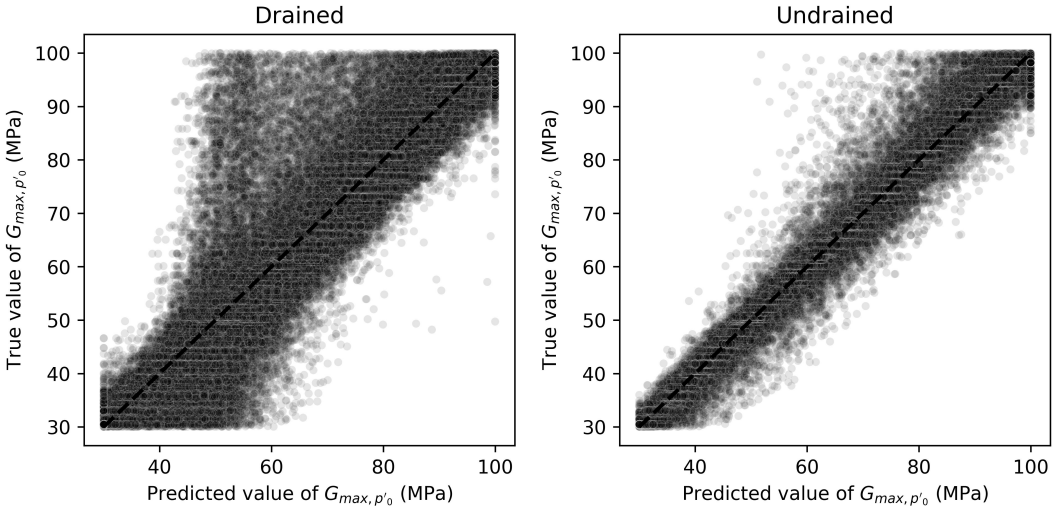

Figure 14. Scatter plots of true and predicted values for G_{max,p'_0} obtained by the best performing algorithm (Ridge-KT) with the 2048×42 training dataset for both drained and undrained tests.

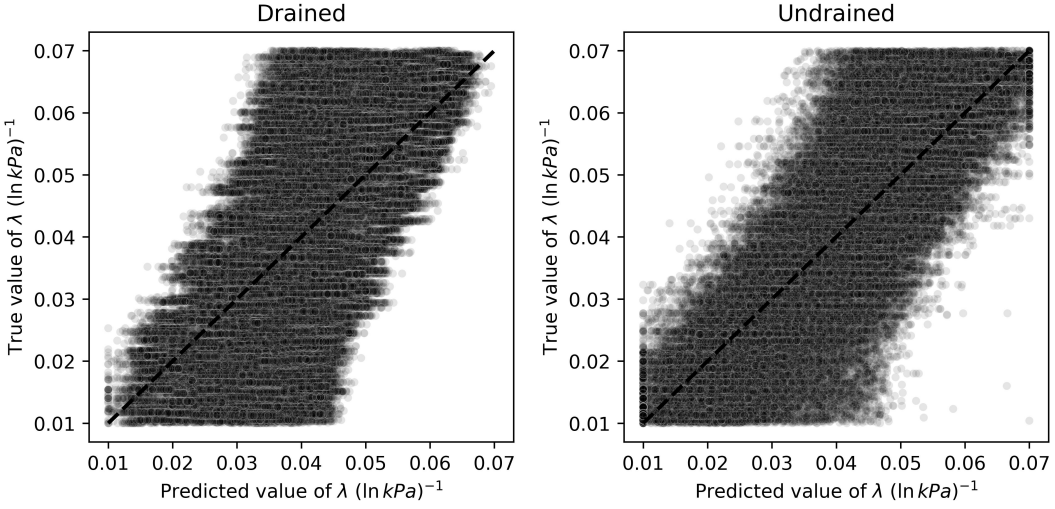

Figure 15. Scatter plots of true and predicted values for λ obtained by the best performing algorithm (Ridge-KT) with the 2048×42 training dataset for both drained and undrained tests.

The opposite situation arises for H_0, which is better predicted when drained tests are considered instead. This is also expected as these types of tests provide a better assessment whether the stress and state–dilatancy properties inferred from the trends in the tests are self-consistent (Jefferies and Been, 2015). Figure 16 presents the results of both ML algorithms, indicating that a

clearer trend is observed when drained tests are used as training and validation datasets. Even though there is also a trend when undrained tests are used, the spread around the identity line is considerable, increasing the MAPE value.

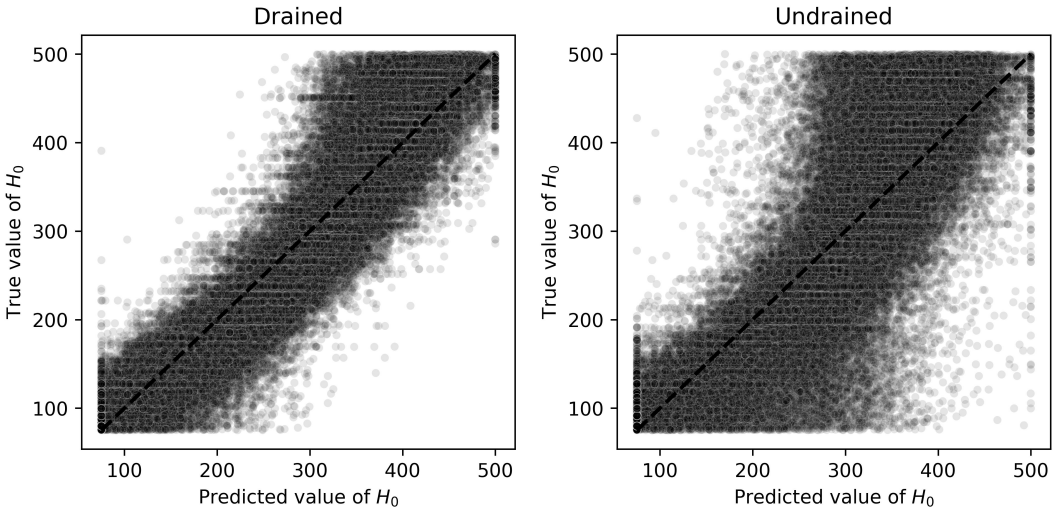

Figure 16. Scatter plots of true and predicted values for H_0 obtained by the best performing algorithm (Ridge-KT) with the 2048×42 training dataset for both drained and undrained tests.

6.1.2 Effect of training sample sizes on the learning task

By analyzing Figures 5 and 6, apparently the overall MAPE slightly increases in the right-bottom corner (large constitutive parameters samples with lower test parameter samples). This is a visual artifice caused by the application of the log-scale to the horizontal axis, which ends up compressing the values on that corner. If the natural scale was considered, one would see that the opposite occurs: large constitutive parameters samples with lower test parameter samples give better results while compared to small constitutive parameters samples with large test parameter samples. Such behavior can be explained by noticing that out of the 14 parameters, 10 correspond to constitutive parameters, so less training samples impair their learning task.

A MAPE comparison is presented in Figures 17 and 18 for both drained and undrained tests with different training sample's diversities (we compare the best performing models obtained by Ridge-KT algorithm, which use the 2048×42 dataset, to two other case: 32×42 and 2048×6 training samples). It is possible to see that the errors of the 10 constitutive parameters exhibit a greater sensitivity to less training samples than the opposite situation with test parameters. Except for OCR, all the other heavily impaired parameters are constitutive ones.

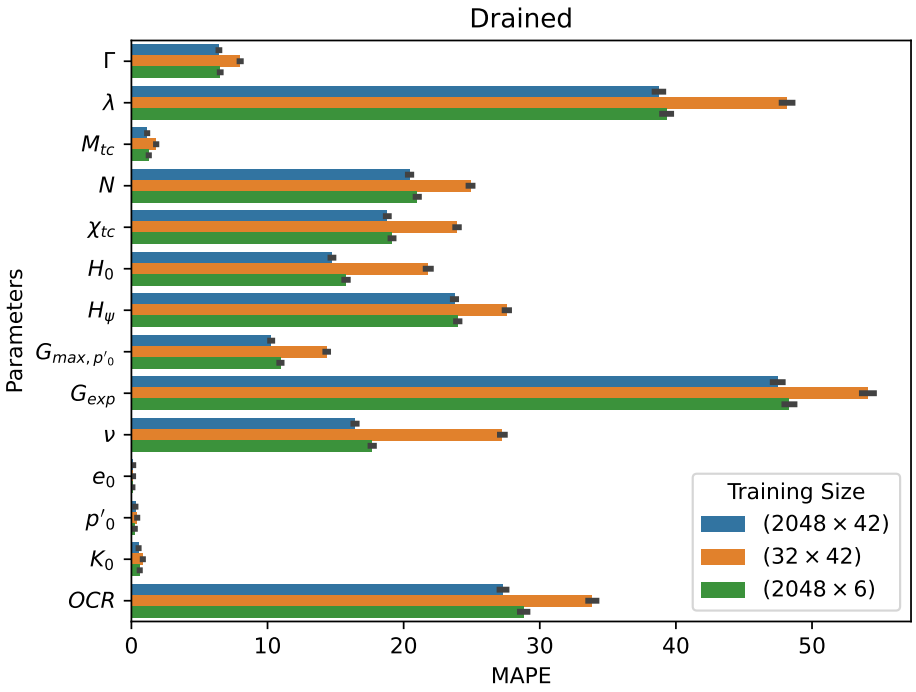

Figure 17. Drained mean absolute percentage errors obtained for each parameter by the best performing algorithm (Ridge-KT) with training datasets of different size.

7 Usage Notes and Codes

In Python, the *h5py* package provides all the necessary tools to interact with the *.h5* files produced and made available in the NorSand4AI dataset. Depending on the intended application, it might be beneficial to down-sample the 4000 × 10 matrix to increase the axial strain increments. This can be accomplished using standard Python packages such as *pandas* and *numpy*. In this section, the codes used to generate the datasets are presented. At first, the following Python packages need to be imported:

```
1: import numpy as np
2: import math
3: import pandas as pd
4: import xlwings as xw
5: import string
6: from skopt.space import Space
7: from skopt.sampler import Lhs
8: from scipy.stats import qmc
9: import os
10: import h5py
```

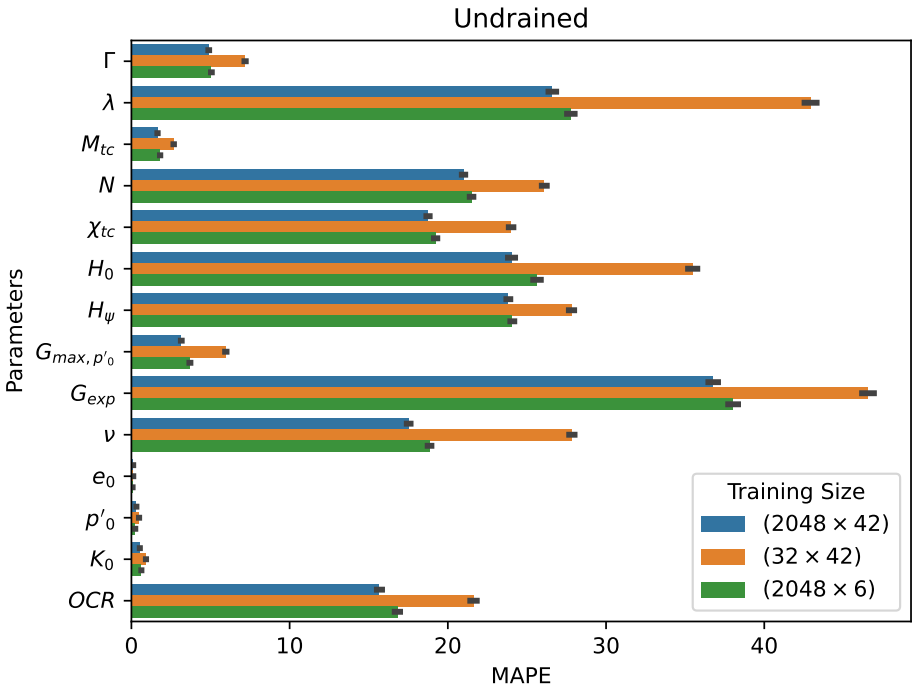

Figure 18. Drained mean absolute percentage errors obtained for each parameter by the best performing algorithm (Ridge-KT) with training datasets of different size.

The packages *numpy*, *math* and *pandas* are required for data manipulation and numeric calculations. The *xlwings* package is needed to bridge Python and Excel. On the other hand, the *string* package is necessary to convert the (row-column) positional encoding to the (row-letter) alphanumeric encoding used in Excel. For the Latin Hypercube sampling procedure, *skopt* is required, while *qmc* from *scipy.stats* is need for the quasi-Monte Carlos sampling. Lastly, for creating folders and files, both *os* and *h5py* should be imported.

Let $dictpos$ be a dictionary that points to the locations in the spreadsheet of the cells corresponding to each input parameter. Additionally, let $dict\_ranges\_material$ and $dict\_ranges\_test$ be dictionaries specifying the sampling ranges of the input parameters. For this paper, these dictionaries are:

```
dictpos = {"Gamma":[6,4],"lambda":[7,4],"Mtc":[14,4], "N":[15,4],
        "Xtc": [16,4],"H0":[17,4],"Hy":[18,4], "Gmax_p0":[21,4],
        "G_exp": [22,4], "nu":[23,4],"Psi_0":[27,4],"p0":[29,4],
        "K0": [30,4], "OCR": [32,4]}
dict_ranges_material = {"Gamma":[0.9,1.4],"lambda":[0.01,0.07],"Mtc":[1.2,1.5],"N":[0.2,0.5],
        "Xtc": [2,5],"H0":[75,500],"Hy":[200,500], "Gmax_p0":[30,100],
        "G_exp": [0.1,0.6], "nu":[0.1,0.3]}
dict_ranges_test = {"Psi_0":[−0.2,0.2],"p0":[50,1000],
```

```
9:          "K0": [0.8,1.2], "OCR": [0.5,3]}
```

7.1 Simply run NorSand in Python

If one seeks to simply run NorSand in Python, the function $run\_NorSand$ can be used. Its inputs are:

- $final\_comp$: input parameters as a numpy array of shape (1,14). The parameters need to be inserted in the same order as $dictpos.keys()$, i.e., ['Gamma', 'lambda', 'Mtc', 'N', 'Xtc', 'H0', 'Hy', 'Gmax_p0', 'G_exp', 'nu', 'Psi_0', 'p0', 'K0', 'OCR'].
- $dictpos$: dictionary to locate the parameters inside the spreadsheet.
- $path\_root$: path of the spreadsheet "NorTxl.xlsm", obtained at http://www.crcpress.com/product/isbn/9781482213683
- $type\_v$: type of the simulation (either "Drained" or "Undrained")

```
1: def run_NorSand(final_comp,dictpos,path_root,type_v):
2:     letters = list(string.ascii_uppercase)
3:     wb = xw.Book(path_root)
4:     app = wb.app
5:     macro_vba = app.macro("'NorTxl.xlsm'!RunSim")
6:     macro_vba_type = app.macro("'NorTxl.xlsm'!ChangeSimMode")
7:     ws = wb.sheets["Params & Plots"]
8:     results_comp = []
9:     for new_v in final_comp:
10:        for nv,ps in zip(new_v,dictpos.values()):
11:            pl,pc = ps
12:            pfinal = letters[pc-1]+str(pl)
13:            ws[pfinal].value = nv
14:        if ws["D34"].value == type_v:
15:            pass
16:        else:
17:            macro_vba_type()
18:        macro_vba()
19:        ws_results = wb.sheets["Txl SimResults"]
20:        np_arr = (ws_results['A4'].expand('table')).value
21:        dd = np.array(np_arr).astype(np.float64)
22:        dict_inpts = {}
23:        for keyv,pvalu in zip(dictpos.keys(),new_v.astype(np.float64)):
```

```
        dict_inpts[keyv] = pvalu
    dict_inpts["Type"] = type_v
    return dict_inpts,pd.DataFrame(dd)
```

This function outputs two entities: a dictionary containing the parameters inserted to run the simulation and a 4000×10 pandas dataframe with simulation results (which are located inside the "Txl SimResults" tab of the xlsm file). The columns are the ones presented in Table 3.

7.2 Generate and save files

To generate the LHS inputs for the NorSandTXL spreadsheet, considering $n\_samples$ soil types and $n\_samples\_2$ initial test conditions, the following code was considered:

```
def gen_NorSand_par_2(dict_ranges_material,dict_ranges_test,n_samples,n_samples_2):
    lhs = Lhs(lhs_type="centered", criterion='maximin')
    lhsinner = Lhs(criterion="ratio")
    space_material = Space([(0, 1.) for x in range(len(dict_ranges_material))])
    space_test = Space([(0, 1.) for x in range(len(dict_ranges_test))])
    x_mat = lhs.generate(space_material.dimensions, n_samples,random_state=11)
    data_inp_mat = (np.array(x_mat).T)
    data_expand_mat = []
    for ind_vals in range(len(dict_ranges_material)):
        vlow,vup = list(dict_ranges_material.values())[ind_vals]
        data_pts = data_inp_mat[ind_vals]
        data_expand_mat.append((vup-vlow)*data_pts + vlow)
    data_expand_mat = np.round(np.array(data_expand_mat),4)
    data_expand_tst_corretos=[]
    for pbb,yv in enumerate(data_expand_mat.T):
        x_tst = lhsinner.generate(space_test.dimensions, n_samples_2,random_state=int(11+2*pbb))
        data_inp_tst = (np.array(x_tst).T)
        data_expand_tst = []
        for ind_vals in range(len(dict_ranges_test)):
            if ind_vals==0:
                data_expand_tst.append(data_inp_tst[ind_vals])
            else:
                vlow,vup = list(dict_ranges_test.values())[ind_vals]
                data_pts = data_inp_tst[ind_vals]
                data_expand_tst.append((vup-vlow)*data_pts + vlow)
        data_expand_tst = np.array(data_expand_tst)
```

```
    data_expand_tst_prov = data_expand_tst.copy()
    data_expand_tst_prov[0] = np.array([(np.clip(yv[2]/(yv[4]*(1+yv[3])),0, yv[2]/(5*yv[4]*(1+yv[3])))+0.2)*lhsv-0.2 for lhsv in
     data_expand_tst_prov[0]])
    data_expand_tst_corretos.append(data_expand_tst_prov)
  data_expand_tst_corretos = np.round(np.array(data_expand_tst_corretos),4)
  final_comp=[]
  for mat_vals,tst_vals in zip(data_expand_mat.T,data_expand_tst_corretos):
    for ti_vals in tst_vals.T:
      final_comp.append(np.concatenate((mat_vals,ti_vals),axis=0))
  return final_comp
```

The quasi-Monte Carlos sampling schemes (Sobol and Halton) can be used to generate the input samples by means of the $gen\_NorSand\_par\_LD$ function, written as:

```
def gen_NorSand_par_LD(dict_ranges_material,dict_ranges_test,n_samples,n_samples_2):
  sampler = qmc.Sobol(d=len(dict_ranges_material), scramble=True,seed=11)
  x_mat = sampler.random_base2(m=int(np.log2(n_samples)))
  data_inp_mat = x_mat.T
  data_expand_mat = []
  for ind_vals in range(len(dict_ranges_material)):
    vlow,vup = list(dict_ranges_material.values())[ind_vals]
    data_pts = data_inp_mat[ind_vals]
    data_expand_mat.append((vup-vlow)*data_pts + vlow)
  data_expand_mat = np.round(np.array(data_expand_mat),4)
  data_expand_tst_corretos=[]
  for pbb,yv in enumerate(data_expand_mat.T):
    samplerinner = qmc.Halton(d=len(dict_ranges_test),scramble=True,seed=int(11+2*pbb))
    x_tst = samplerinner.random(n=n_samples_2)
    data_inp_tst = x_tst.T
    data_expand_tst = []
    for ind_vals in range(len(dict_ranges_test)):
      if ind_vals==0:
        data_expand_tst.append(data_inp_tst[ind_vals])
      else:
        vlow,vup = list(dict_ranges_test.values())[ind_vals]
        data_pts = data_inp_tst[ind_vals]
        data_expand_tst.append((vup-vlow)*data_pts + vlow)
    data_expand_tst = np.array(data_expand_tst)
    data_expand_tst_prov = data_expand_tst.copy()
```

```
        data_expand_tst_prov[0] = np.array([(np.clip(yv[2]/(yv[4]*(1+yv[3])),0,yv[2]/(5*yv[4]*(1+yv[3])))+0.2)*lhsv-0.2 for lhsv in
         data_expand_tst_prov[0]])
        data_expand_tst_corretos.append(data_expand_tst_prov)
    data_expand_tst_corretos = np.round(np.array(data_expand_tst_corretos),4)
    final_comp=[]
    for mat_vals,tst_vals in zip(data_expand_mat.T,data_expand_tst_corretos):
        for ti_vals in tst_vals.T:
            final_comp.append(np.concatenate((mat_vals,ti_vals),axis=0))
    return final_comp
```

On the other hand, to run the NorSandTXL Excel spreadsheet located in $path\_xlsm$ for all the input parameters previously obtained as $final\_comp = gen\_NorSand\_par\_2(\ dict\_ranges\_material, dict\_ranges\_test, n\_samples, n\_samples\_2)$ (or $final\_comp = gen\_NorSand\_par\_LD(\ dict\_ranges\_material, dict\_ranges\_test, n\_samples, n\_samples\_2)$ for the quasi-Monte Carlo sampling of inputs), the following function can be run:

```
def run_NorSand_simus_P(final_comp,dictpos,n_samples_2,path_xlsm):
    letras = list(string.ascii_uppercase)
    wb = xw.Book(path_xlsm)
    app = wb.app
    macro_vba = app.macro("'NorTxl.xlsm'!RunSim")
    macro_vba_type = app.macro("'NorTxl.xlsm'!ChangeSimMode")
    ws = wb.sheets["Params & Plots"]
    results_comp = []
    for idini,new_v in enumerate(final_comp):
        matv = int(math.floor(idini/n_samples_2))
        for nv,ps in zip(new_v,dictpos.values()):
            pl,pc = ps
            pfinal = letras[pc-1]+str(pl)
            ws[pfinal].value = nv
        for type_v in ["Drained","Undrained"]:
            if ws["D34"].value == type_v:
                pass
            else:
                macro_vba_type()
            macro_vba()
            ws_results = wb.sheets["Txl SimResults"]
            np_arr = (ws_results['A4'].expand('table')).value
            path_xlsm_init = ("\\").join(path_xlsm.split("\\")[:-1])
            new_h5_file = path_xlsm_init+'\\Simus\\'+type_v+"\\Par_"+str(matv)+"_"+str(idini)+".h5"
```

```
        new_h5_file_spl = new_h5_file.split("\\")
        for va in range(−3,0):
            try:
                os.mkdir(os.path.join(∗new_h5_file_spl[:va]))
            except:
                pass
        h5f = h5py.File(new_h5_file, 'w')
        dd = h5f.create_dataset('NorSandTXL', data=np.array((ws_results['A4'].expand('table')).value).astype(np.float32),compression
    ='gzip')
        for keyv,pvalu in zip(dictpos.keys(),new_v.astype(np.float32)):
            dd.attrs[keyv] = pvalu
        dd.attrs["Type"] = type_v
        h5f.close()
```

The function $run\_NorSand\_simus\_P$ runs the simulation and also saves the results as *.h5* files in the same folder as the Excel spreadsheet. In this case, the new files are saved following the naming convention and folder structure discussed in the paper.

It is worth noticing that for the LHS sampling with 2000 soil types and 40 test conditions, two values of sampled ψ_0 needed to be reduced due to instabilities in the VBA code calculations. These were:

- $final\_comp[19572][10] = 0.085$ and
- $final\_comp[10929][10] = 0.082$.

On the other hand, for the quasi-Monte Carlos sampling with 2048 soil types and 42 test conditions, five values of sampled ψ_0 needed to be reduced due to the same reasons. These were:

- $final\_comp[56382][10] = 0.0849$,
- $final\_comp[57476][10] = 0.0766$,
- $final\_comp[85371][10] = 0.0955$,
- $final\_comp[34971][10] = 0.08$ and
- $final\_comp[41245][10] = 0.072$.

All the codes previously presented are available as the Jupyter notebook *Sample_and_Run.ipynb* at Ozelim et al. (2023b).

7.3 Analyzing errors during learning tasks

As described in the Methods section, we perform a sample size validation. Considering that the codes for such validation are lengthy, they are presented in Ozelim et al. (2023b). The Jupyter notebook *Sample_size_validation.ipynb* is fully commented to illustrate its usage.

8 Conclusions

Obtaining massive datasets for modelling the behavior of soils is of great interest, not only because new artificial intelligence algorithms can be built, but also to assess the adequacy of newly proposed physically informed models. In the context of critical state approaches, the NorSand model has shown provide a good balance complexity and accuracy. Also, this model is used to assess the liquefaction potential of soils, which is a major cause of high scale disasters lately, such as tailing dams' failures.

In this study, major issues were addressed. Firstly, the paper tackled the challenges associated with the quantity and complexity of synthetic datasets required for nonlinear constitutive modeling of soils. This was achieved by simulating both drained and undrained triaxial tests, resulting in two datasets. The first dataset involved a nested Latin Hypercube Sampling of input parameters, covering 2000 soil types with 40 initial test configurations for each, yielding a total of 160000 triaxial test results. The second dataset employed a nested quasi-Monte Carlo sampling (Sobol and Halton) of input parameters, encompassing 2048 soil types with 42 initial test configurations for each, resulting in a total of 172032 triaxial test results. Each simulation dataset was represented as a matrix of dimensions 4000×10. The study demonstrated that the dataset of 2000 soil types and 40 initial test configurations adequately captured the general behavior of the NorSand model.

Secondly, the paper addressed the issue of the availability of open-source implementations of the NorSand constitutive model. This was achieved by presenting an implementation that connects the well-established VBA implementation to the Python environment. The VBA code served as the "processing kernel" for the new Python implementation, leveraging the extensive testing and validation conducted by Jefferies and Been (2015). This integration allows researchers to harness the full capabilities of Python packages in their analyses involving the NorSand model.

A comprehensive database like the one provided is crucial for developing ML and artificial intelligence models of geotechnical materials. In particular, all geotechnical critical state models involve specific simplifications, with the most apparent being their reliance on 'remoulded' or disturbed soil properties. Understanding the consequences of such structural alterations, especially in terms of their impact on the apparent OCR, poses notable challenges. The effect on the stress ratio (ψ) remains unclear. Through the utilization of physics-informed machine learning and artificial intelligence algorithms, these uncertainties can be thoroughly investigated, uncovering patterns and hidden features within experimental data. We are confident that the results of the present paper are useful assets in this quest, being useful for both academic and industrial communities. Furthermore, researchers interested in modeling sequential data, such as time series, could use this dataset for benchmarking purposes, as the highly non-linear nature of the constitutive model poses a significant challenge to ML and DL techniques.

9 Code and data availability

All data associated with the current submission is available at Ozelim et al. (2023a). Any updates will also be published on Zenodo, and the final DOI cited in the manuscript. The Python code used to generate the NorSandAI dataset is described in the present paper and available at Ozelim et al. (2023b). Besides, the codes used for the learning task considered are also available at Ozelim et al. (2023b)

Author contributions. Conceptualization, L.C.d.S.M.O.; methodology, L.C.d.S.M.O.; software, L.C.d.S.M.O.; validation, L.C.d.S.M.O.; formal analysis, L.C.d.S.M.O., M.D.T.C. and A.L.B.C.; investigation, L.C.d.S.M.O.; writing—original draft preparation, L.C.d.S.M.O.; writing—review and editing, M.D.T.C. and A.L.B.C.; supervision, M.D.T.C.; funding acquisition, L.C.d.S.M.O. All authors have read and agreed to the published version of the manuscript.

Competing interests. The authors declare no competing interests.

Acknowledgements. The authors acknowledge the support of the National Council for Scientific and Technological Development (CNPq Grant 102414/2022-0). This study was also financed in part by the Coordination for the Improvement of Higher Education Personnel (CAPES)—Finance Code 001.

References

Basheer, I. A.: Selection of Methodology for Neural Network Modeling of Constitutive Hystereses Behavior of Soils, Computer-Aided Civil and Infrastructure Engineering, 15, 445–463, https://doi.org/10.1111/0885-9507.00206, 2000.

Bentley: NorSand - PLAXIS UDSM - GeoStudio | PLAXIS Wiki - GeoStudio | PLAXIS - Bentley Communities — communities.bentley.com, https://communities.bentley.com/products/geotech-analysis/w/wiki/52850/norsand---plaxis-udsm, [Accessed 15-11-2023], 2022.

Bergstra, J., Komer, B., Eliasmith, C., Yamins, D., and Cox, D. D.: Hyperopt: a Python library for model selection and hyperparameter optimization, Computational Science & Discovery, 8, 014 008, https://doi.org/10.1088/1749-4699/8/1/014008, 2015.

Feinberg, J. and Langtangen, H. P.: Chaospy: An open source tool for designing methods of uncertainty quantification, Journal of Computational Science, 11, 46–57, https://doi.org/https://doi.org/10.1016/j.jocs.2015.08.008, 2015.

Fu, Q., Hashash, Y. M., Jung, S., and Ghaboussi, J.: Integration of laboratory testing and constitutive modeling of soils, Computers and Geotechnics, 34, 330–345, https://doi.org/10.1016/j.compgeo.2007.05.008, 2007.

Furukawa, T. and Yagawa, G.: Implicit constitutive modelling for viscoplasticity using neural networks, International Journal for Numerical Methods in Engineering, 43, 195–219, https://doi.org/10.1002/(sici)1097-0207(19980930)43:2<195::aid-nme418>3.0.co;2-6, 1998.

Ghaboussi, J., Garrett, J. H., and Wu, X.: Knowledge-Based Modeling of Material Behavior with Neural Networks, Journal of Engineering Mechanics, 117, 132–153, https://doi.org/10.1061/(asce)0733-9399(1991)117:1(132), 1991.

Group, T. H.: Software Using HDF5, https://portal.hdfgroup.org/display/support/Software+Using+HDF5, Accessed on April 24, 2023.

Haghighat, E., Raissi, M., Moure, A., Gomez, H., and Juanes, R.: A physics-informed deep learning framework for inversion and surrogate modeling in solid mechanics, Computer Methods in Applied Mechanics and Engineering, 379, 113 741, https://doi.org/10.1016/j.cma.2021.113741, 2021.

Halton, J. H.: On the efficiency of certain quasi-random sequences of points in evaluating multi-dimensional integrals, Numerische Mathematik, 2, 84–90, https://doi.org/10.1007/bf01386213, 1960.

Hashash, Y. M. A., Jung, S., and Ghaboussi, J.: Numerical implementation of a neural network based material model in finite element analysis, International Journal for Numerical Methods in Engineering, 59, 989–1005, https://doi.org/10.1002/nme.905, 2004.

He, X., He, Q., and Chen, J.-S.: Deep autoencoders for physics-constrained data-driven nonlinear materials modeling, Computer Methods in Applied Mechanics and Engineering, 385, 114 034, https://doi.org/10.1016/j.cma.2021.114034, 2021.

Heider, Y., Wang, K., and Sun, W.: SO(3)-invariance of informed-graph-based deep neural network for anisotropic elastoplastic materials, Computer Methods in Applied Mechanics and Engineering, 363, 112 875, https://doi.org/10.1016/j.cma.2020.112875, 2020.

Itasca Consulting Group, I.: NorSand Model; FLAC3D 7.0 documentation — docs.itascacg.com, https://docs.itascacg.com/flac3d700/common/models/norsand/doc/modelnorsand.html, [Accessed 15-11-2023], 2023.

Jefferies, M. and Been, K.: Soil Liquefaction: A Critical State Approach, Second Edition, CRC Press, 2 edn., https://doi.org/10.1201/b19114, 2015.

Jefferies, M. G.: Nor-Sand: a simle critical state model for sand, Géotechnique, 43, 91–103, https://doi.org/10.1680/geot.1993.43.1.91, 1993.

Jefferies, M. G. and Shuttle, D. A.: Dilatancy in general Cambridge-type models, Géotechnique, 52, 625–638, https://doi.org/10.1680/geot.2002.52.9.625, 2002.

Jung, S. and Ghaboussi, J.: Neural network constitutive model for rate-dependent materials, Computers & Structures, 84, 955–963, https://doi.org/10.1016/j.compstruc.2006.02.015, 2006.

Komer, B., Bergstra, J., and Eliasmith, C.: Hyperopt-Sklearn: Automatic Hyperparameter Configuration for Scikit-Learn, in: Proc. of the 13th Python in Science Conf. (SCIPY 2014), pp. 32–37, https://doi.org/10.25080/Majora-14bd3278-006, 2014.

Lee, S., yuan Hou, K., Wang, K., Sehrish, S., Paterno, M., Kowalkowski, J., Koziol, Q., Ross, R. B., Agrawal, A., Choudhary, A., and keng Liao, W.: A case study on parallel HDF5 dataset concatenation for high energy physics data analysis, Parallel Computing, 110, 102 877, https://doi.org/https://doi.org/10.1016/j.parco.2021.102877, 2022.

Lefik, M. and Schrefler, B.: Artificial neural network as an incremental non-linear constitutive model for a finite element code, Computer Methods in Applied Mechanics and Engineering, 192, 3265–3283, https://doi.org/10.1016/s0045-7825(03)00350-5, 2003.

Montáns, F. J., Chinesta, F., Gómez-Bombarelli, R., and Kutz, J. N.: Data-driven modeling and learning in science and engineering, Comptes Rendus Mécanique, 347, 845–855, https://doi.org/10.1016/j.crme.2019.11.009, 2019.

Mozaffar, M., Bostanabad, R., Chen, W., Ehmann, K., Cao, J., and Bessa, M. A.: Deep learning predicts path-dependent plasticity, Proceedings of the National Academy of Sciences, 116, 26 414–26 420, https://doi.org/10.1073/pnas.1911815116, 2019.

Nti-Addae, Y., Matthews, D., Ulat, V. J., Syed, R., Sempéré, G., Pétel, A., Renner, J., Larmande, P., Guignon, V., Jones, E., and Robbins, K.: Benchmarking database systems for Genomic Selection implementation, Database, 2019, baz096, https://doi.org/10.1093/database/baz096, 2019.

Owen, A. B. and Rudolf, D.: A Strong Law of Large Numbers for Scrambled Net Integration, SIAM Review, 63, 360–372, https://doi.org/10.1137/20M1320535, 2021.

Ozelim, L. C. d. S. M., Casagrande, M. D. T., and Cavalcante, A. L. B.: Database for NorSand4AI: A Comprehensive Triaxial Test Simulation Database for NorSand Constitutive Model Materials, https://doi.org/10.5281/zenodo.8170536, 2023a.

Ozelim, L. C. d. S. M., Casagrande, M. D. T., and Cavalcante, A. L. B.: Codes for NorSand4AI: A Comprehensive Triaxial Test Simulation Database for NorSand Constitutive Model Materials, https://doi.org/10.5281/zenodo.10157831, 2023b.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E.: Scikit-learn: Machine Learning in Python, Journal of Machine Learning Research, 12, 2825–2830, 2011.

Rocscience: NorSand | RS2 | Advanced Constitutive Material Model — rocscience.com, https://www.rocscience.com/learning/norsand-in-rs2-an-advanced-constitutive-material-model, [Accessed 30-10-2023], 2022.

Roscoe, K. H. and Burland, J. B.: On the generalized stress-strain behaviour of 'wet' clay, in: Engineering plasticity, edited by Heyman, J. and Leckie, F., pp. 535–609, Cambridge University Press, Cambridge, 1968.

Schmidt, M. and Lipson, H.: Distilling Free-Form Natural Laws from Experimental Data, Science, 324, 81–85, https://doi.org/10.1126/science.1165893, 2009.

Schofield, A. N. and Wroth, P.: Critical State Soil Mechanics, European civil engineering series, McGraw-Hill, 1968.

Shen, Y., Chandrashekhara, K., Breig, W., and Oliver, L.: Finite element analysis of V-ribbed belts using neural network based hyperelastic material model, International Journal of Non-Linear Mechanics, 40, 875–890, https://doi.org/10.1016/j.ijnonlinmec.2004.10.005, 2005.

Silva, J. P., Cacciari, P., Torres, V., Ribeiro, L. F., and Assis, A.: Behavioural analysis of iron ore tailings through critical state soil mechanics, Soils and Rocks, 45, 1–13, https://doi.org/10.28927/sr.2022.071921, 2022.

Sobol, I.: On the distribution of points in a cube and the approximate evaluation of integrals, USSR Computational Mathematics and Mathematical Physics, 7, 86–112, https://doi.org/10.1016/0041-5553(67)90144-9, 1967.

Sternik, K.: Technical Notoe: Prediction of Static Liquefaction by Nor Sand Constitutive Model, Studia Geotechnica et Mechanica, 36, 75–83, https://doi.org/10.2478/sgem-2014-0029, 2015.

Stoffel, M., Bamer, F., and Markert, B.: Neural network based constitutive modeling of nonlinear viscoplastic structural response, Mechanics Research Communications, 95, 85–88, https://doi.org/10.1016/j.mechrescom.2019.01.004, 2019.

The HDF Group: Hierarchical Data Format, version 5, https://www.hdfgroup.org/HDF5/, 1997-2023.

Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., van der Walt, S. J., Brett, M., Wilson, J., Millman, K. J., Mayorov, N., Nelson, A. R. J., Jones, E., Kern, R., Larson, E., Carey, C. J., Polat, İ., Feng, Y., Moore, E. W., VanderPlas, J., Laxalde, D., Perktold, J., Cimrman, R., Henriksen, I., Quintero, E. A., Harris, C. R., Archibald, A. M., Ribeiro, A. H., Pedregosa, F., van Mulbregt, P., and SciPy 1.0 Contributors: SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python, Nature Methods, 17, 261–272, https://doi.org/10.1038/s41592-019-0686-2, 2020.

Wang, K. and Sun, W.: A multiscale multi-permeability poroplasticity model linked by recursive homogenizations and deep learning, Computer Methods in Applied Mechanics and Engineering, 334, 337–380, https://doi.org/10.1016/j.cma.2018.01.036, 2018.

Woudstra, L.-J.: Verification, Validation and Application of the NorSand Constitutive Model in PLAXIS: Single-stress point analyses of experimental lab test data and finite element analyses of a submerged landslide, Master's thesis, TU Delft Civil Engineering & Geosciences, 2021.

Yang, T., Li, Y.-f., Mahdavi, M., Jin, R., and Zhou, Z.-H.: Nyström Method vs Random Fourier Features: A Theoretical and Empirical Comparison, in: Advances in Neural Information Processing Systems, edited by Pereira, F., Burges, C., Bottou, L., and Weinberger, K., vol. 25, Curran Associates, Inc., https://proceedings.neurips.cc/paper_files/paper/2012/file/621bf66ddb7c962aa0d22ac97d69b793-Paper.pdf, 2012.

Yao, Y., Sun, D., and Matsuoka, H.: A unified constitutive model for both clay and sand with hardening parameter independent on stress path, Computers and Geotechnics, 35, 210–222, https://doi.org/10.1016/j.compgeo.2007.04.003, 2008.

Zhang, N., Zhou, A., Jin, Y.-F., Yin, Z.-Y., and Shen, S.-L.: An enhanced deep learning method for accurate and robust modelling of soil stress–strain response, Acta Geotechnica, https://doi.org/10.1007/s11440-023-01813-8, 2023.

Zhang, P., Yin, Z.-Y., Jin, Y.-F., and Ye, G.-L.: An AI-based model for describing cyclic characteristics of granular materials, International Journal for Numerical and Analytical Methods in Geomechanics, 44, 1315–1335, https://doi.org/10.1002/nag.3063, 2020.

Zhang, P., Yin, Z.-Y., and Jin, Y.-F.: State-of-the-Art Review of Machine Learning Applications in Constitutive Modeling of Soils, Archives of Computational Methods in Engineering, 28, 3661–3686, https://doi.org/10.1007/s11831-020-09524-z, 2021a.

Zhang, P., Yin, Z.-Y., Jin, Y.-F., and Liu, X.-F.: Modelling the mechanical behaviour of soils using machine learning algorithms with explicit formulations, Acta Geotechnica, 17, 1403–1422, https://doi.org/10.1007/s11440-021-01170-4, 2021b.