Automation of LS-DYNA's Material Model Driver for Generation of Training Data for Machine Learning based Material Models

Daniel Sommer, Tarun Kumar Mitruka Vinod Kumar Mitruka, Peter Middendorf

Institute of Aircraft Design, University of Stuttgart
Pfaffenwaldring 31, 70569 Stuttgart, Germany

daniel.sommer@ifb.uni-stuttgart.de
mitruka.tarun@gmail.com
peter.middendorf@ifb.uni-stuttgart.de

1 Abstract

The substitution of classical constitutive material models with data-driven models supported by machine learning techniques could provide a leap in the modelling of materials. The most notable benefits are a faster description of new materials without a tedious manual parameter identification procedure, lower computational time for simulations due to efficient computation within the material model and a more efficient selection of the correct material model for the use-case. The base for any data-driven model is adequate amount and quality of training data. Based on this, machine learning techniques can be used to train neural networks such that they learn the relationship between given input and output. The mapping in the machine learning based material model will be the strain measures to the stresses, similar to classical models. In order to learn the stress-strain relationship for materials, training data is generated from existing material models in LS-DYNA, as the direct extraction from materials testing is not possible due to the impossibility of local stress measurements. To generate training data, an existing model could be implemented in own code, single-element simulations could be performed, or even data from component simulations could be extracted. This study shows a way to generate training data for almost any material model available in LS-DYNA by using and automating the integrated Interactive Material Model Driver. This enables access to the unbiased stress response of the black-box material models, by providing an arbitrary progression of components of the displacement gradient in time as input, to be evaluated at a single integration point. Automation is added on top of the embedded Material Model Driver regarding the generation of strain paths for 2D and 3D cases. It is followed by automatic execution, data extraction and preparation of data for machine learning. Advantages and disadvantages over other strategies for training data generation will be highlighted, compared and finally an exemplary material model using neural networks trained on data generated with the Automated Material Model Driver is shown. The goal is that networks learnt through such a training data can also be used as a surrogate to develop any new material model.

2 Introduction

Classical constitutive material models available today require conducting numerous varieties of experiments to obtain certain material parameters required by the specific model in order to have a suitable fit to the experiment data. Apart from this, a high level of expertise is required in order to choose the appropriate material model for a given material. This usually consumes a large amount of time during the development phase of any project. Machine learning techniques are recently gaining popularity to develop material models from experimental data, thereby eliminating the process of computing material parameters or selecting the classical material model. The main idea behind the motivation of this work is to find a machine learning model which can represent classical analytical models and on which then transfer-learning can be used to describe a new material in the finite element simulation quickly without the drawbacks outlined before.

In literature several publications can be found starting in the late 90s with a peak in recent years, after some time of fewer publications in that area. For example, Palau et al. [1] studied the effects of recurrent and filter-based parametrization techniques to develop a feasible neural network to replace an elasto-plastic material model. Usually, the networks trained from existing material models are used as surrogates to train a material model from the experimental data for the material of interest. One such way of using a surrogate material model neural network was proposed by Ghaboussi et al. [2] already in 1998. Further examples trying to formulate a complete material model using machine learning are [3,

4, 5], others substitute parts of classical models or do parameter identification with machine learning techniques [6, 7] each with different strategies of generating training data.
The most desirable benefit of a Machine Learning based Material Model (MLMM) would be that the experimental data can be directly fed to a pre-trained neural network and a new material model can then be instantly developed by re-training. It is important to know that even though one achieves good time-cost benefits, it is difficult to understand the actual physical meaning of the hidden layers and weights used in a neural network, which requires means of rendering the MLMM explicable for an engineer. For an MLMM to perform more accurately, it has to be trained with crucial amount of high-quality data. This way, errors can be minimized in all types of loading scenarios, thereby making the model robust during an actual application. In this paper, the primary focus is to generate data using the Interactive Material Model Driver (IMMD) and automating the process to capture many loading scenarios in a fast way using `*MAT_024` and `*MAT_SAMP-1` as an example. Later a simple application is discussed, where the data generated with the automated IMMD is used to train a neural network. It is significant that this methodology is not restricted to plastic or hypoelastic models but can be used for any black-box material model in LS-DYNA.

3 Interactive Material Model Driver

The Interactive Material Model Driver (IMMD) is an interface available in LS-DYNA to directly capture the material behavior for a given load scenario for shell and solid material models, as documented in appendix K of the LS-DYNA manual [8]. Schwer [9] highlights the early birth of the IMMD from the fact that constitutive models do not depend on the origin of the strain increments, which would normally be the finite element cycle, and hence IMMD works on user specified strain paths. He also comments on the fact that not all laboratory tests are reproduced using an IMMD simulation as for example, if one has experimental data from uniaxial tests, the corresponding experimental lateral strain data are mostly missing for a precise execution of IMMD. IMMD results in a pure stress-strain behaviour which are free from additional noises due to element formulation, time integration schemes or hourglass control among others. This results in the unbiased response of the original material model as implemented in LS-DYNA. One more limitation of IMMD commented by Schwer [9] is that IMMD only uses strain as input. If it rather uses stresses as input, then one could set lateral stresses as zero and run a smoother uniaxial test simulation and thereby avoiding the issue mentioned earlier.
IMMD calls the material subroutines in LS-DYNA directly and evaluates the stress response for the given strain. The IMMD interface is called on automatically if there is no `*NODE` or `*ELEMENT` keyword in the input deck for a Symmetric Multi-Processing (SMP) execution. For a Massively Parallel Processing execution (MPP), the keyword `*CONTROL_MPP_MATERIAL_MODEL_DRIVER` is to be used. It returns the constitutive response at an integration point analogous to actual applications. In the most recent versions of LS-DYNA the IMMD is only available for SMP Linux versions and is not available for other compilations of the solver, where the IMMD invocation results in various errors. For this paper `ls-dyna_smp_d_R12_0_0_x64_redhat65_ifort160` was used.
The input to LS-DYNA for IMMD consists of a single material model, nine load curves which correspond to time history of the components of displacement gradient, an end time for a simulation, element type and plotting information. The element type is important for choosing the correct implementation – shell or solid – of the actual material model, the element itself has no influence as noted earlier. The nine displacement gradient components are $\partial u/\partial x, \partial v/\partial y, \partial w/\partial z, \partial u/\partial y, \partial v/\partial x, \partial u/\partial z, \partial w/\partial x, \partial v/\partial z, \partial w/\partial y$, where u, v, w are the displacement in x, y, z directions respectively. Thus, we obtain a normal strain in e.g. x-direction as $\varepsilon_x = \partial u/\partial x$ whereas the tensorial shear component is $\varepsilon_{xy} = {}^1/_2\,({}^{\partial u}/_{\partial y} + {}^{\partial v}/_{\partial x})$. For shell elements, the strain increments specified normal to the mid surface by the user is over-written by IMMD, as this is calculated within the shell material models themselves in order to meet the requirement of the through-thickness stress component being zero. It is also possible to use the batch mode feature to execute certain interactive commands in a sequential order and view the desired output quantities. This feature enables for a smoother automation of the tool for data generation which is also commented later in this paper. With this, a set of commands is provided by an mmd.bat, which are executed to e.g. extract the stress response. Further details on the interactive commands and usage of IMMD can be found in appendix K of the manual [8]. The automation discussion in this paper is beyond the capabilities of interactive batch mode feature.

4 Data Generation

Since the data generated by IMMD should include all possible types of deformation for the robustness of the material model, the following approach was chosen. An effective strain was computed (similar to the effective plastic strain computed by LS-DYNA, but including the elastic and volumetric parts) as follows:

$$\varepsilon_{eff} = \sqrt{\frac{2}{3}(\varepsilon_{xx}^2 + \varepsilon_{yy}^2 + \varepsilon_{zz}^2 + 2\varepsilon_{xy}^2 + 2\varepsilon_{yz}^2 + 2\varepsilon_{zx}^2)} \quad (1)$$

The tensorial shear components are replaced with $\varepsilon_{ij} = {}^{\gamma_{ij}}/_{2}$ because Eq. 1 consists of all tensorial components, whereas the output for shear components written by LS-DYNA uses the engineering shear strain. Squaring on either side, we can re-write the above expression as

$$\frac{3}{2}\varepsilon_{eff}^2 = \varepsilon_{xx}^2 + \varepsilon_{yy}^2 + \varepsilon_{zz}^2 + \frac{\gamma_{xy}^2}{2} + \frac{\gamma_{yz}^2}{2} + \frac{\gamma_{zx}^2}{2} \quad (2)$$

This equation can be visualized as a sphere of radius $\sqrt{{}^{3}/_{2}}\,\varepsilon_{eff}$ if the shear components are equal to zero or can be generalized to an ellipsoid with a non-zero shear component. For the load cases discussed later in the paper, an $\varepsilon_{eff} = 1$ was chosen. For larger deformations, ε_{eff} can be increased. Next, we will discuss different ways in which Eq. 2 was used to define the boundary for the strain paths which were given as an input to the IMMD. By generating strain paths in such manner, we cover the entire strain space, regardless of the fact that if such a strain scenario may not be exhibited by any element in realistic applications. For example, some cases may violate proper lateral straining behavior. Even though the straining behavior is not realistic, the material response from IMMD is still valid and thus can be beneficial when such strain cases can be related to a constrained element or an element in contact. The displacement gradients are given to yield a fully populated strain tensor, i.e. strain space is not constructed to give only principal strains, as many material models have a stress response dependent on the shear components as well. If a material model without a connection of shear strains to the stress response is considered, strain paths could be generated in principle space only and the training data later augmented by a rotation into non-principal space on stress and strain. But since popular models such as `SAMP` or `GISSMO` have a shear-strain-dependent stress response, this most general approach was always used.
At the end of this section, we will discuss the automation procedure adopted to run IMMD with these generated strain paths and obtain the corresponding material response for a chosen `*MAT_024` (`*MAT_PIECEWISE_LINEAR_PLASTICITY`) material card.

Before, it is important to understand how the points were chosen on the sphere/ellipsoid such that loading always occurs until a given ε_{eff}, in order for the strain paths to be comparable. The normal and shear strain components were uniquely considered as an axis of a cartesian coordinate system. For plane strain case, only components in the XY-plane were considered. Since the radius of the sphere/ellipsoid is fixed, we compute the strain components at desired time increments using a spherical coordinate system such that Eq. 2 is satisfied at the end time of simulation. Apart from the radius ε_{eff}, two angles in a 3D case and 5 angles in a 6D case is required to compute the strain components using the spherical coordinate system. In the interest of encapsulating the whole strain space, we set the $n-1$ angles in the range 0° and 180°, and the n^{th} angle in the range 0° and 360°. The angle increment is defined by the user, depending on the desired number of paths. An illustration of points on the sphere using this increment approach can be envisaged using Fig. 1, left. It is observed that choosing angles using such an approach makes the data points denser near the poles. This could create troubles during training of the neural network as it creates an unnecessary bias in the data set. Hence for a three-dimensional data set, the angles for the spherical coordinate system were picked according to the algorithm mentioned by Deserno [10]. This uniformly distributes the points selected on the sphere as shown in Fig. 1, right. With the points on the sphere/ellipsoid in hand, one can arrive at this point in any desired fashion. A suitable time step size is selected for the generation of strain paths. However, for training a neural network, the time step size is determined by `DT` from `*DATABASE_BINARY_D3PLOT` keyword as it denotes the output from IMMD. The time step size can also be considered as a critical parameter while training a neural network, especially for explicit dynamics simulations. The strain path

techniques implemented to generate data later using IMMD are discussed in the following.

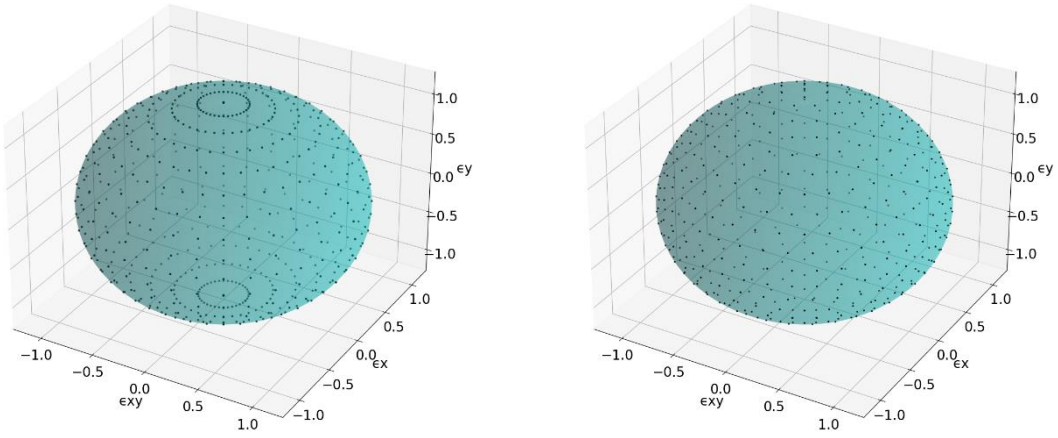


Fig. 1: Non-Uniform Distribution of Data Points (left); Uniform Distribution of Data Points (right)

4.1 Linear Monotonic Strain Paths

Linear strain path denotes that the strain components are varying linearly with time. Monotonic corresponds to the facts that each component is a monotonically increasing/decreasing function and there is no change in the direction of loading until it reaches ε_{eff}. Since the whole strain space is covered, we develop cases where we have loading only in a particular direction, i.e., uniaxial and biaxial tensile/compression cases and also pure shear cases. Strain paths under such a category can be generated for a plane strain case and for three dimensionally loaded case. For a plane strain case, again the user has an option to select points on the sphere uniformly or with high density near the poles. Fig. 2 shows linear monotonic paths (plane strain) inside an ellipsoid and the individual load components while Fig. 3, left shows the components for a 3D case. When presenting stress/strain paths, up to three paths are highlighted in the following figures while the rest is only shown in greyscale. The strain paths exceed beyond the surface of the sphere in order to have points beyond the end time of the simulation in LS-DYNA. In other words, one reaches the surface of the sphere at the end time of an IMMD simulation. From Fig. 3, left, it is observed that the strain paths for normal components have lesser slope variants than those of the shear components. It arises from the technique used to implement the spherical coordinate system. The projected components of the spherical coordinate system can easily be altered to arrive at a denser data for the normal components. An example on how the projections from the spherical coordinates are arrived for the individual load curves can be understood from Eq. 3 to 5 for a plane strain case. For shear, we arrive at the spherical coordinate projection and then equally spilt the component to shear loading components and hence a factor of 0.5 is included. α and β are the angles calculated either through the methods of uniform or non-uniform distribution and r is the radius of the sphere/ellipsoid calculated as discussed earlier. $\sqrt{2}$ in Eq. 5 is to compensate for the factor of 2 in Eq. 2 in order to obtain an ellipsoid.

$$\frac{\partial u}{\partial x} = r * \sin\alpha * \sin\beta \tag{3}$$

$$\frac{\partial v}{\partial y} = r * \sin\alpha * \cos\beta \tag{4}$$

$$\frac{\partial u}{\partial y} = \frac{\partial v}{\partial x} = 0.5 * r * \cos\alpha * \sqrt{2} \tag{5}$$

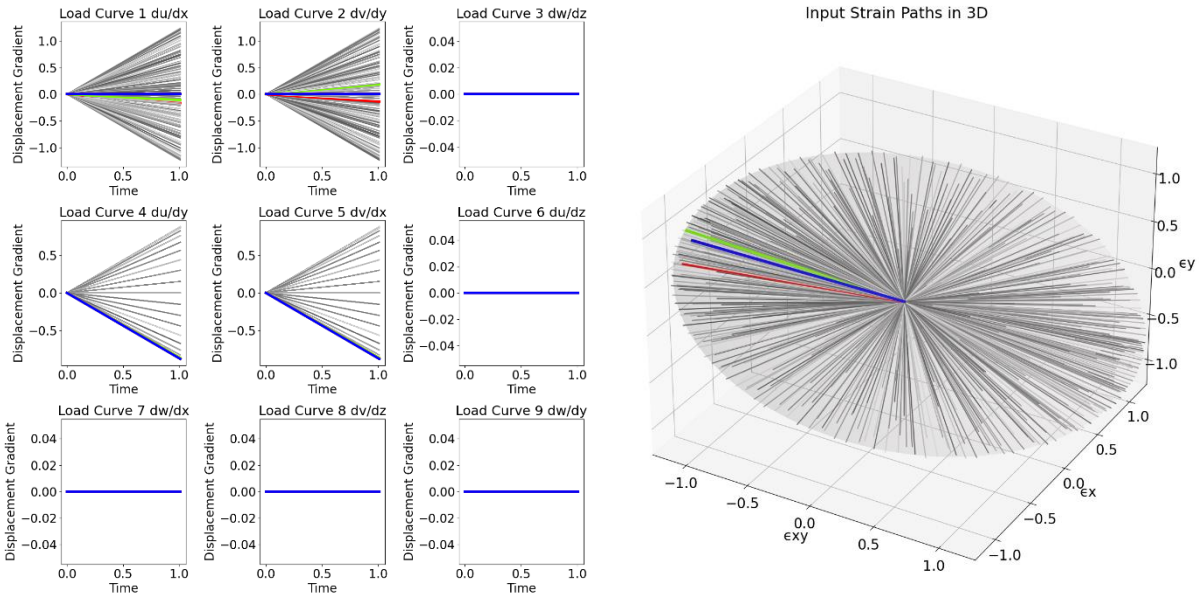


Fig. 2: Linear Monotonic Plane Strain Paths with Non-Uniform Distribution (left), Paths shown in the Ellipsoid (right)

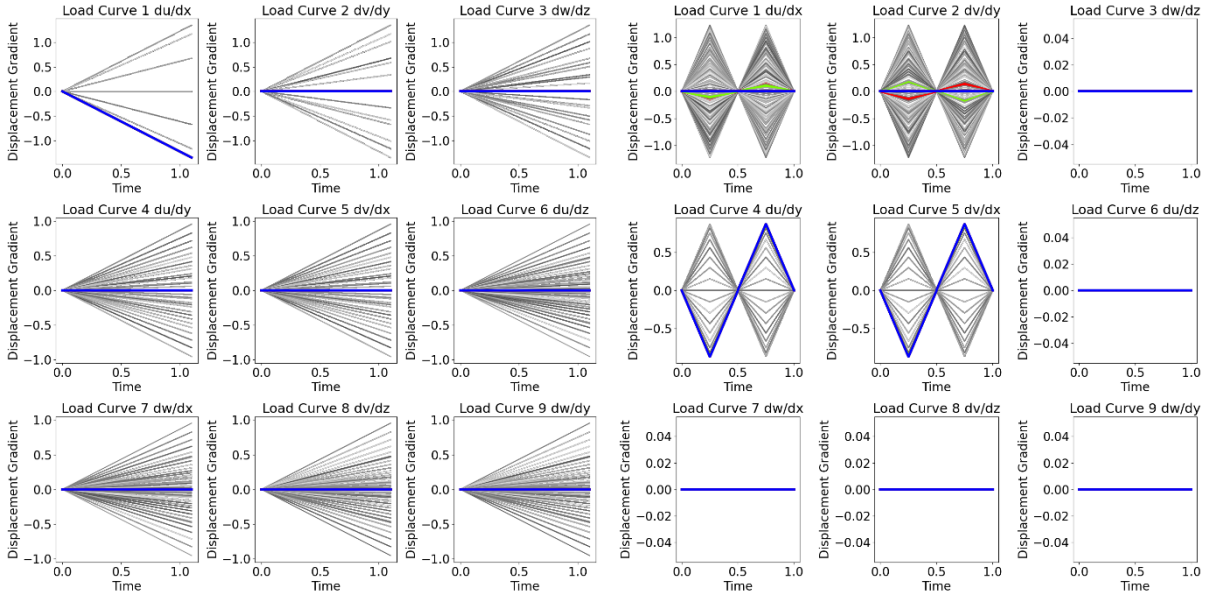


Fig. 3: Linear Monotonic Strain Paths in 3D (left), Linear Cyclic Plane Strain Paths (right)

4.2 Linear Cyclic Strain Paths

Cyclic strain paths symbolize a saw-tooth shaped loading path. This variety of data set can be generated for plane strain cases for both types of points selections on the sphere. If we consider only the first quadrant in a 2D projected data set, the paths are generated such that it first undergoes tension until a quarter of the simulation time $t_{end}/4$. Then it is followed by compression up to the same magnitude until $3t_{end}/4$ with zero-deformation state at $t_{end}/2$. It again achieves a zero-deformation state at t_{end}. A pictorial depiction of the cyclic strain paths can be seen in Fig. 3, right.

4.3 Sinusoidal Strain Paths

The intension of the sinusoidal load paths is to include non-linearity in the data set. The sinusoidal strain paths follow a similar track as the cyclic path, but instead of the increments being linear, it is computed such that the path resembles a sinus wave. This load curve can not only be generated for plane strain

case but also for loading in all directions. Fig. 4 depicts a sinusoidal load curve. Even though the paths appear linear in the projected view of the ellipsoid, the sinusoidal behavior is present in the individual components of the load curve. This adds a variable strain-rate over the path on top of the non-linearity of the progression of individual components.

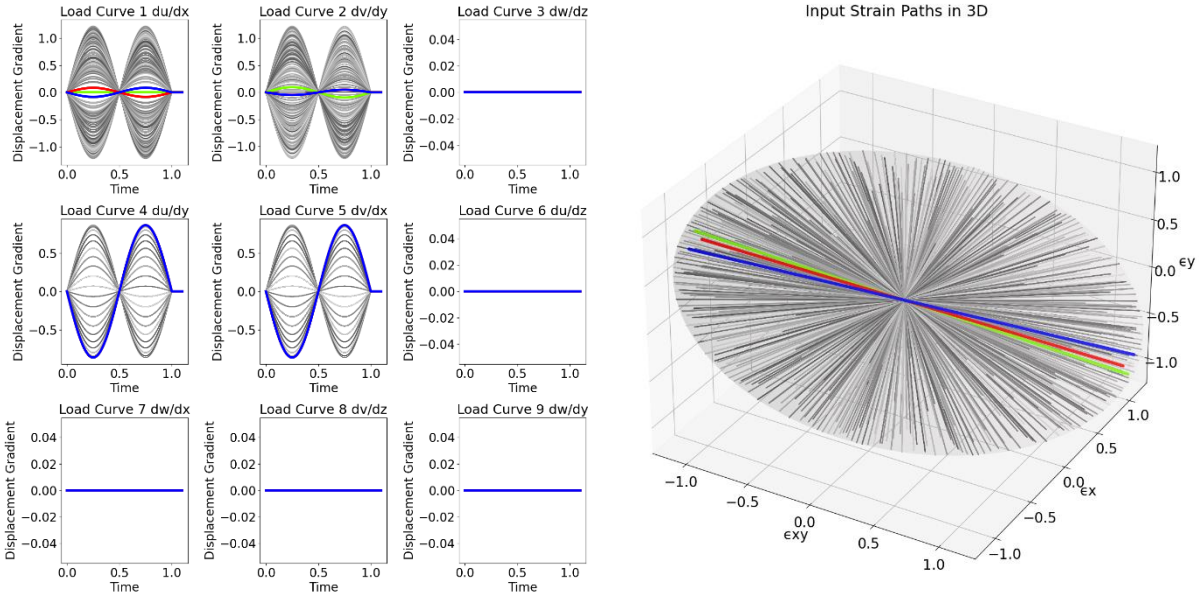


Fig. 4: Sinusoidal Plane Strain Paths with Uniform Distribution (left), Paths shown in the Ellipsoid (right)

4.4 Random Strain Paths

This kind of load curve is implemented only for plane strain conditions. Here, the user decides a number of random points m to be picked from the beginning till end of the simulation. These points are picked in random in the whole sphere/ellipsoid space and the final point is decided as described earlier which satisfies Eq. 2. The increments are linear up to this particular point and then switches directions until all m random switches are considered. The maximum value of a strain component is always within the range (-r, r), where r is the radius of the sphere. Fig. 5 illustrates such random strain paths with $m = 5$. It is observed that random paths train the neural network against extreme loading conditions better and thereby makes the material model more resilient.

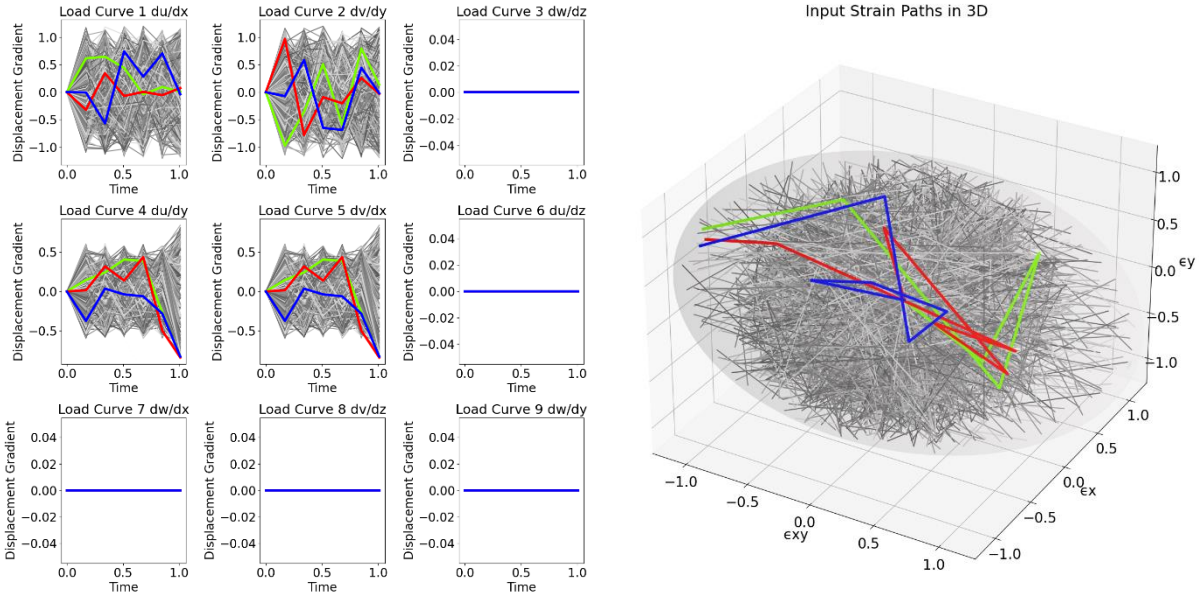


Fig. 5: Random Strain Paths with Uniform Distribution (left), Paths shown in the Ellipsoid (right)

4.5 Random Walking Strain Paths

Another way of including randomness to the data was choosing the increments randomly within the sphere at every time increment instead of considering m points. This resembles like a person taking a walk till the desired location but constantly changing direction at every step taken. Fig. 6 shows such 500 random walking paths.

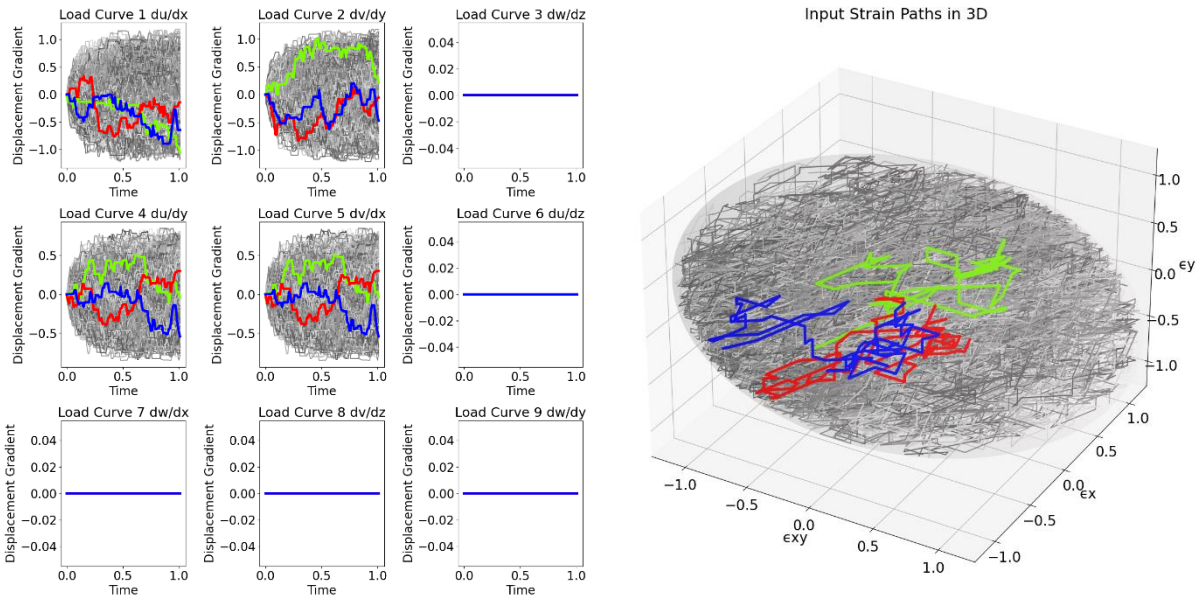


Fig. 6: Random Walking Paths (left), Paths shown in the Ellipsoid (right)

It is to note that the application of focus is a 2D case and hence many load cases were restricted to plane strain type. One could easily extend the idea to a more general 3D loading case and also capture plane strain conditions on other planes (YZ and ZX). Also, it is not necessary to include only sinusoidal curves for non-linearity. It is also possible to choose random or fixed points in the strain space and use B-Splines or other non-linear curves and try to fit data accordingly.

5 Automation of IMMD

Python was used to create a framework which creates the necessary keyword files for an IMMD simulation, run the simulation and also post-process the data at the end generating desired plots. The user only provides an XML file for the current run of the batch in IMMD. It contains various information, e.g. type of load curve, end time, angle increment, number of time steps for strain path generation, appropriate scaling factors for the load curves. This simplifies the switch between small and large deformations with the existing strain paths generated and makes the data management more flexible. Once the input XML file is filled and executed, the necessary keyword files are created. The *qd.cae.dyna* library was used to create keyword files which not only include the load curves but also other keywords required describing element type, material cards and more. This is followed by running the IMMD in LS-DYNA using the *subprocess* library in Python. Due to the batch mode feature, we can directly write the stress, strain, history variables and current time to a Parquet file using the *pyarrow* library. The Parquet format is used as it is convenient to read parquet files for post-processing in Python, filtering data as per user-demands and the file sizes are very small for such huge amounts of data compared to other file storage types available. Plots are automatically created to visualize the data and saved in the mentioned locations. Thus, the whole process of data generation is automated and the user can execute and combine datasets for different loading scenarios. An invocation of LS-DYNA and thus the IMMD is however needed for each strain path, where initialization and licensing take most of the execution time. As a remedy LS-DYNA is invoked in parallel and can also be executed on a remote cluster. Alternatives to this would require an adaptation of IMMD within LS-DYNA directly which will be outlined in our summary. Once the training data set is ready, a neural network can be trained to develop a MLMM. Fig. 7 shows the load curves converted to six strain components and corresponding stress results obtained from IMMD for the chosen `*MAT_024` material. This example is simulated using shell elements

and hence a z-direction strain is observed even for a plane strain case, there by maintaining the stresses in the z-direction as zero.

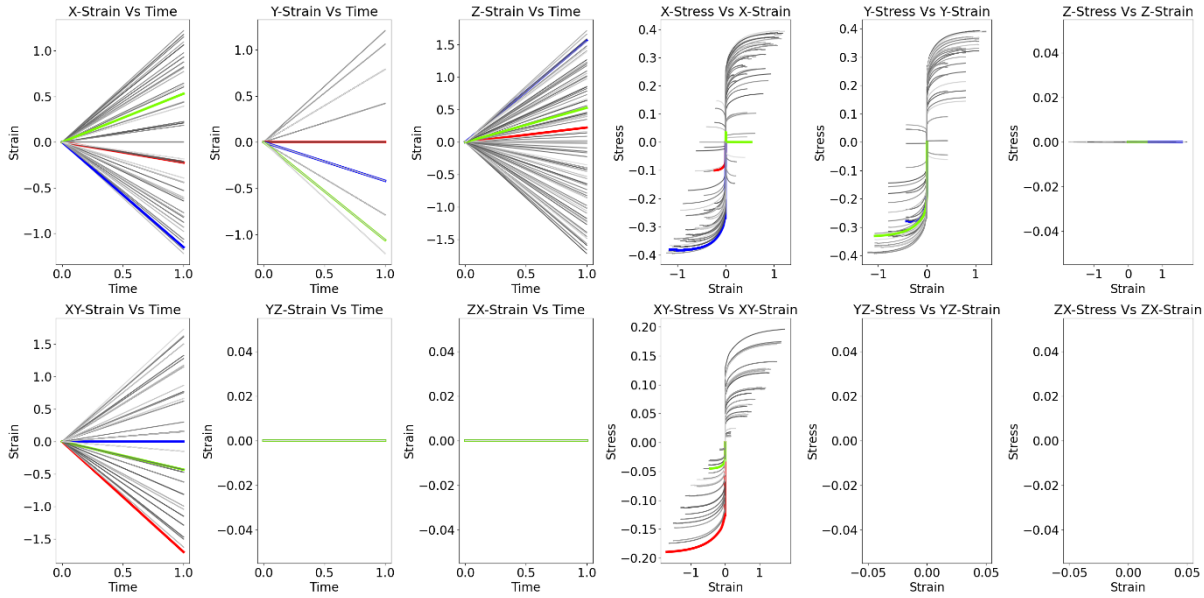


Fig. 7: Results from IMMD – Strain Vs Time (Left), Stress Vs Strain (Right)

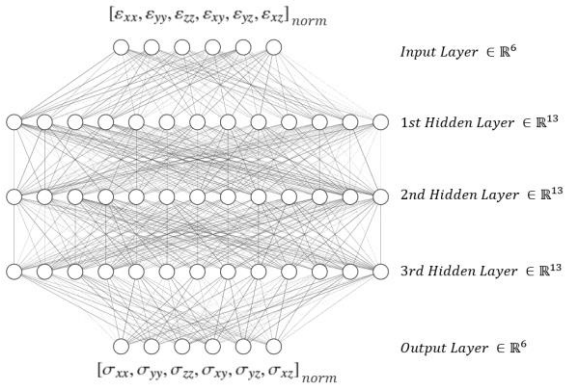
6 Application

The focus of this paper is the automation of IMMD as a programmable interface to generate stress-strain training data from existing (commercial) material models. Usage of such training data is subject to a journal paper currently under review. For completeness we will give two brief examples on how IMMD-data can be used to formulate a machine learning model.

The models are trained using TensorFlow/Keras within Python and are then transferred to LS-DYNA using our own user-material subroutine. Since user-materials in LS-DYNA are to be implemented in Fortran, this gap was overcome by implementing two approaches: One mode is saving the TensorFlow model to an ASCII-File containing network architecture, weights, biases and normalization parameters. This is read dynamically upon initialization of the solver and the neural network is then available for prediction and even further training within Fortran. Recompilation is thus not needed and the user-material generalizes for any new machine learning model. The direct implementation in Fortran makes this approach very fast, but special or new features from Keras which shall be used for the models must be re-implemented in the Fortran library. Using the alternative socket-mode of our user-material, a python listener is started with the beginning of a new simulation to which our user-material connects. Within the user-material inputs are then sent during every timestep trough this socket and prediction is done in Python with any library, as desired. Predicted stresses and histories are then sent back to the MLMM, where it is fed back into the finite element cycle. While this bypasses the need of transferring models from different languages and reimplementation of new routines from Python to Fortran, this approach is slower. However, for academic research this is an important feature and provides a tool for prototyping. The implementation thus provides a scalable and dynamic interface for machine learning models in a commercial finite element code like LS-DYNA.

6.1 Hypoelastic Machine Learning based Material Model

To show the basic capability of MLMMs, linear monotonic strain paths without unloading were used to generate stress-strain pairs to different equivalent strains using our automation of IMMD. A baseline neural network was setup after a brief hyperparameter optimization, as shown in Fig. . Input into the network are normalized total strains and after passing through three hidden layers the output layers give the normalized total stresses and after rescaling to actual stresses are returned by the material subroutine.



Hyperparameter	
Hidden layers	3
Neurons/layer	13
Activation function	tanh
Batch size	200
Loss function	MSE
Optimizer	Adam

Fig. 8: Architecture of the neural network and corresponding hyperparameters

With this model, plasticity won't be capturable, thus we obtain a hypoelastic model out of *MAT_024, without representing real plasticity. For this purpose, either a different network architecture would be required, which will be shown briefly in the next use-case – or another set of inputs, by which plasticity can be manifested within the network (e.g. as proposed by [1]).
After training in TensorFlow/Keras and transfer to our user-material-interface for MLMMs, the trained model was evaluated using different test-cases. It is important to note that these present real results of an actual simulation and not pre-defined strain-paths. Here, the next strain increment depends on the correct prediction of the MLMM. Fig. 9 shows good agreement on a patch of elements subjected to a tensile load. Within the elastic region the error is a maximum misprediction 0.05 N compared to the force calculated using *MAT_024, which grows to a maximum of 0.3 N when we enter the plastic/hypoelastic region. Overall the forces have an average deviation of $2.958 \cdot 10^{-2}$ N, thus describing force and energy in good agreement with the original model – on new strain paths. Even on coupon-level in an open-hole tension specimen, the machine learning surrogate shows similar von Mises-stress in terms of distribution and magnitude.

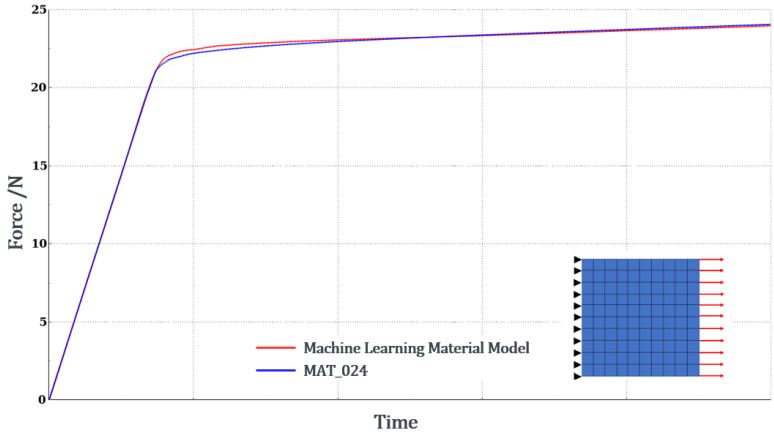


Fig. 9: Response of MLMM compared with original MAT_024 on a patch of elements in tension

6.2 Machine-Learning based Material Model for Polymers

To highlight the massive capability of the use of the training data from the automated IMMD, one more example given here is using **MAT_SAMP-1* as a base to generate stress-strain pairs for an injection molding polymer using different plasticity curves for tension, compression and shear. Again, paths to different effective strains with different progressions, linear and random, were used to train an artificial network containing gated recurrent units (GRUs) within a series of feed-forward hidden layers, i.e. a recurrent neural network (RNN). These units are capable of keeping a history of past outputs of each neuron. With this technique the network is able to capture true plasticity with loading and unloading. An exemplary stress response from the original **MAT_SAMP-1* model is compared with the trained MLMM on IMMD data as a surrogate for the former in Fig. 10. Three changes in loading are made, where each loading direction enters the plastic domain. The RNN MLMM follows this path, which was not part of the training, correctly and predicts plastification correctly even after unloading and re-entering the plasticity.

The use of recurrent units requires several special implementations to keep memory to a minimum within the FE-analysis and has several obstacles when strain increments are outside of the training domain – i.e. when the time step is very small or loading is at high speeds. This is beyond the scope of this publication focused on training data trough IMMD and will be part of a future publication.

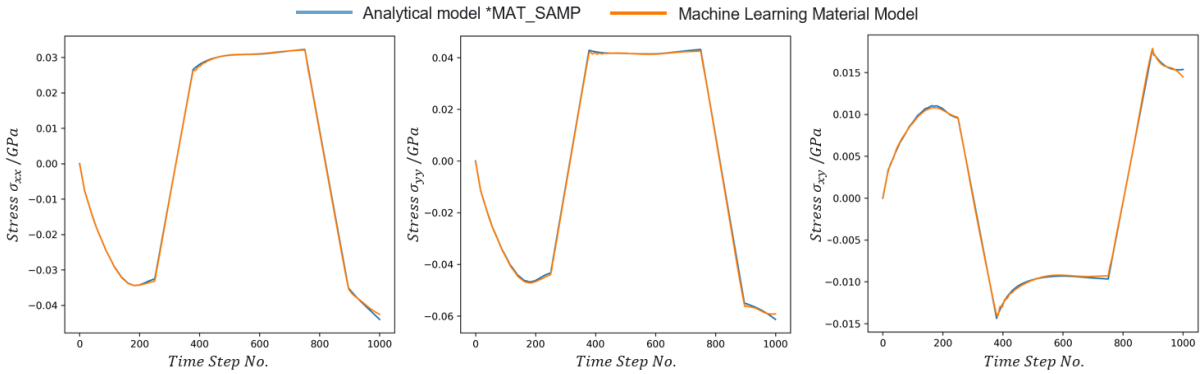


Fig. 10: Stress progression over time step for original MAT_SAMP and MLMM surrogate for one path

7 Summary

This method substitutes setting up simulations for of a variety of load cases without having the trouble of appropriate boundary condition definition, it enables access to closed-source models and is easier to test than re-implementations of analytical models in own subroutines for different loading scenarios. But the main benefit of generating training data with this strategy, compared to e.g. excessive number of single elements under different loading conditions or stress-strain responses from component simulations are is the direct access to the unbiased stress response from the actual material model without interference of element formulation, hourglass control, time-stepping schemes, post-processing and more. This training data will contain the cleanest learning experience from closed-source material models possible, whereas other ways will produce a prediction from learning data with additional noise/bias before it is biased again by effects within an FE-analysis.
On the downside, an input keyword file for each strain path has to be generated and LS-DYNA with IMMD is invoked for it. Depending on the desired coverage of the strain space, this means large amounts of data and more importantly much time for initialization and licensing. A change to IMMD within the solver would alleviate this drawback of the method, e.g. if IMMD would accept multiple paths at once or recognize the `*CASE` keyword. But then the IMMD is a fairly old, unknown tool within LS-DYNA and as stated before only available in specific trunks of the development. Nevertheless, it would be beneficial to add a few more features to this veteran in order to make it faster to generate training data for modern machine learning material models, when using our automation.
However, with the presented automation of the Interactive Material Model Driver in LS-DYNA we are now able to extract the stress-strain response of any material model within the solver. This paper highlighted five different implementations of defining strain paths to obtain different quality and type of training data. With this data several MLMM were trained, two of which were described briefly to show the feasibility on learning using the data generated with our method. Further extension with other strategies of strain path generation are easily implementable and research effort can be dedicated towards consolidating the best machine learning approach on this training data – especially for new materials beyond the training experience.

This work is funded by the *Federal Ministry for Economic Affairs and Energy of Germany* for project *DigiBody* under identification 19I19003C and project AIMM under identification 19I20024E.

8 Literature

[1] T. Palau, A. Kuhn, S. Nogales, H. Böhm and A. Rauh, “A neural network based elasto-plasticity material model,” *ECCOMAS 2012 - European Congress on Computational Methods in Applied Sciences and Engineering,* pp. 8861-8870, 01 2012.

[2] J. Ghaboussi, D.A.Pecknold, M.Zhang and R.M.Haj-Ali, “Autoprogressive training of neural network constitutive models,” *International Journal for Numerical Methods in Engineering Vol. 41 No. 1,* pp. 105-126, 1998.

[3] D. Huang, J. N. Fuhg, C. Weißenfels and P. Wriggers, “A machine learning based plasticity model using proper orthogonal decomposition,” *Computer Methods in Applied Mechanics and Engineering, Vol. 365,* 2020.

[4] T.Furukawa and M.Hoffman, “Accurate Cyclic Plastic Analysis Using a Neural Network Material Model,” *Engineering Analysis with Boundary Elements, Vol. 28, No. 3,* pp. 195-204, 2004.

[5] M. B. Gorji, M. Mozaffar, J. N. Heidenreich, J. Cao and D. Mohr, “On the potential of recurrent neural networks for modeling path dependent plasticity,” *Journal of the Mechanics and Physics of Solids, Vol. 143,* 2020.

[6] L. Wu, V. D. Nguyen, N. G. Kilingar and L. Noels, “A recurrent neural network-accelerated multi-scale model for elasto-plastic heterogeneous materials subjected to random cyclic and non-proportional loading paths;,” *Computer Methods in Applied Mechanics and Engineering, Vol. 369,* 2020.

[7] D. P. Jang, P. Fazily and J. W. Yoon, “Machine learning-based constitutive model for J2- plasticity,” *International Journal of Plasticity, Vol. 138,* 2021.

[8] Livermore Software Technology, LS-DYNA Keyword User's Manual - Volume 1, LS-DYNA R11 r:13109, 2020.

[9] L. Schwer, “Interactive Material Model Driver: a Little Known LS-DYNA Capability,” *Fea Information INC. Worldwide News for March 2008,* pp. 9-16, 1 2008.

[10] M. Deserno, “How to generate equidistributed points on the surface of a sphere,” *Max-Planck-Institut für Polymerforschung, Mainz,* p. 1, 09 2004.