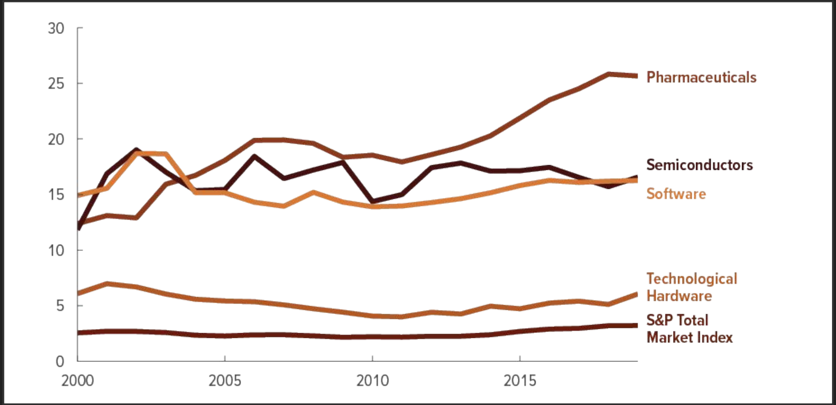
Pharmaceutical Stress vs. Strain Simulations Using Machine Learning

Juan Mendoza

Motivation

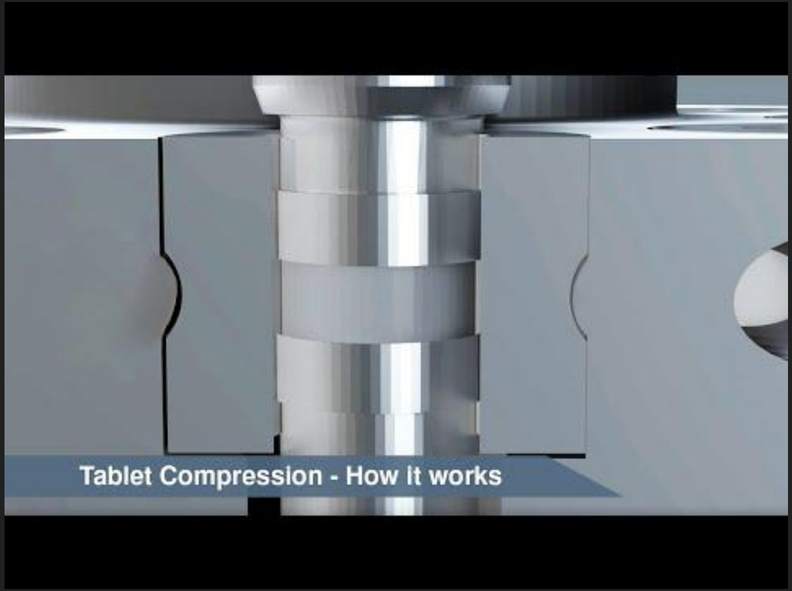
- Why a pharmaceutical simulation?
  - “In 2019, the pharmaceutical industry spent $83 billion dollars on R&D…about 10 times what the industry spent per year in the 1980s” (Congressional Budget Office).
  - Pre-pandemic!
- Why AI?
  - Endless applications
  - Can improve research in every field
  - Can improve everyday life



Congressional Budget Office, using data from Bloomberg, limited to U.S. firms as identified by Aswath Damodaran, “Data: Breakdown” (accessed January 13, 2020), https://tinyurl.com/yd5hq4t6. See www.cbo.gov/publication/57025#data.

Problem Statement

- What is the compaction behavior of pharmaceutical powders given their properties?
- What is the strain vs. stress relationship?
- Applications
  - Pharmaceutical manufacturing of tablets
  - Metal forming



“Tablet Compression - How it works animation”.

Related Work

Simulating Compaction Behavior with the Discrete Element Method

- Pros
  - No analytical solution exists for modeling compaction behavior for more than two particles
  - Expanded our knowledge of how these systems behave
- Cons
  - The behavior of each particle is calculated by Newton’s Second Law
  - Slow and computationally expensive
  - Requires high performance computing


ICE Publishing is part of the Institution of Civil Engineers

ICE Virtual Library
essential engineering knowledge

Home | Journals | Books | Subjects | Information | News | About

Home › Géotechnique › List of Issues › Volume 29, Issue 1 › A discrete numerical model for granular assemblies

Géotechnique
ISSN 0016-8505 | E-ISSN 1751-7656
Volume 29 Issue 1, March 1979, pp. 47-65

A discrete numerical model for granular assemblies

Authors: P. A. Cundall*, , and O. D. L. Strack†,

Author Affiliations

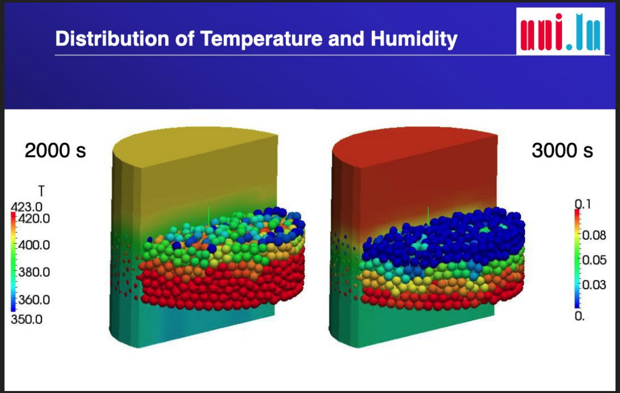
https://doi.org/10.1680/geot.1979.29.1.47
Published Online: May 25, 2015
Keywords: geotechnical engineering

Key: Open access content · Subscribed content · Free content · Trial content

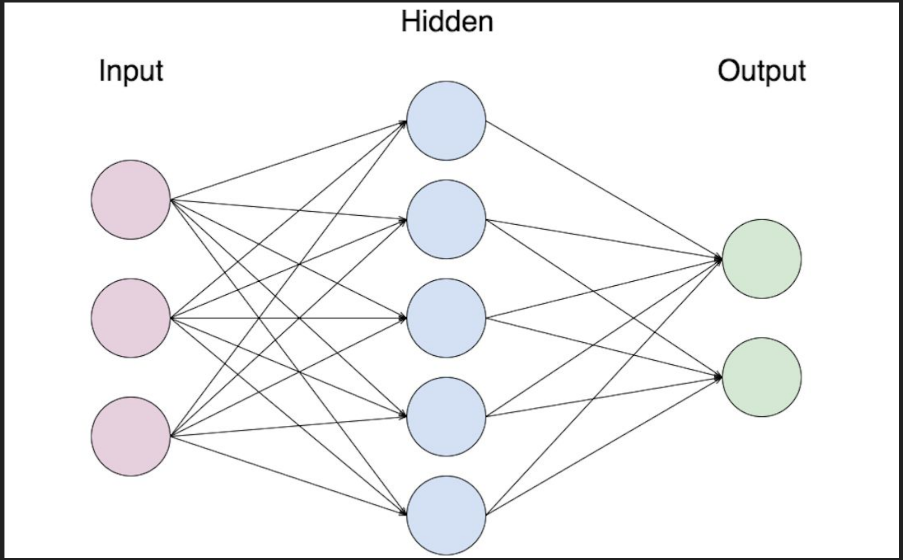
Abstract

The distinct element method is a numerical model capable of describing the mechanical behaviour of assemblies of discs and spheres. The method is based on the use of an explicit numerical scheme in which the interaction of the particles is monitored contact by contact and the motion of the particles modelled particle by particle. The main features of the distinct element method are described. The method is validated by comparing force vector plots obtained from the computer program BALL with the corresponding plots obtained from a photoelastic analysis. The photoelastic analysis used for the comparison is the one applied to an assembly of discs by De Josselin de Jong and Verruijt (1969). The force vector diagrams obtained numerically closely resemble those obtained photoelastically. It is concluded from this comparison that the distinct element method and the program BALL are valid tools for research into the behaviour of granular assemblies.

La méthode des éléments distincts est un modèle numérique capable de décrire le comportement mécanique de l'assemblage de



Contributions



Proposal

- We propose a new approach that speeds up the process to Θ (1) time complexity
- Relies on machine learning and particularly neural networks
- Build a dedicated dataset
- Evaluate our approach to show that it is efficient and speedy

Outline

- Background Material
- Our Approach
- Evaluation
- Conclusion

Background Material

Strain vs Stress

DEM simulations

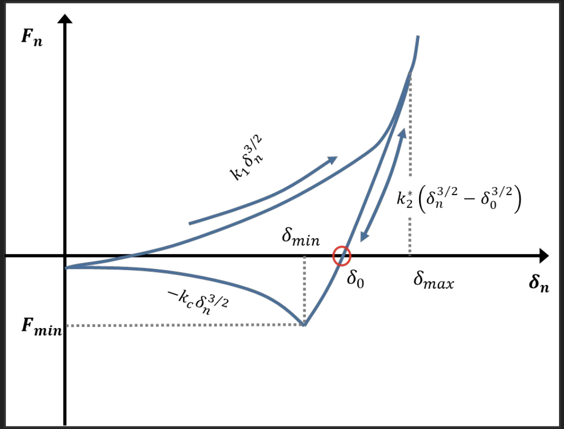
- Given some properties, we can simulate the relationship between strain and stress over time

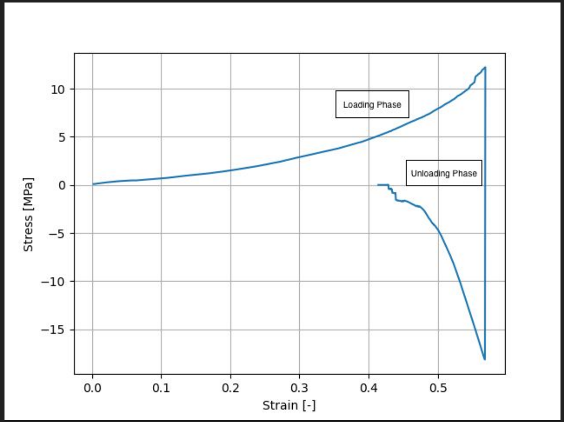
Inputs

- Properties of the material (Ex. Loading Plasticity Value, Unloading Plasticity, Coefficient Adhesion)

Outputs

- A function of strain and stress over time





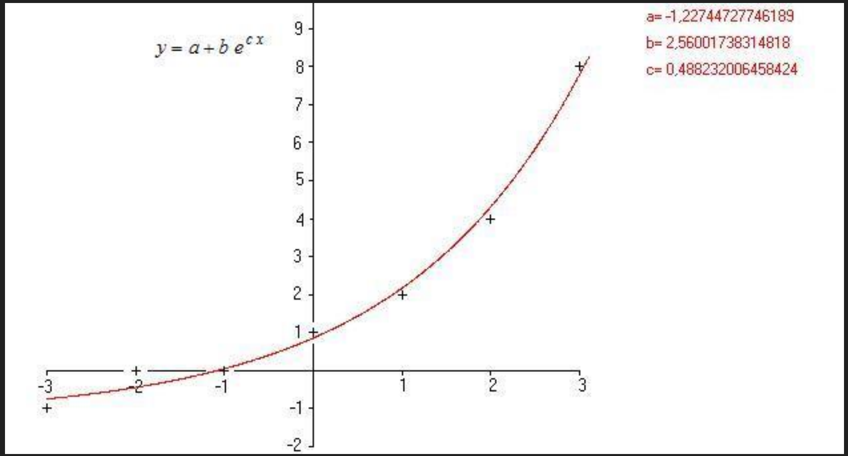
Exponential Regression

What is it?

- Statistical method to find an exponential relationship between two variables

Optimization

- Try to find coefficients that minimize the error between the curve and the points



Neural Networks

- Subset of machine learning
- Typically used for supervised learning
  - Inputs and outputs are given to infer a function between them
- Neural networks learn through backpropagation
  - A form of gradient descent
  - The network adjusts its weights based on the error between its predictions and the training outputs
  - These adjustments can make its predictions more accurate

Regression with a deep neural network (DNN)

In the previous section, you implemented two linear models for single and multiple inputs.

Here, you will implement single-input and multiple-input DNN models.

The code is basically the same except the model is expanded to include some "hidden" non-linear layers. The name "hidden" here just means not directly connected to the inputs or outputs.

These models will contain a few more layers than the linear model:

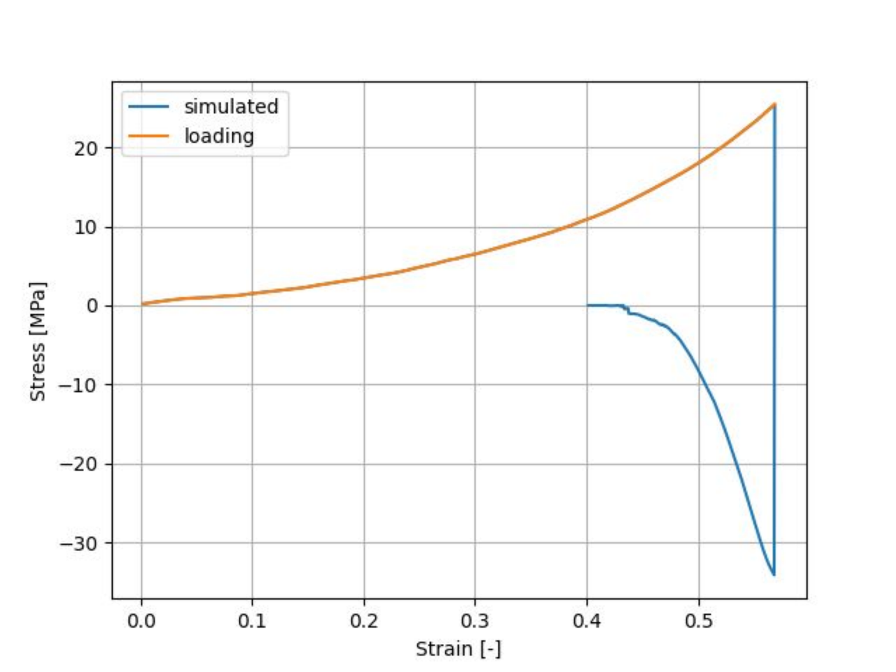
- The normalization layer, as before (with `horsepower_normalizer` for a single-input model and `normalizer` for a multiple-input model).
- Two hidden, non-linear, `Dense` layers with the ReLU (`relu`) activation function nonlinearity.
- A linear `Dense` single-output layer.

Both models will use the same training procedure so the `compile` method is included in the `build_and_compile_model` function below.

```
def build_and_compile_model(norm):
  model = keras.Sequential([
      norm,
      layers.Dense(64, activation='relu'),
      layers.Dense(64, activation='relu'),
      layers.Dense(1)
  ])

  model.compile(loss='mean_absolute_error',
                optimizer=tf.keras.optimizers.Adam(0.001))
  return model
```

Approach

Our Approach

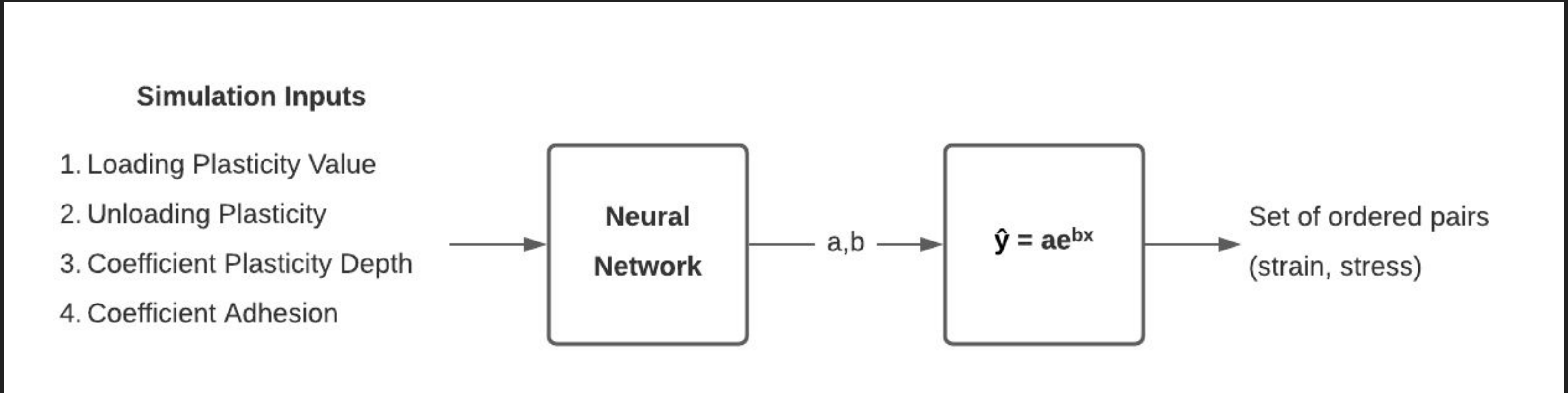


Constraints

- Focus on the loading phase
- Focus on families of inputs that give simulation results that are roughly exponential

Our Approach

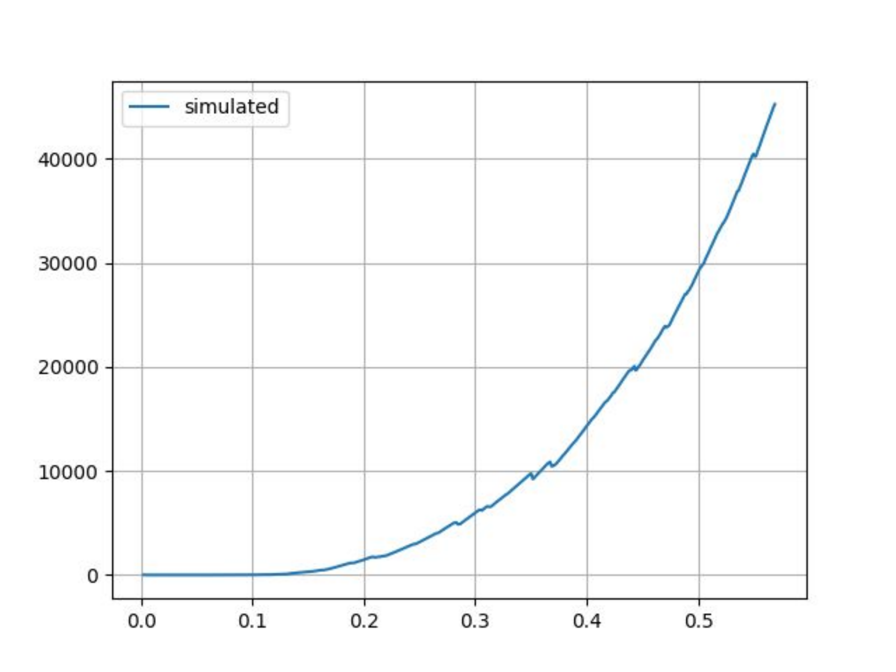
End product



The Training Process

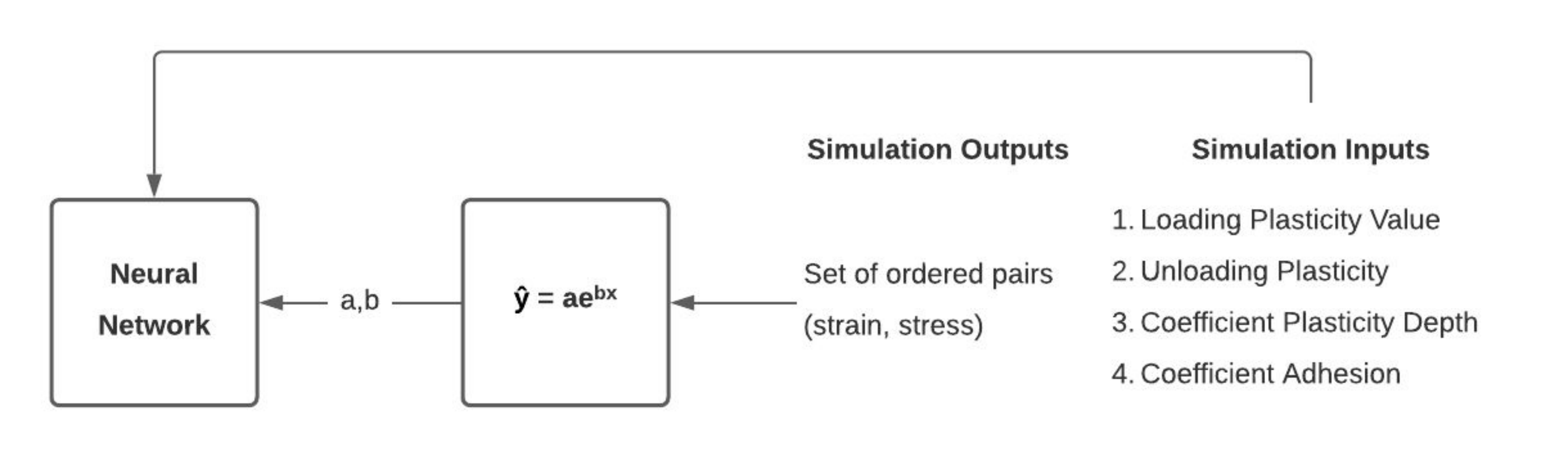
Steps

- Collect simulation samples
- Cut everything after loading phase
- Use exponential regression to get a and b
- Create training dataset



Our Approach

Training Process



Collect Data

Collaborators

- Worked with Kostas Giannis, a PhD student from Technische Universität Braunschweig · Institut für Partikeltechnik - Center of Pharmaceutical Engineering (PVZ)
- Wrote a Python script to generate random input files for the simulation to run on HPC
- Generated random values within a realistic range for the following parameters: Loading Plasticity Value, Unloading Plasticity, Coefficient Plasticity Depth, Coefficient Adhesion

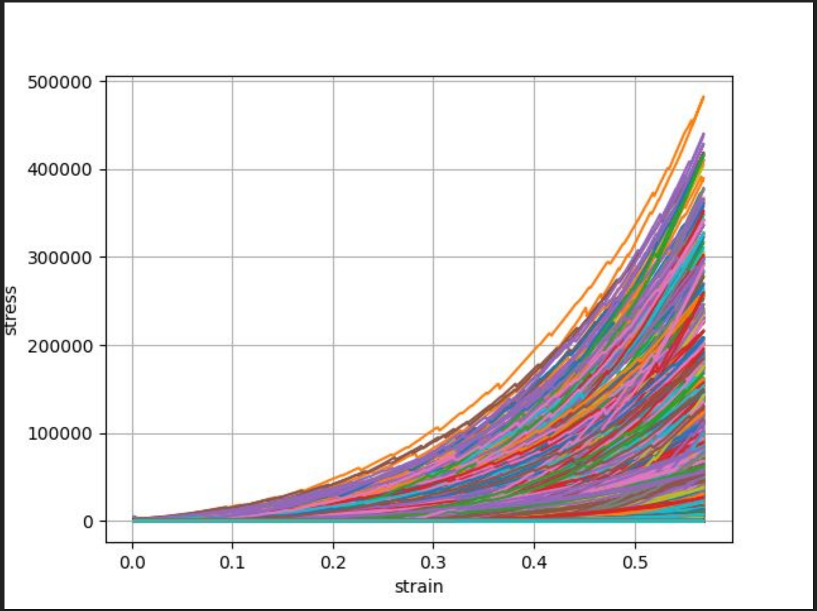
```
# 2000 0.4 1e5 1e5 0 2e4 0e4 0e4 .. 0.5 0.5 0 0 .8. 1546.26490773707 309.252981547414 0 0  154.6 30.9  0.5 0


fix         m1 all property/global youngsModulus peratomtype 2.58e8
fix         m2 all property/global poissonsRatio peratomtype 0.3
#fix             m33 all property/global kn peratomtypepair 2  0.4e8 0.4e8 0.4e8 0.4e8
fix         m3 all property/global LoadingStiffness peratomtypepair 1   28216.9295019
fix         m4 all property/global UnloadingStiffness peratomtypepair 1   12941.9090378
fix             m5 all property/global coefficientPlasticityDepth peratomtypepair 1 0.205465141219
#fix             m44 all property/global kt peratomtypepair 2 100 100 100 100
fix             m6 all property/global gamman peratomtypepair 1 100
fix             m7 all property/global gammat peratomtypepair 1 100
fix         m27 all property/global coefficientRestitution peratomtypepair 1 0.352
fix         m8 all property/global coefficientFriction peratomtypepair 1 0.561
fix         m9 all property/global coefficientAdhesionStiffness peratomtypepair 1 1.69903424655
fix         m10 all property/global pullOffForce peratomtypepair 1 0
fix         m11 all property/global FluidViscosity peratomtypepair 1  0.5
fix         m12 all property/global  coeffFrictionStiffness peratomtypepair 1  0
fix         m13 all property/global  FrictionViscosity peratomtypepair 1 0.01
fix             m144 all property/global coefficientRollingFriction peratomtypepair 1 0.3
#New pair style
# AM: or no history?
#pair_style gran model hertz/stiffness tangential history
pair_style  gran model luding tangential tan_luding  rolling_friction cdt #rolling_friction luding
#rolling_friction luding
pair_coeff  * *

variable dt equal  0.000000001
```

Results

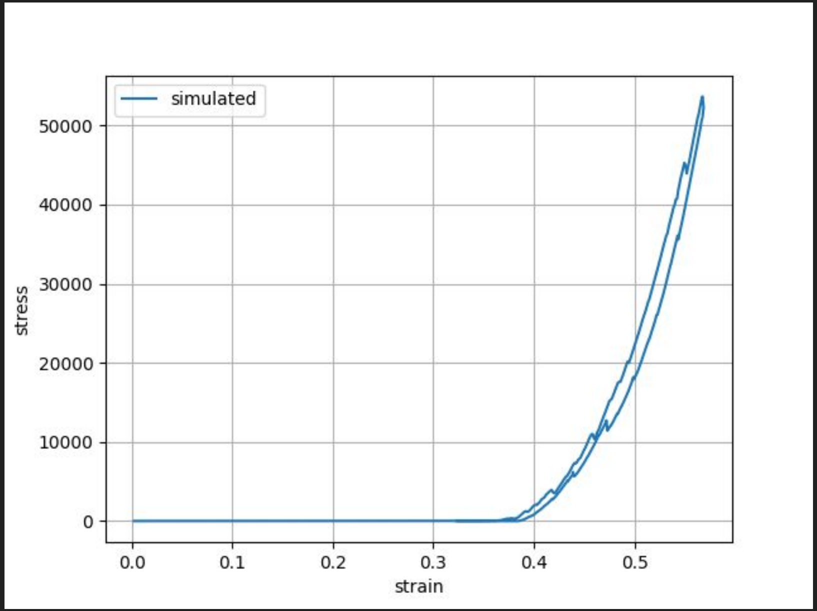
- Gathered a total of 3500 simulation runs

Collect Data

All Stress vs. Strain Simulation Graphs

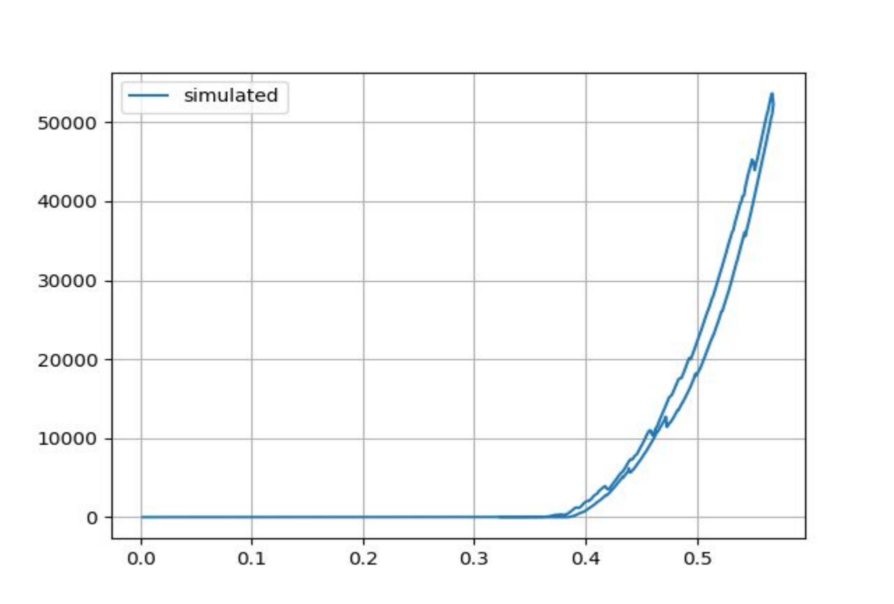


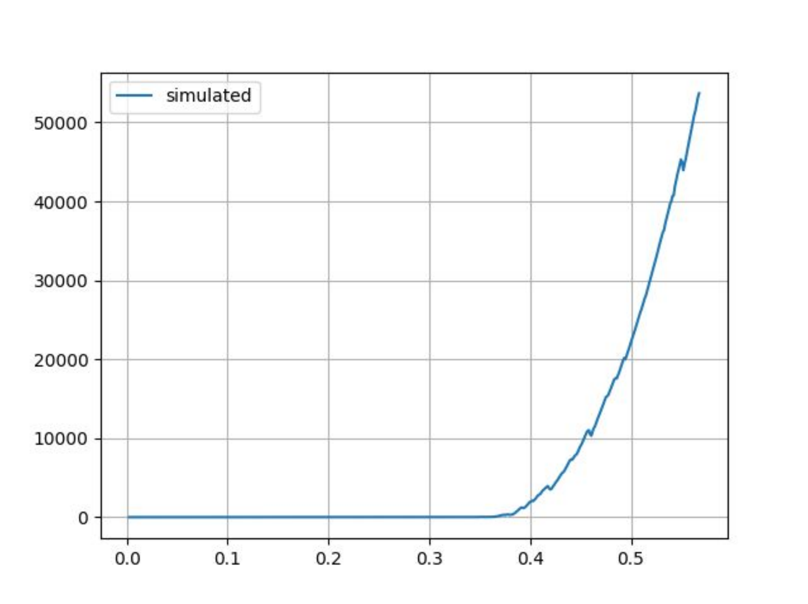
Single Stress vs. Strain Graph



Crop Images

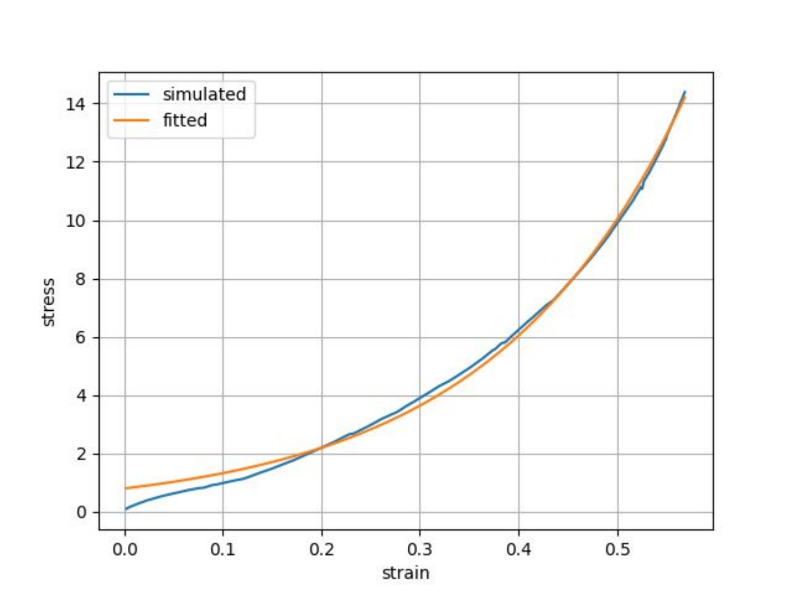
We know at what strain the mass stress will be, but I still just split the graph at the max anyway.

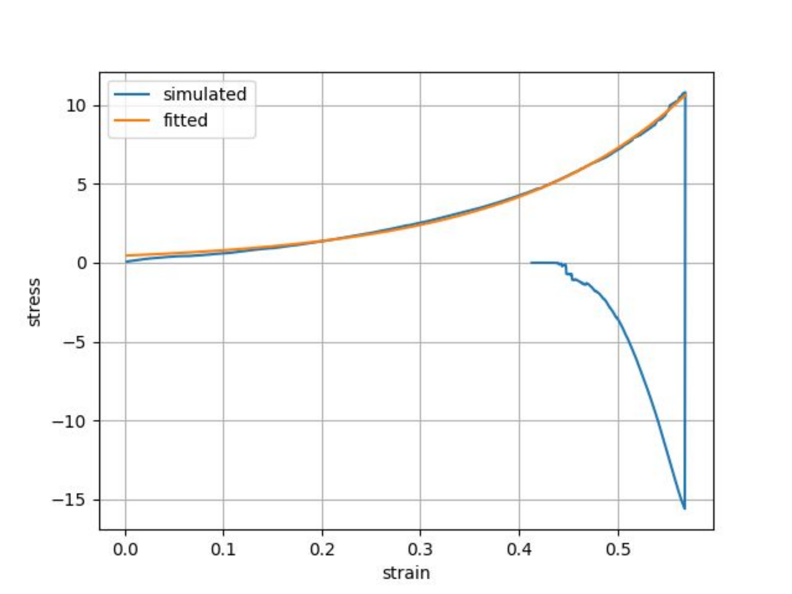




Fit exponential

Used the Python curve_fit function to fit an exponential function and return it's corresponding coefficients.





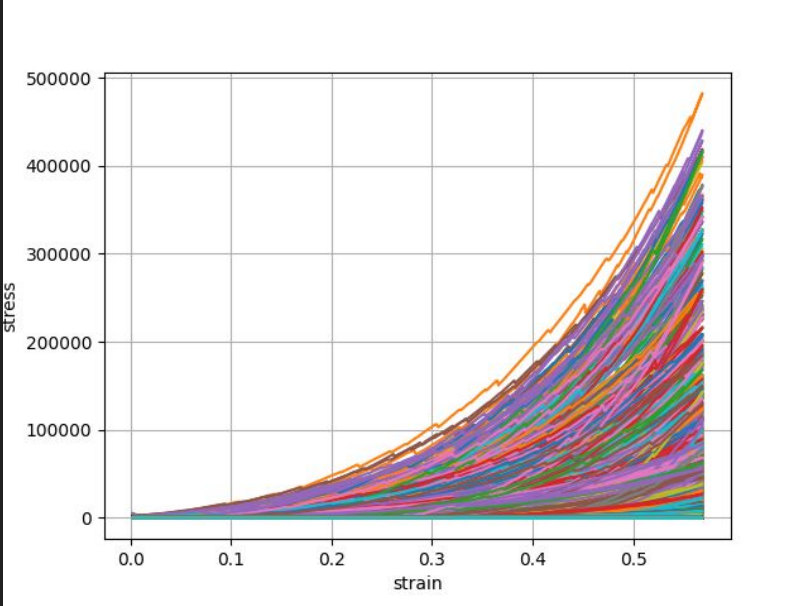
NN Dataset

- Neural Network Training Examples
  - Loading Plasticity Value [N / m^2]
  - Unloading Plasticity [-]
  - Coefficient Plasticity Depth [-]
  - Coefficient Adhesion [-]
  - Exponential Coefficient 1 [-]
  - Exponential Coefficient 2 [-]
- Total number of training examples: 3028

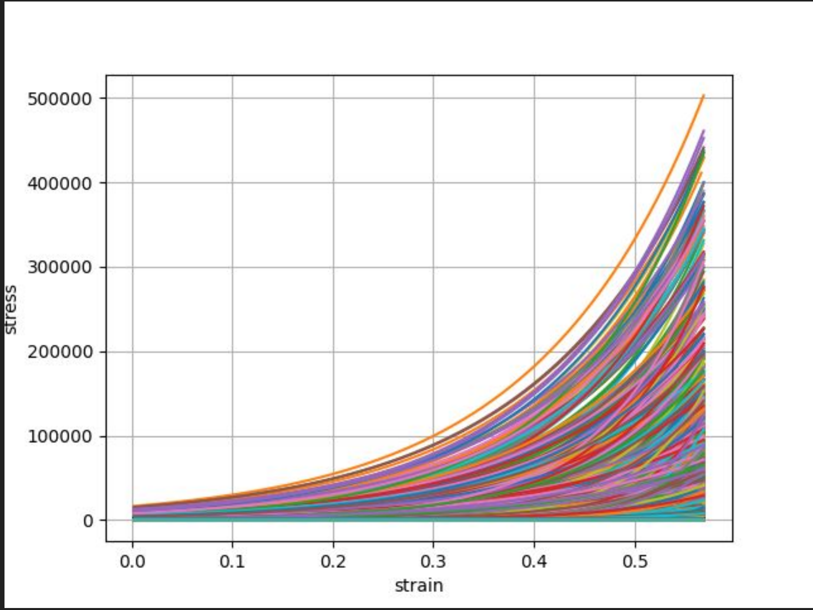
I	Lpv	Up	Cpd	Ca	Out1	Out2
1439	94856857.97	2750.46	0.35	1.47	22.02	12.8
2770	33391.33	3594.62	0.24	1.3	0.17	8.61
1999	5092.52	13542.69	0.14	1.34	0.39	6.75
2000	37621.07	8201.76	0.41	1.06	0.01	14.96
2510	98707.69	18034.11	0.79	1.48	0	5.67
1213	1961458.41	16604.76	0.56	4.02	0	57.43
34	68165456.88	7345.27	0.05	1.53	5588.41	5.89
182	19203427.18	14065.16	0.36	2.16	18.51	13.16
689	79757175.45	10285.03	0.42	1.37	2.43	17.91
2055	31509.24	7725.66	0.42	0.57	0	16.04
2468	40591.29	14170.07	0.26	3.01	0.56	9.23
613	1968707.18	13021.79	0.34	4.8	3.9	11.87
2376	99847.95	2346.25	0.98	1.97	0	5
232	80467763.38	261.33	0.62	1.09	0.46	8.14
1355	22354431.84	5809.06	0.84	0.63	0.37	5.39
814	70362612.32	5100.31	0.51	2.14	0	34.76
1653	81545750.75	657.81	0.61	3.58	0	18.45
1288	18608869.41	13708.41	0.99	2.74	0.35	5.11
1958	61976.04	9948.9	0.85	1.64	0	5.11
2777	92624.25	4529.3	0.2	1.7	1.05	7.89
1298	24919959.78	5554.9	0.55	4.97	0	50.07
2254	53038.52	10344.96	0.47	0.79	0	22.25
1090	85431346.7	18500.18	0.76	2.59	1.39	5.51
3010	93079.34	8294.53	0.36	4.62	0.05	13.16
2757	59441.99	6807.48	0.41	1.99	0	15.69
2678	33370.18	734.09	0.99	3.38	0	4.96
694	8788628.99	11706.75	0.35	1.35	7.74	13.01
9	36086796.12	12128.59	0.83	1.32	0.59	5.44
576	98179095.97	1296.83	0.67	1.84	0.89	6.98
1223	3028413.47	14513.54	0.35	3.75	4.57	12.43
817	57054097.57	17039.12	0.27	3.68	736.21	9.59
2431	35262.39	13723.76	0.56	3.08	0	38.14
442	14572064.11	3349.24	0.48	3.81	0	24.8
2939	71861.62	4520.86	0.14	0.88	1.67	6.9
2811	54021.66	7273.68	0.19	1.83	1.14	7.69
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]

Fun Graphs

All Stress vs. Strain Simulation Graphs

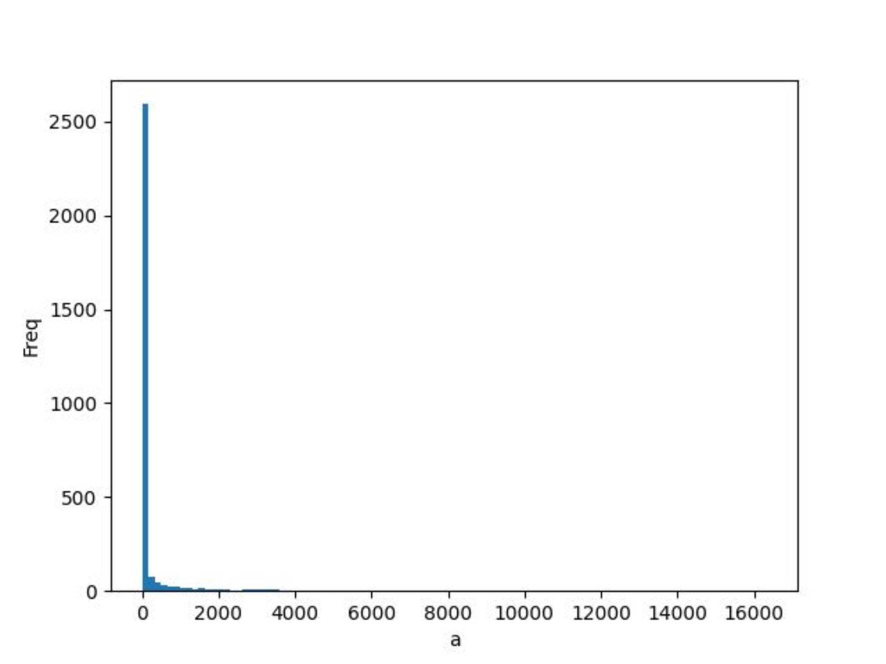


All Exponential Fit Graphs

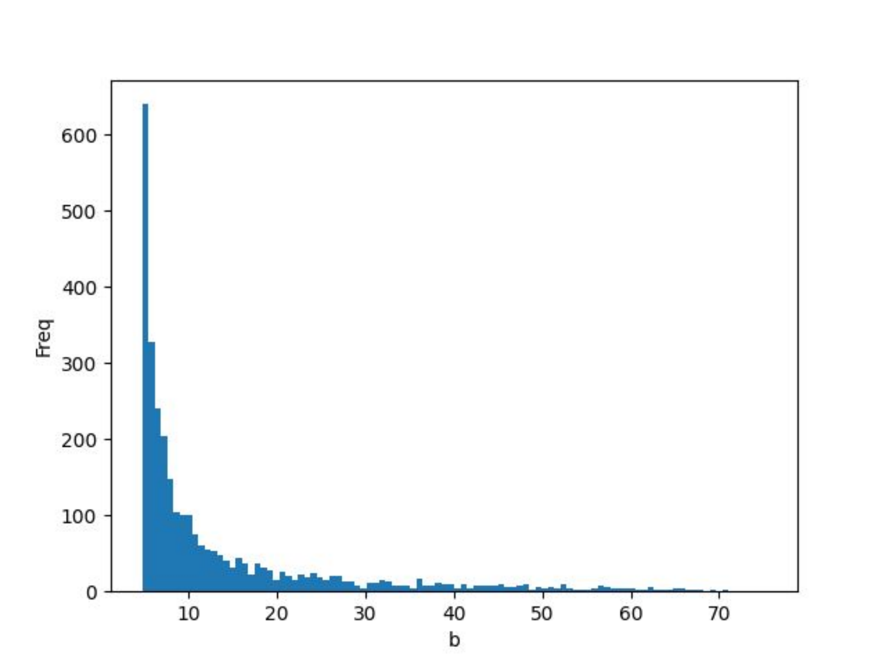


Fun Graphs

Frequency Distribution of a

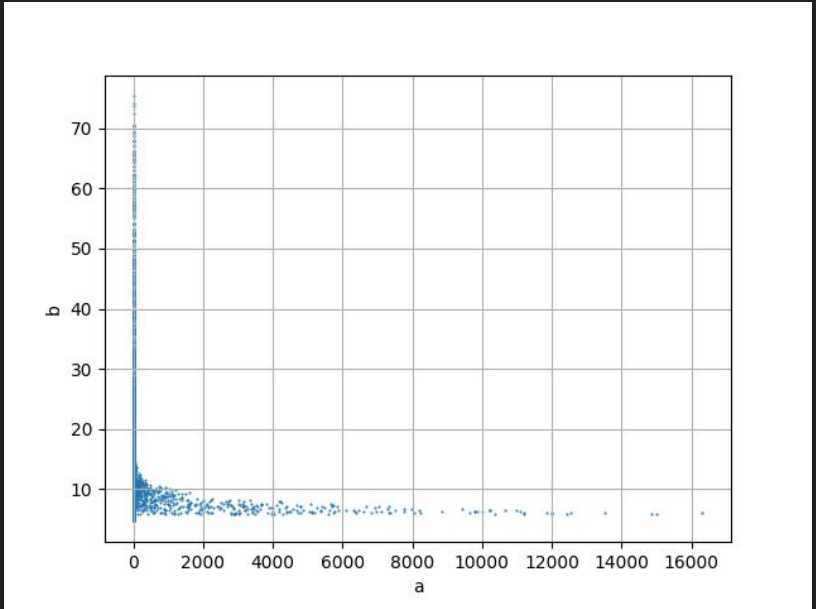


Frequency distribution of b



Fun Graphs

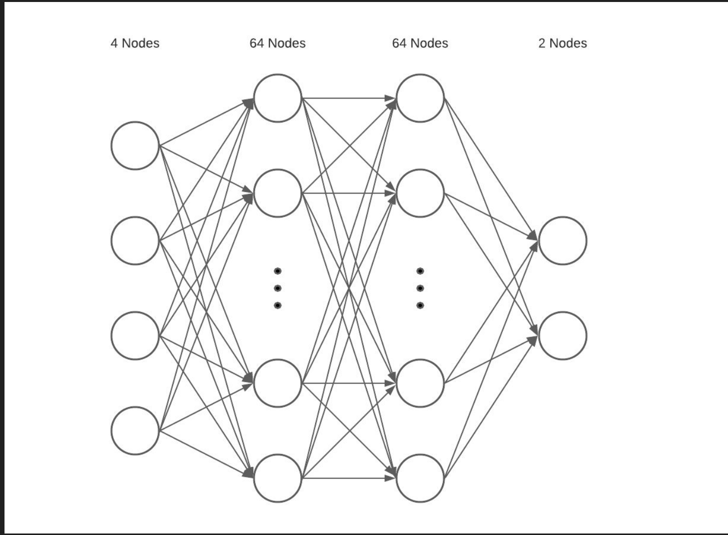
Scatter Plot of a vs. b



Train a NN

Neural Network Architecture

- Layers
  - 4 nodes in input layer
  - 64 nodes in first hidden layer
  - 64 nodes in second hidden layer
  - 2 nodes in output layer
  - Concluded after experimenting with different architectures
- Used Adam (Adaptive Moment Estimation) optimization algorithm
- All ReLU (Rectified Linear Unit) activation functions
- Allocation
  - 45% of these records go to training
  - 25% of these records go to validation
  - 20% of these records go to testing
- Early stopping
  - Stop backpropagation when validation error hits minimum



Evaluation

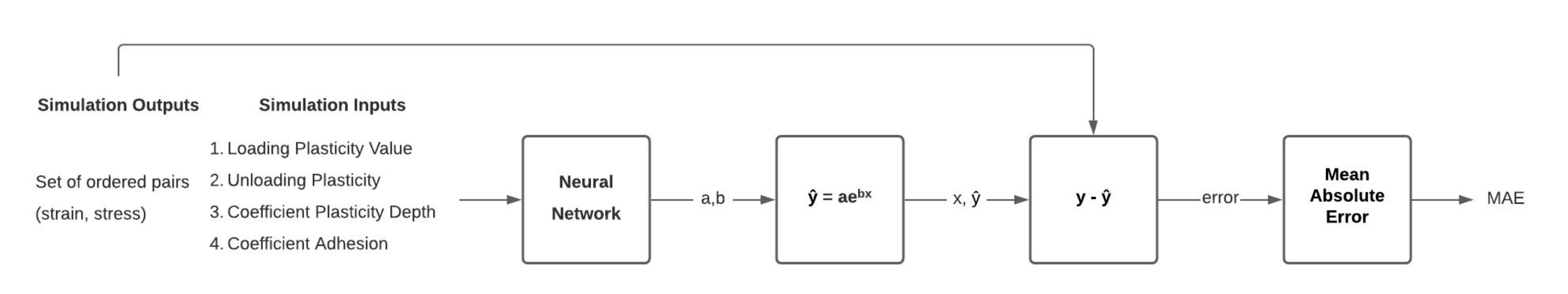
Evaluation Methodology

Evaluate each component

- Exponential fitting
- NN (lambda evaluation results)

Evaluate the whole method

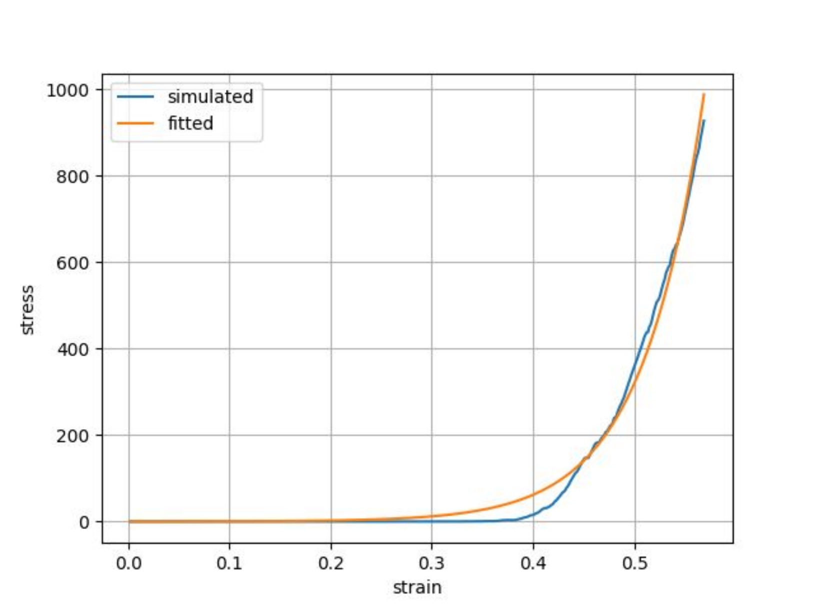
- The whole method



Exponential Fitting

Residuals

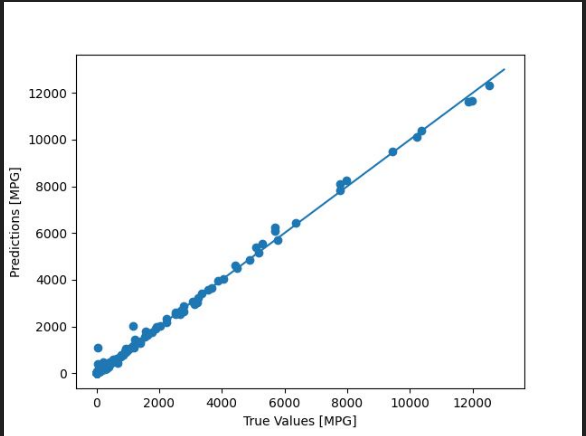
Root Mean Square Error	745.5558711149811
Mean Absolute Error	470.00821986935165
Mean Square Error	555853.5569540182

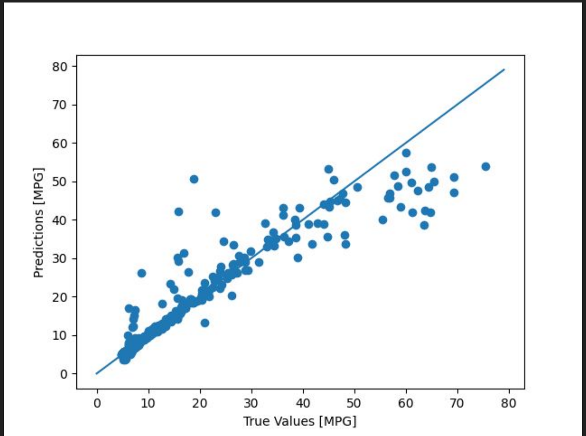


Lambda

a b

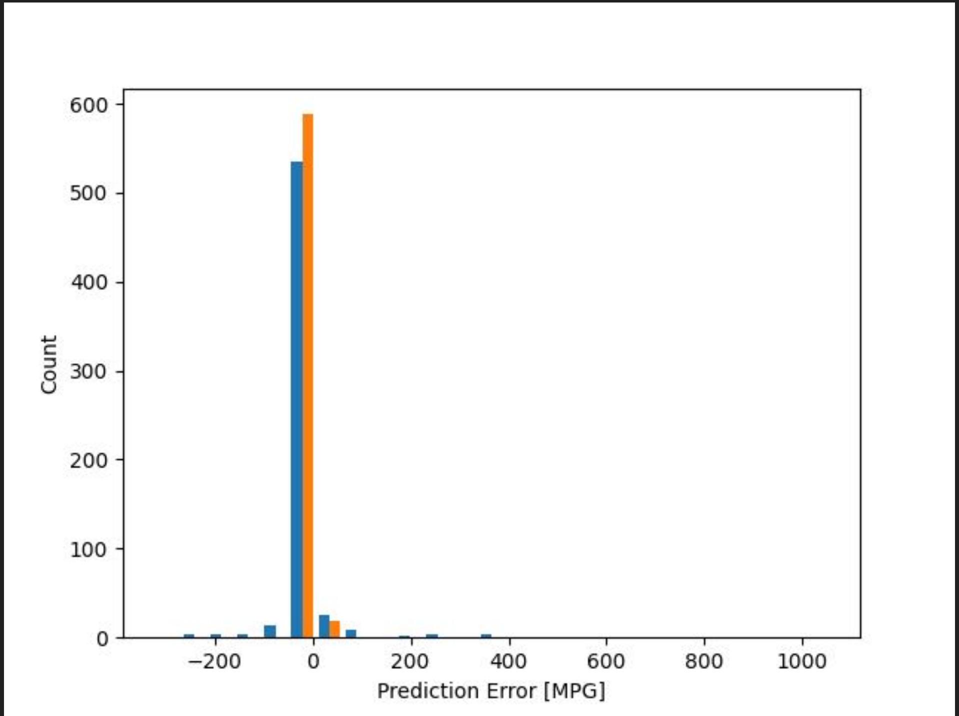
Root Mean Square Error	156.48366958051417	Root Mean Square Error	2.2242482487163016
Mean Absolute Error	110.97848067559875	Mean Absolute Error	1.743066214136309
Mean Square Error	24487.13884538354	Mean Square Error	4.947280271917535





Lambda

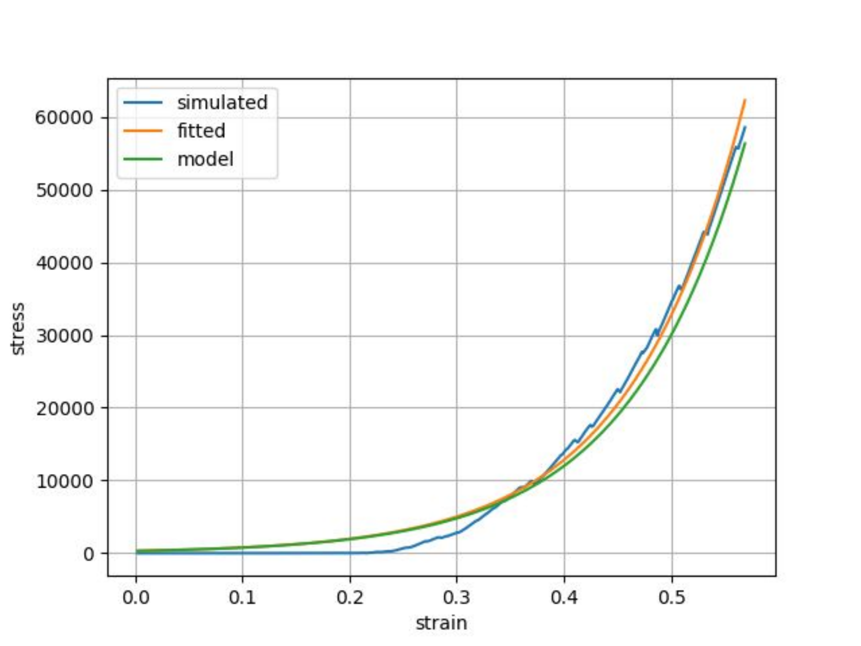
Error Histogram



Methodology

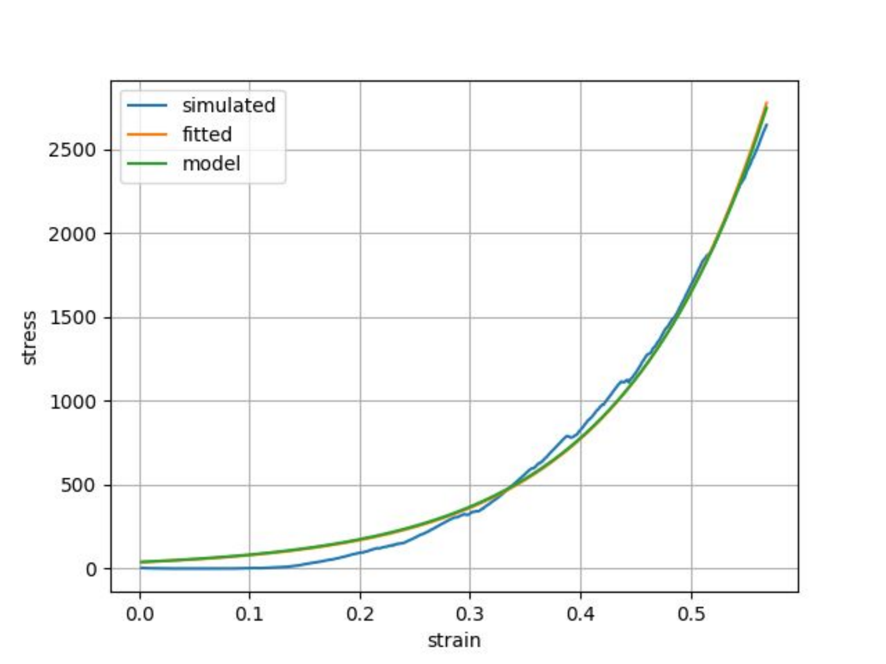
Residuals

Root Mean Square Error	54.87599563758551
Mean Absolute Error	9.76554860746884
Mean Square Error	3011.374897216304

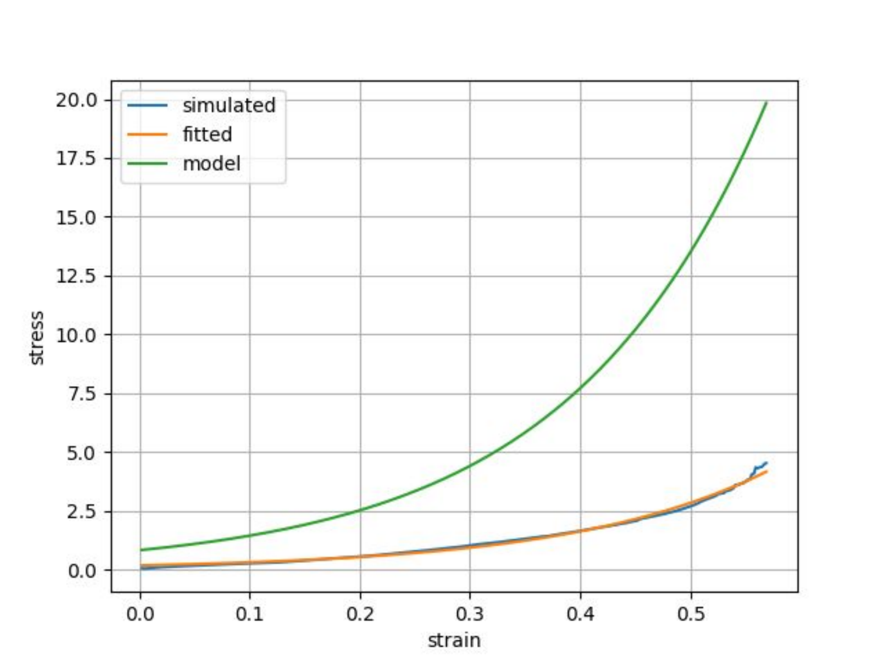


Noteworthy

Good C:



Bad :C



Conclusions

Some great stuff

- Our approach does not require HPC after training to run

Future Work

- Unloading phase
- Could try this process for other simulations, possibly non-exponential

Fin.