Machine Learning models' construction for the load behavior of composite materials in the undamaged zone. Research Report of UGAL for the project Leap-Re D3T4H2S

Contract: 11/2024 din 21.03.2024
Research Study, 2024

https://www.d3t4h2s.ugal.ro/index.php

This study details the research activities conducted by the UGAL research team between Mars 21st, 2024, and December 31st, 2024. The archive ***WorkMLModels.zip*** associated with this report contains the scripts and files supporting the research presented; they can also be used to follow this report easily.

As a partner in this project, UGAL is mainly responsible for two types of tasks:

1. The first task involves constructing Machine Learning (ML) models to predict:
   - The thermo-mechanical behavior of the composite materials used for producing hydrogen tanks (material responses).
   - The tank response: in this case, we use a "full model" of the tank.

(2) The second task will partially cover the optimization task of implementing the *Expert tool for real-time evaluation and optimization of hydrogen storage vessels*. Because optimization is generally a complex task, we have to prepare theoretical and practical "tools," especially in this scientific context, even though optimization will be one of the final stages of our project.

PART I

Machine Learning Models for the Traction Test

This part presents a prospective work concerning the predictions we can make using ML models and data collected from specific material tests (traction tests) or simulations. At the same time, we tested the programming resources and toolboxes that we relied on in our implementations. The informatic resources are provided by the MATLAB 2024 platform and endowed with appropriate toolboxes.

1. General Objective and Available Data

Complex mechanical tests concerning the materials envisaged to be used for the hydrogen tanks' transportation were carried out by our colleagues from S VERTICAL (Mourad NACHTANE) and

ENSTA Bretagne (Prof. Mostapha TARFAOUI). The tensile test, one of the most often used mechanical characterization, was made on an INSTRON 5969 test machine, comprising two tensile bits between which the specimen is placed. Some partial test data was put at our disposal in our attempt to generate a Machine Learning (ML) model for *stress-strain dependence* during the tensile test of *different specimens*.

The data collected refers to 12 specimens loaded within different stress and strain ranges. Linear segments approximated the results, as shown in Figure 1.

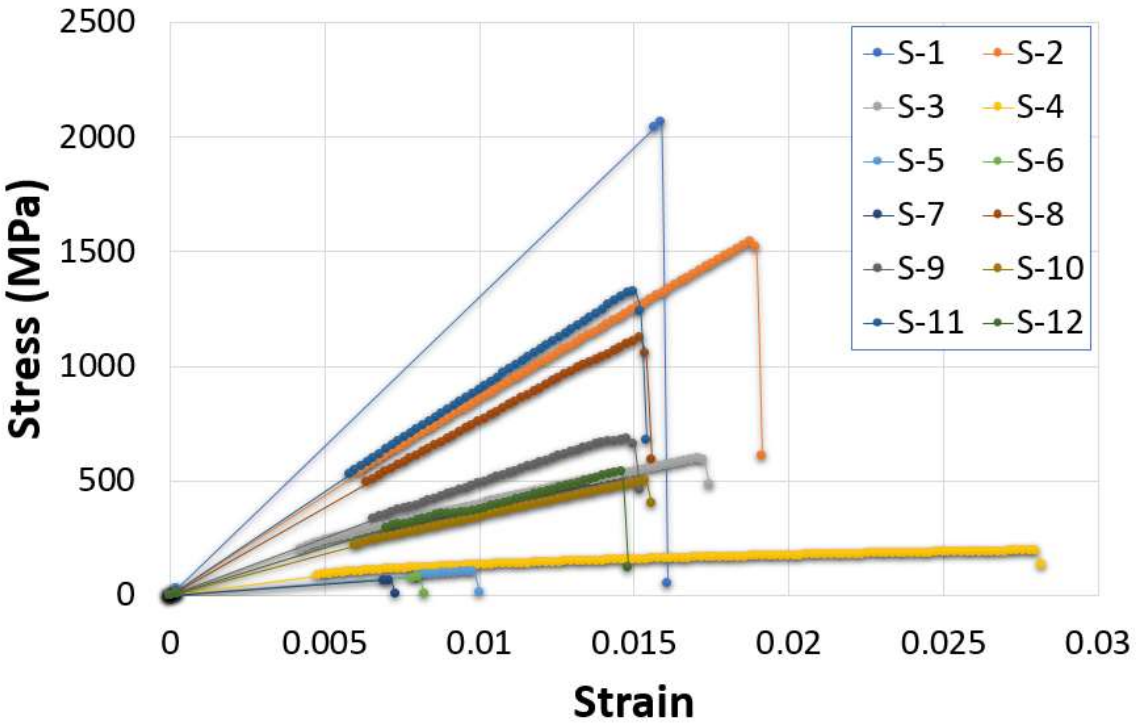


Figure 1. The stress-strain dependence during the tensile test of the twelve specimens.

The table below shows the min and max limits of the Stress and Strain parameters for the twelve specimens.

	Strain (min)	Strain (max)	Stress (min)	Stress (max)
S-1	0	0.01587015	0	2064.24019
S-2	0	0.01874477	0	1540.16043
S-3	0	0.01705347	0	593.258184
S-4	0	0.02798728	0	192.457154
S-5	0	0.00981914	0	101.786008
S-6	0	0.00803761	0	77.3136431
S-7	0	0.00709987	0	63.9780266
S-8	0	0.01537992	0	1121.76279
S-9	0	0.01478241	0	680.072911
S-10	0	0.01536462	0	499.329696
S-11	0	0.01499824	0	1327.03122
S-12	0	0.0146055	0	540.079359

Table 1. Limits of Strain and Stress Values

Every specimen is composed of 16 composite layers having different orientations. The following sequence could characterize a specimen state during the tensile test:

$$\alpha_1,\ \alpha_2,\ldots,\alpha_{16},\ Strain,\ Stress \qquad (**)$$

Each of the twelve tested specimens has a specific angle combination $\alpha_i, i=1,\cdots,16$ that will be called a pattern.

Unit system: MPa (t, mm, s, °C, N, MPa, mJ)
Essai traction
Norme: UNT-ASTMD3039
S-1: $[0]_{16}$
S-2: $[\pm 20]_8$
S-3: $[\pm 30]_8$
S-4: $[\pm 45]_8$
S-5: $[\pm 60]_8$
S-6: $[\pm 70]_8$
S-7: $[90]_{16}$
S-8: $[0/45/0/90/0/-45/0/45]_s$
S-9: $[45/0/-45/90]_{2s}$
S-10: $[45/-45/0/45/-45/90/45/-45]_{2s}$
S-11: $[0/30/0/90/0/-30/0/30]_{2s}$
S-12: $[60/0/-60/90]_{2s}$

Table 2. The patterns of the specimens.

For each specimen, the tensile test collected pairs of values Stress – Strain that can placed on a specific straight line whose equation is given in the table below.

Specimen	Equation
S-1	$F_1(x)$= 130393x + 4e-07
S-2	$F_2(x)$ = 81800x + 29,48
S-3	$F_3(x)$ = 46733x + 0,5042
S-4	$F_4(x)$ = 17588x + 0,00055
S-5	$F_5(x)$ = 10691x + 0,0192
S-6	$F_6(x)$ = 9624,4x
S-7	$F_7(x)$ = 9132x+0.0264
S-8	$F_8(x)$ = 76670x + 0,054
S-9	$F_9(x)$ = 50158x + 0,1227
S-10	$F_{10}(x)$ = 36049x + 0,4325
S-11	$F_{11}(x)$ = 90481x + 0,0401
S-12	$F_{12}(x)$ = 41686x

Table 3. The linear functions where the measurements are placed.

2. The Specific Objectives of this Research

As mentioned, the main objective is to construct an ML model to apprehend all the measurements described before and predict the Stress value for any pattern and Strain value. Our work's specific objectives are:

1. Generate a dataset big enough to construct the ML model to generalize the response for any pattern and adequate Strain value.
2. Construct a parametric model (e.g., the multiple linear regression) that is easy to understand and apply and can be used for comparison with the following models.
3. Construct some nonparametric models (SVM, decision trees, Gaussian process regression, and neural networks), analyze their accuracy, and compare them to the parametric model.
4. Choose the more accurate parametric model that could be used in further research.

3. A dataset generation for constructing and testing different ML models.

We used three ideas to yield a dataset that can be used to train and test the ML models.

- A uniformly distributed noise perturbs the patterns; that is, it affects each orientation angle with a value belonging to [-d, d] (e.g., d=2 grads). This perturbation models the imprecision in achieving the layer's orientation but also diversifies the orientation values to make the generalization possible.
- We consider M (e.g., M=30) data points generated by M Strain values for each specimen. The corresponding M measured Stress values are obtained using the corresponding linear function F_i.

$$\text{Stress}_k = F_i(\text{Strain}_k) \quad k=1,\cdots,M;\ i=1,\cdots,12.$$

- Each time a data point is generated, the pattern is perturbated.

Finally, *our dataset would have **12*M** data points*.

Remark: We have a *data-generating process*, using a probability distribution, that meats the *independent and identically distribution assumptions*. The training and test sets will be generated independently using the same probability distribution.

3.1 Generation of the dataset

The program **H2Data2** constructs each data point's pattern, functions, strain, and stress values. It also generates the *design matrix* called BigData.

The vector PATTERN contains the layers' orientations for each specimen.

```
PATTERN=zeros(12,16);
PATTERN(1,:)=[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.];
PATTERN(2,:)=[20. -20. 20. -20. 20. -20. 20. -20. 20. -20. 20. -20. 20. -20. 20. -20.];
PATTERN(3,:)=[30. -30. 30. -30. 30. -30. 30. -30. 30. -30. 30. -30. 30. -30. 30. -30.];
PATTERN(4,:)=[45. -45. 45. -45. 45. -45. 45. -45. 45. -45. 45. -45. 45. -45. 45. -45.];
PATTERN(5,:)=[60. -60. 60. -60. 60. -60. 60. -60. 60. -60. 60. -60. 60. -60. 60. -60.];
PATTERN(6,:)=[70. -70. 70. -70. 70. -70. 70. -70. 70. -70. 70. -70. 70. -70. 70. -70.];
PATTERN(7,:)=[90. 90. 90. 90. 90. 90. 90. 90. 90. 90. 90. 90. 90. 90. 90. 90.];
PATTERN(8,:)=[0. 45. 0. 90. 0. -45. 0. 45. 45. 0. -45. 0. 90. 0. 45. 0.];
PATTERN(9,:)=[45. 0. -45. 90. 45. 0. -45. 90. 90. -45. 0. 45. 90. -45. 0. 45.];
PATTERN(10,:)=[45. -45. 0. 45. -45. 90. 45. -45. -45. 45. 90. -45. 45. 0. -45. 45.];
PATTERN(11,:)=[0. 30. 0. 90. 0. -30. 0. 30. 30. 0. -30. 0. 90. 0. 30. 0.];
PATTERN(12,:)=[60. 0. -60. 90. 60. 0. -60. 90. 90. -60. 0. 60. 90. -60. 0. 60.];
```

Implementation:
- F: array of function handles
- strain: matrix with the values of the strain for each specimen and M=30 abscissae
-nc=18;

```
- BigData=zeros(12*M,nc);  % matrix with 360 data points

- PAT=PATTERN(Sk,:)+ random('unif',-delta,delta,[1,16]);
- BigData(i,:)=[PAT, strain(Sk,j), F{Sk}(strain(Sk,j))];

- save('WS_data360',' BigData',' F',' strain',' delta',' M');
```

Data point #220 would be, for example, the following [18,1] vector

```
[ 1.391     45.331     0.34471    91.703     0.30031    -46.96     1.2375     45.435     44.92
0.92624  -45.968  -0.075962    88.909    -1.8056     43.677    -0.96621  0.0051266   393.11].
```

3.2 Construction of the tables for training and testing

The program **H2Construction** constructs the tables for training and testing. It splits the lines of the BigData matrix into two matrices, Dtrain and Dtest.

Each specimen generated 30 (M) data points. The first 25 and last 5 lines will be added to the DTrain and DTest matrices, respectively. The two matrices will be converted into TableTest and TableTest, respectively.

TableTrain ← DTrain

TableTest ← DTest

4. Construction of Step Vise models using datasets

The multiple linear regression models we considered also included nonlinear terms (products of predictors) and used a step-wise technique to construct those models. The model maintains linearity in terms of its coefficients.

```
1. Adding x1, FStat = 227.8572, pValue = 1.239117e-38
2. Adding St, FStat = 216.2564, pValue = 3.720389e-37
3. Adding x1:St, FStat = 367.5712, pValue = 8.032568e-54
4. Adding x12, FStat = 53.9613, pValue = 2.01116e-12
5. Adding x12:St, FStat = 95.8075, pValue = 9.11615e-20
6. Adding x13, FStat = 236.636, pValue = 1.49516e-39
7. Adding x13:St, FStat = 148.9911, pValue = 5.765152e-28
8. Adding x12:x13, FStat = 72.9692, pValue = 7.48834e-16
9. Adding x9, FStat = 65.4199, pValue = 1.66602e-14
10. Adding x9:St, FStat = 36.892, pValue = 3.92447e-09
11. Adding x2, FStat = 16.1382, pValue = 7.52075e-05
12. Adding x1:x9, FStat = 14.5624, pValue = 0.000166095
13. Adding x1:x12, FStat = 5.6396, pValue = 0.018218
14. Adding x2:St, FStat = 5.2191, pValue = 0.023075
15. Removing x12:x13, FStat = 0.24892, pValue = 0.61822
```

Linear regression model:

Ss ~ 1 + x1*x9 + x1*St + x2*x13 + x2*St + x9*St + x12*St + x13*St

Estimated Coefficients:

	Estimate	SE	tStat	pValue
(Intercept)	66.289	16.348	4.0548	6.4742e-05
x1	-1.0564	0.90931	-1.1618	0.24629
x2	5.299	1.5056	3.5196	0.00050263
x9	-2.1988	0.58916	-3.7322	0.000229
x12	-1.5438	0.30825	-5.0083	9.6246e-07
x13	0.42446	0.401	1.0585	0.29071
St	1.2418e+05	1901.7	65.3	7.0906e-174
x1:x9	**0.04764**	**0.012162**	**3.9171**	**0.00011217**
x1:St	**-1320.7**	**54.246**	**-24.346**	**1.1118e-71**
x2:x13	**-0.065347**	**0.01489**	**-4.3885**	**1.607e-05**
x2:St	**-176.64**	**54.038**	**-3.2687**	**0.0012122**
x9:St	**-170.77**	**29.761**	**-5.7382**	**2.4366e-08**
x12:St	**666.06**	**38.01**	**17.523**	**3.3209e-47**
x13:St	**-307.24**	**37.886**	**-8.1096**	**1.5046e-14**

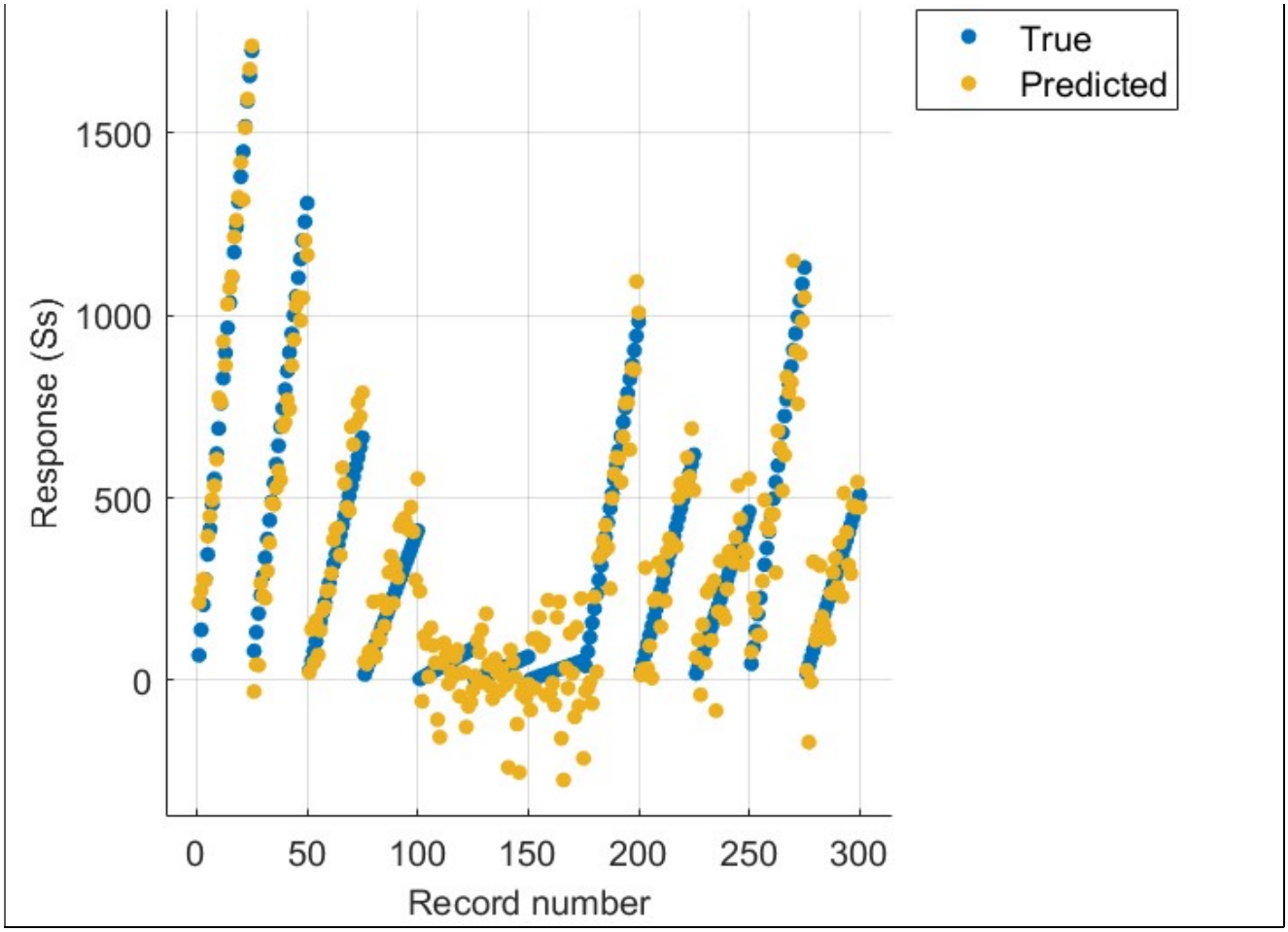


Figure 2. Linear regression model- Response versus true values

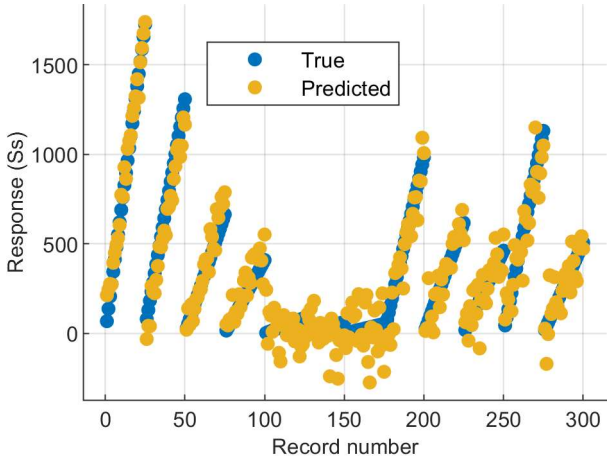


Figure 22. Predicted versus real values for the training data set

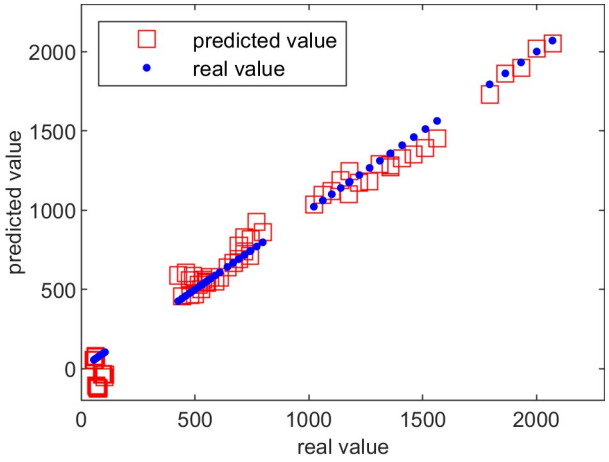


Figure 23. Predicted versus real values for the test data set

4.1 Comparison of the true stress values to the predicted values

	Stress	uPred
1	1793.4	1771.2

2	1862.4	1812.9
3	1931.4	1963.1
4	2000.4	1966.8
5	2069.4	2062.7
6	1358.4	1234.1
7	1409.5	1256.4
8	1460.6	1339.7
9	1511.7	1409.9
10	1562.8	1472.7
11	691.2	765.74
12	717.77	837
13	744.33	830.2
14	770.9	881.83
15	797.46	972.66
16	426.61	455.18
17	443.02	583.99
18	459.42	525.39
19	475.83	588.2
20	492.24	567.19
21	**90.999**	**-46.329**
22	**94.498**	**-23.934**
23	**97.997**	**-29.726**
24	**101.5**	**-18.482**
25	**105**	**-27.37**
26	**67.043**	**-83.925**
27	**69.621**	**-118.72**
28	**72.2**	**-102.75**
29	**74.779**	**-121.22**
30	**77.357**	**-129.23**
31	56.218	27.051
32	58.379	57.544
33	60.54	60.965
34	62.701	44.883
35	64.862	68.942
36	1022	1067.1
37	1061.3	1105.3
38	1100.6	1182.4
39	1139.9	1202.6
40	1179.2	1205.6
41	642.72	608.92
42	667.43	655.97
43	692.15	655.66
44	716.86	755.1
45	741.58	713.76
46	480.46	486.48
47	498.92	462.78
48	517.39	531.04
49	535.85	565.14
50	554.31	608.58
51	1176.2	1176.3
52	1221.4	1169.8
53	1266.6	1207.7
54	1311.9	1219.9
55	1357.1	1284.2
56	527.67	535.89
57	547.96	557.76
58	568.26	546.71
59	588.55	571.09
60	608.84	620.91

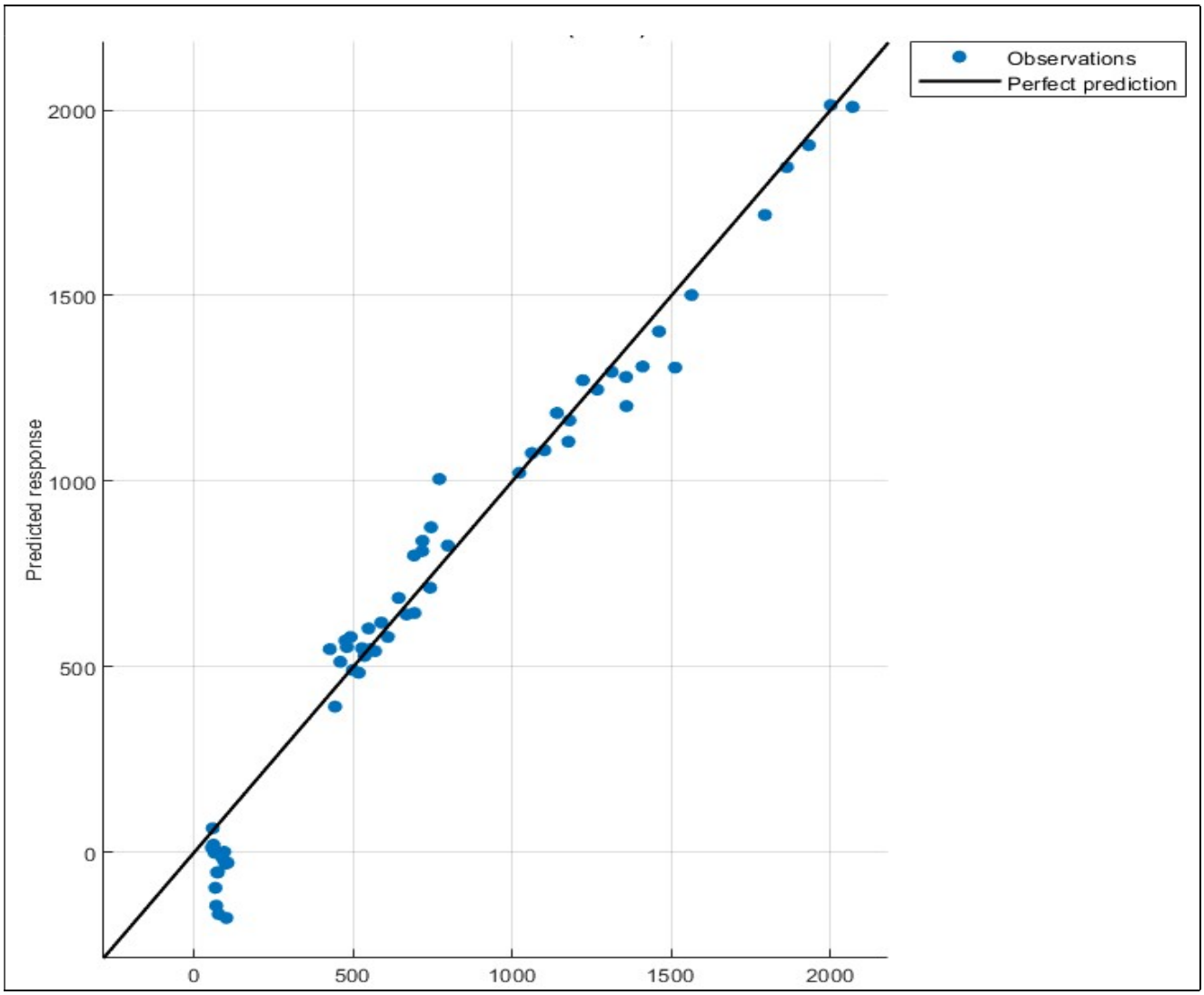


Figure 3. Stress-predicted values versus the true measured values.

4.2 Comparison between real and predicted stress values for the specimen #7 and all the 30 strain's values

Strain	Stress	uPred
0.00023666	2.1876	7.5603
0.00047332	4.3488	9.0623
0.00070999	6.51	10.564
0.00094665	8.6712	12.066
0.0011833	10.832	13.568
0.00142	12.994	15.07
0.0016566	15.155	16.572
0.0018933	17.316	18.074
0.00213	19.477	19.576
0.0023666	21.638	21.078
0.0026033	23.8	22.58
0.0028399	25.961	24.082
0.0030766	28.122	25.584
0.0033133	30.283	27.085

0.0035499	32.444	28.587
0.0037866	34.606	30.089
0.0040233	36.767	31.591
0.0042599	38.928	33.093
0.0044966	41.089	34.595
0.0047332	43.25	36.097
0.0049699	45.412	37.599
0.0052066	47.573	39.101
0.0054432	49.734	40.603
0.0056799	51.895	42.105
0.0059166	54.056	43.607
0.0061532	56.218	45.109
0.0063899	58.379	46.611
0.0066265	60.54	48.113
0.0068632	62.701	49.614
0.0070999	64.862	51.116

Figure 4 shows a deviated linear placement.

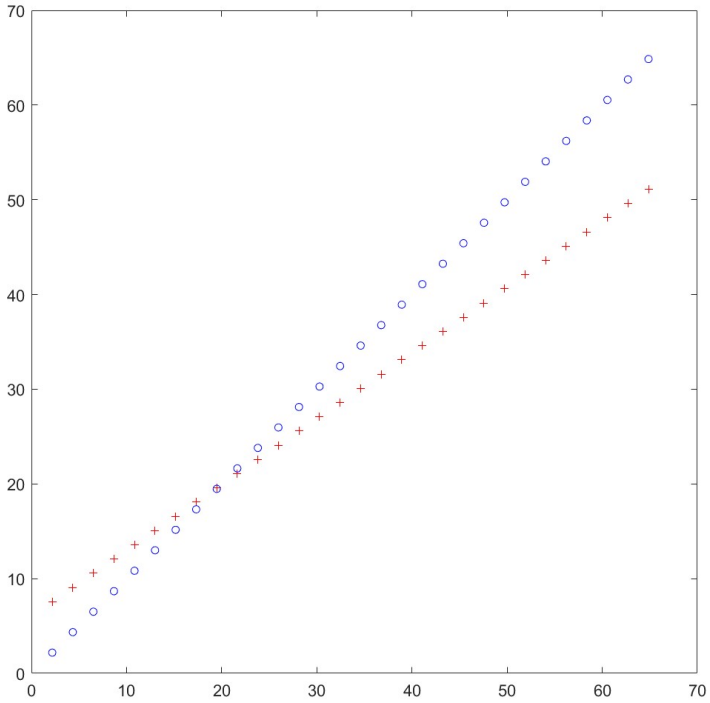


Fig 4. Comparison between real (blue) and predicted stress values(red) for specimen #7 and all the 30 strain values.

5. Models based on Support Vector Machines

5.1 First SVM model

Program: H2_modelSVM4

The training parameters are:

Standardized data=yes, Kernel function= Quadratic, Kernel scale= Automatic, Box constraint= Automatic, Epsilon= Auto

****

Nr	LM	RealValue	SVM4
1	1732	1793.4	1684.9
2	1860.7	1862.4	1815.7
3	1898.5	1931.4	1916.3
4	2020.2	2000.4	1946.3
5	2053	2069.4	1966
6	1280.4	1358.4	1237.9
7	1325.2	1409.5	1259.7
8	1350.6	1460.6	1291.7
9	1390.2	1511.7	1268.4
10	1453.7	1562.8	1416.4
11	776.57	691.2	782.42
12	830.27	717.77	771.33
13	817.05	744.33	794.64
14	924.42	770.9	969.99
15	862.47	797.46	816.63
16	590.41	426.61	451.63
17	458.26	443.02	249.64
18	604.86	459.42	372.73
19	560.09	475.83	485.59
20	585.09	492.24	325.44
21	-37.373	90.999	39.86
22	-34.88	94.498	39.649
23	-32.447	97.997	-14.256
24	-52.129	101.5	-48.742
25	-37.333	105	-24.972
26	-103.89	67.043	-14.329
27	-114.62	69.621	-68.909
28	-120.05	72.2	-42.349
29	-124.01	74.779	-50.366
30	-120.17	77.357	-73.159
31	56.668	56.218	40.865
32	57.595	58.379	32.068
33	50.58	60.54	17.733
34	79.963	62.701	57.463
35	70.683	64.862	-2.7203
36	1034.2	1022	1009.6
37	1097.1	1061.3	1047.3
38	1121.6	1100.6	1086.4
39	1190.6	1139.9	1123.2
40	1244.3	1179.2	1149.4
41	638.1	642.72	612.3
42	668.63	667.43	589.98

```
    43      692.83      692.15      529.37
    44      742.13      716.86      648.78
    45      711.31      741.58      522.45
    46      465.52      480.46      567.94
    47      468.47      498.92      493.39
    48      530.87      517.39      464.42
    49       560.6      535.85      523.57
    50      541.03      554.31      580.87
    51      1101.1      1176.2        1108
    52      1174.1      1221.4      1208.3
    53      1182.4      1266.6      1204.2
    54      1291.1      1311.9        1281
    55      1271.3      1357.1      1275.3
    56      500.97      527.67       506.3
    57      550.47      547.96      629.11
    58      576.83      568.26      602.14
    59      548.51      588.55       617.1
    60      576.88      608.84      679.57
mdlLoss=
       11770
RMSE (root-mean-square error) Validation=       70.507
RMSE test =              97.712
```

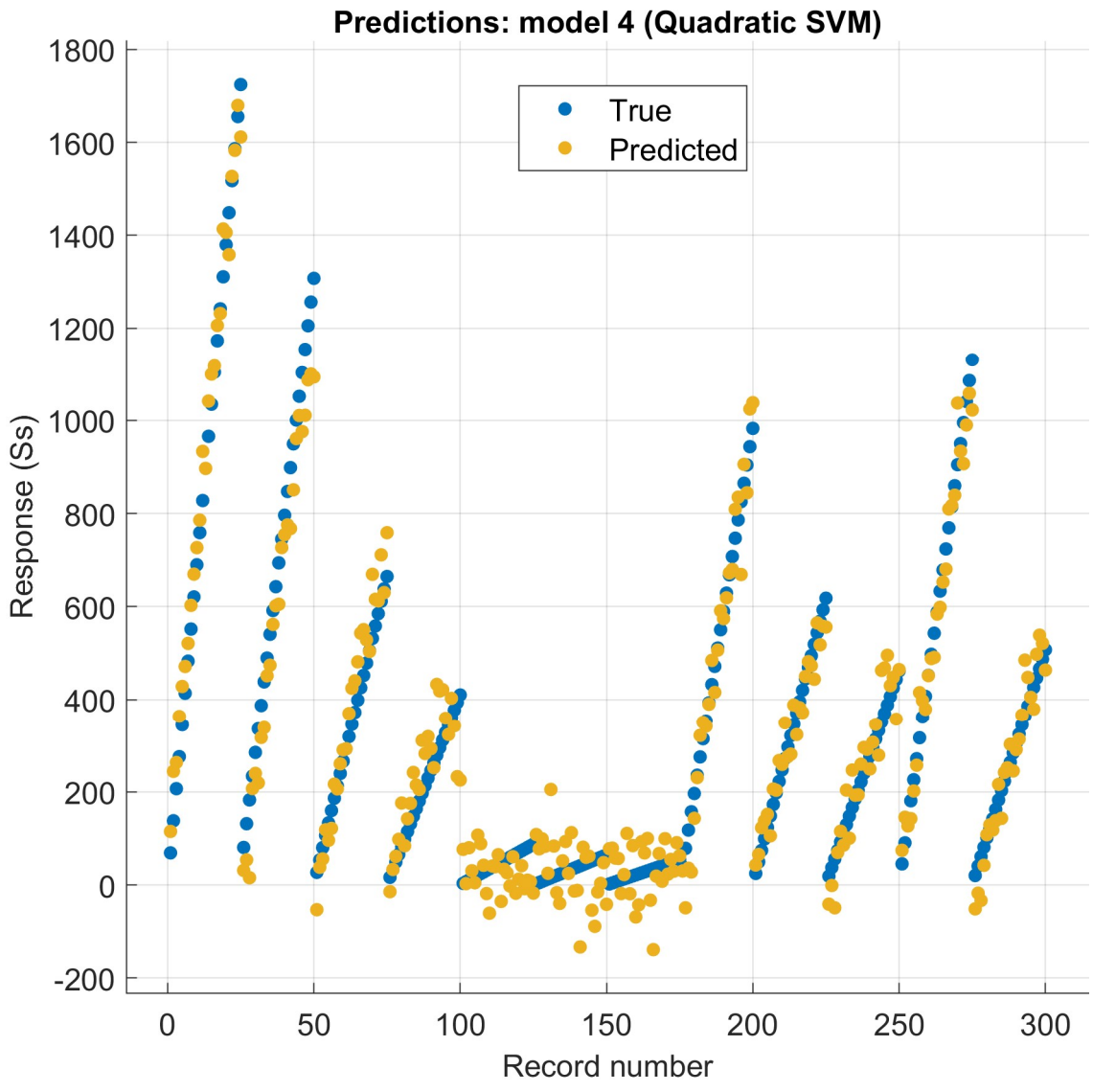
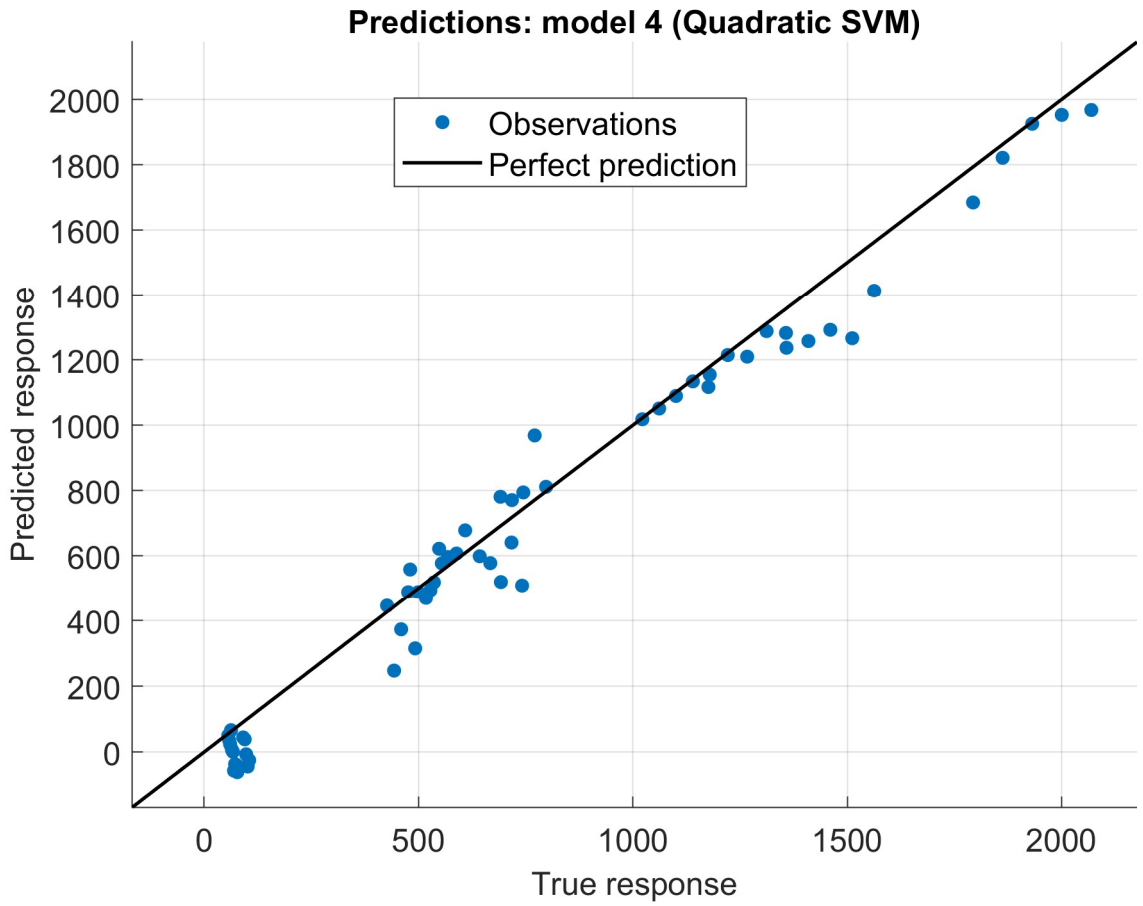


Figure 5. SVM4 model- Response versus true values




```
>> H2_useSVM4: for specimen #7
    strain     RealStress      SVM4
    ______     __________     ______
    0.001        9.1584       17.452
    0.002         18.29       23.366
    0.003        27.422        27.53
    0.004        36.554       29.945
    0.005        45.686       30.611
    0.006        54.818       29.527
    0.007         63.95       26.694
```

SVM4 Model with Optimization of Hyperparameters

****

Iter	Eval result	Objective: log(1+loss)	Objective runtime	BestSoFar (observed)	BestSoFar (estim.)	BoxConstraint	KernelScale	Epsilon	KernelFunction	PolynomialOrder	Standardize
1	Best	11.761	0.24632	11.761	11.761	50.879	-	5506.1	linear	-	true
2	Best	11.269	0.057955	11.269	11.306	0.0010394	-	55.75	linear	-	false
3	Best	11.101	0.18005	11.101	11.101	3.707	-	1.7182	linear	-	false
4	Accept	11.761	0.090178	11.101	11.102	22.21	322.44	2744.3	gaussian	-	false
5	Best	11.087	0.056985	11.087	11.087	0.051545	-	0.32655	linear	-	false
6	Accept	11.316	8.6758	11.087	11.087	0.0017306	-	0.38008	polynomial	2	false
7	Accept	11.761	0.055333	11.087	11.087	22.54	0.16624	2799.6	gaussian	-	false
8	Accept	11.103	0.077886	11.087	11.087	0.0094022	-	0.55794	linear	-	false
9	Accept	11.135	14.203	11.087	11.088	925.29	-	0.3297	linear	-	false
10	Best	11.076	0.0685	11.076	11.077	0.36097	-	0.329	linear	-	false
11	Accept	11.096	0.10543	11.076	11.085	1.1846	-	0.33354	linear	-	false
12	Best	11.075	0.056443	11.075	11.082	0.24861	-	0.32781	linear	-	false
13	Best	11.074	0.057453	11.074	11.081	0.22007	-	0.32747	linear	-	false
14	Accept	11.761	0.051278	11.074	11.081	0.0629	-	32063	linear	-	false
15	Accept	11.835	0.068064	11.074	11.076	26.375	0.0010591	0.36085	gaussian	-	false
16	Best	11.071	0.049445	11.071	11.071	0.11624	-	1.568	linear	-	false
17	Accept	11.078	0.055216	11.071	11.073	0.17758	-	0.90556	linear	-	false
18	Accept	11.077	0.054596	11.071	11.073	0.13152	-	5.2493	linear	-	false
19	Accept	11.086	0.050962	11.071	11.075	0.056664	-	2.6373	linear	-	false
20	Accept	11.074	0.068646	11.071	11.073	0.27179	-	2.7135	linear	-	false

Iter	Eval result	Objective: log(1+loss)	Objective runtime	BestSoFar (observed)	BestSoFar (estim.)	BoxConstraint	KernelScale	Epsilon	KernelFunction	PolynomialOrder	Standardize
21	Accept	11.073	0.060873	11.071	11.073	0.25692	-	2.5645	linear	-	false
22	Accept	11.077	0.074086	11.071	11.073	0.57366	-	7.8012	linear	-	false
23	Accept	11.072	0.065127	11.071	11.072	0.27767	-	3.2501	linear	-	false
24	Accept	28.526	14.596	11.071	11.075	956.21	-	0.71929	polynomial	3	false
25	Best	10.54	0.068093	10.54	10.541	58.131	-	0.32647	linear	-	true
26	Accept	11.835	0.073609	10.54	10.541	35.687	0.0015134	0.52674	gaussian	-	true
27	Accept	11.761	0.046824	10.54	10.541	31.597	0.0064668	32451	gaussian	-	true
28	Best	7.9513	4.5132	7.9513	7.9515	937.68	-	0.45123	polynomial	2	true
29	Accept	11.761	0.042463	7.9513	7.9524	0.0010158	-	30111	polynomial	2	true
30	Accept	11.761	0.045145	7.9513	7.9526	0.0010001	-	28809	polynomial	3	true

____

Optimization completed.
MaxObjectiveEvaluations of 30 reached.
Total function evaluations: 30
Total elapsed time: 54.9026 seconds
Total objective function evaluation time: 43.9151

Observed objective function value = 7.9513
Estimated objective function value = 7.9526
Function evaluation time = 4.5132

Estimated objective function value = 7.9526
Estimated function evaluation time = 4.5076

mdlLoss= 8492.6

Nr	LM	RealValue	SVM
1	1732	1793.4	1703.7
2	1860.7	1862.4	1836.2
3	1898.5	1931.4	1851
4	2020.2	2000.4	1942.6
5	2053	2069.4	1978.7
6	1280.4	1358.4	1181.9
7	1325.2	1409.5	1319.7
8	1350.6	1460.6	1334.2
9	1390.2	1511.7	1318.2
10	1453.7	1562.8	1405.9
11	776.57	691.2	752.59
12	830.27	717.77	809.04
13	817.05	744.33	821.8
14	924.42	770.9	874.66
15	862.47	797.46	845.05
16	590.41	426.61	456.78
17	458.26	443.02	312.86
18	604.86	459.42	364.27
19	560.09	475.83	311.76
20	585.09	492.24	436.92
21	-37.373	90.999	-2.4286
22	-34.88	94.498	-4.0187
23	-32.447	97.997	-69.107
24	-52.129	101.5	-109.96
25	-37.333	105	-23.563
26	-103.89	67.043	-63.279
27	-114.62	69.621	-120.44
28	-120.05	72.2	-96.765
29	-124.01	74.779	-59.83
30	-120.17	77.357	-101.09
31	56.668	56.218	54.417
32	57.595	58.379	59.222
33	50.58	60.54	40.54
34	79.963	62.701	51.446
35	70.683	64.862	48.108
36	1034.2	1022	970.44
37	1097.1	1061.3	1062.4
38	1121.6	1100.6	1055.3
39	1190.6	1139.9	1159.2
40	1244.3	1179.2	1161.8
41	638.1	642.72	637.65
42	668.63	667.43	649.23
43	692.83	692.15	658.06
44	742.13	716.86	689.85
45	711.31	741.58	642.64
46	465.52	480.46	468.55
47	468.47	498.92	454.03
48	530.87	517.39	494.98
49	560.6	535.85	496.77
50	541.03	554.31	524.12

51	1101.1	1176.2	1125
52	1174.1	1221.4	1185.4
53	1182.4	1266.6	1220.6
54	1291.1	1311.9	1282.4
55	1271.3	1357.1	1291.5
56	500.97	527.67	495.97
57	550.47	547.96	533.09
58	576.83	568.26	543.67
59	548.51	588.55	555.36
60	576.88	608.84	579.53

Remark: A small increase in quality

5.2 SVM with Hyperparameters' Optimization and grade=2

SVM Model 2

(Model SVM 22)-

The model is trained using the MATLAB function `fitrsvm` as below:

```
mdlSVM2=fitrsvm(TableTrain, "Ss", 'Standardize', true,...
'kernelfunction', 'polynomial', 'PolynomialOrder',2,
'KernelScale',3.001,
'BoxConstraint',0.8143
'Epsilon',4.0715
'OptimizeHyperparameters',' all');
```

mdlLoss=

6448.2

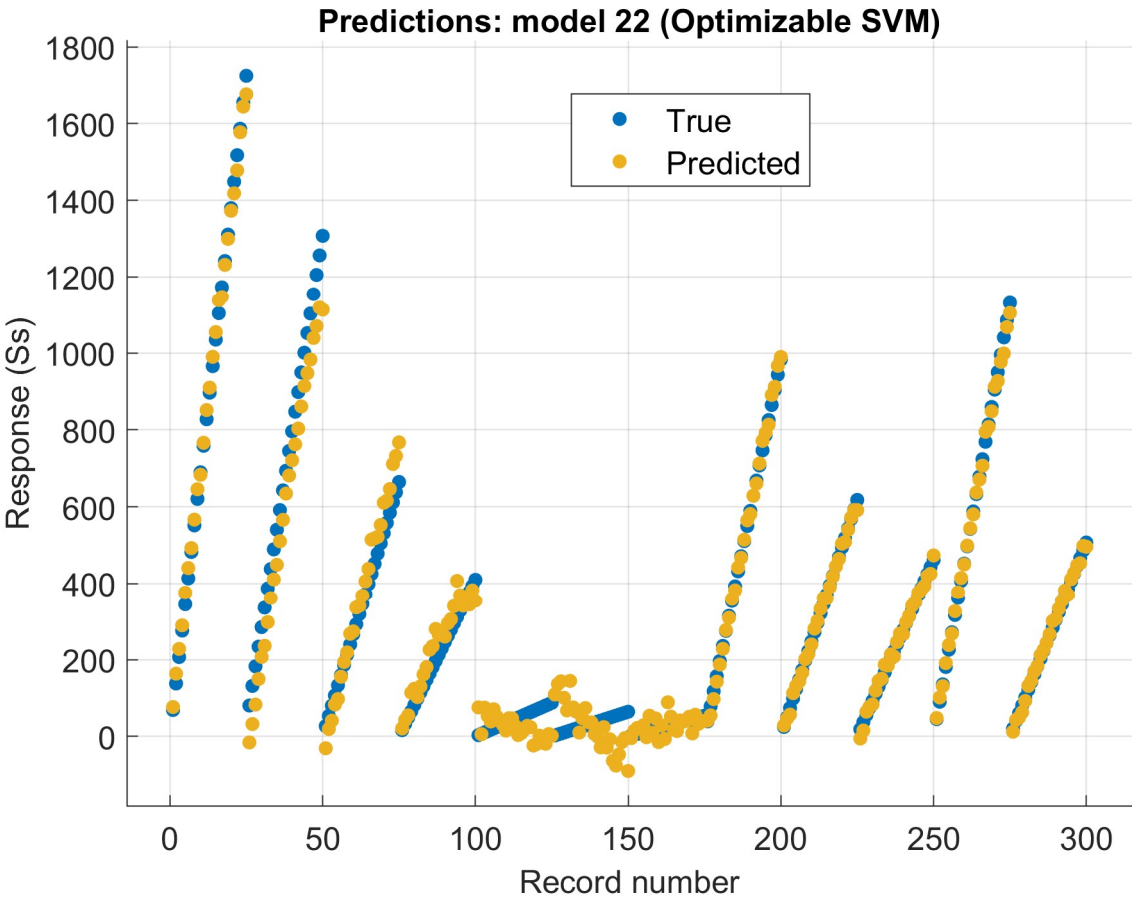
Nr	LM	RealValue	SVM	
1	1732	1793.4	1722.4	
2	1860.7	1862.4	1752	
3	1898.5	1931.4	1779.1	
4	2020.2	2000.4	1786.9	
5	2053	2069.4	1795.6	
6	1280.4	1358.4	1299.1	
7	1325.2	1409.5	1325.1	
8	1350.6	1460.6	1325.9	
9	1390.2	1511.7	1296	
10	1453.7	1562.8	1283.2	
11	776.57	691.2	695.27	
12	830.27	717.77	749.01	
13	817.05	744.33	787.79	
14	924.42	770.9	772.34	
15	862.47	797.46	754.7	
16	590.41	426.61	417.99	
17	458.26	443.02	431.28	
18	604.86	459.42	438.98	
19	560.09	475.83	441.98	
20	585.09	492.24	441.92	
21	-37.373	90.999	93.074	**A SVM model that gives good predictions in this range.**
22	-34.88	94.498	95.691	
23	-32.447	97.997	100.18	
24	-52.129	101.5	105.26	
25	-37.333	105	110.17	

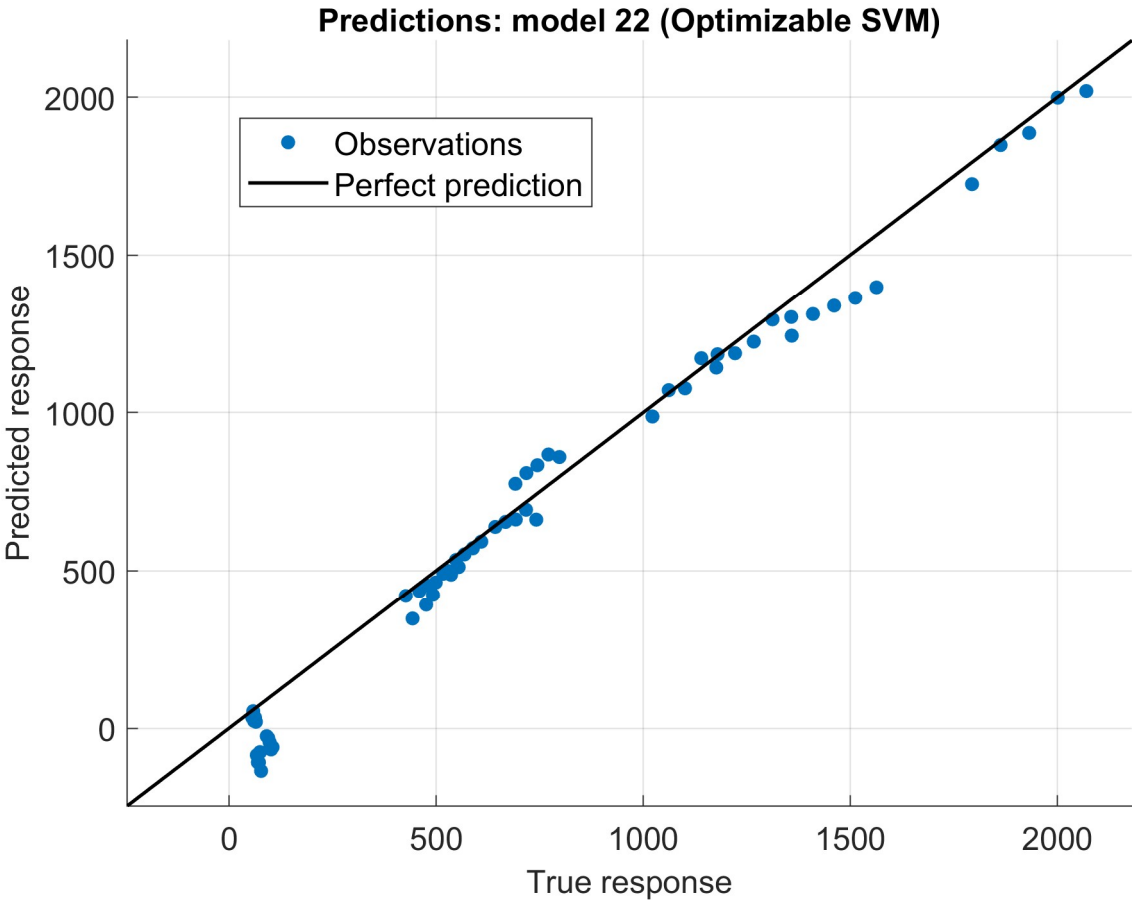
```
    26    -103.89     67.043      68.82
    27    -114.62     69.621     71.944
    28    -120.05      72.2      74.032
    29    -124.01     74.779     77.312
    30    -120.17     77.357     80.144
    31     56.668     56.218     57.801
    32     57.595     58.379     61.307
    33      50.58      60.54     63.944
    34     79.963     62.701     66.879
    35     70.683     64.862     69.335
    36     1034.2       1022     986.38
    37     1097.1     1061.3     1024.8
    38     1121.6     1100.6     1045.1
    39     1190.6     1139.9     1055.9
    40     1244.3     1179.2     1068.4
    41      638.1     642.72     635.92
    42     668.63     667.43     647.59
    43     692.83     692.15     663.69
    44     742.13     716.86     680.33
    45     711.31     741.58     688.16
    46     465.52     480.46     474.78
    47     468.47     498.92     489.04
    48     530.87     517.39      499.4
    49      560.6     535.85     512.94
    50     541.03     554.31      521.8
    51     1101.1     1176.2     1146.3
    52     1174.1     1221.4     1179.7
    53     1182.4     1266.6     1205.6
    54     1291.1     1311.9     1224.8
    55     1271.3     1357.1     1232.6
    56     500.97     527.67     519.48
    57     550.47     547.96     543.12
    58     576.83     568.26     564.46
    59     548.51     588.55     565.74
    60     576.88     608.84     581.46

RMSEValid =       5.1788
RMSETest  =       80.301
Size: 40kB

>> H2_UseSVM2

  strain    RealStress    SVM2
   0.001        9.1584     8.1594
   0.002         18.29     16.018
   0.003        27.422     25.319
   0.004        36.554     35.301
   0.005        45.686     45.576
   0.006        54.818     56.146
   0.007         63.95     67.326
```


Predictions: model 22 (Optimizable SVM)
True
Predicted
Response (Ss)
Record number
0
50
100
150
200
250
300
200
400
600
800
1000
1200
1400
1600
1800


Predictions: model 22 (Optimizable SVM)
Observations
Perfect prediction
Predicted response
True response
0
500
1000
1500
2000

5.3 SVM Model with cubic Kernel function –(SVM16)

Non-optimizable version:

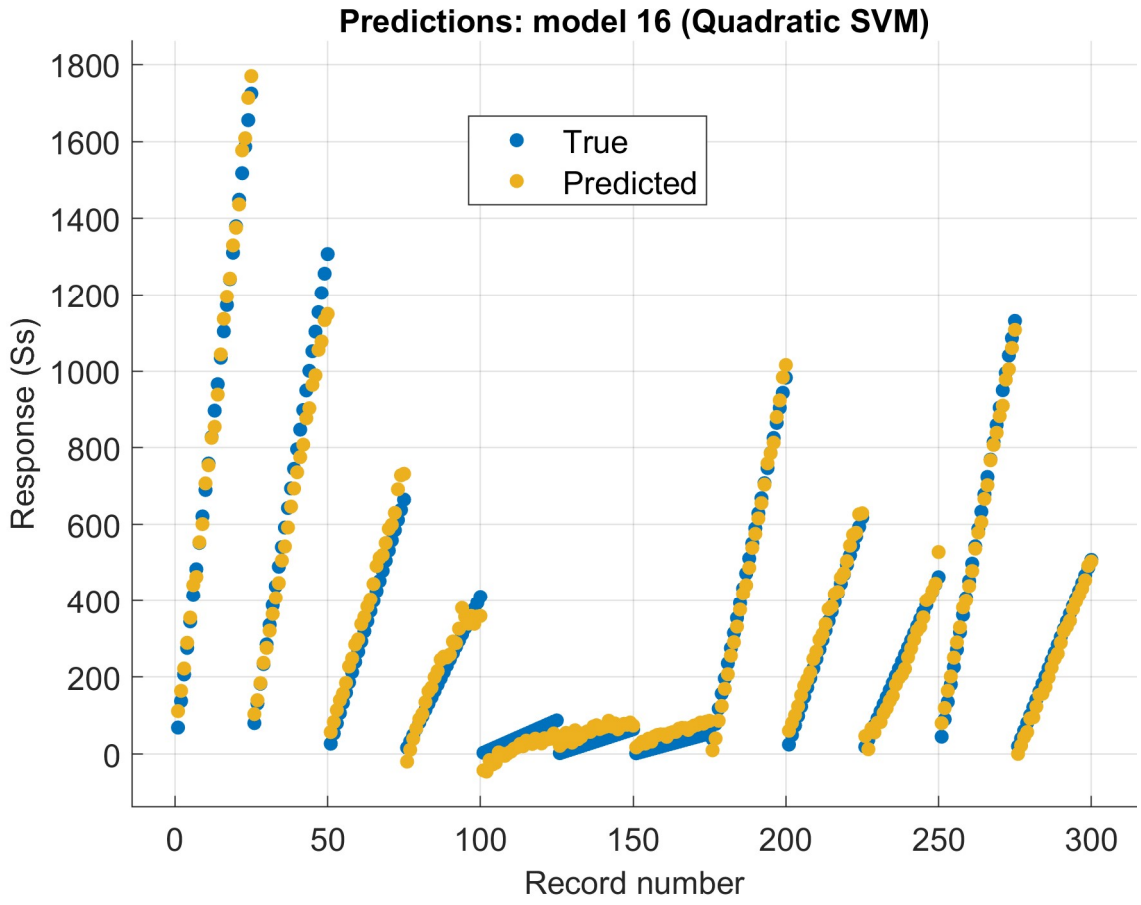
```
'KernelFunction', 'polynomial', ...
'PolynomialOrder', 3, ...
'KernelScale', 3.001, ...
'BoxConstraint', Automatic, ...
'Epsilon', Auto, ...
'Standardize', true);
```

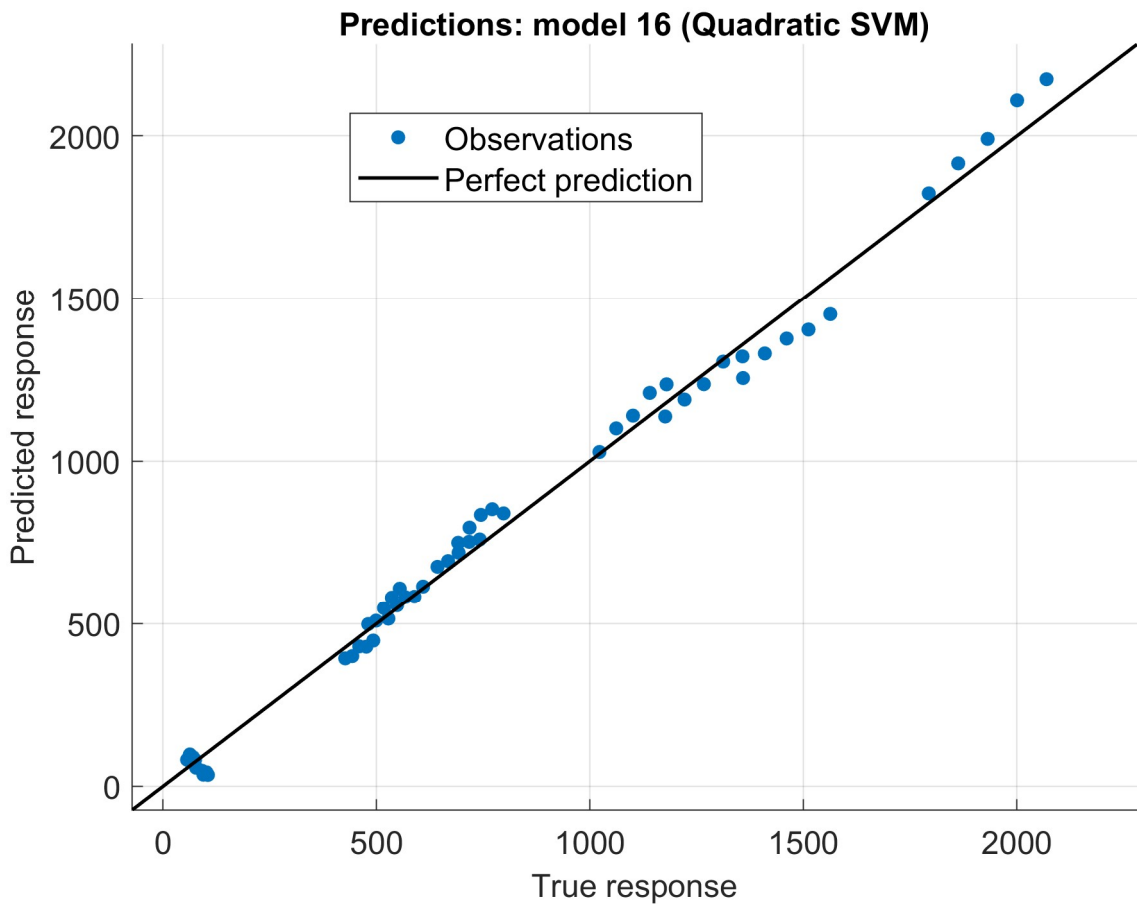
Nr	LM	RealValue	SVM16
1	1732	1793.4	1823.5
2	1860.7	1862.4	1915.7
3	1898.5	1931.4	1990.9
4	2020.2	2000.4	2109.4
5	2053	2069.4	2174.2
6	1280.4	1358.4	1254.2
7	1325.2	1409.5	1329.6
8	1350.6	1460.6	1375.3
9	1390.2	1511.7	1403.1
10	1453.7	1562.8	1450.6
11	776.57	691.2	749.72
12	830.27	717.77	796.12
13	817.05	744.33	835.09
14	924.42	770.9	852.53
15	862.47	797.46	839.77
16	590.41	426.61	392.61
17	458.26	443.02	399.61
18	604.86	459.42	429.55
19	560.09	475.83	428.53
20	585.09	492.24	447.42
21	-37.373	90.999	47.679
22	-34.88	94.498	35.616
23	-32.447	97.997	39.292
24	-52.129	101.5	42.684
25	-37.333	105	34.877
26	-103.89	67.043	82.007
27	-114.62	69.621	89.711
28	-120.05	72.2	81.055
29	-124.01	74.779	81.242
30	-120.17	77.357	56.943
31	56.668	56.218	81.491
32	57.595	58.379	84.83
33	50.58	60.54	89.598
34	79.963	62.701	97.737
35	70.683	64.862	93.759
36	1034.2	1022	1027.8
37	1097.1	1061.3	1100.1
38	1121.6	1100.6	1139.1
39	1190.6	1139.9	1208.7
40	1244.3	1179.2	1234.9
41	638.1	642.72	676.06
42	668.63	667.43	693.53
43	692.83	692.15	719.39
44	742.13	716.86	752.67
45	711.31	741.58	759.86
46	465.52	480.46	498.21
47	468.47	498.92	508.85
48	530.87	517.39	547.56

```
    49      560.6      535.85      580.99
    50     541.03      554.31      608.88
    51     1101.1      1176.2      1136.5
    52     1174.1      1221.4      1188.4
    53     1182.4      1266.6      1235.4
    54     1291.1      1311.9      1304.7
    55     1271.3      1357.1      1320.7
    56     500.97      527.67      515.08
    57     550.47      547.96      556.47
    58     576.83      568.26      582.27
    59     548.51      588.55      583.73
    60     576.88      608.84      615.13

RMSEValid =      34.945

RMSETest =       52.108
```






```
>> H2_useSVM16
```

strain	RealStress	SV16
0.001	9.1584	33.037
0.002	18.29	45.31
0.003	27.422	56.359
0.004	36.554	66.258
0.005	45.686	75.08
0.006	54.818	82.898
0.007	63.95	89.786

6. Neural Network Models

6.1 First NN Model1

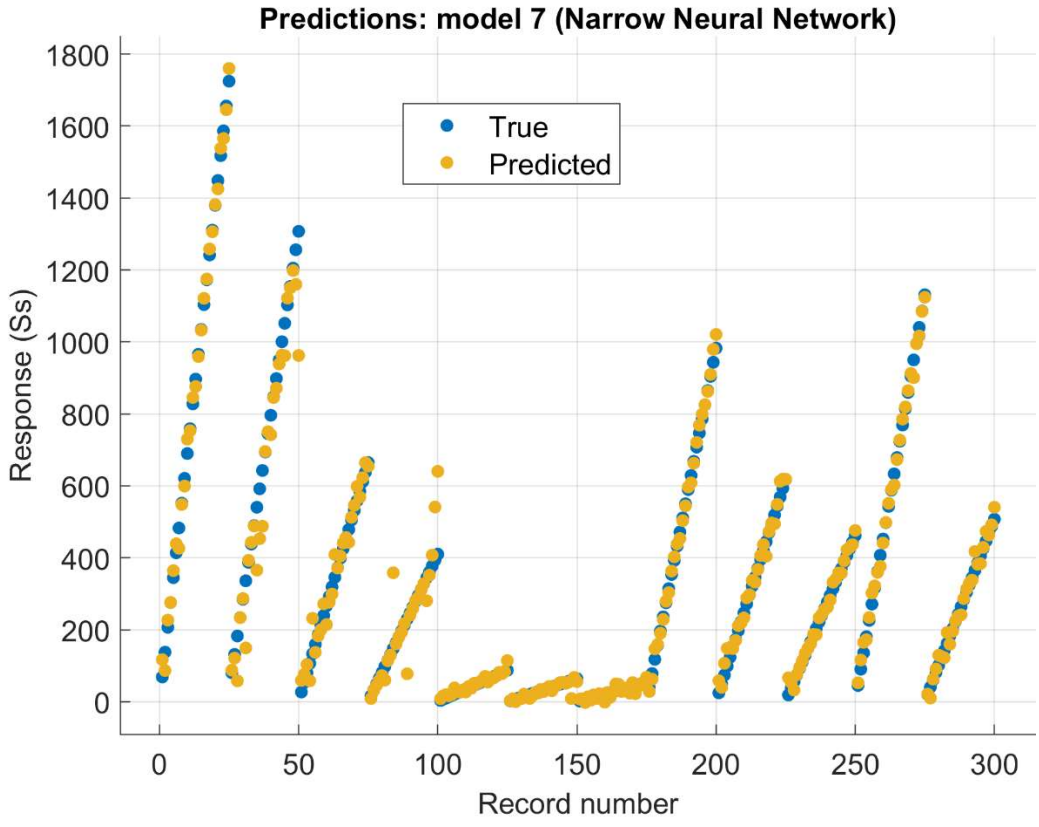
The NN has one hiden layer.

```
....'LayerSizes', 10,
    'Activations', 'relu',
    'Lambda', 0,
    'IterationLimit', 1000,
    'Standardize', true);
>> H2_modelNN1
```

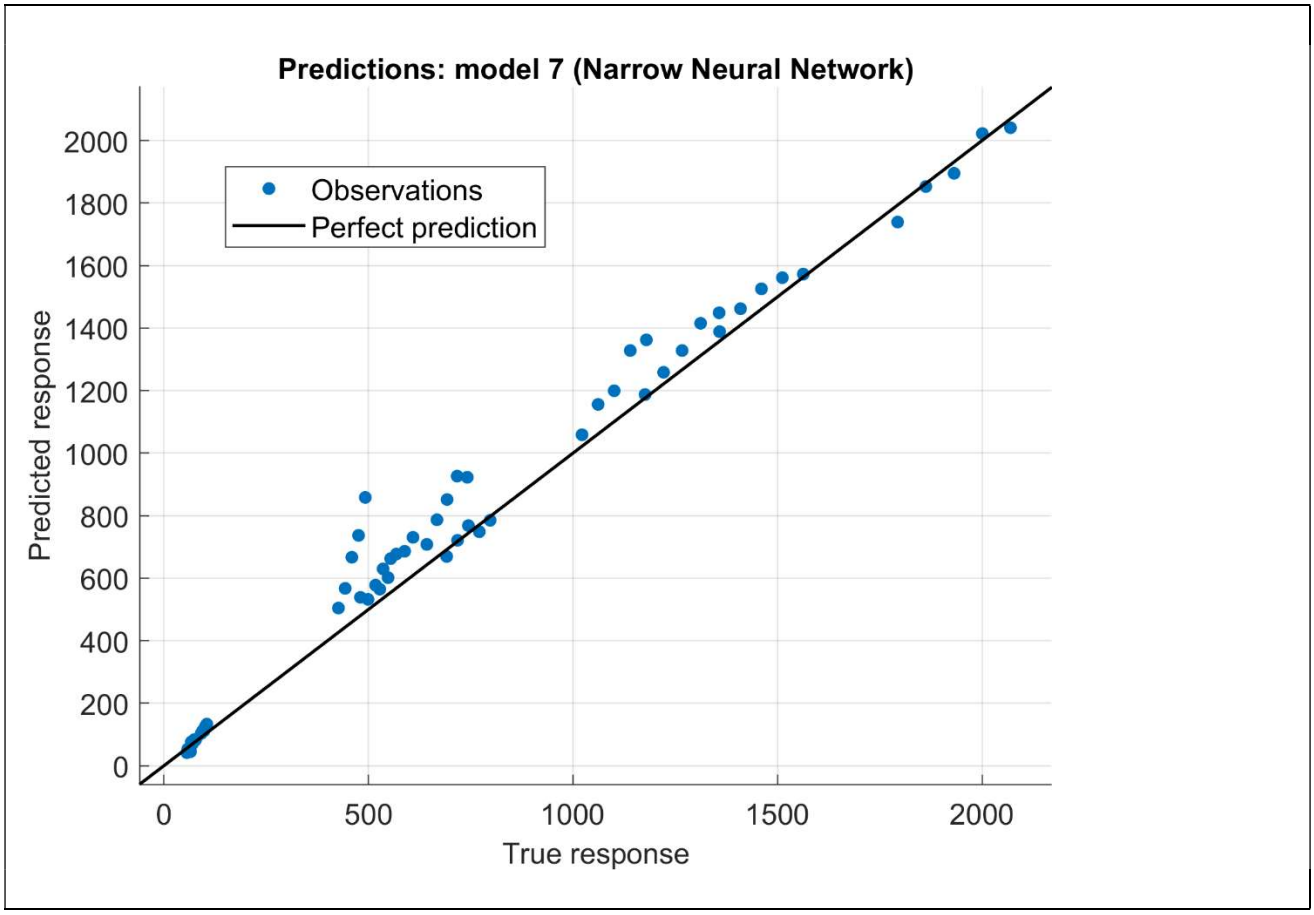
****

Nr	LM	RealValue	NN
1	1732	1793.4	1768.9
2	1860.7	1862.4	1855.2
3	1898.5	1931.4	1911
4	2020.2	2000.4	2003.1
5	2053	2069.4	2055.2
6	1280.4	1358.4	1359.6
7	1325.2	1409.5	1437.7
8	1350.6	1460.6	1484.2
9	1390.2	1511.7	1494.5
10	1453.7	1562.8	1536.2
11	776.57	691.2	655.52
12	830.27	717.77	749.93
13	817.05	744.33	798.6
14	924.42	770.9	787.92
15	862.47	797.46	810.36
16	590.41	426.61	474.05
17	458.26	443.02	541.8
18	604.86	459.42	652.92
19	560.09	475.83	695.52
20	585.09	492.24	782.71
21	-37.373	90.999	99.325
22	-34.88	94.498	94.041
23	-32.447	97.997	95.63
24	-52.129	101.5	106.8
25	-37.333	105	94.882
26	-103.89	67.043	57.712
27	-114.62	69.621	58.986
28	-120.05	72.2	54.844
29	-124.01	74.779	55.906
30	-120.17	77.357	56.941
31	56.668	56.218	58.724
32	57.595	58.379	58.315
33	50.58	60.54	63.18
34	79.963	62.701	61.615

35	70.683	64.862	68.378
36	1034.2	1022	995.85
37	1097.1	1061.3	1033.1
38	1121.6	1100.6	1082.3
39	1190.6	1139.9	1179.1
40	1244.3	1179.2	1190.7
41	638.1	642.72	631.49
42	668.63	667.43	658.34
43	692.83	692.15	707.84
44	742.13	716.86	756.03
45	711.31	741.58	779.49
46	465.52	480.46	529.22
47	468.47	498.92	574.27
48	530.87	517.39	625.46
49	560.6	535.85	710.36
50	541.03	554.31	789.52
51	1101.1	1176.2	1135.2
52	1174.1	1221.4	1192.4
53	1182.4	1266.6	1231.3
54	1291.1	1311.9	1298.1
55	1271.3	1357.1	1298.5
56	500.97	527.67	538.22
57	550.47	547.96	596.48
58	576.83	568.26	641.77
59	548.51	588.55	653.03
60	576.88	608.84	714.61



The program `H2_UseNN1` gives the predictions for specimen #7 and all the strain values.




```
>> H2_UseNN1
    strain    RealStress    Predicted
    ______    __________    _________

    0.001       9.1584       3.1531          ?????
    0.002        18.29        25.99
    0.003       27.422       34.617
    0.004       36.554       36.614
    0.005       45.686       45.903
    0.006       54.818       55.192
    0.007        63.95       64.481
```

6.2 NN Model1 (Direct)

```
Characteristics: "Standardize", true;
"LayerSizes",10;
"Activations", "relu";
IterationLimit=100,Lambda=0
```

```
>> H2_modelNN1D

RMSEValid=
       23.985
```

Nr	RealValue	NND
1	1793.4	1751.2
2	1862.4	1877
3	1931.4	1919.2b
4	2000.4	2035.6
5	2069.4	2067.1
6	1358.4	1347.6
7	1409.5	1435.9
8	1460.6	1471
9	1511.7	1496.8
10	1562.8	1532.9
11	691.2	658.42
12	717.77	719.43
13	744.33	748.96
14	770.9	781.51
15	797.46	761.74
16	426.61	563.99
17	443.02	598.33
18	459.42	686.84
19	475.83	736.81
20	492.24	829.11
21	90.999	33.024
22	94.498	18.488
23	97.997	37.633
24	101.5	30.626
25	105	51.785
26	67.043	35.395
27	69.621	32.712
28	72.2	52.718
29	74.779	62.093
30	77.357	38.266
31	56.218	49.459
32	58.379	57.133
33	60.54	57.705
34	62.701	56.72

```
    35     64.862      57.527
    36       1022      1044.1
    37     1061.3      1098.4
    38     1100.6      1128.5
    39     1139.9      1236.8
    40     1179.2      1248.9
    41     642.72      627.12
    42     667.43      665.68
    43     692.15      686.29
    44     716.86      696.36
    45     741.58      688.86
    46     480.46      553.68
    47     498.92      610.35
    48     517.39       674.5
    49     535.85      737.42
    50     554.31      798.96
    51     1176.2        1125
    52     1221.4      1191.8
    53     1266.6      1219.3
    54     1311.9      1305.5
    55     1357.1      1288.1
    56     527.67      544.26
    57     547.96      584.32
    58     568.26       625.9
    59     588.55      624.68
    60     608.84      635.05

RMSE=
       90.388
mdlLoss = 4101.1
```

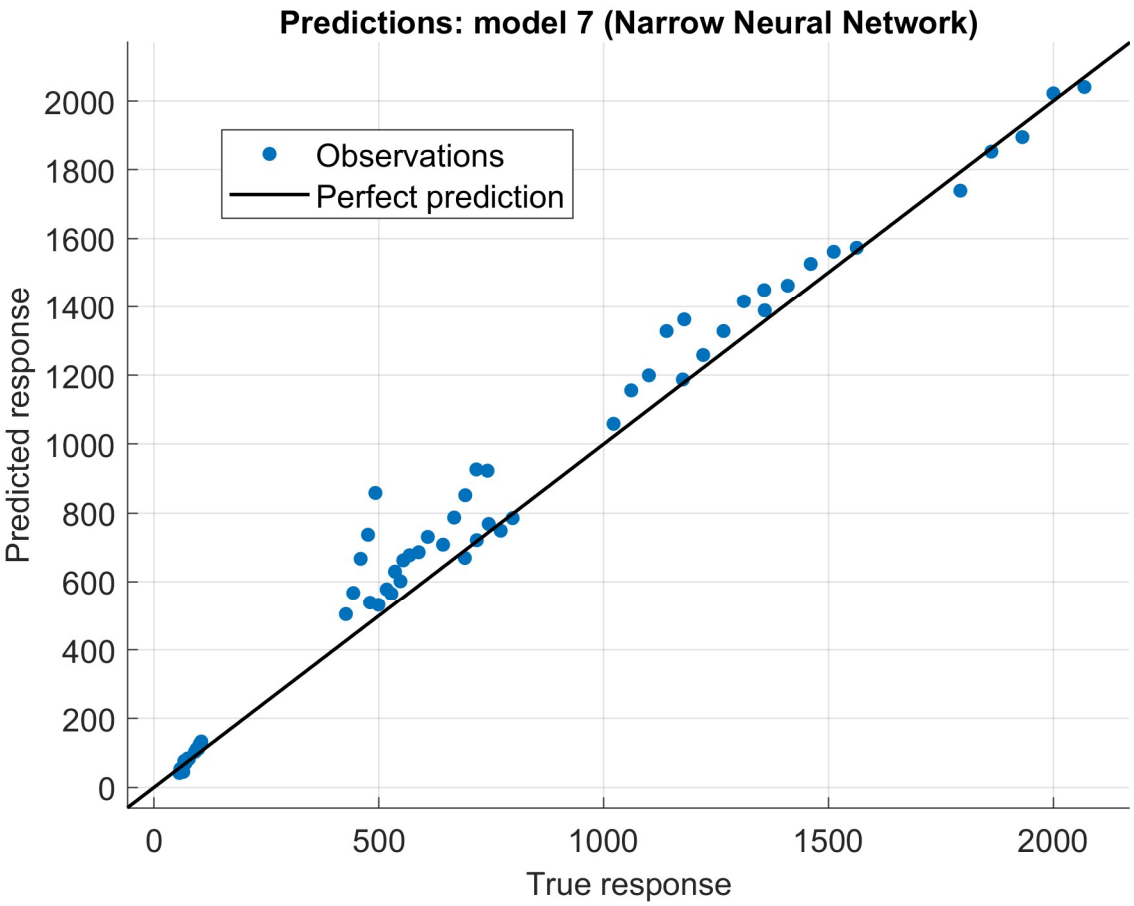


Figure 10_2_NN1 Direct

6.3 The second NN - Model10

NN with optimization

```
    'LayerSizes', [166 280 298], ...
    'Activations', 'relu', ...
    'Lambda', 3.63e-08, ...
    'IterationLimit', 1000, ...
    'Standardize', true);
```

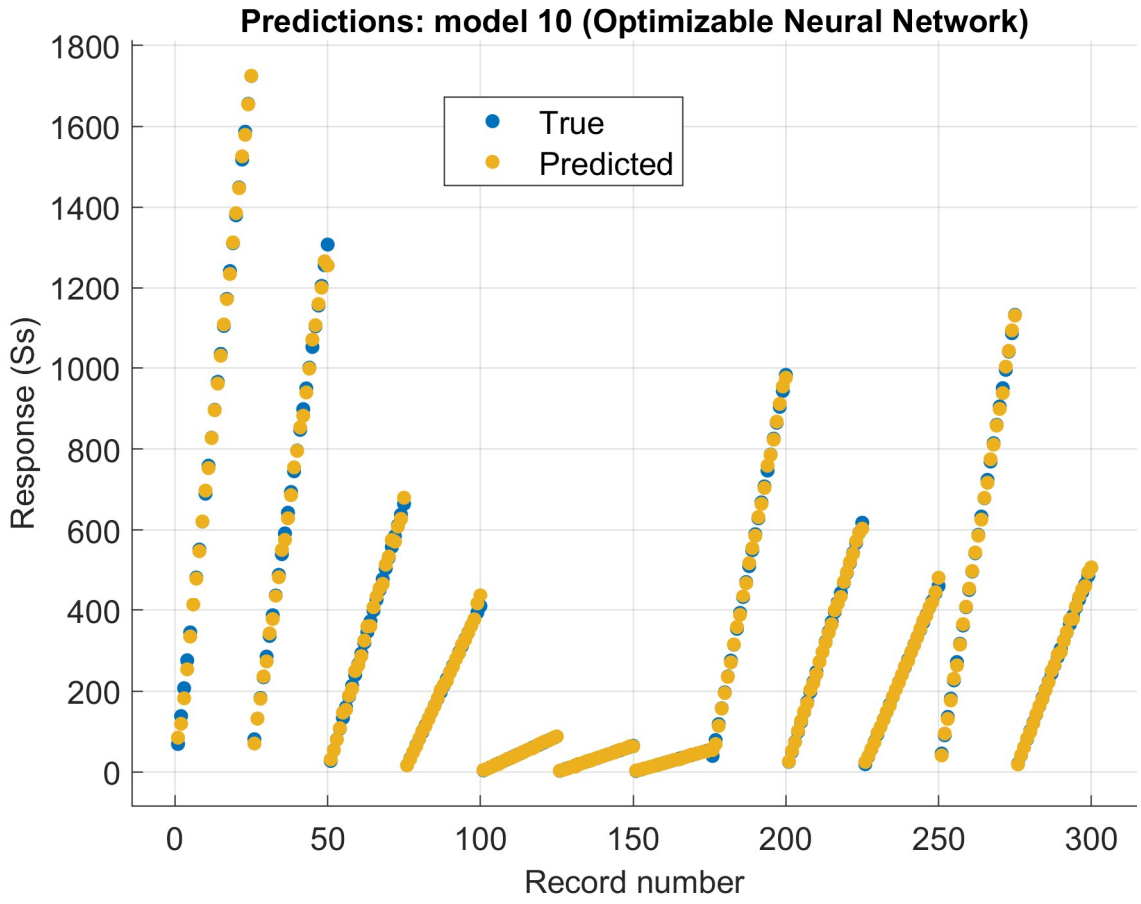
>> H2_modelNN2

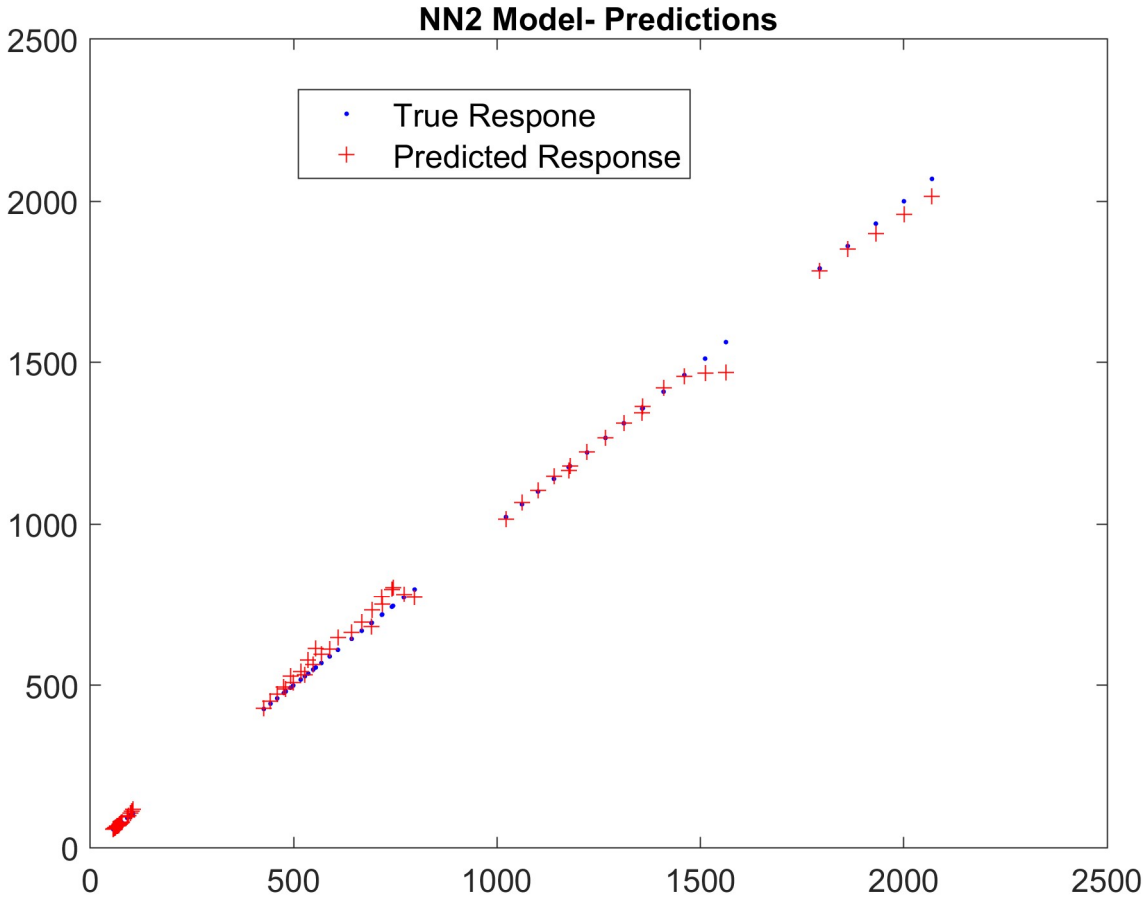
```
RMSE(Valid)=
      6.735
```

Nr	LM	RealValue	NN
1	1732	1793.4	1769.9
2	1860.7	1862.4	1825.2
3	1898.5	1931.4	1859.4
4	2020.2	2000.4	1918.9
5	2053	2069.4	1958
6	1280.4	1358.4	1363.9
7	1325.2	1409.5	1424.6
8	1350.6	1460.6	1449.5
9	1390.2	1511.7	1448.2
10	1453.7	1562.8	1442.6
11	776.57	691.2	670.6
12	830.27	717.77	723.63
13	817.05	744.33	774.62
14	924.42	770.9	742.99
15	862.47	797.46	732.08
16	590.41	426.61	426.63
17	458.26	443.02	445.04
18	604.86	459.42	465.19
19	560.09	475.83	481.25
20	585.09	492.24	510.72
21	**-37.373**	**90.999**	**92.233**
22	**-34.88**	**94.498**	**96.111**
23	**-32.447**	**97.997**	**102.73**
24	**-52.129**	**101.5**	**107.33**
25	**-37.333**	**105**	**113.38**
26	**-103.89**	**67.043**	**67.583**
27	**-114.62**	**69.621**	**70.387**
28	**-120.05**	**72.2**	**73.247**
29	**-124.01**	**74.779**	**76.896**
30	**-120.17**	**77.357**	**80.282**
31	56.668	56.218	55.927
32	57.595	58.379	57.377
33	50.58	60.54	59.053
34	79.963	62.701	60.786
35	70.683	64.862	62.169

36	1034.2	1022	1015.5
37	1097.1	1061.3	1073.3
38	1121.6	1100.6	1116.5
39	1190.6	1139.9	1168.5
40	1244.3	1179.2	1211.7
41	638.1	642.72	652.63
42	668.63	667.43	671.7
43	692.83	692.15	701.27
44	742.13	716.86	724.54
45	711.31	741.58	746.62
46	465.52	480.46	472.74
47	468.47	498.92	481.76
48	530.87	517.39	499.55
49	560.6	535.85	522.78
50	541.03	554.31	541.7
51	1101.1	1176.2	1152.9
52	1174.1	1221.4	1196
53	1182.4	1266.6	1231.5
54	1291.1	1311.9	1280.4
55	1271.3	1357.1	1302.3
56	500.97	527.67	525.6
57	550.47	547.96	555.78
58	576.83	568.26	578.95
59	548.51	588.55	586.37
60	576.88	608.84	609.34

Optimizer: Bayesian optimization
RMSE(test)=34.385
MSE(Test)=1182.3
MAE(Test)=19.82
Model size= 1MB


Predictions: model 10 (Optimizable Neural Network)
True
Predicted
Response (Ss)
Record number
0
200
400
600
800
1000
1200
1400
1600
1800
0
50
100
150
200
250
300


NN2 Model- Predictions
True Respone
Predicted Response
0
500
1000
1500
2000
2500

```
The program H2_UseNN2 gives the predictions for specimen #7 and
all the strain values and recalls the NN1's predictions.

>> H2_UseNN2
     The following table compares the predictions made by the two
neural networks.

    strain      NN1       RealStress      NN2
    ______    _______     __________    _______

    0.001     3.1531        9.1584       9.756
    0.002      25.99         18.29      18.649
    0.003     34.617        27.422      27.809
    0.004     36.614        36.554      36.448
    0.005     45.903        45.686      45.113
    0.006     55.192        54.818      54.097
    0.007     64.481         63.95      60.975
```

Conclusion

- The proposed way to enrich the dataset (the data-generating process) seems to be effective for treating this problem of using ML prediction.
- The accuracy of the predictions also depends on the accuracy of the initial data furnished by the measurements.
- An analysis should be conducted concerning some parameters' values (d, M, the number of specimens) to study the parameters' contribution to the models' accuracy. This analysis is reported for an ulterior stage of the project if it will be needed.
- The set of predictions must be enriched to validate the ML models over very large zones of the prediction space.

7. GENERALIZATION TEST USING THE MODEL NN2.

So far, the generalization accuracy of ML models has been tested using the testing datasets reserved for this objective. The testing dataset comes from the same initial traction tests; they have the same real physical support. The real generalization power of the ML model would be proven for data points that the ML model has never "seen".

In this section, we shall test the generalization power of the NN2 model, i.e., the best-generated model, considering data points corresponding to fictitious specimens.

Hypothesis: The data points are fictitious; they belong neither to the training nor the test data.

Objectif: To test the generalization aptitude of the NN2 model.

The script `H2_UseNN2_generalization` considers data points that are generated by changing the two middle orientations of specimen 8 (…45, 45…) with 4 pairs of values ([40,40], [20,20], [20,-20], [-20,-20])). It predicts the Stress for 30 strain values.

The **`strain`** column shows the 30 values of the strain equally spaced out in the given range. The predicted stress value is given in the **`pred_NN2`** column.
The column `ss_real_8` shows the stress value for specimen 8 at the same strain. This value can be used to compare the two specimens to see whether the change produced by the two new orientations is realistic.

```
>> H2_UseNN2_generalization
```

The fictitious specimens are "neighboring" S8:
PATTERN(8,:)=[0. 45. 0. 90. 0. -45. 0. **45. 45**. 0. -45. 0. 90. 0. 45. 0.];

New pattern: the middle values are changed
NEW PATTERN=[0. 45. 0. 90. 0. -45. 0. **40. 40.** 0. -45. 0. 90. 0. 45. 0.];

strain	ss_real_8	pred_NN2
0.00051266	39.36	44.474
0.0010253	78.666	85.151
0.001538	117.97	123.93
0.0020507	157.28	162.53
0.0025633	196.58	201.91
0.003076	235.89	243.06
0.0035886	275.2	283.33
0.0041013	314.5	322.53
0.004614	353.81	359.95
0.0051266	393.11	398.18
0.0056393	432.42	438.08
0.006152	471.73	479.12
0.0066646	511.03	518.58
0.0071773	550.34	557.92
0.00769	589.64	597.44
0.0082026	628.95	638.04
0.0087153	668.26	676.85
0.009228	707.56	715.06
0.0097406	746.87	753.4
0.010253	786.17	792.45

0.010766	825.48	832.01
0.011279	864.78	873.07
0.011791	904.09	913.47
0.012304	943.4	954.75
0.012817	982.7	1000.3
0.013329	1022	1048.9
0.013842	1061.3	1094.9
0.014355	1100.6	1135.9
0.014867	1139.9	1178.1
0.01538	1179.2	1222.1

New pattern: the middle values are changed
NEW PATTERN=[0. 45. 0. 90. 0. -45. 0. **20. 20.** 0. -45. 0. 90. 0. 45. 0.];

strain	ss_real_8	pred_NN2
0.00051266	39.36	64.802
0.0010253	78.666	104.27
0.001538	117.97	143.65
0.0020507	157.28	184.09
0.0025633	196.58	225.35
0.003076	235.89	266.53
0.0035886	275.2	308.15
0.0041013	314.5	349.8
0.004614	353.81	394.45
0.0051266	393.11	439
0.0056393	432.42	482.58
0.006152	471.73	522.31
0.0066646	511.03	562.62
0.0071773	550.34	602.58
0.00769	589.64	641.33
0.0082026	628.95	680.05
0.0087153	668.26	718.55
0.009228	707.56	757.06
0.0097406	746.87	797.97
0.010253	786.17	841.82
0.010766	825.48	887.6
0.011279	864.78	935.27
0.011791	904.09	982.03
0.012304	943.4	1027.7
0.012817	982.7	1075.9
0.013329	1022	1123.7
0.013842	1061.3	1164.5
0.014355	1100.6	1206.3
0.014867	1139.9	1248.5
0.01538	1179.2	1289.5

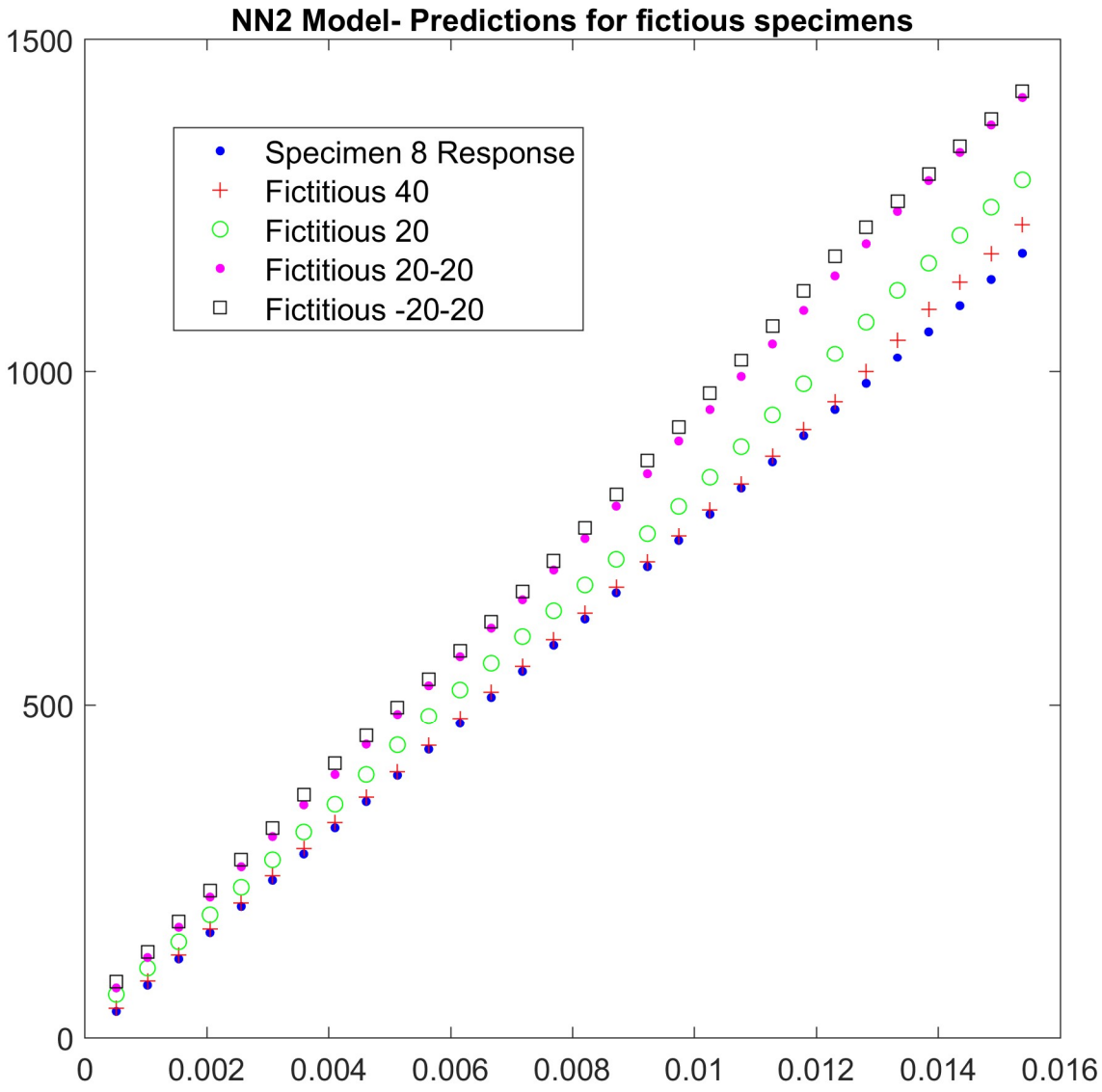
New pattern: the middle values are changed
NEW PATTERN=[0. 45. 0. 90. 0. -45. 0. **20. -20.** 0. -45. 0. 90. 0. 45. 0.];

strain	ss_real_8	pred_NN2
0.00051266	39.36	74.537
0.0010253	78.666	120.06
0.001538	117.97	165.44
0.0020507	157.28	210.68
0.0025633	196.58	256.08
0.003076	235.89	301.33
0.0035886	275.2	348.86
0.0041013	314.5	394.38
0.004614	353.81	439.93
0.0051266	393.11	485.3
0.0056393	432.42	528.65
0.006152	471.73	572.35
0.0066646	511.03	615.23
0.0071773	550.34	657.91
0.00769	589.64	702.49
0.0082026	628.95	749.76
0.0087153	668.26	798.28
0.009228	707.56	847.01
0.0097406	746.87	896.05
0.010253	786.17	943.3
0.010766	825.48	993
0.011279	864.78	1043.2
0.011791	904.09	1093.5
0.012304	943.4	1145.3
0.012817	982.7	1193.5
0.013329	1022	1242.2
0.013842	1061.3	1288.4
0.014355	1100.6	1330.6
0.014867	1139.9	1371.8
0.01538	1179.2	1413

New pattern: the middle values are changed
NEW PATTERN=[0. 45. 0. 90. 0. -45. 0. **-20. -20.** 0. -45. 0. 90. 0. 45. 0.];

strain	ss_real_8	pred_NN2
0.00051266	39.36	83.895
0.0010253	78.666	128.42
0.001538	117.97	174.22
0.0020507	157.28	220.58
0.0025633	196.58	266.74

0.003076	235.89	314.29
0.0035886	275.2	364.12
0.0041013	314.5	411.43
0.004614	353.81	453.36
0.0051266	393.11	495.92
0.0056393	432.42	538.56
0.006152	471.73	581.36
0.0066646	511.03	625.21
0.0071773	550.34	670.05
0.00769	589.64	716.49
0.0082026	628.95	766.19
0.0087153	668.26	816.25
0.009228	707.56	867.16
0.0097406	746.87	917.29
0.010253	786.17	967.76
0.010766	825.48	1017.8
0.011279	864.78	1070.1
0.011791	904.09	1122.9
0.012304	943.4	1174.6
0.012817	982.7	1218.4
0.013329	1022	1257.1
0.013842	1061.3	1298.5
0.014355	1100.6	1339.8
0.014867	1139.9	1381.1
0.01538	1179.2	1422.4



The difficulty is in validating the predictions in this stage. Verifying the predictions using a simulation platform or, even better, making real tensile tests would be necessary. Our colleagues from P2 will accomplish this task.

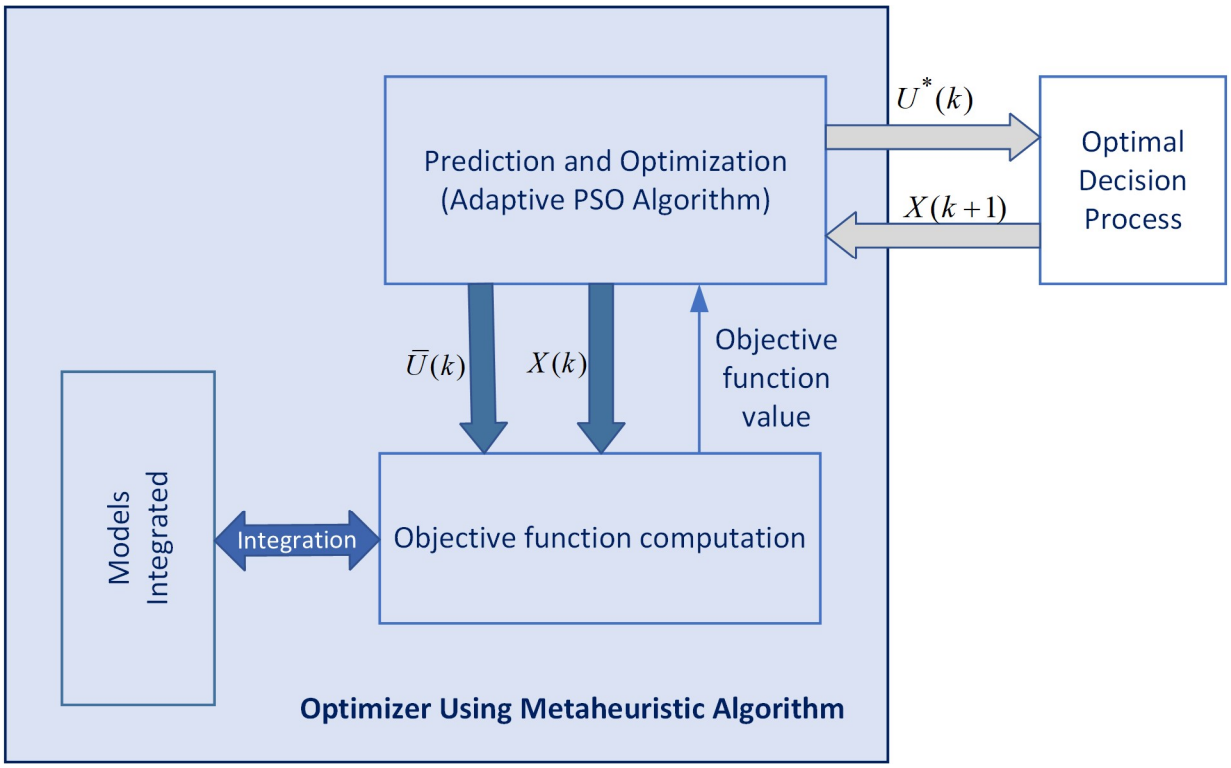
PART II

8. ML algorithms that Emulate Metaheuristic Algorithms for optimal decision-making

Employing Metaheuristic algorithms for the optimal decision-making process.

Some optimization problems need a metaheuristic algorithm (MA) in searching for the optimal solution, especially when the optimization function has difficult characteristics (distributed parameters, nonlinearities, etc.). That is a vast subject, already treated in the literature, that renders MAs realistic candidate tools for the optimization modules (*optimizers*). These tools are robust and flexible but sometimes involve very large computation efforts. There is an interesting and effective approach that allows replacing an MA algorithm with an ML algorithm only in the execution phase to reduce drastically the computation effort.

This report's authors have proposed the analysis of possible "equivalence" between ML algorithms and MA within two simulated studies.



II.1 Metaheuristic algorithm for optimal decision-making

In Figure II.1, the optimal decision-making is presented as a closed-loop evolution, where:

- X ($X \in \boldsymbol{X}$) is the set of variables whose evolution must be optimized, and U^* is the optimal decision variable set.
- The objective function is a function of vectors X and U, which must be optimized.
- The variable k covers the situation when the decision process is recursive. If this is not the case, the indexation (k and k+1) must not be considered.

In this figure, the optimizer is based on the Particle Swarm Optimization metaheuristic (an adaptive version, Adaptive PSO Algorithm). Any other metaheuristic can be used; a very realistic way

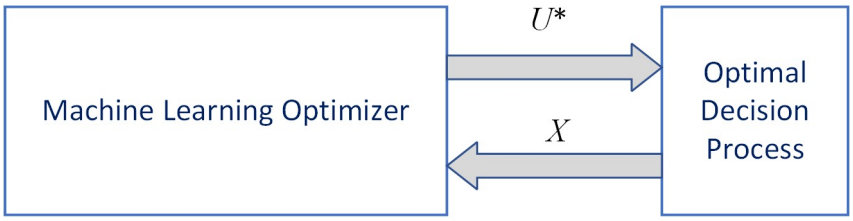
to construct optimizers is to use Genetic Algorithms (Evolutionary Algorithms). Generally speaking, we can use an appropriate MA for the optimal decision problem at hand.

Roughly speaking, the optimizer responds to the following problem: "When the process is characterized by the variables set X, what are the values of the decision variables U that optimize the objective function?". The optimizer finds the solution: U*.

The following paper presents research results obtained in the framework of this project, and it describes the principle of the possible equivalence between ML algorithms and MAs for optimal decision-making.

Mînzu, V.; Arama, I.; Rusu, E. **Machine Learning Algorithms That Emulate Controllers Based on Particle Swarm Optimization**—An Application to a Photobioreactor for Algal Growth. Processes 2024, 12, 991, https://doi.org/10.3390/pr12050991.

Remark 1: As a general principle, if an optimizer is constructed using an MA, then we can achieve an ML algorithm that "captures" the optimality of the MA. The ML algorithm can emulate the MA (Figure II.2).



II.2 The ML optimizer that is "equivalent" to an MA optimizer

We can construct a new optimizer, called ML optimizer, which is simpler than the previous one because it contains only the trained ML model without a searching process or integrations.

In the mentioned paper, this principle is applied to a specific optimal decision process, the optimal control of a dynamic process. (Because a specific control structure was adopted, a process model is considered as an example: a photobioreactor). *All of these do not affect the generality of the presentation.*

Remark 2: The paper presents general principles and implementation aspects of the ML algorithm, which "captures" the optimality of the APSOA, even though it exemplifies the procedure using a particular optimizer.

Why is our desideratum to replace the APSOA (or any MA) optimizer?

Owing to extensive searchings for the optimal solution inside space $\boldsymbol{X}$ and possible numerous numerical integrations, there is a big computational effort that leads to a large optimizer execution time. The main motivation of this work was to decrease the computational effort and, consequently, the optimizer execution time. This work proposed replacing the APSOA with an ML model that has "learned" the optimal behavior of the APSOA.

Remark 3. The training data are obtained through simulations of a large enough set of decision-making using the MA optimizer (here, the APSOA).

The data set generation supposes a large enough number of optimizer's simulations using APSOA. After each simulation of the optimizer, a couple of vectors (X, U*) are recorded (see Figure II.1); that is a data point. The simulations are conducted in the initial phase to collect the data points for the training and testing.

A design procedure is given below.

Design Procedure

1. Write the "APSOA Optimizer" program for the considered optimization problem based on the APSOA, which finds the quasi-optimal solution U^* for a given initial vector X.
2. Repeat M times the execution of "APSOA Optimizer" to produce M sequences (X, U) and save them in a data structure.
3. Choose, construct, and test an algorithm called "Optimizer_ML" that emulates the optimal behavior of the APSOA. This step is repeated until an accurate and appropriate model is found.
4. Integrate the "Optimizer_ML" into the final optimization program.

The design procedure can also be followed mutatis mutandis when the optimization problem is solved using evolutionary algorithms (such as genetic algorithms). The program at step 1 must use the new MA to search for the quasi-optimal solution. Details for this case can be found in the following paper, having the same authors:

Mînzu, V.; Arama, I. *A Machine Learning Algorithm That Experiences the Evolutionary Algorithm's Predictions—An Application to Optimal Control;* Mathematics 2024, 12(2), 187. https://doi.org/10.3390/math12020187.

The new optimizer should preserve the optimal behavior of the decision-making process. In identical conditions, the ML decision process must also be quasi-optimal.

Remark 4. The "Optimizer_ML" emulates the "APSOA Optimizer," which means that both have the same behavior; they give near identical quasi-optimal solutions.

Because the linear regression could seem much too simple, we have also studied other types of models (trees, support vector machines, Regression Neural Networks, and Gaussian processes) trying to improve capturing the optimality, the final target being that the designed ML optimizer would better approach the optimal solution. For the case study presented in the article mentioned above, the obtained models perform less than those of the Linear Regression and RNN models. In other optimization cases, many kinds of ML algorithms must also be analyzed, considering the ML optimizer's size. It is a design matter.

Conclusion:

- When we solve a new optimization problem, sometimes we need an MA (PSO, EA, etc.) that searches for the optimal solution U^* inside of the optimization space $\boldsymbol{U}$. If the optimizer's execution time is not acceptable, we can use the approach presented before to create an ML optimizer. However, initially, we need the MA to search for the optimal solution.
- The ML optimizer replaces the MA optimizer only in execution when the optimal decision is made. So, the optimizer's execution time is diminished.

The article mentioned above gives more details concerning the principle and the implementation of this approach. These details are not repeated in this presentation. In addition, the Supplementary Materials attached to this article contain all the MATLAB programs that can be used in our next work.

PART III

9. The generalization power of the ML algorithms already developed.

This part describes how we can test the generalization power of ML models already developed in the first part of this report. The generalization power of the ML model would be proven for data points that the model has never "seen"; that is, they belong neither to the training data nor the test data. So, the first problem is to generate data points situated in the model's

The following text was written to propose an article with authors from UGAL and P2 in the future.

4. Implementation of Machine Learning Models

This section is dedicated to the intricate process of generating ML models for the stress-strain dependence during the tensile test of different specimens. The initial step involves preparing the datasets for the learning process and constructing the parametric and nonparametric ML models.

As mentioned, the main objective is to construct ML models that would apprehend all the measurements described before and predict the Stress value for any pattern (combination of orientations) and a given Strain value. Our work's specific objectives were set with the anticipation of achieving this ultimate goal in mind:

1. Generate a dataset of significant size. This dataset will be used to construct the ML model, enabling it to provide a generalized response for any pattern and an appropriate Strain value.
2. Construct a parametric model (e.g., multiple linear regression) that is easy to understand and apply and can be compared with the following models.
3. Construct some nonparametric models (SVM, decision trees, Gaussian process regression, and neural networks), analyze their accuracy, and compare them to the parametric model. Out of many ML models investigated, we chose to present two nonparametric models (SVM and Regression NN). The last ones yielded four trained and tested models, the most effective and appropriate to the considered data set.
4. Carefully select the most accurate parametric models that could be used in further research, providing a solid foundation for future studies.

Remark 1: For our problem, many SVM and Regression NN models could be constructed, some of them having potentially superior capabilities to predict the behavior of the stencils. Our objective was not to find their best but to validate our approach: to prove that ML models can accurately predict stress-strain behavior.

4.1 Data Preparation

*** This part is covered by PART 1.***

4.2 Construction of ML Models

4.2.1 A multiple linear regression model

The first ML model constructed to fit the data set is parametric: a multiple linear regression model that allows the possibility of including nonlinear terms as the interactions, that is, the product of predictor variables. The model maintains its linearity about its coefficients.

Out of the linear regression models developed in our work, we present only that based on the step-wise regression strategy. The latter consists of adding or removing features from a constant model. In the MATLAB system used in our implementation, this strategy is implemented by a specialized function `step-wise` (T), which returns a model that fits the dataset in T.

The features of this model are named $x_1,\ldots,x_{16}$ for the orientations α_1, $\alpha_2,\ldots$, α_{16}, and `St` and `Ss` for the Strain and Stress, respectively. Appendix A describes the results obtained using the step-wise regression strategy. One of the best linear regression models for Stress has the following structure:

```
Ss ~ 1 + x1 + x2 + x9 + x12 + x13 + St +
     + x1*x9 + x1*St + x2*x13 + x2*St + x9*St + x12*St + x13*St
```

Besides the intercept and terms corresponding to six predictors, all the other terms are interactions of the predictors. Its coefficients are given in Appendix A as well. Figure 22 presents the predicted and training values for all 300 training records (data points). Figure 23 shows a global image of the model's generalization efficiency using the 60 data points.

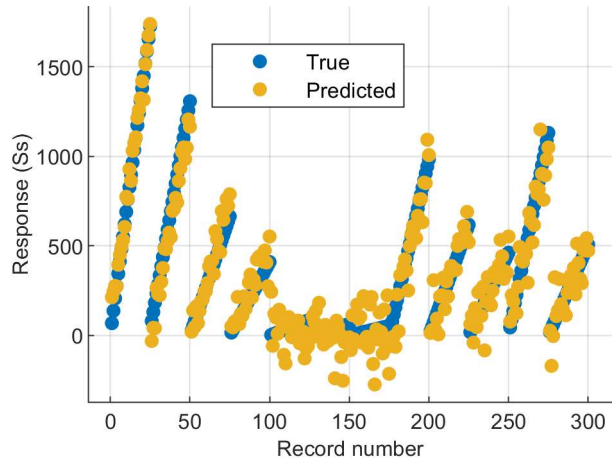


Figure 22. Predicted versus real values for the training data set

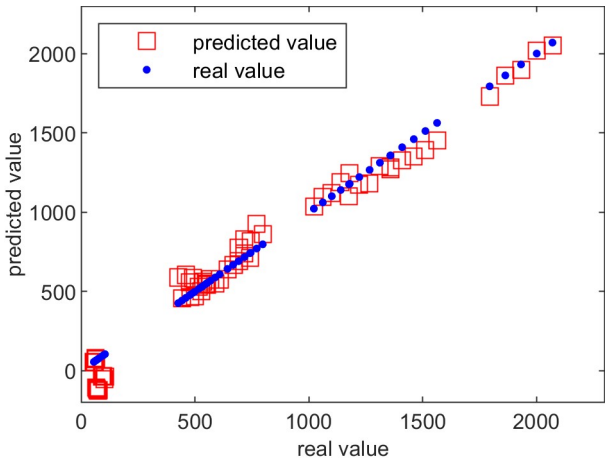


Figure 23. Predicted versus real values for the test data set

Usually, the training and test process results can be characterized by statistics, allowing the ML models constructed using the same data set to be compared. The statistics given in Table 22 for different ML models are the Root mean squared error (RMSE), R-squared, and Mean absolute error (MAE) values (see [77]).

	Statistics	**SW Linear Regression**	**SVM Regression 1**	**SVM Regression 2**	**RNN1**	**RNN2**
Training results	RMSE	52.045	34.93	46.903	41.135	6.375
	R-Squared	0.98	0.99	0.98	0.99	1.
	MAE	37.42	28.324	31.44	19.132	3.9465
Test results	RMSE	91.091	52.108	86.383	99.206	34.385
	R-Squared	0.97	0.99	0.98	0.97	1.
	MAE	66.667	43.38	66.255	67.875	19.829
Model size		22 kB	16 kB	40 kB	8 kB	1 MB

Table 22. Statistics of the training and test process results and model size for different ML models.

Let's notice the statistics characterizing the Step-Wise Linear Regression model presented in this sub-section, given in the column **SW Linear Regression**. These will allow us to ascertain the superiority of the next proposed ML models. Besides the first column, Table 22 also presents the statistics of the four trained and tested ML models described in the next subsections. The model size is also given.

Although the regression model has good predictions for most specimens, it does not give good predictions for those with a small stress range; their behavior is not accurately "learned." Figures 22 and 23 show certain data points for which the predicted Stress value is negative. There are ten such data points, which we shall call critical, corresponding to the specimens whose Stress range is very narrow. These bad predictions are given in Table 23.

Stress True value	90.999	94.498	97.997	101.5	105.	67.043	69.621	72.2	74.779	77.357
Stress Predicted value	-46.32	-23.93	-29.72	-18.4	-27.3	-83.92	-118.7	-102.	-121.2	-129.2

Table 23. The set of the worst predictions made by the SW Linear Regression Model.

This fact led to investigating *nonparametric* ML models capable of overpassing this drawback.

Remark 2: Besides ameliorating the statistics, improving the critical data points' predictions was challenging for the new ML models. The following subsections present only models with better prediction capabilities, including the critical data points.

4.2.2 Support Vector Machine Models

Support Vector Machine generated good ML models for our problem. Out of the SVM models constructed in our work, we present, in the sequel, only two SVM models responding to our objectives. The first SVM Model, called **SVM Regression 1**, has a cubic Kernel function and a set of intern Hyperparameters (as defined within MATLAB system – see [88]): PolynomialOrder, Standardize, KernelScale, BoxConstraint, Epsilon. Appendix A gives details concerning the Hyperparameters of the SVM Regression 1.

This model was trained and tested using the Regression Learner application (see [88]), which led to the results presented in Table 22, column **SVM Regression 1**. The RMSE values are smaller than those of the **SW linear Regression** model, proving that the SVM works better. Figures 24 and 25 illustrate this statement if compared to Figures 22 and 23.

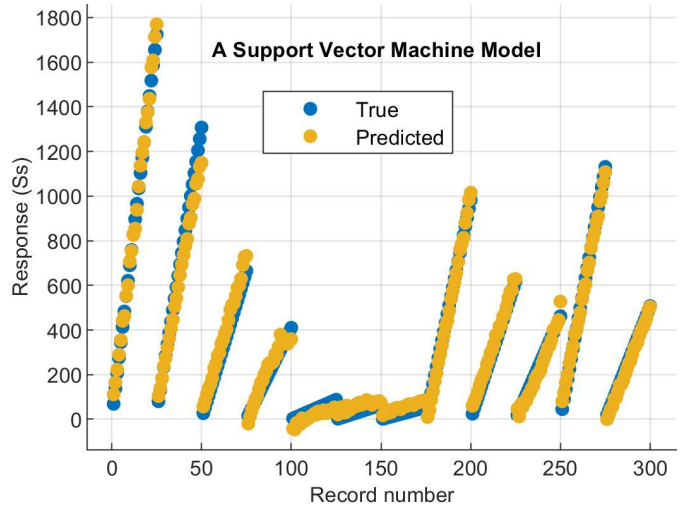


Figure 24. Predicted versus real values for the training data set - SVM Regression 1

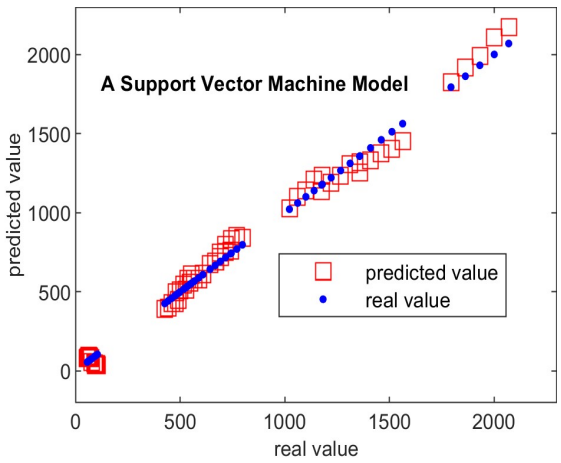


Figure 25. Predicted versus real values for the test data set - SVM Regression 1

The predictions for the critical data points are given in Table 24. Although they are not very good, they are better than those in Table 23, at least because they are positive.

Stress True value	90.999	94.498	97.997	101.5	105.	67.043	69.621	72.2	74.779	77.357
Stress Predicted value	47.679	35.616	39.292	42.684	34.877	82.007	89.711	81.055	81.242	56.943

Table 24. The set of the critical predictions made by the SVM Model 1.

We also constructed another SVM model that predicts very well the critical data points, whose statistics are given in a column called **SVM Regression 2**. The Hyperparameters' Model is the following:

Preset: Optimisable SVM,
Kernel function: Quadratic
Kernel scale: Automatic.

The training process uses Bayesian optimization to optimize the combination of hyperparameters.

Table 25 shows excellent predictions for the critical data points, but the statistics corresponding to this new SVM model are inferior to those of the **SVM Regression 1.**

Stress True value	90.999	94.498	97.997	101.5	105.	67.043	69.621	72.2	74.779	77.357
Stress Predicted value	93.074	95.691	100.18	105.26	110.17	68.82	71.944	74.032	77.312	80.144

Table 25. The set of the critical predictions made by the SVM Model 2.

Moreover, the new model size, 40 kB, is larger than the first. In conclusion, owing to most of its characteristics, the **SVM Regression 1** model can be considered better than the second one.

4.2.3 Regression Neural Network Models

This subsection presents two other nonparametric ML models using Regression Neural Networks. The first one uses a Narrow Neural Network in MATLAB system terminology. Its statistics are shown in column **RNN 1** of Table 22.

Details concerning the model RNN 1 are given in Appendix A. The hyperparameters are optimized using heuristic procedures. The RMSE and MAE values are better than the previous models in Table 22, showing better predicting accuracy. Moreover, the model's size is the smallest of all presented ML models, having 8 kB.

The predicted Stress values for the critical data points are very good, like those presented in Table 25, proving that this problem is also solved.

The more accurate prediction is obtained using another Regression NN, the **RNN 2** model, whose statistics are displayed in the last column of Table 22. Details concerning the model **RNN 2** are given in Appendix A. It is an RNN with 3 layers whose hyperparameters are found using Bayesian optimization.

Table 26 shows that the predictions for the critical data points are very good, the best in comparison with the previous models.

Stress True value	90.999	94.498	97.997	101.5	105.	67.043	69.621	72.2	74.779	77.357
Stress Predicted value	92.233	96.111	102.73	107.33	113.38	67.583	70.387	73.247	76.896	80.282

Table 26. The set of the critical predictions made by the RNN 2.

The price to pay for the very good accuracy of RNN 2 is the larger size of the model, which is 1 MB.

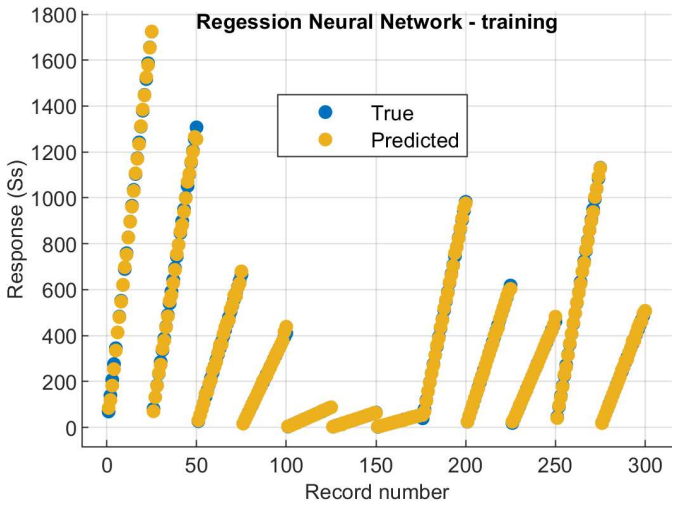


Figure 26. Predicted versus real values - the training of RNN2

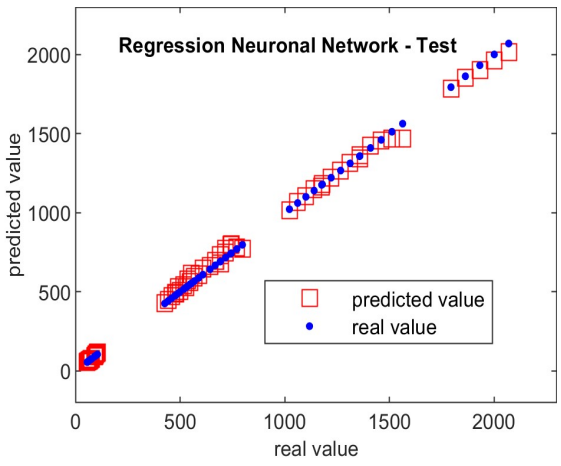


Figure 27. Predicted versus real values - the test of RNN2

Figures 26 and 27 show the efficiency of the training and test processes, respectively, and prove that RNN2 is the more accurate ML model for the given data set. According to how the ML model is used, RNN1 can replace RNN2 and be a good solution for our prediction problem due to its small model size and good accuracy.

10. 5. Forecasting New Stratification Combinations

This section will present how to exploit the ML models presented in Section 4, that is, to replicate and predict the behavior of carbon fiber-epoxy composites for different orientations, including novel stratification combinations.

5.1 Stress-strain predictions for new combinations

So far, the generalization accuracy of ML models has been tested using the testing datasets reserved for this objective. The testing dataset comes from the same initial traction tests; they have the same real physical support. The generalization power of the ML model would be proven for data points that the model has never "seen"; that is, they belong neither to the training data nor the test data.

The RNN2 model was used as the most performant ML model for stress prediction in our tests.

This subsection considers the case when a new combination has the same structure as one that already contributed to the ML model construction, differing from this in only a few layer orientations. For example, we can generate novel stratification combinations derived from the S8's pattern as "neighbors" of this one: the middle sequence 45/45 is replaced by α/α. The new pattern, denoted Snew, is given below:

Snew = [0/45/0/90/0/-45/0/α/α/0/-45/0/90/0/45/0].

Under this hypothesis, the ML model can cover and predict the new combination's behavior.

Four values (denoted α) have been considered that generated the four stratifications presented below for the composite materials. Figure 50 presents the predicted and real Stress values for α=45° and, for comparison, the predicted and simulated Stress values for α=40°. The blue curve is shorter because it stops at the beginning of the damaged zone; the Digimat VA simulation program can determine the latter. Currently, the predictions do not consider the damaged zones.

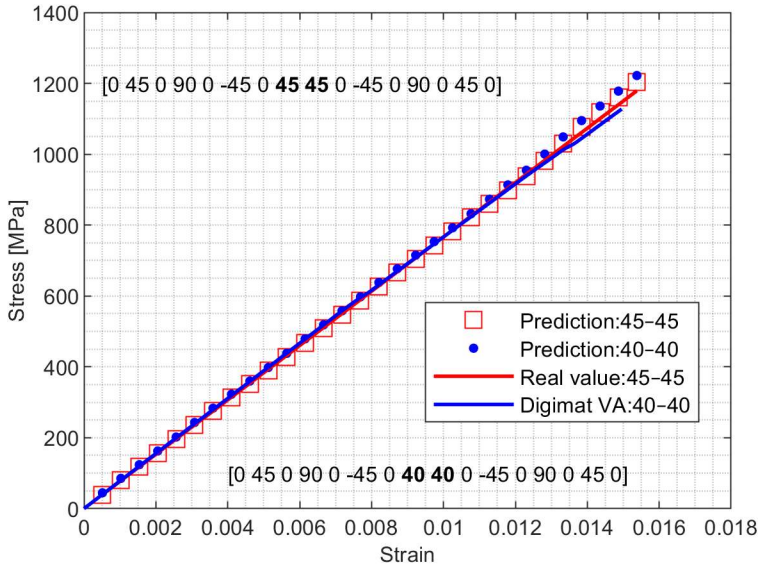


Figure 51. Predicted and real/simulated Stress values for α=45° and α=40°.

Figure 51 presents the predicted and simulated Stress values for α=20° and, for comparison, α=30°. The continuous blue and red curves are shorter because they stop at the beginning of the damaged zone.

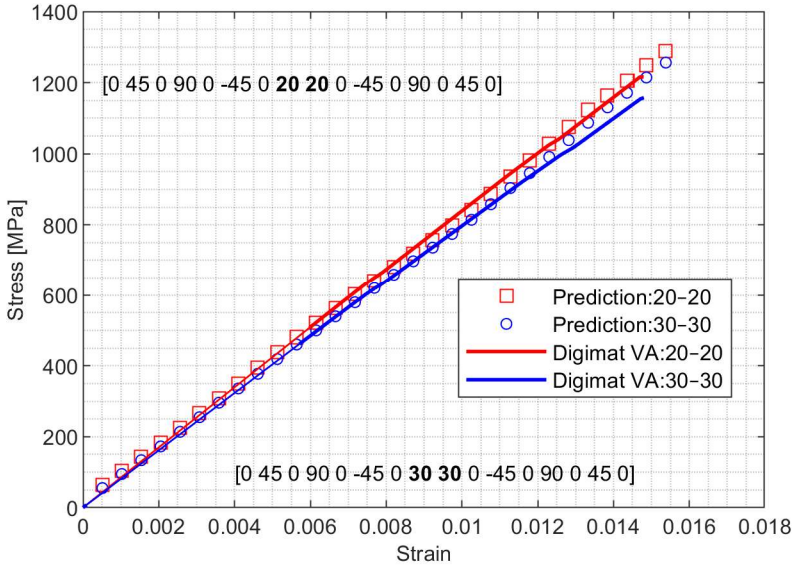


Figure 52. Predicted and simulated Stress values for α=20° and α=30°.

Figures 51 and 52 involve the following remark.

Remark 3. Two aspects can be stated:

- The predictions made by the ML model are very good inside the considered elasticity zones.
- Predictions give more significant errors at the end of the elasticity zones while remaining within acceptable limits.

Beyond the opportunity to compare predictions to real/simulated values, these examples based on the Snew pattern suggest how to solve a possible peculiar problem that seeks the most resistant stratification having a given pattern.

5.1 Stress-strain predictions for new random combinations

In this part of our work, we considered specimens with randomly generated layer orientations, which, in other words, did not contribute to the dataset used to construct the ML model. Then, we compared the predictions for these specimens made by the ML model with DIGIMAT simulation results. Only four specimens with randomly generated combinations are considered to make this presentation easy to follow. To remain inside the ML model's generalization area, we first chose four base specimens submitted to tensile tests, PAT2, PAT6, PAT9, and PAT11, that contributed to the dataset used to train the ML model. They have different behaviors in the space Strain−Stress. Each layer orientation of these specimens was modified independently using a uniformly distributed perturbation in the range [-4°, +4°]. The resulting specimens with randomly generated orientations are NEW2, NEW6, NEW9, and NEW11, which are quite different from the initial ones but remain in the model representation area. For example, Figure 53bis shows the sixteen orientation values of NEW2 and base specimen PAT2.

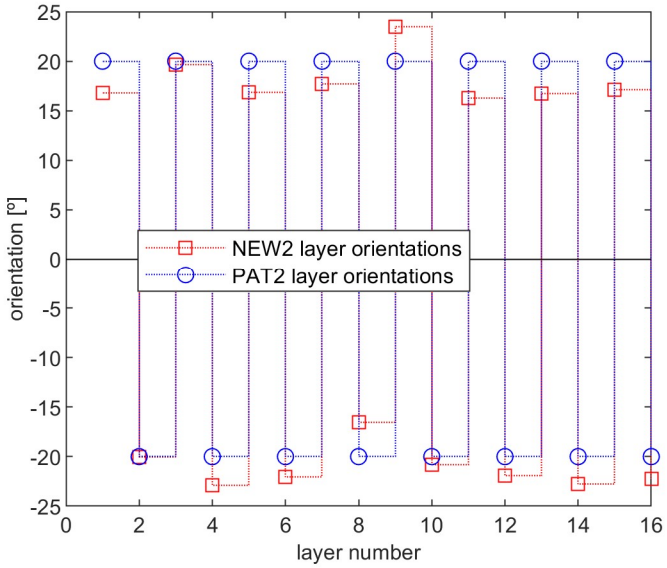


Figure 53bis. Layer orientations of NEW2 and PAT2

Table A1 in APPENDIX B gives the layer orientations for all the base and perturbated specimens and the difference between them (DIFF2, DIFF6, DIFF9, DIFF11). Because the base specimens contributed to the dataset used to train and test the ML model, their stress predictions given strain values are already accurate. The accuracy of the stress prediction must be verified for the new specimens by comparing them with the values given by the DIGIMAT simulations.

Figure 5.3 presents all the curves obtained through simulation and prediction and ascertains the very good prediction accuracy of the ML model made for the four new specimens. The pairs of curves having the same color prove that there is a small prediction error for all strain values.

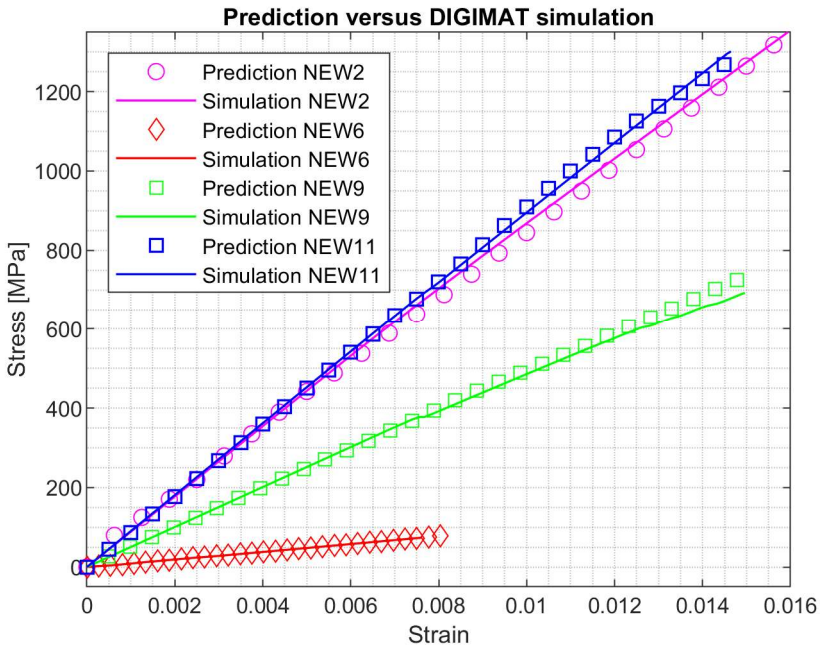


Figure 53. Comparison between the predicted and DIGIMAT values for the four randomly generated specimens.

To zoom in on the prediction error, Figure 54 shows the prediction relative error in a certain number of points situated in the Strain range considered in Figure 53.

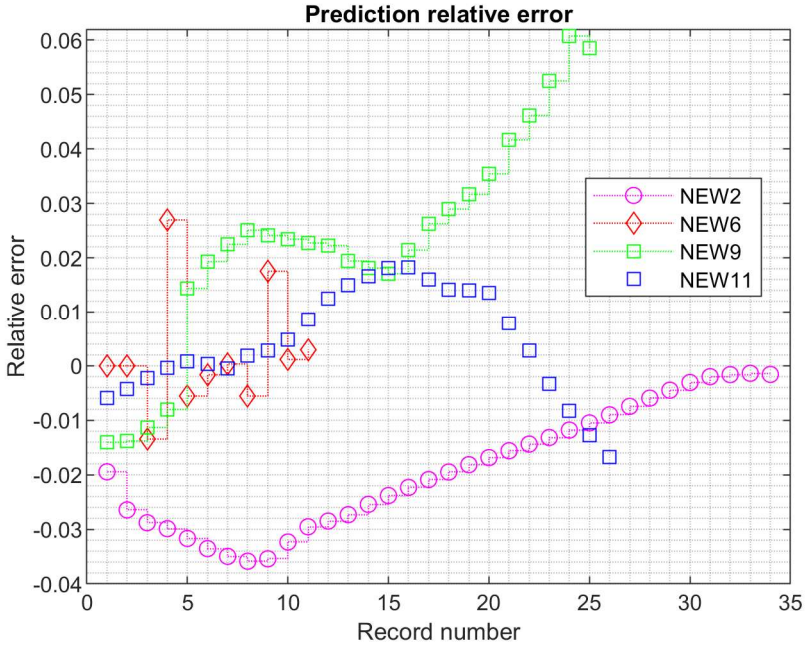


Figure 54. Relative prediction errors for the four randomly generated specimens.

The following equation gives the prediction relative error:

$$\text{prediction relative error} = \frac{\text{predicted stress value - simulated stress value}}{\text{simulated stress value}}$$

The relative error is placed in the interval [-0.04, 0.06], which means the prediction accuracy is greatly satisfactory. The proposed ML model has a good generalization power if it is appropriately employed, that is,

- the specimens are inside the ML model representation domain and
- the strain values are inside the range corresponding to the elasticity zone.

Remark 4. Generally, there is no procedure to verify if the first constraint is rigorously met. Depending on the dataset and practical application, the user can consider a certain ML model representation domain a priori and estimate if this constraint is met. After that, reliable predictions can be made.

APPENDIX A

Details concerning the SW Linear Regression model

The regression model is obtained using the step-wise function:

```
model ← stepwise(T)
```

*** This part is already presented in PART 1 ***

Details concerning the SVM Regression 1

Hyperparameters Model:
Preset Quadratic SVM
Kernel function: Cubic
Kernel scale 3.001
Box constraint: Automatic
Epsilon Auto
Standardisation data: Yes
Optimiser: Not applicable

Details concerning model RNN 1

Hyperparameters Model:
Preset Narrow NN
Number of fully connected layers 1
First Layer size 10
Activation ReLU
Iteration limit 1000
Regularisation strength (lambda) 0
Standardisation data: Yes
Optimiser: Not Applicable

Details concerning model RNN 2

Preset Optimizable NN
Iteration limit: 1000
Optimiser: Bayesian optimisation

The training of RNN 2 is made using the `fitrnet` function:

```
RegNN = fitrnet(...
```

```
predictors, ...
response, ...
'LayerSizes', [166 280 298], ...
'Activations', 'relu', ...
'Lambda', 3.6315e-08, ...
'IterationLimit', 1000, ...
'Standardise', true);
```

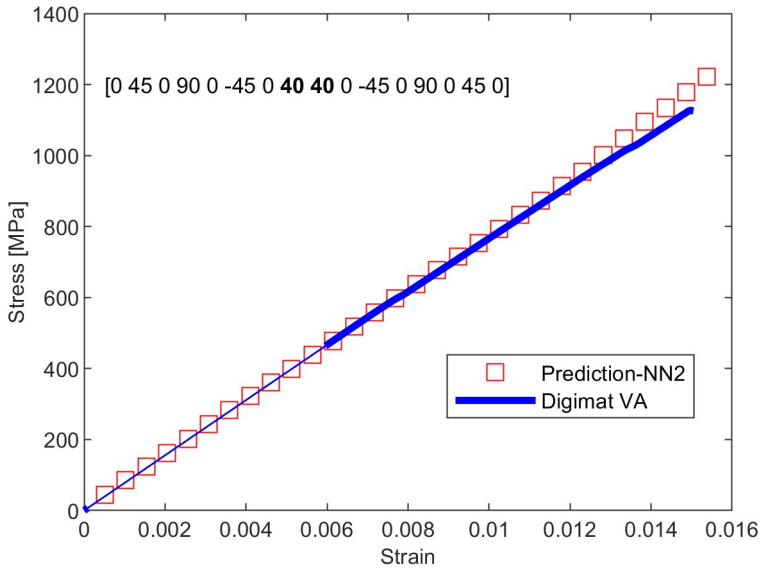


Figure 5.1 Prediction versus simulation for the new pattern (a)

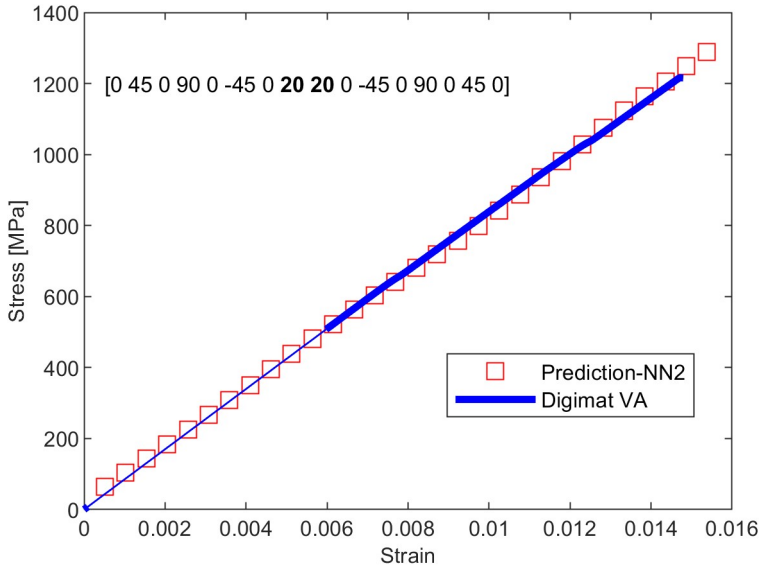


Figure 5.2 Prediction versus simulation for the new pattern (b)

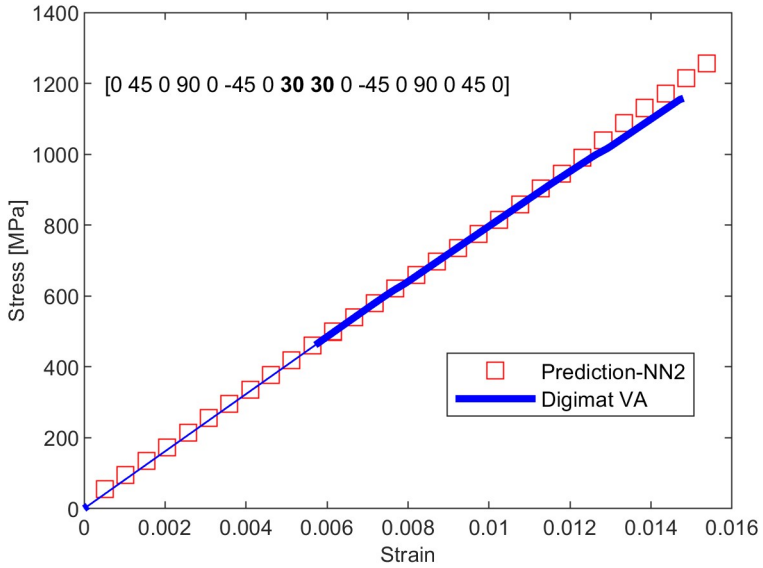


Figure 5.3 Prediction versus simulation for the new pattern (c)

APPENDIX B

	ang1	ang2	ang3	ang4	ang5	ang6	ang7	ang8
PAT2	20.00	-20.00	20.00	-20.00	20.00	-20.00	20.00	-20.00
NEW2	16.80	-20.05	19.67	-22.92	16.85	-22.06	17.68	-16.53
DIFF2	-3.20	-0.05	-0.33	-2.92	-3.15	-2.06	-2.32	3.47
PAT6	70.00	-70.00	70.00	-70.00	70.00	-70.00	70.00	-70.00
NEW6	71.47	-69.27	68.30	-70.32	73.18	-71.33	67.00	-68.95
DIFF6	1.47	0.73	-1.70	-0.32	3.18	-1.33	-3.00	1.05
PAT9	45.00	0.00	-45.00	90.00	45.00	0.00	-45.00	90.00
NEW9	46.59	3.31	-42.30	88.52	48.57	-1.31	-44.54	88.57
DIFF9	1.59	3.31	2.70	-1.48	3.57	-1.31	0.46	-1.43
PAT11	0.00	30.00	0.00	90.00	0.00	-30.00	0.00	30.00
NEW11	-3.49	26.87	-0.14	89.83	2.48	-33.05	3.61	32.53
DIFF11	-3.49	-3.13	-0.14	-0.17	2.48	-3.05	3.61	2.53

	ang9	ang10	ang11	ang12	ang13	ang14	ang15	ang16
PAT2	20.00	-20.00	20.00	-20.00	20.00	-20.00	20.00	-20.00
NEW2	23.49	-20.79	16.26	-21.91	16.71	-22.76	17.14	-22.26
DIFF2	3.49	-0.79	-3.74	-1.91	-3.29	-2.76	-2.86	-2.26
PAT6	70.00	-70.00	70.00	-70.00	70.00	-70.00	70.00	-70.00
NEW6	69.55	-73.38	67.92	-67.99	73.58	-72.32	69.08	-70.79
DIFF6	-0.45	-3.38	-2.08	2.01	3.58	-2.32	-0.92	-0.79
PAT9	90.00	-45.00	0.00	45.00	90.00	-45.00	0.00	45.00
NEW9	90.23	-48.59	3.12	45.28	92.95	-41.92	-1.90	42.29
DIFF9	0.23	-3.59	3.12	0.28	2.95	3.08	-1.90	-2.71
PAT11	30.00	0.00	-30.00	0.00	90.00	0.00	30.00	0.00
NEW11	29.95	1.22	-28.86	-0.13	92.30	-0.28	26.92	-0.82
DIFF11	-0.05	1.22	1.14	-0.13	2.30	-0.28	-3.08	-0.82

Table A.1 The layers' orientations of the basic and new random specimens and their differences.