Cloud Publications
International Journal of Advanced Packaging Technology
2016, Volume 4, Issue 1, pp. 169-176, Article ID Tech-555
ISSN 2349-6665, doi 10.23953/cloud.ijapt.21

Research Article Open Access

Predicting Corrugated Box Compression Strength Using an Artificial Neural Network

Siripong Malasri, Prasanth Rayapati and Divya Kondeti

Engineering Management Graduate Program, Christian Brothers University, 650 East Parkway South, Memphis, TN, USA

Correspondence should be addressed to Siripong Malasri, pong@cbu.edu

Publication Date: 29 January 2016

DOI: https://doi.org/10.23953/cloud.ijapt.21

Abstract McKee formula has been widely used to predict the compression strength of corrugated boxes. An experimental verification, published in early 2015, showed the inaccuracy of the formula. McKee formula left out several important factors, including box height, temperature, and humidity. An artificial neural network, CBU-BOX1, was developed based on 74 cases of cubical RSC single-wall corrugated boxes from 3”x3”x3” to 36”x36”x36”. Box height, temperature and humidity data were included in the network development. CBU-BOX1 performance ranged from 0% to 26.3% error with an average error of 6.9% while McKee formula performance ranged from 0.6% to 149.3% error with an average error of 28.7%. However, CBU-BOX1 performance dropped significantly when it was used for rectangular boxes. Out from twelve test cases of rectangular boxes, the formula resulted in an average error of 25.7% while CBU-BOX1 resulted in 34.8%. Thus, the network is unacceptable for rectangular boxes. In this study, 67 more cases were added to the previous 74 cases. Out of 141 cases, 43 were rectangular boxes. CBU-BOX2 significantly outperformed McKee formula with an average error of 9.21% versus 30.79%.
Keywords *Artificial Neural Network; Compression Strength; McKee Formula; Corrugated Boxes*

1. Introduction

McKee formula has been widely used to predict the compression strength of corrugated boxes. An experimental verification [1] showed the inaccuracy of the formula, which left out several important factors, including box height, temperature, and humidity.

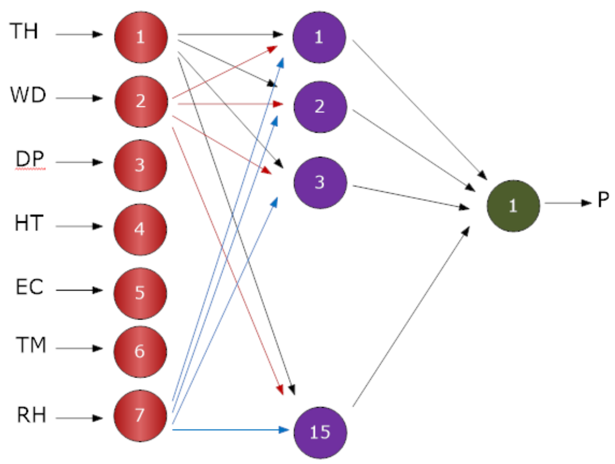
An artificial neural network is software capable of learning from examples. It has been successfully used in transport packaging [2]. The first version of the box compression strength neural network [3], CBU-BOX1, was developed based on data of 74 cases of cubical RSC single-wall corrugated boxes. This neural network outperformed McKee formula with an error range of 0% - 26.3% versus 0.6% - 149.3% with an average error of 6.9% versus 28.7%. However, when CBU-BOX1 was applied to rectangular boxes, its performance was worse than that of McKee formula [4].

In this study additional cubical and rectangular boxes at different temperature and humidity were added. The second version of box compression strength neural network, CBU-BOX2, was developed.

2. Materials and Methods

Twenty-four cubical boxes were added to the previous 74 cases, which gave a total of 98 cubical boxes. Forty-three rectangular boxes were also added. Thus, a total of 141 cases of RSC single-wall corrugated boxes were included in this study. An ECT test was performed for each box compressed. Some boxes were placed in a temperature/humidity chamber at different combinations of temperature and humidity, while some boxes were placed at room condition, which was about the standard test environment of 73°F and 50% RH. Box dimensions (width/depth/height) ranged from 3" to 36" with a temperature range from 66°F to 104°F, and a humidity range from 48% to 80%.

A feed-forward fully- connected Backpropagation neural network shown in Figure 1 was used. The numbers of input and out neurons were controlled by the collected data, i.e., seven input parameters and one output parameter. The number of hidden neurons was arbitrary and was chosen as 15 in this work. Other training parameters were shown in Figure 2. The sigmoid function, $y = \frac{1}{1+e^{-x}}$, was used to generate an output (y) of each hidden and output cell from a weighted sum of connection weight and input vectors (x). NeuroShell2 [5] was used to train CBU-BOX2 neural network. Figure 3 shows some features of NeuroShell2 software. Once training reached a satisfactory performance, a generic source code was generated for software application development in any programming language. Fourteen samples out of 141 cases of training data are shown in Table 1. In the "Mark" column, "T" was used for 114 training cases and "V" for validation for 27 cases.



Input	Output
Wall Thickness (in) - TH Box Width (in) – WD Box Depth (in) – DP Box Height (in) – HT ECT (lb/in) – EC Temperature (°F) – TM Relative Humidity (%) - RH	Maximum Load (lb) - P

Figure 1: *CBU-BOX2 Neural Network Configuration*

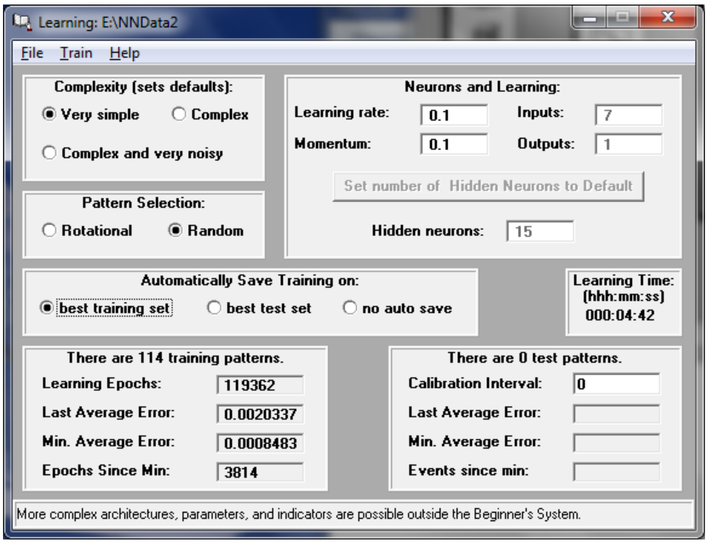


Figure 2: *Training Parameters*

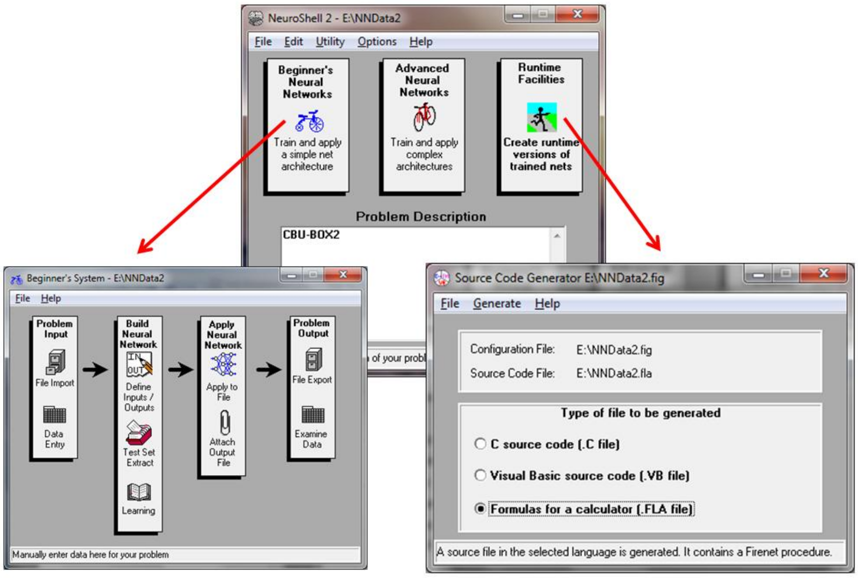


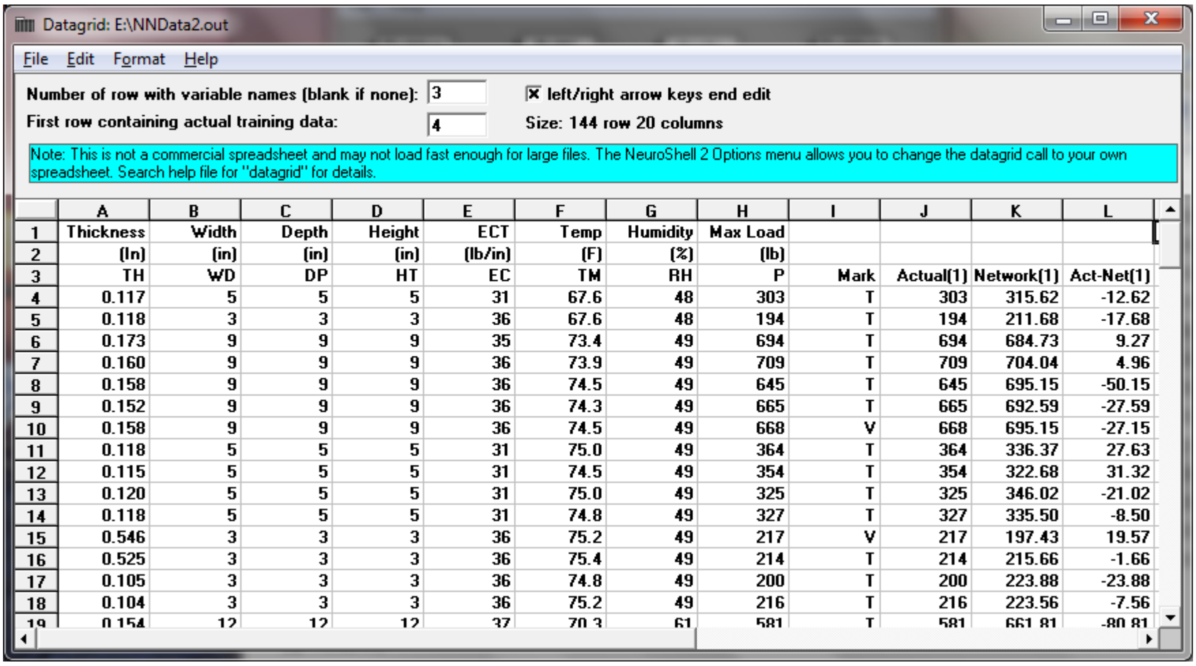
Figure 3: *NeuroShell2 Neural Network Development Software*

Table 1: *Training Data Samples*

TH	WD	DP	HT	EC	TM	RH	P	Mark
0.16	9	9	9	17	83.1	58	341	V
0.145	9	9	9	17	78.4	57	344	T
0.151	12	12	12	14	79.2	58	411	T
0.148	12	12	12	14	88.3	57	439	T
0.151	12	12	12	14	81.1	57	462	T
0.156	16	16	16	17	89.8	56	589	V
0.156	16	16	16	17	88.2	57	534	T
0.159	16	16	16	17	87.9	57	592	T
0.141	4	4	12	15	72.0	53	319	T
0.141	4	4	12	17	72.5	53	275	T
0.082	4	4	4	16	71.4	53	232	V
0.082	4	4	4	14	71.8	53	215	T
0.058	4	4	6	24	72.0	53	355	T
0.098	4	4	6	17	72.3	53	334	T

3. Results and Discussion

Figure 4 shows a partial output generated by CBU-BOX2 neural network. For each of the 141 cases, NeuroShell2 compared the predicted strength, i.e., Network(1), and the actual strength, i.e., Actual(1). It also outputted the difference of the two figures, i.e., Act-Net(1).



Datagrid: E:\NNData2.out

File Edit Format Help

Number of row with variable names (blank if none): 3

First row containing actual training data: 4

left/right arrow keys end edit

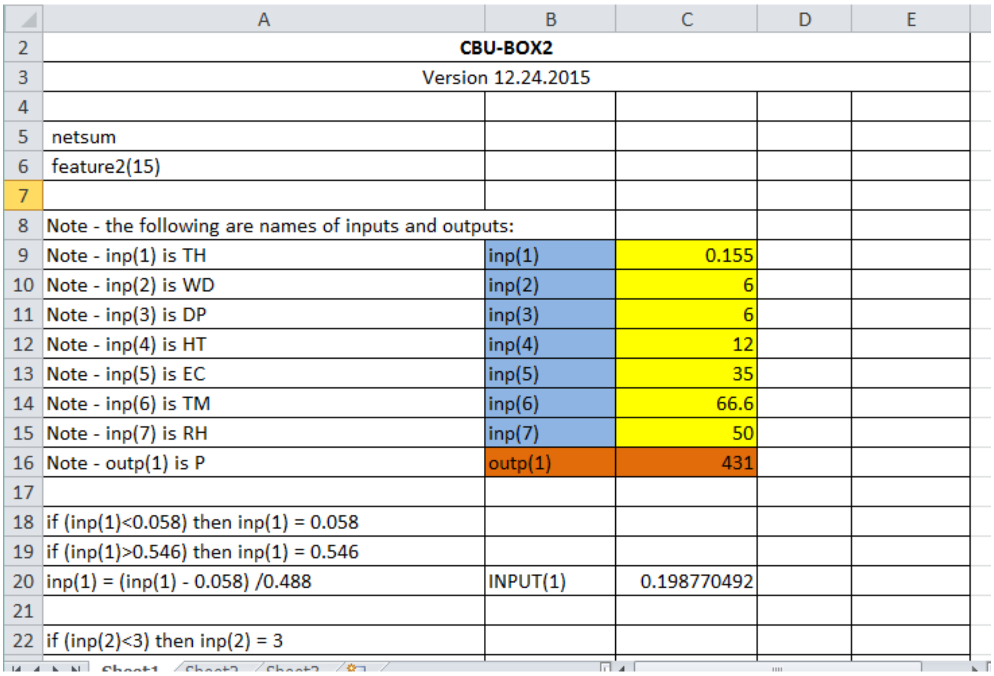
Size: 144 row 20 columns

Note: This is not a commercial spreadsheet and may not load fast enough for large files. The NeuroShell 2 Options menu allows you to change the datagrid call to your own spreadsheet. Search help file for "datagrid" for details.

	A	B	C	D	E	F	G	H	I	J	K	L
1	Thickness	Width	Depth	Height	ECT	Temp	Humidity	Max Load				
2	(In)	(in)	(in)	(in)	(lb/in)	(F)	(%)	(lb)				
3	TH	WD	DP	HT	EC	TM	RH	P	Mark	Actual(1)	Network(1)	Act-Net(1)
4	0.117	5	5	5	31	67.6	48	303	T	303	315.62	-12.62
5	0.118	3	3	3	36	67.6	48	194	T	194	211.68	-17.68
6	0.173	9	9	9	35	73.4	49	694	T	694	684.73	9.27
7	0.160	9	9	9	36	73.9	49	709	T	709	704.04	4.96
8	0.158	9	9	9	36	74.5	49	645	T	645	695.15	-50.15
9	0.152	9	9	9	36	74.3	49	665	T	665	692.59	-27.59
10	0.158	9	9	9	36	74.5	49	668	V	668	695.15	-27.15
11	0.118	5	5	5	31	75.0	49	364	T	364	336.37	27.63
12	0.115	5	5	5	31	74.5	49	354	T	354	322.68	31.32
13	0.120	5	5	5	31	75.0	49	325	T	325	346.02	-21.02
14	0.118	5	5	5	31	74.8	49	327	T	327	335.50	-8.50
15	0.546	3	3	3	36	75.2	49	217	V	217	197.43	19.57
16	0.525	3	3	3	36	75.4	49	214	T	214	215.66	-1.66
17	0.105	3	3	3	36	74.8	49	200	T	200	223.88	-23.88
18	0.104	3	3	3	36	75.2	49	216	T	216	223.56	-7.56
19	0.154	12	12	12	37	70.3	61	581	T	581	661.81	-80.81

Figure 4: *A Partial Output Generated by NeuroShell2*

A generic source code was generated by NeuroShell2 (Appendix A). An Excel spreadsheet developed based on the source code is shown in Figure 5.



	A	B	C	D	E
2	CBU-BOX2				
3	Version 12.24.2015				
4					
5	netsum				
6	feature2(15)				
7					
8	Note - the following are names of inputs and outputs:				
9	Note - inp(1) is TH	inp(1)	0.155		
10	Note - inp(2) is WD	inp(2)	6		
11	Note - inp(3) is DP	inp(3)	6		
12	Note - inp(4) is HT	inp(4)	12		
13	Note - inp(5) is EC	inp(5)	35		
14	Note - inp(6) is TM	inp(6)	66.6		
15	Note - inp(7) is RH	inp(7)	50		
16	Note - outp(1) is P	outp(1)	431		
17					
18	if (inp(1)<0.058) then inp(1) = 0.058				
19	if (inp(1)>0.546) then inp(1) = 0.546				
20	inp(1) = (inp(1) - 0.058) /0.488	INPUT(1)	0.198770492		
21					
22	if (inp(2)<3) then inp(2) = 3				

Figure 5: *CBU-BOX2 Excel Application*

The CBU-BOX2 performance was evaluated and compared with McKee formula performance in Table 3, Figure 6, and Appendix B.

Table 3: *CBU-BOX2 & McKee Formula Performance Comparison*

Category	Number Of Cases	<=5% Error	>5% to 10% Error	>10% to 20% Error	>20% Error	*Error Range (Average)*
CBU-BOX2 (All Cases)	141	60 or 42.55%	30 or 21.28%	35 or 24.82%	16 or 11.35%	*0% – 56.98% (Avg = 9.21%)*
McKee (All Cases)	141	13 or 9.22%	16 or 11.35%	24 or 17.02%	88 or 62.41%	*0.33% – 149.27% (Avg =30.79%)*
CBU-BOX2 (Cases with V Mark)	27	11 or 40.74%	4 or 14.81%	6 or 22.22%	6 or 22.22%	*1.26% - 40.28% (Avg = 11.30%)*
McKee (Cases with V Mark)	27	1 or 3.70%	2 or 7.41%	7 or 25.93%	17 or 62.96%	*2.08% - 149.27% (Avg = 37.46%)*
CBU-BOX2 (Rectangular)	43	20 (46.51%)	8 (18.60%)	11 (25.58%)	4 (9.30%)	*0.06% - 36.49% (Avg = 8.27%)*
McKee (Rectangular)	43	2 (4.65%)	2 (4.65%)	5 (11.63%)	34 (79.07%)	*3.14% - 62.59% (Avg = 37.05%)*
CBU-BOX2 (Cubical)	98	40 (40.82%)	22 (22.45%)	24 (24.49%)	12 (12.24%)	*0% - 56.98% (Avg = 9.47%)*
McKee (Cubical)	98	11 (11.22%)	14 (14.29%)	19 (19.39%)	54 (55.10%)	*0.33% - 149.27% (Avg = 28.04%)*

Figure 6: *CBU-BOX2 & McKee Formula Performance Comparison*

4. Conclusion

As seen from Table 3, Figure 6, and Appendix B, CBU-BOX2 significantly outperforms McKee formula, including the unseen cases, i.e., those cases with a “V” mark. Table 3 also shows that CBU-BOX2 performs well for both rectangular and cubical boxes. The 141 cases used to develop CBU-BOX2 cover wide ranges of box sizes and conditions, i.e., width/depth/height ranges from 3” to 36”, temperature range from 66°F to 104°F, and humidity range from 48% to 80%.

In order to make the neural network more comprehensive, material (virgin versus recycled) should be added as an input parameter. This can be accomplished by testing recycled corrugated boxes of different sizes and environmental conditions. With additional data, a new version of neural network can be trained and developed.

References

[1] Aloumi, B., Alnashwan, W., Malasri, S., Othmani, A., Kist, M., Sampson, N., Polania, S., Sanchez-Luna, Y., Johnson, M. and Fotso, R. *Experimental Verification of McKee Formula.* International Journal of Advanced Packaging Technology. 2015. 3 (1) 129-137.

[2] Malasri, S. *Applications of Neural Networks in Transport Packaging.* Proceedings of the PACKCON 2015 Conference. 2015. 15-20.

[3] Malasri, S., Baroth, S., Cherukuri, S., Gurram, S., Vadlapatla, H. and Moats, R. *Estimating Corrugated Box Strength.* Proceedings of the PACKCON 2015 Conference. 2015. 11-14.

[4] Wadlington, C., Nguyen, N., Zheng, Zhihong, Loving, B. and Malasri, S. *Validation of CBU-BOX1 Neural Network.* Proceedings of the PACKCON 2015 Conference. 2015. 21-24.

[5] Ward Systems Group, Inc. NeuroShell2, http:www.wardsystems.com/neuroshell2.asp (As of December 25, 2015)

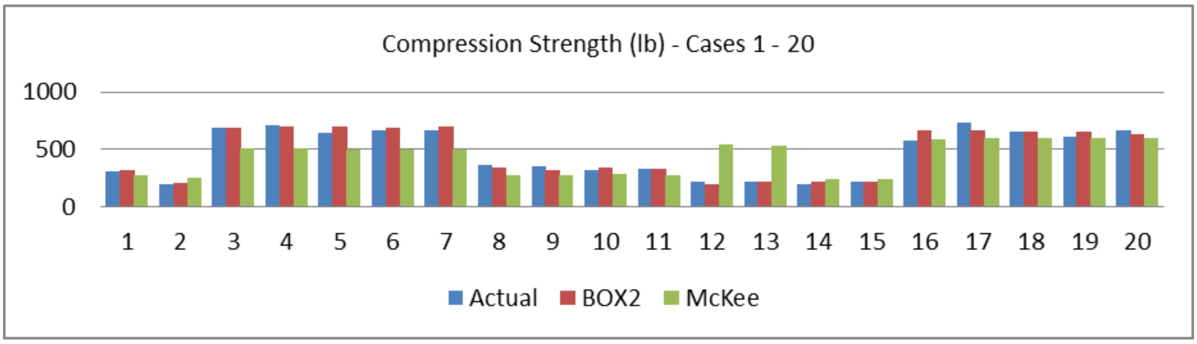
Appendix A Generic Source Code Generated by NeuroShell2

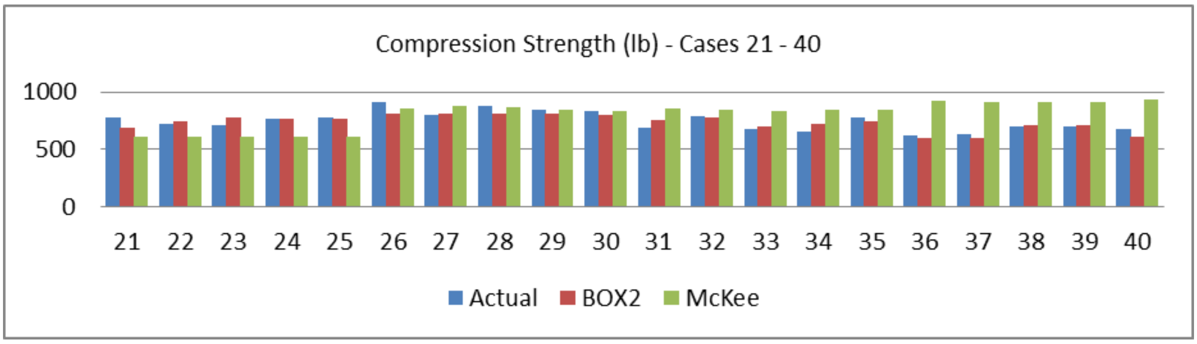
```
netsum
feature2(15)
Note - inp(1) is TH
Note - inp(2) is WD
Note - inp(3) is DP
Note - inp(4) is HT
Note - inp(5) is EC
Note - inp(6) is TM
Note - inp(7) is RH
Note - outp(1) is P
if (inp(1)<0.058) then inp(1) = 0.058
if (inp(1)>0.546) then inp(1) = 0.546
inp(1) = (inp(1) - 0.058) /0.488
if (inp(2)<3) then inp(2) = 3
if (inp(2)>36) then inp(2) = 36
inp(2) = (inp(2) - 3) /33
if (inp(3)<3) then inp(3) = 3
if (inp(3)>36) then inp(3) = 36
inp(3) = (inp(3) - 3) /33
if (inp(4)<3) then inp(4) = 3
if (inp(4)>36) then inp(4) = 36
inp(4) = (inp(4) - 3) /33
if (inp(5)<12) then inp(5) = 12
if (inp(5)>40) then inp(5) = 40
inp(5) = (inp(5) - 12) /28
if (inp(6)<66) then inp(6) = 66
if (inp(6)>104) then inp(6) = 104
inp(6) = (inp(6) - 66) /38
if (inp(7)<48) then inp(7) = 48
if (inp(7)>80) then inp(7) = 80
inp(7) = (inp(7) - 48) /32
netsum = netsum + inp(4) * -0.6943372
netsum = netsum + inp(5) * -0.8443508
netsum = netsum + inp(6) * 1.534944
netsum = netsum + inp(7) * -7.410185
feature2(4) = 1 / (1 + exp(-netsum))
netsum = -0.6371768
netsum = netsum + inp(1) * -63.56625
netsum = netsum + inp(2) * -10.43079
netsum = netsum + inp(3) * -9.066187
netsum = netsum + inp(4) * -3.963259
netsum = netsum + inp(5) * -20.94891
netsum = netsum + inp(6) * 11.87389
netsum = netsum + inp(7) * 31.26726
feature2(5) = 1 / (1 + exp(-netsum))
netsum = -7.49949
netsum = netsum + inp(1) * 0.6761363
netsum = netsum + inp(2) * -7.165943
netsum = netsum + inp(3) * -5.918962
netsum = netsum + inp(4) * 2.023466
netsum = netsum + inp(5) * -0.4293676
netsum = netsum + inp(6) * 8.238652E-03
netsum = netsum + inp(7) * -5.399711
feature2(6) = 1 / (1 + exp(-netsum))
netsum = -9.673411
netsum = netsum + inp(1) * 57.20142
netsum = netsum + inp(2) * 52.36789
netsum = netsum + inp(3) * 38.6922
netsum = netsum + inp(4) * -86.51951
netsum = netsum + inp(5) * 0.2131806
netsum = netsum + inp(6) * 7.545617
netsum = netsum + inp(7) * -31.17518
netsum = netsum + inp(3) * -4.812082
netsum = netsum + inp(4) * 0.6202544
netsum = netsum + inp(5) * -0.610349
netsum = netsum + inp(6) * 0.9391114
netsum = netsum + inp(7) * -5.874597
feature2(11) = 1 / (1 + exp(-netsum))
netsum = -2.139933
netsum = netsum + inp(1) * -14.47617
netsum = netsum + inp(2) * -4.172575
netsum = netsum + inp(3) * 14.07384
netsum = netsum + inp(4) * -26.67177
netsum = netsum + inp(5) * 9.429369
netsum = netsum + inp(6) * 11.32987
netsum = netsum + inp(7) * -4.766322
feature2(12) = 1 / (1 + exp(-netsum))
netsum = -0.2524938
netsum = netsum + inp(1) * -1.072484
netsum = netsum + inp(2) * 8.087201E-02
netsum = netsum + inp(3) * 20.43565
netsum = netsum + inp(4) * -11.70324
netsum = netsum + inp(5) * 0.3484668
netsum = netsum + inp(6) * -56.12461
netsum = netsum + inp(7) * -2.363098
feature2(13) = 1 / (1 + exp(-netsum))
netsum = -3.89846
netsum = netsum + inp(1) * -1.38506
netsum = netsum + inp(2) * -5.276136
netsum = netsum + inp(3) * -3.536175
netsum = netsum + inp(4) * 0.1051574
netsum = netsum + inp(5) * 1.241501
netsum = netsum + inp(6) * 3.577389
```

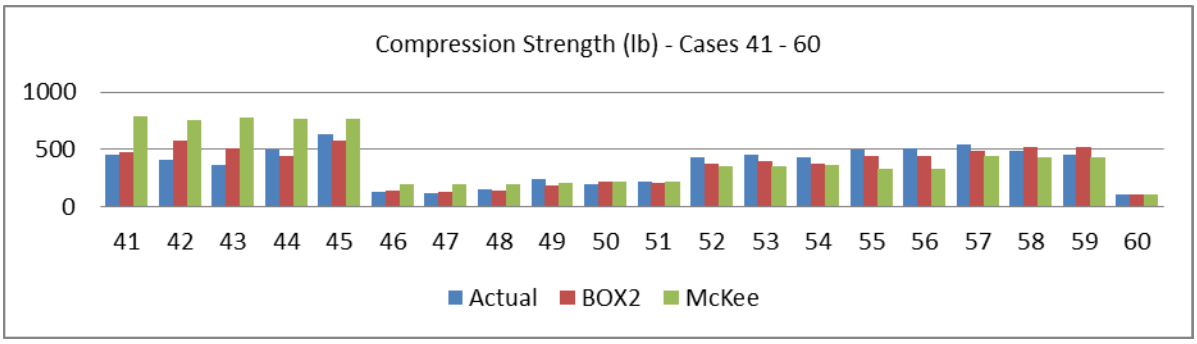
```
netsum = -7.247163
netsum = netsum + inp(1) * -7.162368E-02
netsum = netsum + inp(2) * -6.672145
netsum = netsum + inp(3) * -5.509887
netsum = netsum + inp(4) * 1.965163
netsum = netsum + inp(5) * -0.1127764
netsum = netsum + inp(6) * -0.3749207
netsum = netsum + inp(7) * -6.057019
feature2(1) = 1 / (1 + exp(-netsum))
netsum = -6.879015
netsum = netsum + inp(1) * -3.016899
netsum = netsum + inp(2) * -5.380744
netsum = netsum + inp(3) * -3.580803
netsum = netsum + inp(4) * -0.4355183
netsum = netsum + inp(5) * -0.9137687
netsum = netsum + inp(6) * 1.621481
netsum = netsum + inp(7) * -7.314968
feature2(2) = 1 / (1 + exp(-netsum))
netsum = 2.58671
netsum = netsum + inp(1) * -56.1168
netsum = netsum + inp(2) * 19.70152
netsum = netsum + inp(3) * 18.78237
netsum = netsum + inp(4) * -23.49325
netsum = netsum + inp(5) * 13.29584
netsum = netsum + inp(6) * -8.186071
netsum = netsum + inp(7) * -13.79474
feature2(3) = 1 / (1 + exp(-netsum))
netsum = -6.930517
netsum = netsum + inp(1) * -3.43881
netsum = netsum + inp(2) * -5.543357
netsum = netsum + inp(3) * -3.266341
feature2(7) = 1 / (1 + exp(-netsum))
netsum = -6.69485
netsum = netsum + inp(1) * -2.424634
netsum = netsum + inp(2) * -5.845432
netsum = netsum + inp(3) * -3.707195
netsum = netsum + inp(4) * -0.2126636
netsum = netsum + inp(5) * -0.6274015
netsum = netsum + inp(6) * 1.344668
netsum = netsum + inp(7) * -6.710152
feature2(8) = 1 / (1 + exp(-netsum))
netsum = -7.186853
netsum = netsum + inp(1) * -1.09149
netsum = netsum + inp(2) * -6.539968
netsum = netsum + inp(3) * -4.775326
netsum = netsum + inp(4) * 0.8580441
netsum = netsum + inp(5) * -0.1386947
netsum = netsum + inp(6) * 0.6256868
netsum = netsum + inp(7) * -6.080272
feature2(9) = 1 / (1 + exp(-netsum))
netsum = -1.865983
netsum = netsum + inp(1) * 1.98675
netsum = netsum + inp(2) * -11.25155
netsum = netsum + inp(3) * -10.4184
netsum = netsum + inp(4) * 1.585183
netsum = netsum + inp(5) * -0.5645384
netsum = netsum + inp(6) * 1.33543
netsum = netsum + inp(7) * -3.745443
feature2(10) = 1 / (1 + exp(-netsum))
netsum = -6.47867
netsum = netsum + inp(1) * -0.8656681
netsum = netsum + inp(2) * -5.996781
netsum = netsum + inp(7) * -6.608476
feature2(14) = 1 / (1 + exp(-netsum))
netsum = -7.510636
netsum = netsum + inp(1) * 2.029926
netsum = netsum + inp(2) * -7.943251
netsum = netsum + inp(3) * -7.275569
netsum = netsum + inp(4) * 2.87489
netsum = netsum + inp(5) * -1.476773
netsum = netsum + inp(6) * 0.559886
netsum = netsum + inp(7) * -3.235952
feature2(15) = 1 / (1 + exp(-netsum))
netsum = -0.2801921
netsum = netsum + feature2(1) * -0.4374026
netsum = netsum + feature2(2) * 3.453336E-02
netsum = netsum + feature2(3) * 1.700875
netsum = netsum + feature2(4) * -0.1481685
netsum = netsum + feature2(5) * -0.9338183
netsum = netsum + feature2(6) * -0.5930923
netsum = netsum + feature2(7) * 1.477526
netsum = netsum + feature2(8) * 0.1232549
netsum = netsum + feature2(9) * -0.5360605
netsum = netsum + feature2(10) * -5.806566
netsum = netsum + feature2(11) * 0.4093191
netsum = netsum + feature2(12) * -2.210154
netsum = netsum + feature2(13) * -2.185017
netsum = netsum + feature2(14) * 2.029893
netsum = netsum + feature2(15) * -6.077104E-03
outp(1) = 1 / (1 + exp(-netsum))

outp(1) = 810 *  (outp(1) - .1) / .8  + 103
```

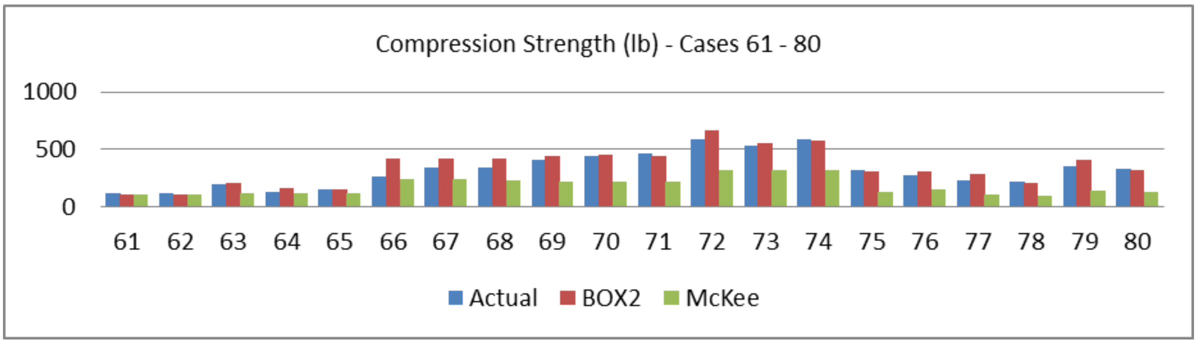
Appendix B Graphical Comparison of Actual, CBU-BOX2, and McKee Formula for 141 Cases



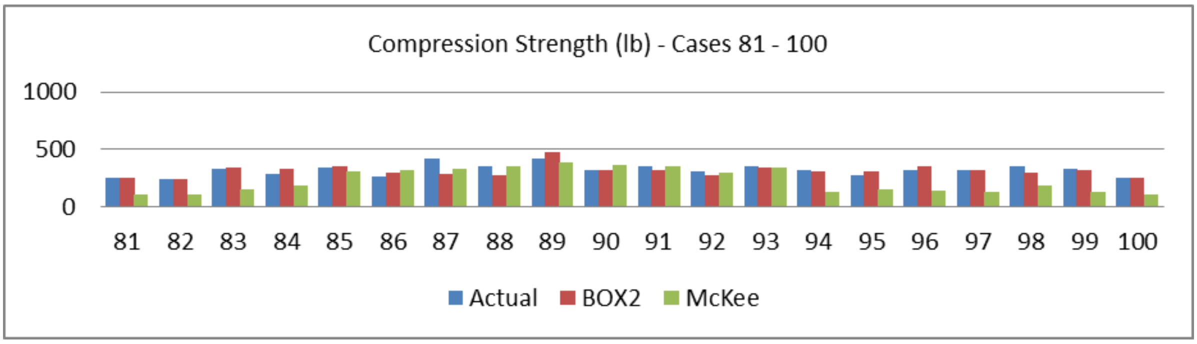




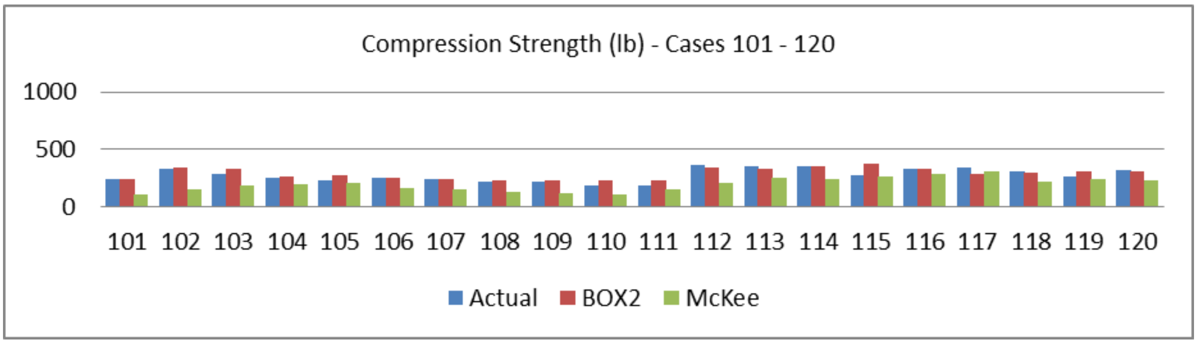
Compression Strength (lb) - Cases 41 - 60
1000
500
0
41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60
Actual BOX2 McKee



Compression Strength (lb) - Cases 61 - 80
1000
500
0
61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80
Actual BOX2 McKee



Compression Strength (lb) - Cases 81 - 100
1000
500
0
81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100
Actual BOX2 McKee



Compression Strength (lb) - Cases 101 - 120
1000
500
0
101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120
Actual BOX2 McKee