Report - APL405

Aryan Sharma 2020ME21196
Nihal Pushkar 2020ME10947
Radhika Agawan 2020ME10956

April 19, 2022

Nomenclature

$\boldsymbol{\sigma}$	Stress tensor
A	Area of cross-section (m^2)
E	Young's modulus (MPa)

Contents

1 Introduction

Soft robotics is the study of designing, controlling, and fabricating robots made of soft materials rather than hard links. Unlike rigid metal, ceramic, and hard plastic robots, soft robots' compliance may increase their safety while operating near humans.

Soft robotics, a new technology that has arisen in recent years, has added flexibility and adaptability to rigid robots that were previously unavailable. In soft robotics, three-dimensional (3D) printing has evolved into four-dimensional (4D) printing, with the fourth dimension referring to the printed mechanism's time-dependent reaction to varied stimuli such as heat, electricity, magnetism, and pneumatic pressure.

You should add the citations as described here [**?**]. You can refer to this document for more latex symbols[1].

1.1 Actuators

1.1.1 Soft Pneumatic Actuators (SPA)

In this study, 4D-printed soft pneumatic actuators (SPA) are used which are invented by Harvard's Whitesides Research Group. They are composed of finger-like structures with bellows that inflate when compressed, allowing them to extend and bend.

Inside an elastomer, they are made up of a number of channels and chambers. When pressured, these channels expand, causing motion. Modifying the shape of the embedded chambers and the material qualities of their walls changes the nature of the motion.

When a PneuNets actuator is pressured, the most compliant (least stiff) areas expand first. If the PneuNet is made of a single, homogeneous elastomer, for example, the thinnest structures will experience maximum expansion. Designers may pre-program the actuator's behavior by choosing wall thicknesses that will provide the required motion. These actuators are less expensive, lighter, easier to make, adaptive, flexible, and deformable than stiff alternatives.

[1] https://wch.github.io/latexsheet/

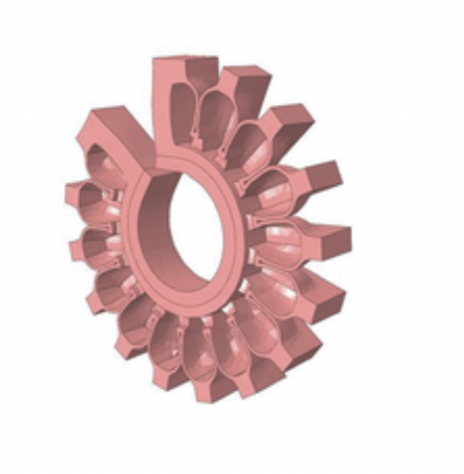


Figure 1: Soft Pneumatic Actuators (SPA).

Furthermore, the softness and flexibility of the material reduce the effect on human skin and other delicate items and surfaces. In a closed-loop system, these sorts of soft robot actuators were used to construct a soft surgical manipulator.

This technology's promise also rests in its application to the rehabilitation of a wrist and finger exoskeleton that aids joint mobility. Because of its capacity to make soft robots and actuators with intricate inner structures, 3D/4D printing is being researched. Ninjaflex material was chosen because it can be 3D printed without air bubbles and has hyperelastic qualities that give flexibility and sensitivity to applied stress.

Figure 2: SPA cross section.

2 Problem statement

One of the key issues in 4D printing soft robots and actuators is modeling and predicting their motion, which is complicated by the material's non linearity. In many cases, a linear analytical model fails to effectively predict the actuation behavior of 4D printed actuators, but numerical simulations using nonlinear material principles enhance accuracy. Using machine learning (ML) algorithms based on numerical findings, on the other hand, might assist reduce time and effort throughout the design process.

3 Methodology

The following methodology has two subsections, one being the data generation part and the other one being the Neural Network implementation part.

3.1 Data Generation

There was need to generate data for the neural network to train and model upon. The original paper had used real models of SPA to measure the bending angles. Since this method was not feasible, we used GIBBON-FEBio to generate the simulations for the model of SPA at different bending angles. The simulations are carried out using MATLAB software.

What are FEBio and GIBBON and their uses?

FEBio: FEBio is a software tool for nonlinear finite element analysis in biomechanics and biophysics and is specifically focused on solving nonlinear large deformation problems in biomechanics and biophysics.

GIBBON: GIBBON (The Geometry and Image-Based Bioengineering add-ON) is an open-source MATLAB toolbox by Kevin M. Moerman and includes an array of image and geometry visualization and processing tools and is interfaced with free open source software such as TetGen, for robust tetrahedral meshing, and FEBio for finite element analysis.

Since MATLAB code wasn't calibrated to generate the bending angles directly, they were measured manually from the deformation image in MATLAB simulation.

The bending angle was measured manually using AUTODESK - Inventor and then DATA was recorded in the excel.

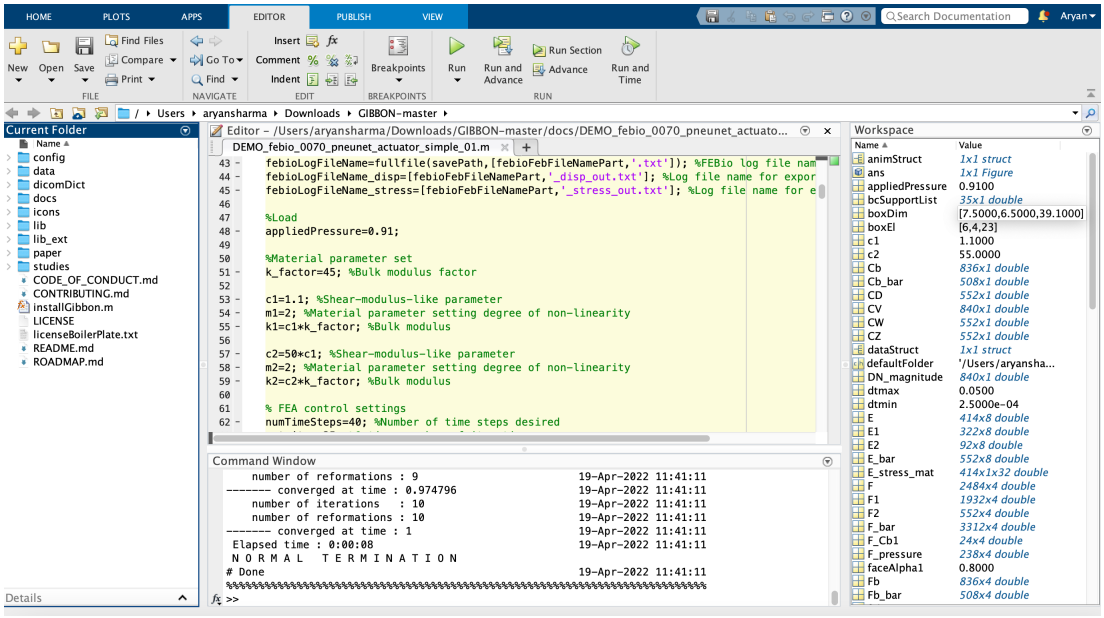


Figure 3: Screenshot of MATLAB Simulation

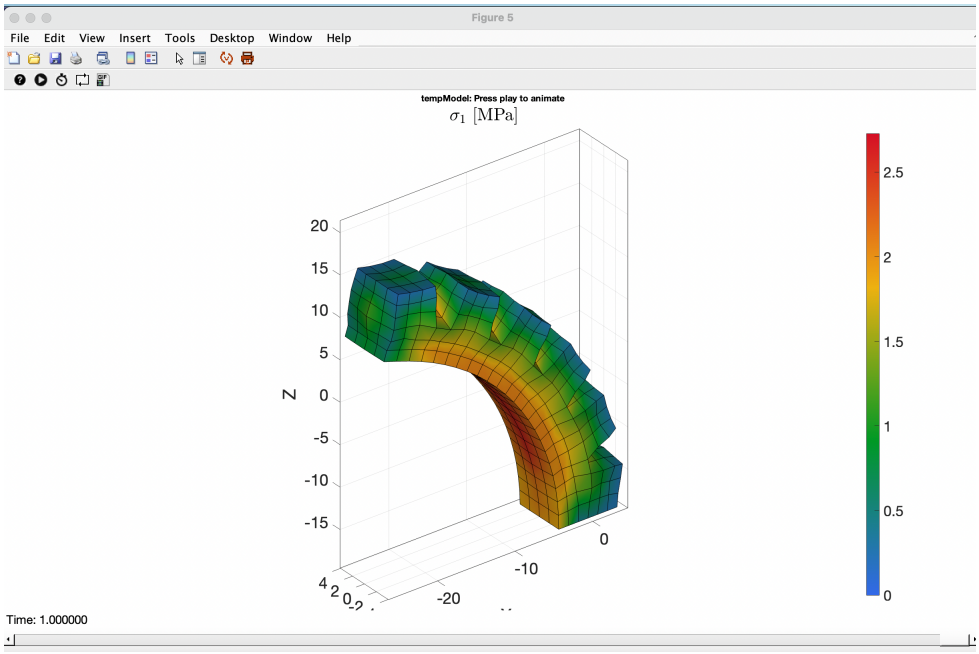


Figure 4: Simulated model of the SPA

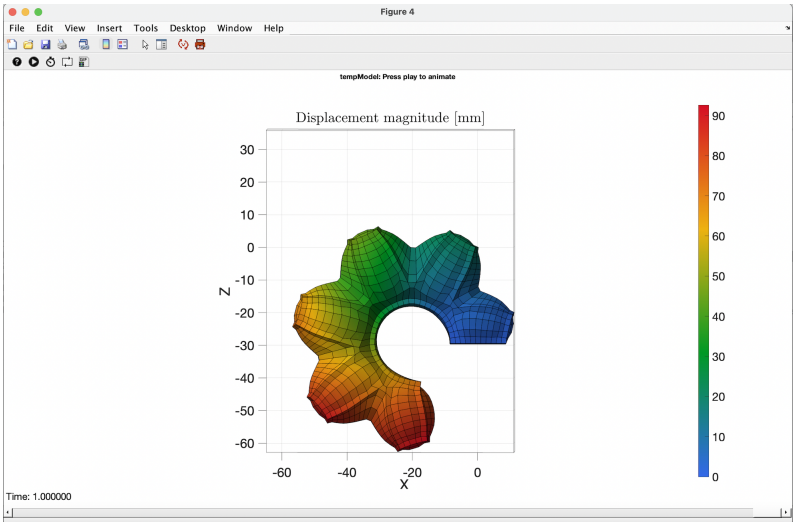


Figure 5: Simulated model of the SPA

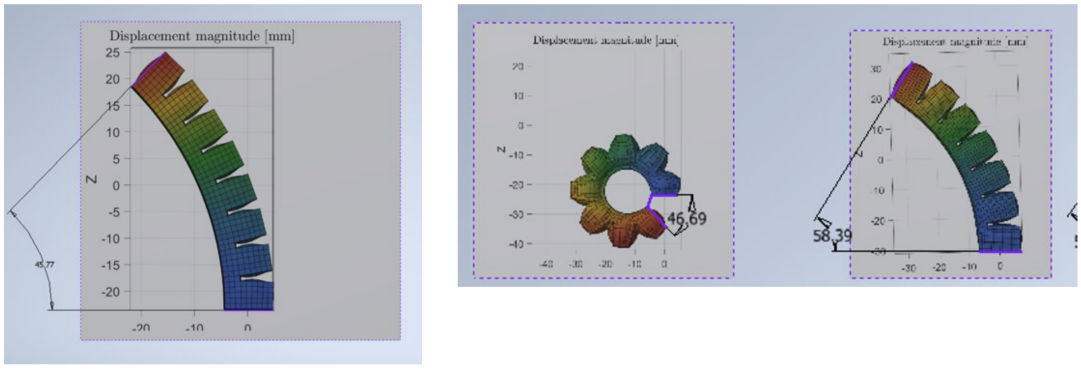


Figure 6: Bending angle measured using AutoDesk

	A	B	C	D	E	F	G	H	I	J
1	ding Angle (output/degrees).	Geometrical Parameters(mm)			Material Parameters(MPa)				Input Pressure (MPa)	Pressure Squared.
2		Length	Hieght	Width	c1	c2	k1	k2		
3	313.3074597	47	9	6	1	50	50	2500	0.5	0.25
4	256.2392149	47	9	6	1	50	50	2500	0.45	0.2025
5	202.5384498	47	9	6	1	50	50	2500	0.4	0.16
6	159.4775408	47	9	6	1	50	50	2500	0.35	0.1225
7	121.8218075	47	9	6	1	50	50	2500	0.3	0.09
8	92.0757569	47	9	6	1	50	50	2500	0.25	0.0625
9	65.4412224	47	9	6	1	50	50	2500	0.2	0.04
10	47.4231918	47	9	6	1	50	50	2500	0.15	0.0225
11	27.8312913	47	9	6	1	50	50	2500	0.1	0.01
12	13.5432489	47	9	6	1	50	50	2500	0.05	0.0025
13	58.3924987	62	13.5	9	1	5	50	250	0.05	0.0025
14	54.6111503	63	9.6	8	2	10	20	200	0.1	0.01
15	130.3207412	63	9.6	8	0.4	2.8	8	56	0.05	0.0025
16	131.6871729	99	11	10	4	40	120	1200	0.3	0.09
17	133.3828676	99	11	10	4	40	80	800	0.3	0.09
18	135.6532264	99	14	10	4	40	80	800	0.3	0.09

Figure 7: Bending angle measured using AutoDesk

3.2 Neural Network Generation

The neural network that was designed has 2 hidden layers. The 1st hidden layer has 37 neurons. The second hidden layer has 29 neurons.

We have also used a combinations of activation functions, to try to find out the best one for our neural network.

Using these configuration a high accuracy was obtained without adding the regularization term. Running the code several times with different no. of neurons and hidden layer to reach to this conclusion.

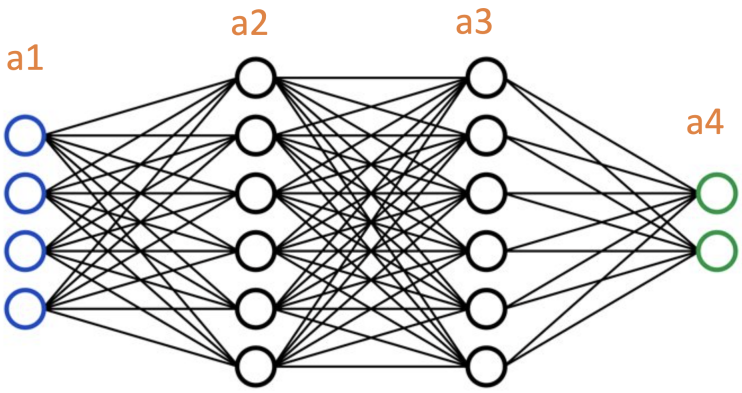


Figure 8: Illustration of NN

In the figure given above, a1-a2 has sigmoid activation function , a2-a3 has sigmoid and a3-a4 has ReLu.

Other activation functions such as tanh were tried but they gave a greater error and were thus discarded.

For the a3 a4 layer we used ReLu since angle should be positive. Hence used ReLu.

Since the data set is small Normal activation function can't be used since we don't have the true mean and standard deviation of the activation layers.

3.3 Cost Function

For faster convergence we needed high gradient cost functions.

Tried making a faster converging gradient function by using exponentials of absolute difference/higher order polynomials.

Faced the issue of overloading, that is the value went extremely high when the optimizer started its work.

Finally decided to take mean bi-quadratic function as it had higher gradient than mean square error function and didn't explode in the optimizer while training for the optimizing the weights and biases.

3.4 Types of Optimizers used and how they were selected

The optimizers considered were TNC(Truncated Newton), L-BFGS(limited-memory BFGS), CG(conjugate gradient method).All of these are minimizing functions.

Gradient compatible optimizers were used since we need to use back-propagation.

L-BFGS is a variant of BFGS that allows the usage of box constraints for any parameter, however, since we did not have any such constraints, we considered BFGS better for our purpose.

The fundamental limitation of using CG is that it is much slower, that is, it needs multiple cycles to reach the minimum because of the way it works.

Out of the remaining(TNC, CG and BFGS), BFGS(Broyden–Fletcher–Goldfarb–Shanno) optimizer was used.

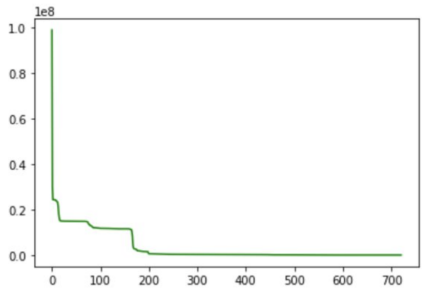
The main criteria was by running all possible combinations of the optimizer-activation layer. Finally based on test errors the BFGS was selected.

3.5 Testing the Neural Network

3.5.1 Graphs for TNC Optimizer

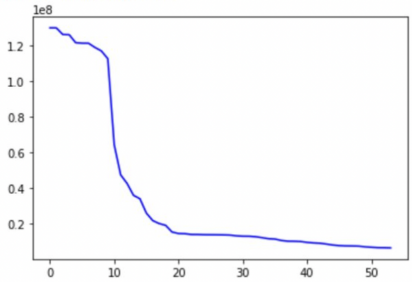
```
plt.plot(li,'g')
# sigmoid, TNC, f(x) = x^4
print(res.message)
```

```
Converged (|f_n-f_(n-1)| ~= 0)
```



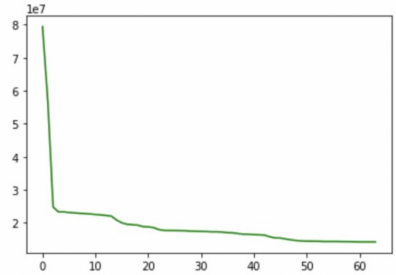

```
plt.plot(li,'b')
# ReLu, TNC, f(x) = x^4
print(res.message)
```

```
Linear search failed
```




```
# normal(0.3, 0.5), TNC, f(x) = x^4
plt.plot(li,'g')
print(res.message)
```

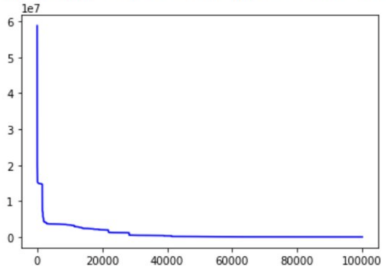
```
Converged (|f_n-f_(n-1)| ~= 0)
```



3.5.2 Graphs for CG

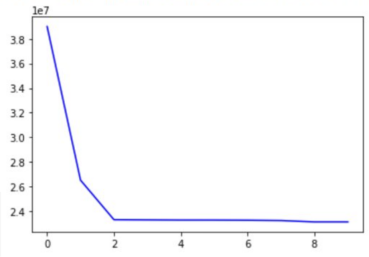
```
plt.plot(li,'b')
# sigmoid, CG, f(x) = x^4
print(res.message)
```

```
Maximum number of iterations has been exceeded.
```



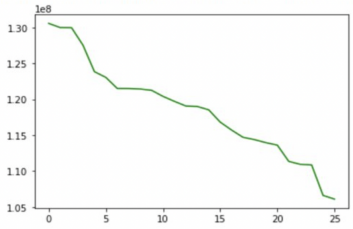

```
# normal(0.3, 0.5), CG, f(x) = x^4
plt.plot(li,'b')
print(res.message)
```

```
Desired error not necessarily achieved due to preci
```




```
plt.plot(li,'g')
# ReLu, CG, f(x) = x^4
print(res.message)
```

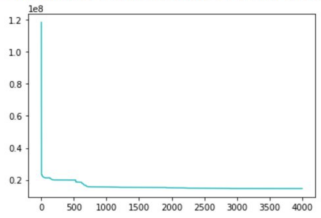

```
Desired error not necessarily achieved due to precision loss.
```



3.5.3 Graphs for L- BFGS

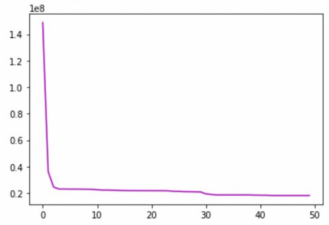
```
# normal(0.6,1), L-BFGS-B, f(x) = x^4
plt.plot(li,'c')
print(res.message)
```

```
b'CONVERGENCE: REL_REDUCTION_OF_F_<=_FACTR*EPSMCH'
```



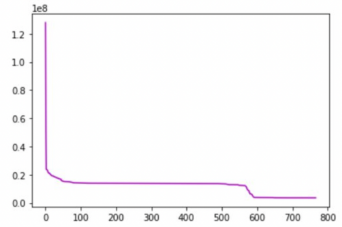

```
# normal(0.3, 0.5), L-BFGS-B, f(x) = x^4
plt.plot(li,'m')
print(res.message)
```

```
b'ABNORMAL_TERMINATION_IN_LNSRCH'
```




```
# sigmoid, L-BFGS-B, f(x) = x^4
plt.plot(li,'m')
print(res.message)
```

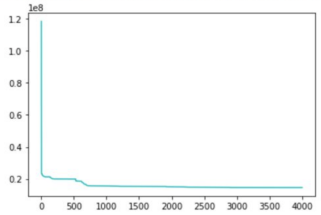
```
b'CONVERGENCE: REL_REDUCTION_OF_F_<=_FACTR*EPSMCH'
```



3.5.4 Graphs for BFGS

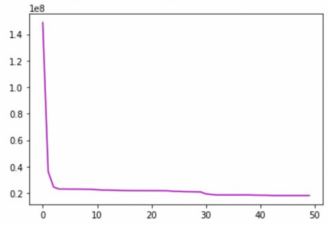
```
# normal(0.6,1), L-BFGS-B, f(x) = x^4
plt.plot(li,'c')
print(res.message)
```

```
b'CONVERGENCE: REL_REDUCTION_OF_F_<=_FACTR*EPSMCH'
```



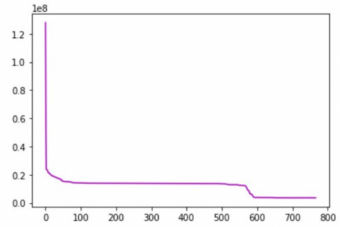

```
# normal(0.3, 0.5), L-BFGS-B, f(x) = x^4
plt.plot(li,'m')
print(res.message)
```

```
b'ABNORMAL_TERMINATION_IN_LNSRCH'
```




```
# sigmoid, L-BFGS-B, f(x) = x^4
plt.plot(li,'m')
print(res.message)
```

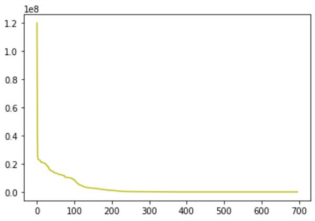
```
b'CONVERGENCE: REL_REDUCTION_OF_F_<=_FACTR*EPSMCH'
```



3.5.5 Graphs for BFGS continued

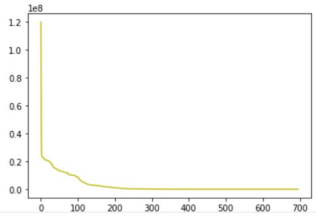
```
# sigmoid, BFGS, f(x) = x^4, lambda_ 0.01
plt.plot(li,'y')
print(res.message)
print(res.fun)
```

```
Desired error not necessarily achieved due to precision loss.
5.264708715328892
```



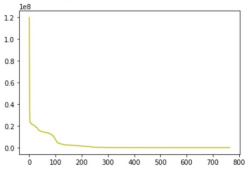

```
# sigmoid, BFGS, f(x) = x^4, lambda_ = 0.1
plt.plot(li,'y')
print(res.message)
print(res.fun)
```

```
Desired error not necessarily achieved due to precision loss.
5.264708715328892
```



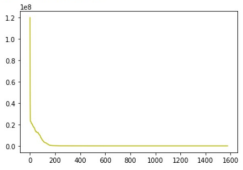

```
# sigmoid, BFGS, f(x) = x^4, lambda_ = 0.005
plt.plot(li,'y')
print(res.message)
print(res.fun)
```

```
Desired error not necessarily achieved due to precision loss.
5.3487808562453365
```



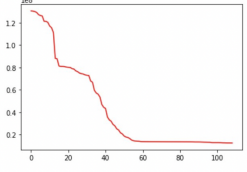

```
# sigmoid, BFGS, f(x) = x^4, lambda_ = 0.001
plt.plot(li,'y')
print(res.message)
print(res.fun)
```

```
Desired error not necessarily achieved due to precision loss.
0.583793756391613 6
```



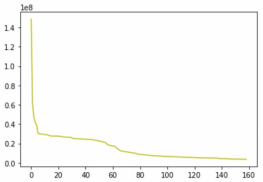

```
# ReLu, BFGS, f(x) = x^4
plt.plot(li,'r')
print(res.message)
```

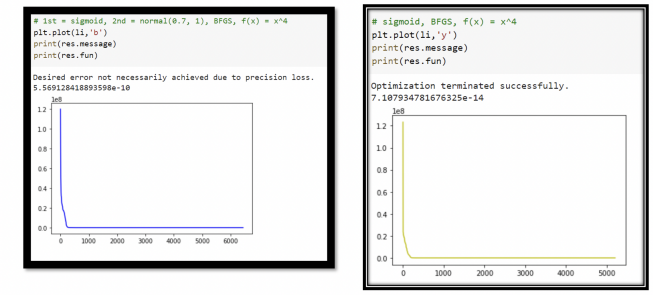
```
Desired error not necessarily achieved due to precision loss.
```




```
# normal(0.3, 0.5), BFGS, f(x) = x^4
plt.plot(li,'y')
print(res.message)
```

```
Desired error not necessarily achieved due to precision loss.
```





3.5.6 Prediction Function

Prediction function

- For 2 hidden layer both sigmoid activated, with final activation as ReLu.

```
def p2hl(w1, w2, w3, X):
  m=X.shape[0]
  num_labels= w3.shape[0]
  a1= np.concatenate([np.ones((m,1)), X], axis=1) #Input layer
  a2= sigmoid(a1.dot(w1.T)) #Hidden Layer
  a2= np.concatenate([np.ones((a2.shape[0],1)), a2 ], axis=1) #Activation of 2nd layer
  a3= sigmoid(a2.dot(w2.T))
  a3= np.concatenate([np.ones((a3.shape[0],1)), a3 ], axis=1) #Activation of 3rd layer
  a4 = relu(a3.dot(w3.T))
  p= a4
  return p
```

4 Results and discussion

- For training set, mean absolute error in degrees.

```
# for 2 hidden layers
e = 0
p_h2t = p2hl(w1h2,w2h2,w3h2,X)   # X is training data
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:4: RuntimeWarning: overflow encountered in exp
  after removing the cwd from sys.path.
```

```
el = p_h2t - np.reshape(y,(len(y),1))
```

```
np.sum(np.abs(el))/len(y)
```

```
0.0003915180242717138
```

- For testing set, mean absolute error in degrees.

```
e = 0
p_h2 = p2hl(w1h2,w2h2,w3h2,X1)   # X1 is testing data
y1 = np.reshape(y1,(len(y1),1))
print(np.sum(np.abs(p_h2-y1))/y1.shape[0])
```

```
0.0005622265888931756
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:4: RuntimeWarning: overflow encountered in exp
  after removing the cwd from sys.path.
```

5 Future work

By altering the meshing algorithm one can calibrate the MATLAB GIBBON FEBio code to directly generate the bending angle.

Algorithms that could predict whether the simulation on given input parameters will work or not.

Training a machine learning model that can predict blocked force/torque for a given set of input parameters in a SPA.

References