SMT4DL 2020, Chennai Mathematical Institute

Final Report on Verification of a Neural Network that predicts the frequency of a given Signal

Deeksha and Soumodev

December 29, 2020

CONTENTS

1 INTRODUCTION

Estimation of the frequency, f of a noisy signal has been one of the main problems in the field of signal processing and communications, due to its vast applications including power systems, communications, and radar.
Many theoretical techniques have been proposed to solve this problem; examples include discrete Fourier transform, least squares methods and phase-locked loops.
Then why use Neural Network?
We can use Neural Network since one of the issues with these theoretical techniques is that it is quite tedious to calculate every time for each input. Using Neural Network, we can train it by feeding some input data and once the Neural Network has some acceptable accuracy, we can use the model to calculate the frequency.
Here, the time is required only to train the model. Once trained sufficiently, it will be easy to get an output for a given input fed to the network. Neural networks (NNs), which belong to the family of deep-learning methods, can derive meaningful results from complicated and complex problems, and may detect patterns that human beings do not see in data; finding f of a noisy signal is a good example of such a problem. We know that a noisy signal is related to its f, but mathematical identification of that relation can be difficult.
But a Neural Network comes with its own share of problems. One of the biggest problems with Neural network is robustness and consistency. To address these problems (atleast partially), we will use Z3 in order to verify certain properties for which if the trained model is working correctly (i.e., giving the required frequency) or not. Thus, we will provide some specific constraints to the input and check the validity of the model.
Note: The link to the code is given below [1]

[1] **Link to the code: COLAB-LINK**

2 Problem Statement

We want to find the frequency, f of a noisy sinusoidal wave

$$\mathbf{S}\,(\mathbf{t}) = \mathbf{A}\left(\sin(\mathbf{2}\pi\mathbf{ft} + \psi) + \Omega(\mathbf{t})\right) \tag{2.1}$$

where A is amplitude,
t is time,
ψ is phase,
and Ω is zero mean Gaussian noise with a variance of σ^2.

The signal-to-noise ratio (SNR) is the ratio of signal power P_s to noise power P_n and can be expressed in decibels as:

$$SNR = \frac{P_s}{P_N} = \left(\frac{A}{\sigma}\right)^2 \tag{2.2}$$

$$SNR_{dB} = 10.log_{10}(SNR) = 10.log_{10}[\left(\frac{A}{\sigma}\right)^2] \tag{2.3}$$

$$\sigma^2 = \frac{A^2}{10^{\frac{SNR_{dB}}{10}}} \tag{2.4}$$

So given SNR_{dB} and A we can obtain the variance that is needed for calculating the noise function.

3 Data Preparation for the Neural Network

To generate data sets, we have taken frequencies ranging from 1kHz to 5kHz.
The amplitude, A is set to 1. SNR is set to 25, as it is a typical setting for a good signal, thus σ = 0.5 (calculated from equation (2.4)) .The noise is calculated using Gaussian distribution.
The phase change is randomly assigned to the waves, in order to train our model for phase change. This is done by creating an array of 500 consecutive readings, with a time lapse of 4.1667 e^{-05}s, for a given signal, then randomly taking 50 consecutive readings from it and using that as our data.
Since the signal is fed with noise using Guassian distribution it is highly improbable that we get two signals that are equivalent in each point w.r.t. the amplitude, this ensures that there is no redundant data.
We add an extra point in the input of each data in the dataset. This point is set to 1 for all the inputs and this is added because we want to get rid of the biases and using this extra point we achieve the shift that the biases intended.
The data is appended with the frequency of the signal corresponding to those 50 points and the frequency is normalized to lie between 0 and 1. *Why normalize?*
The neural networks tend to work better for data within 0 and 1 for regression problems.
By this method(took 4000 of the raw data with 500 points and for each such data took 50 different consecutive points of length 50, so 4000x50=200000) we generated a dataset of size 2,00,000. Of these data we have taken, 70% as training data, 20% as validation data set, and 10% as the test data set, to train our model, neural network.
We used Pandas to split the dataset that we generated randomly to get the training, testing and validation data.

3.1 Sample Data

We are adding two sample data from the dataset, one without adding any noise and another with the same frequency but with added noise. *[Note: We are only considering the first 50 points of the input and ignoring the extra input used to simulate the biases.]*

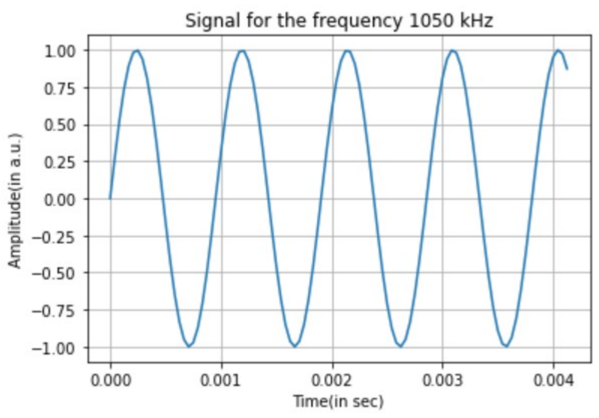


Figure 3.1: This is a signal taken from the generated dataset before adding any noise. The frequency of the signal is 1050 KHz.

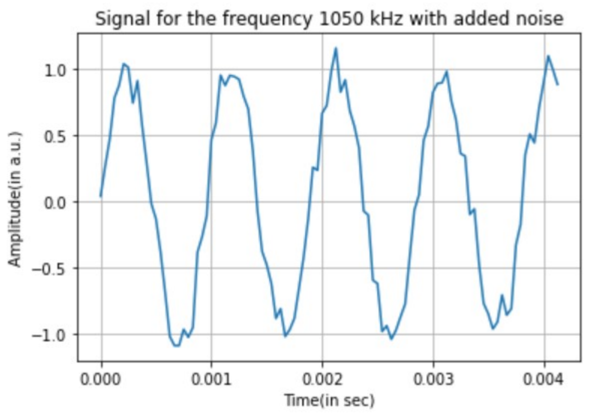


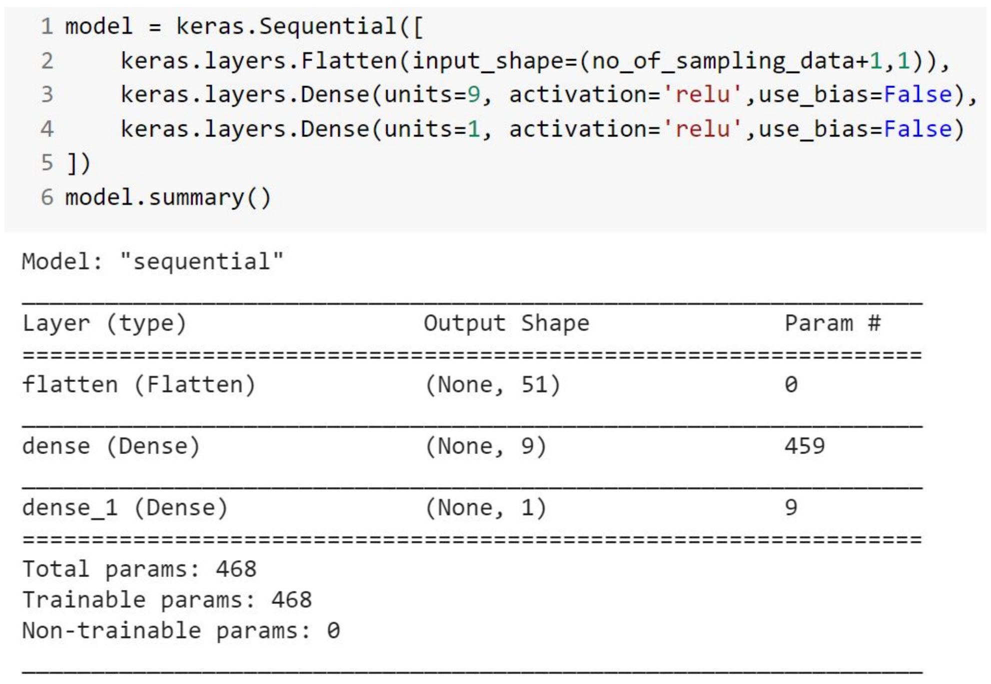
Figure 3.2: This is a signal taken from the generated dataset(the same signal as in Figure. 3.1) after adding some guassian noise of SNR=25

4 The Neural Network

The Network Configuration Our Neural Network have 3 layers.
The first layer is the input layer, comprising 51 neuron(as defined in the previous section). The hidden layer has 9 neurons.
The last layer is the output layer with 1 neuron, that predicts the frequency. Here, We have used *Relu* as activation for all layers, where Relu can be defined as:

$$Relu(x) = Max(0, x)$$

The optimizer used is "Adam" optimizer.



```
model = keras.Sequential([
    keras.layers.Flatten(input_shape=(no_of_sampling_data+1,1)),
    keras.layers.Dense(units=9, activation='relu',use_bias=False),
    keras.layers.Dense(units=1, activation='relu',use_bias=False)
])
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten (Flatten)            (None, 51)                0
_________________________________________________________________
dense (Dense)                (None, 9)                 459
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 9
=================================================================
Total params: 468
Trainable params: 468
Non-trainable params: 0
_________________________________________________________________
```

Figure 4.1: This is the model description obtained from *Keras.model.summary()*

Calculating the Loss We have used MSE(Mean Squared Error) as a metric to measure the loss of the model.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(Y_i - P_i)^2 \tag{4.1}$$

where n is the number of measurements, Y_i are the real values and P_i are the predicted values.

CALCULATING THE ACCURACY We have also measured its performance using accuracy with 5% as tolerance. The accuracy is calculated as follows:

$$Accuracy = \frac{1}{n}\sum_{i=1}^{n} |P_i - Y_i| \leq T \times Y_i \tag{4.2}$$

where n is the number of measurements, P_i is a predicted value, Y_i is the actual value and T is the threshold value. Here the threshold was set to 5%
Note: Here we are finding the average number of data points, for which our network predicts within the permissible threshold.
All codes were written in Python with the help of TensorFlow and Keras packages for modelling the Neural Network. The model summary can be seen in Figure.4.1. Calculations were performed on google colab.

4.1 RESULTS ACHIEVED FROM THE NETWORK

Accuracy of the model on training and validation data is given in figure 4.2. It can be seen that accuracy of 0.9272 i.e. 92.72% in training data and 0.9360 i.e. 93.60% is achieved by the model. When used under training data, accuracy was found to be 93.235%.

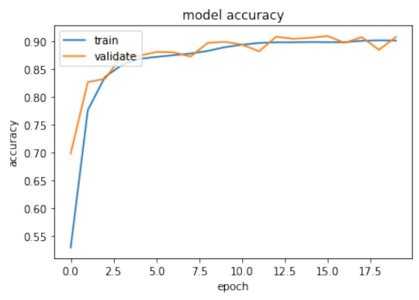


Figure 4.2: Training accuracy and validation accuracy as the model trains

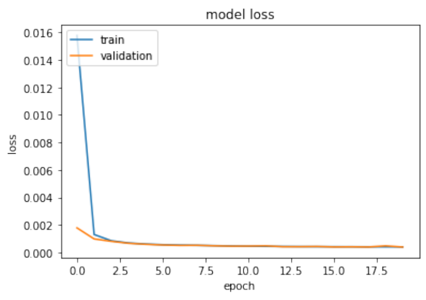


Figure 4.3: Training loss and validation loss as the model trains

The lowest recorded loss were $2.9979e^{-04}$ on the training dataset (at epoch 20) and $2.7994e^{-04}$ on the validation dataset (at epoch 17) in normalised values. The evaluated loss in test data

was $2.835e^{-04}$ in normalised values.The model showed better progress in the initial epoch and got saturated near the end.
To show the model's functionality on both low and high frequencies, we present the prediction accuracy of the model at one frequency at the low end of the frequency and one from the high end. The model performed well on both frequencies. The model predicted 1265.87Hz at real f=1224.7Hz (phase =154, error= 5.34%) and 4158.531Hz at real f=4129.1Hz (phase= 13, error= 1.74%),

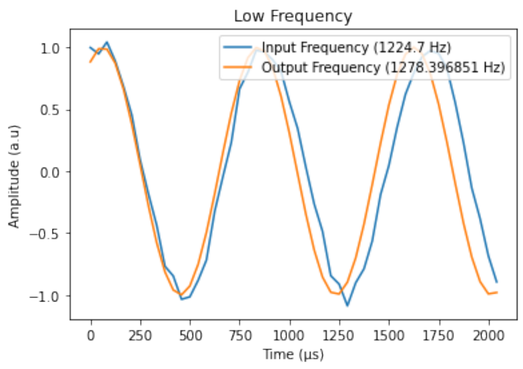


Figure 4.4: Behaviour of model with low frequency

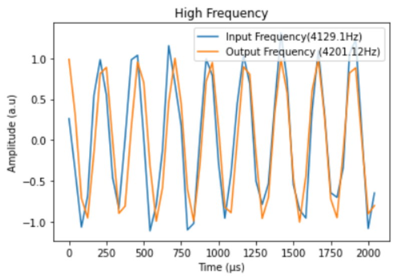


Figure 4.5: Behaviour of model with high frequency

5 Abstraction of neural Network

Though we have few neurons in each layers, we want to generalise to, any number of neurons in each layer. Hence abstracted our neural network such that it has atmost four neurons in each layers. Here we used CEGAR algorithm to check a given property by abstracting the network.

5.1 Implementation

In our project , we experimented cegar algorithm for one property: Reversal of wave. The main cegar was implemented using the following functions:

posneg: It splits the neuron based on pos/neg Input: Weights of neural network Output: weights new neural network and the indication of pos and neg neurons

incdec: it splits the neuron based on inc/dec neuron

Input: weights, indication of pos and neg of neurons

Output:weights of the new neural network, status of the neural network (which indicates whether a given neuron is pos/inc (PI), pos/dec(PD), neg/inc(NI) and neg/dec(ND) for every hidden layer)

It is to be noted that the output network of posnef function and output network of the incdec function will be equivalent to the original neural network. From here on whenever refer this equivalent split network as original network. And we pass our network as its weights to function.

splitting neurons: it gives the status/indicator as to how the network is supposed to be split based on some heuristic. Here we split the neuron with maximum value difference given an input.

Input: original network, abstracted network, input values , current status of the abstracted network

Output: status with which the neurons are supposed to be splitted

merge_neuron: It merges the given network based on the status given. That is for inc neuron maximum of incoming and sum of outgoing and for dec neuron minimum of incoming and sum of outgoing.

Input:Original network, status

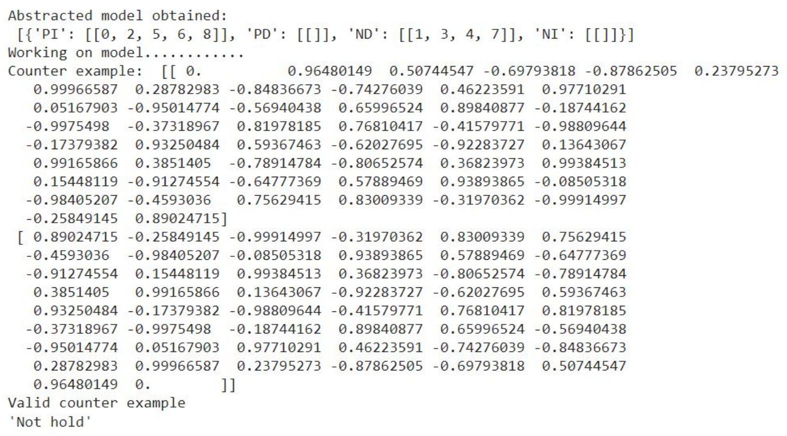
Output:merged network

It is to be noted that given a status from splitting neuron and original weights, the merge neuron can act as splitting of network which can be used in refinement of network.

With the help of the above functions we implemented CEGAR. Here we refine our abstract network with respect to a spurious counter example till it is no longer a counter example for this refined network.

Our aim here is to find if there is an input to our original network such that its its predicted value is greater that the threshold when compared to the reversed input. It there is the we can say that the property : frequency is same when reversed , doesnot hold.

Result:It was noted that when full range of frequency was used , it was abstracted to the original network. Hence we constrained our frequency range to 4000-5000 Hz



```
Abstracted model obtained:
 [{'PI': [[0, 2, 5, 6, 8]], 'PD': [[]], 'ND': [[1, 3, 4, 7]], 'NI': [[]]}]
Working on model...........
Counter example:  [[ 0.          0.96480149  0.50744547 -0.69793818 -0.87862505  0.23795273
   0.99966587  0.28782983 -0.84836673 -0.74276039  0.46223591  0.97710291
   0.05167903 -0.95014774 -0.56940438  0.65996524  0.89840877 -0.18744162
  -0.9975498  -0.37318967  0.81978185  0.76810417 -0.41579771 -0.98809644
  -0.17379382  0.93250484  0.59367463 -0.62027695 -0.92283727  0.13643067
   0.99165866  0.3851405  -0.78914784 -0.80652574  0.36823973  0.99384513
   0.15448119 -0.91274554 -0.64777369  0.57889469  0.93893865 -0.08505318
  -0.98405207 -0.4593036   0.75629415  0.83009339 -0.31970362 -0.99914997
  -0.25849145  0.89024715]
 [ 0.89024715 -0.25849145 -0.99914997 -0.31970362  0.83009339  0.75629415
  -0.4593036  -0.98405207 -0.08505318  0.93893865  0.57889469 -0.64777369
  -0.91274554  0.15448119  0.99384513  0.36823973 -0.80652574 -0.78914784
   0.3851405   0.99165866  0.13643067 -0.92283727 -0.62027695  0.59367463
   0.93250484 -0.17379382 -0.98809644 -0.41579771  0.76810417  0.81978185
  -0.37318967 -0.9975498  -0.18744162  0.89840877  0.65996524 -0.56940438
  -0.95014774  0.05167903  0.97710291  0.46223591 -0.74276039 -0.84836673
   0.28782983  0.99966587  0.23795273 -0.87862505 -0.69793818  0.50744547
   0.96480149  0.        ]]
Valid counter example
'Not hold'
```

Figure 5.1: Output for reversal of wave using CEGAR when range is limited

6 SOME DISCUSSIONS ON Z3 VERIFICATION

For the verification portion, We can divide it into 3 parts:

THE INPUT DATA **First**, We tried taking any arbitrary input to the network. But it was pointless for the network to predict the frequency of something that is arbitrary and doesn't depict any signal.Although the verification was considerably faster. So,**the main objective** we want to cover here is that we cannot give any arbitrary input to the network. We want to get the input of the network to comply with **some sinewave** with some added noise.
To verify that the data given as an input is indeed a sinewave, we need to check whether the 50 input points form a probable sinewave.
But we know that a sinewave is non-linear and it is not possible to handle non-linear functions in Z3. *So what should we do?*
What we can do is to **approximate the sinewave** by evaluating it using *Taylor Series Expansion* upto some terms so that we get a plausible signal.

$$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \frac{x^9}{9!} \ldots$$

By using this expansion, we might be able to tell Z3 that the input has to be like a sinewave but because of the complex equation it takes some considerable amount(not feasible) of time for Z3 to verify.
Also by just considering the approximated sinewave as defined, we weren't able to get a good approximate when we tried to check the approximated function over a signal $S = A\sin(2\pi f t + \phi)$. Then, we realized that the value that sin would evaluate can be $> 2\pi$ so in order to generalize it to some value that is within the range 0 to 2π of a sine function we implemented a MOD simulation in Z3 (since Z3 doesn't have a MOD function) so that we are able to get a meaningful result.

We took the approximation to be upto 8 terms and got a good estimate as shown in Figure **??**iii)Approx.

But the problem was that giving such a complex constraint to Z3 was making the verification take too long or sometimes even failed(the RAM got overloaded and crashed).

Thus, to easy things up for the Z3 solver, we tried **a third constraint** over continuity of the consecutive sampling points that is the difference between two consecutive points in the input will be within some threshold where the threshold depends on the frequency of the wave.

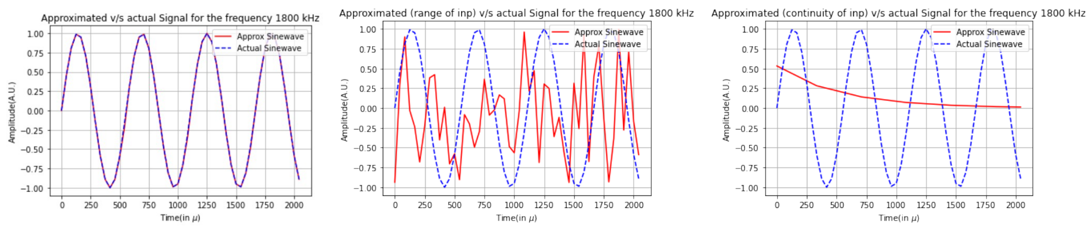


Figure 6.1: The Signal obtained by approximating the sinewave by taylor series/random range of inputs/continuity constraint upto 8 terms w.r.t. the actual wave of frequency 1800KHz without noise

THE NETWORK MODEL In order to get the predicted value of the Neural Network, We need to model the Trained Network using Z3. To achieve that, we extracted the weights and biases from the model that we trained and used a method GET_MODEL() which invokes a feedforward function complying with Z3 variables, such that when given the weights the method will output a solver,an optimizer, an input and an output, where the input and output are basically Z3 variables and the solver would hold the constraints that our Neural Network learned while training and the optimizer will also hold the same constraints.
Then why use both optimizer and solver?
We want to verify some maximizing properties like *the maximum phase change so that the frequency remains in a given threshold*, and optimizer allows us to use a built-in maximize function that can take a z3 variable and maximize its values so that all the constraints still holds good.
We also defined a modified GET_SOLVER_MOD() that takes in Weights and and creates two simulations of the network and gives two sets of input and outputs and a common solver.
We defined this because we wanted to compare the output of the network for a given input as well as some modification over the given input(like reversal/flipping) in order to get the verification w.r.t. the actual predicted output(over the given input).

THE PROPERTIES THAT WE TRIED TO VERIFIED

- **Phase Change:**The phase of a signal S is the angle ϕ, in degrees or radians that the waveform has shifted from a certain reference point along the horizontal zero axis. We define the verification problem as:
  What is the maximum change allowed in the phase of the signal so that the Network still

predicts the frequency within a given threshold?
The threshold that we took was $\pm 1\%$ of the original frequency.
We also tried another variant of the same problem where the verification is *whether there is a phase value that results in the frequency to go beyond the threshold.*
In order to do the verification(s), we took the weights and biases from the Trained Network and fed it to the Z3 function GET_MODEL() as described in the previous paragraph.
This way, we get the input variables and output variable, a solver and an optimizer that constraints the input and output to comply with our trained Network.
The input is then constrained with what was described in the first paragraph(i.e., a function that approximates a sinewave).
We also put a constraint that the range of the sampling points(i.e., the input variables) must be between -1 to +1 as the amplitude is fixed to 1.
We constrain the frequency to be between the permissible range of 1KHz to 5KHz.
Now in order to address the later problem, we have taken the phase to be a Z3 variable and constrained it to be MOD value with respect to 2π and checked the model where every constraint was taken over the solver. To address the former problem, we took the same variable with the same constraint but now over the optimizer and added the maximize constraint on the phase variable and checked the model.

- **Reversal of wave:** Property: Here we want to check if frequencies are independent of the direction in which, signal is read, That is, if we reflect the signal then the frequency predicted by our model remains the same or not. To check this property, the sample points are reversed and fed as input to our model. We first generate signal using z3 approximated sine wave. Let the generated signal be of the form $[a_0, a_1, ...a_{50}]$, then $[a_{50}, a_{49}, ..., a_1]$ is fed as input to our Z3network. Similar to above verification, we ensure our variables comply with our trained model by using Z3 function GET_MODEL(), also , frequency and input is constrained within the range . Here we used solver to find if there exists an input for which the predicted output is not within the threshold.

- **Flipping of wave:** We define 'flipping' of a wave, as its reflection with respect to x-axis. Property:Here we want to check if there exists a frequency,f, for which when the signal is flipped and given as input to our network , the network outout is greater that the tolerance. To check this property, the sign of the sample points are flipped, and fed as input to our model. We first generate signal using z3 approximated sine wave. Let the generated signal be of the form $[a_0, a_1, ...a_{50}]$,then $[-a_0, -a_1, ..., -a_{50}]$ is fed as input to our Z3network. Constraints over neural network is similar to the above verification. Frequency is constrained within the range. Here we used solver to find if there exists a frequency for which the predicted out is not within the threshold.

- **Maximum Amplitude:** For a given signal the amplitude of a signal is the maximum displacement of the signal with respect to the x-axis. Property: Any change in the amplitude doesn't affect the frequency of the signal. We try to find what is the minimum change in the amplitude that leads to the a change in the frequency by some threshold(=5% of the upper range of the frequency that the network can verify), if any. Since it uses optimize function and the calculation is quite heavy for Z3. We try to check a lesser constrained variant of the above problem where we check whether there exists some amplitude other than the given one for which the change in the frequency exceeds the threshold limit using the solver.

6.1 Results obtained from the verification

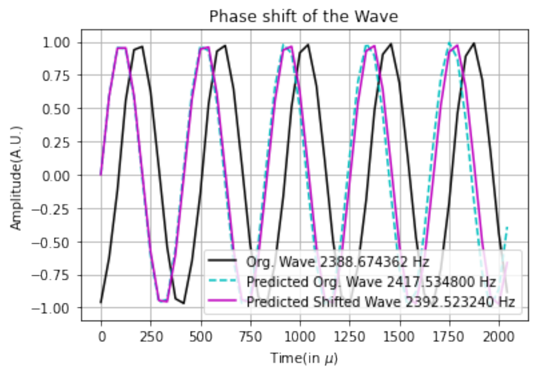


Figure 6.2: The signal obtained by the Z3 solver which has a phase shift that exceeds the threshold of ±5%

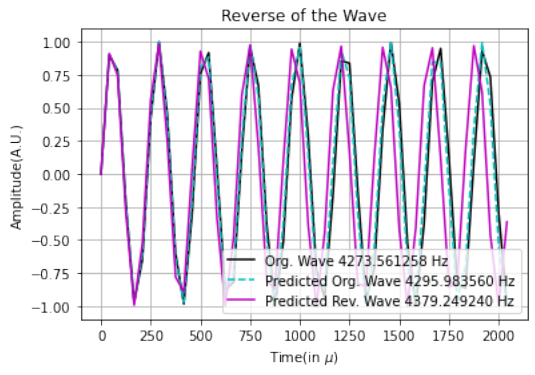


Figure 6.3: The signal obtained by the Z3 solver which is the reversed of the original wave and that exceeds the threshold of ±5%

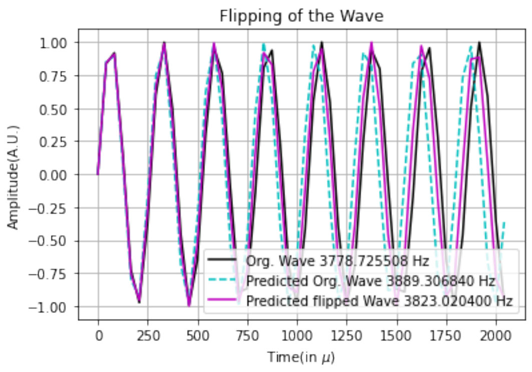


Figure 6.4: The signal obtained by the Z3 solver which is the flipped of the original wave and that exceeds the threshold of ±5%

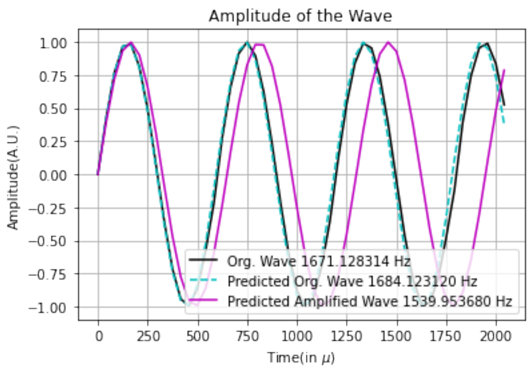


Figure 6.5: The signal obtained by the Z3 solver which has a different amplitude w.r.t. original wave that exceeds the threshold of ±5%