In [11]:

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn import metrics
```

In [12]:

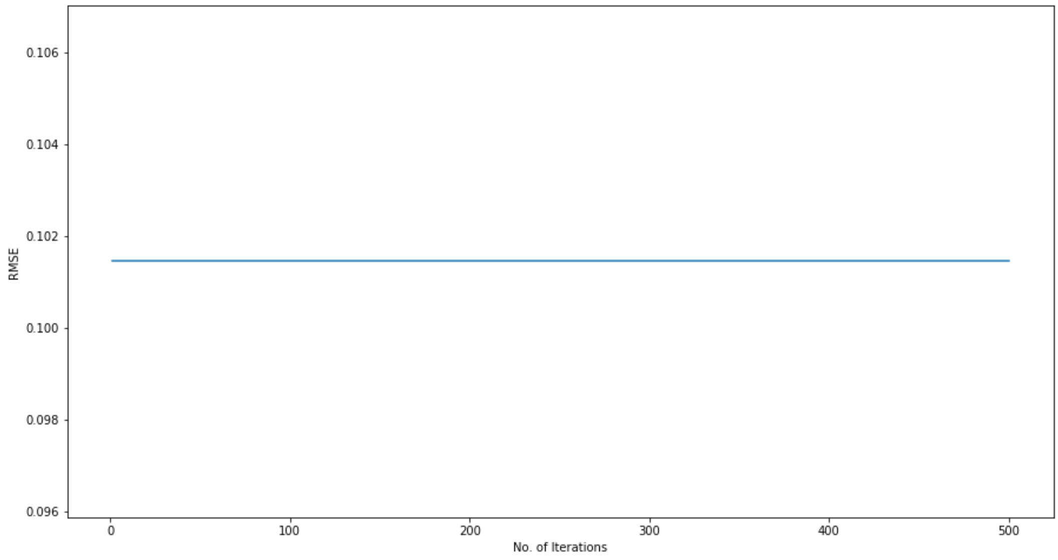
```
'''
training_inputs = np.array([[3, 1.5],
                            [2, 1],
                            [4, 1.5],
                            [3, 1],
                            [3.5, 5],
                            [2, 0.5],
                            [5.5, 1],
                            [1, 1],
                            [4.5, 1]])
training_outputs = np.array([[1, 0, 1, 0, 1, 0, 1, 0, 1]]).T
'''

dataset = pd.read_csv('UTM_EnvironmentalDataSet_2018_4.csv')
X = dataset.iloc[:,12:14]
Y = dataset.iloc[:,15]

X_train,X_test,Y_train,Y_test = train_test_split(X,Y,test_size=0.2)
training_inputs = X_train.values.tolist()
testing_inputs = X_test.values.tolist()
training_outputs = Y_train.values.tolist()
testing_outputs = Y_test.values.tolist()
final_weight = []


training_inputs = np.array(training_inputs)
training_outputs = np.array([training_outputs]).T
testing_inputs = np.array(testing_inputs)
testing_outputs = np.array([testing_outputs]).T

print(testing_outputs.shape)
```

```
(12, 1)
```

In [13]:
```
def sigmoid(x):
    return 1/(1 + np.exp(-x))

def RMSE(z):
    loss = np.sum((z - training_outputs) ** 2)
    #print(loss)
    return np.sqrt(loss / 9)

def ANN(w):
    z1 = sigmoid(np.dot(training_inputs, w[0:2]))
    z2 = sigmoid(np.dot(training_inputs, w[2:4]))
    z3 = sigmoid(np.dot(training_inputs, w[4:6]))
    #print(np.array([z1, z2, z3]))
    z4 = sigmoid(np.dot(np.array([z1, z2, z3]).T, w[6:9]))

    #print(z4)
    output=np.resize(z4,(45,1))
    #print(output)
    return RMSE(output)

# Define Optimization Problem
problem = {
        'CostFunction': ANN,
        'nVar': 1,
        'VarMin': 0,   # Alternatively you can use a "numpy array" with nVar
elements, instead of scalar
        'VarMax': 1,    # Alternatively you can use a "numpy array" with nVa
r elements, instead of scalar
    };

def PSO(problem, MaxIter = 100, PopSize = 100, c1 = 1.4962, c2 = 1.4962, w =
0.7298, wdamp = 1.0):

    best = 0
    R = np.zeros(MaxIter)
    I = np.zeros(MaxIter)
    # Empty Particle Template
    empty_particle = {
        'position': None,
        'velocity': None,
        'cost': None,
        'best_position': None,
        'best_cost': None,
    };

    # Extract Problem Info
    CostFunction = problem['CostFunction']
    VarMin = problem['VarMin']
    VarMax = problem['VarMax']
    nVar = problem['nVar']

    # Initialize Global Best
    gbest = {'position': None, 'cost': np.inf}

    # Create Initial Population
    pop = []
    for i in range(0, PopSize):
        pop.append(empty_particle.copy())
        pop[i]['position'] = np.random.rand(9, 1)
        pop[i]['velocity'] = np.zeros(nVar)
        pop[i]['cost'] = CostFunction(pop[i]['position'])
        pop[i]['best_position'] = pop[i]['position'].copy()
        pop[i]['best_cost'] = pop[i]['cost']

        if pop[i]['best_cost'] < gbest['cost']:
            gbest['position'] = pop[i]['best_position'].copy()
            gbest['cost'] = pop[i]['best_cost']
```

In [14]:
```
if __name__ == '__main__':

    gbest, pop = PSO(problem, MaxIter = 500, PopSize = 50, c1 = 1.5, c2 = 2,
w = 1, wdamp = 0.995)
```

```
Iteration 0: Best Cost = 0.10144974125151822
Iteration 1: Best Cost = 0.10144974125151822
Iteration 2: Best Cost = 0.10144974125151822
Iteration 3: Best Cost = 0.10144974125151822
Iteration 4: Best Cost = 0.10144974125151822
Iteration 5: Best Cost = 0.10144974125151822
Iteration 6: Best Cost = 0.10144974125151822
Iteration 7: Best Cost = 0.10144974125151822
Iteration 8: Best Cost = 0.10144974125151822
Iteration 9: Best Cost = 0.10144974125151822
Iteration 10: Best Cost = 0.10144974125151822
Iteration 11: Best Cost = 0.10144974125151822
Iteration 12: Best Cost = 0.10144974125151822
Iteration 13: Best Cost = 0.10144974125151822
Iteration 14: Best Cost = 0.10144974125151822
Iteration 15: Best Cost = 0.10144974125151822
Iteration 16: Best Cost = 0.10144974125151822
Iteration 17: Best Cost = 0.10144974125151822
Iteration 18: Best Cost = 0.10144974125151822
Iteration 19: Best Cost = 0.10144974125151822
Iteration 20: Best Cost = 0.10144974125151822
Iteration 21: Best Cost = 0.10144974125151822
Iteration 22: Best Cost = 0.10144974125151822
Iteration 23: Best Cost = 0.10144974125151822
Iteration 24: Best Cost = 0.10144974125151822
Iteration 25: Best Cost = 0.10144974125151822
Iteration 26: Best Cost = 0.10144974125151822
Iteration 27: Best Cost = 0.10144974125151822
Iteration 28: Best Cost = 0.10144974125151822
Iteration 29: Best Cost = 0.10144974125151822
Iteration 30: Best Cost = 0.10144974125151822
Iteration 31: Best Cost = 0.10144974125151822
Iteration 32: Best Cost = 0.10144974125151822
Iteration 33: Best Cost = 0.10144974125151822
Iteration 34: Best Cost = 0.10144974125151822
Iteration 35: Best Cost = 0.10144974125151822
Iteration 36: Best Cost = 0.10144974125151822
Iteration 37: Best Cost = 0.10144974125151822
Iteration 38: Best Cost = 0.10144974125151822
Iteration 39: Best Cost = 0.10144974125151822
Iteration 40: Best Cost = 0.10144974125151822
Iteration 41: Best Cost = 0.10144974125151822
Iteration 42: Best Cost = 0.10144974125151822
Iteration 43: Best Cost = 0.10144974125151822
Iteration 44: Best Cost = 0.10144974125151822
Iteration 45: Best Cost = 0.10144974125151822
Iteration 46: Best Cost = 0.10144974125151822
Iteration 47: Best Cost = 0.10144974125151822
Iteration 48: Best Cost = 0.10144974125151822
Iteration 49: Best Cost = 0.10144974125151822
Iteration 50: Best Cost = 0.10144974125151822
Iteration 51: Best Cost = 0.10144974125151822
Iteration 52: Best Cost = 0.10144974125151822
Iteration 53: Best Cost = 0.10144974125151822
Iteration 54: Best Cost = 0.10144974125151822
Iteration 55: Best Cost = 0.10144974125151822
Iteration 56: Best Cost = 0.10144974125151822
Iteration 57: Best Cost = 0.10144974125151822
Iteration 58: Best Cost = 0.10144974125151822
Iteration 59: Best Cost = 0.10144974125151822
Iteration 60: Best Cost = 0.10144974125151822
Iteration 61: Best Cost = 0.10144974125151822
Iteration 62: Best Cost = 0.10144974125151822
Iteration 63: Best Cost = 0.10144974125151822
Iteration 64: Best Cost = 0.10144974125151822
Iteration 65: Best Cost = 0.10144974125151822
Iteration 66: Best Cost = 0.10144974125151822
Iteration 67: Best Cost = 0.10144974125151822
```




```
Best population:

 [[1.        ]
 [0.64965334]
 [0.35722428]
 [0.9424739 ]
 [1.        ]
 [0.35074416]
 [0.        ]
 [0.        ]
 [0.        ]]
```

In [18]:
```python
weight = gbest['position']

z1 = sigmoid(np.dot(testing_inputs, weight[0:2]))
z2 = sigmoid(np.dot(testing_inputs, weight[2:4]))
z3 = sigmoid(np.dot(testing_inputs, weight[4:6]))
#print(np.array([z1, z2, z3]))
z4 = sigmoid(np.dot(np.array([z1, z2, z3]).T, weight[6:9]))

y_pred=np.resize(z4,(12,1))

explained_variance=metrics.explained_variance_score(testing_outputs, y_pred)
print(explained_variance)
```

```
-2.220446049250313e-16
```

In [22]:
```python
max_error=metrics.max_error(testing_outputs, y_pred)
print(max_error)
```

```
0.04580000000000001
```

In [26]:
```python
r2_score=metrics.r2_score(testing_outputs, y_pred)
print(r2_score)
```

```
-8422.327453769325
```

In [28]:
```python
mean_squared_log_error=metrics.mean_squared_log_error(testing_outputs, y_pre
d)
print(mean_squared_log_error)
```

```
0.0009422440426689332
```

In [35]:
```
mean_absolute_error=metrics.mean_absolute_error(testing_outputs, y_pred)
print(mean_absolute_error)
```

```
0.04534166666666667
```

In [ ]: