MC_Data_ESN

January 2, 2020

1 Predicting Mackey Glass using Echo State Neural Network

1.0.1 Importing Required Libraries

[1]:
```
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import ESN
import pandas as pd
```

1.0.2 Set seed for random weights generator

[2]:
```
def set_seed(seed=None):
    """Making the seed (for random values) variable if None"""

    # Set the seed
    if seed is None:
        import time
        seed = int((time.time()*10**6) % 4294967295)
    try:
        np.random.seed(seed)
    except Exception as e:
        print( "!!! WARNING !!!: Seed was not set correctly.")
        print( "!!! Seed that we tried to use: "+str(seed))
        print( "!!! Error message: "+str(e))
        seed = None
    print( "Seed used for random values:", seed)
    return seed
```

[3]:
```
## Set a particular seed for the random generator (for example seed = 42), or␣
↪use a "random" one (seed = None)
# NB: reservoir performances should be averaged accross at least 30 random␣
↪instances (with the same set of parameters)
seed = 42 #None #42
```

[4]:
```
set_seed(seed) #random.seed(seed)
```

```
Seed used for random values: 42
```

[4]: 42

[5]:
```
initLen = 1000
trainLen = initLen + 900
testLen = 10000
```

[6]:
```
df =  pd.read_excel(r'C:\Users\INFO-DSK-02\Desktop\Lorentz Multi Dimension␣
↪Prediction-Phase-2\Final_Version\3D_ReservoirComputing\Input\Mackey Glass␣
↪Data\MCglass.xlsx', index = False)
```

[7]:
```
df.head()
```

[7]:

	t	x
0	0.0	1.200000
1	0.1	1.188060
2	0.2	1.176238
3	0.3	1.164535
4	0.4	1.152947

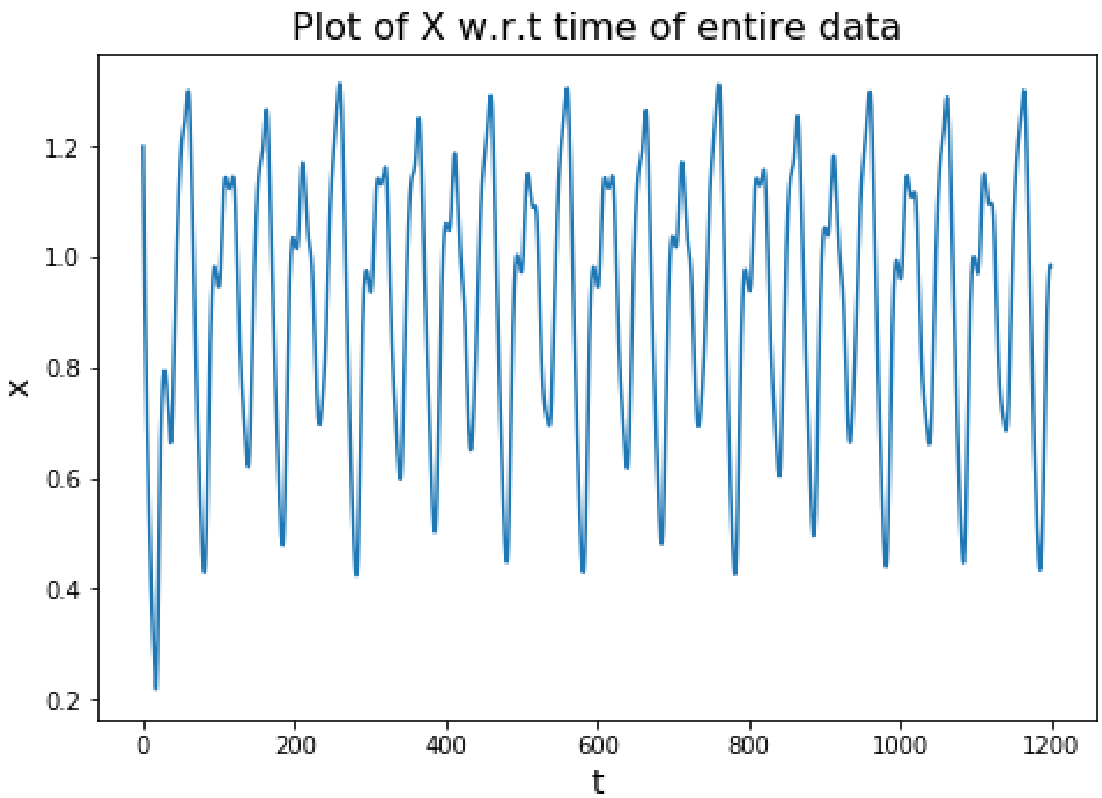
2 EDA

[8]:
```
import os
if not os.path.exists(r"C:\Users\INFO-DSK-02\Desktop\Lorentz Multi Dimension␣
↪Prediction-Phase-2\Final_Version\3D_ReservoirComputing\images\Mackey Glass"):
    os.mkdir(r"C:\Users\INFO-DSK-02\Desktop\Lorentz Multi Dimension␣
↪Prediction-Phase-2\Final_Version\3D_ReservoirComputing\images\Mackey Glass")
```

[9]:
```
from matplotlib import rcParams
rcParams.update({'figure.autolayout': True})
fig = plt.figure()
ax=fig.add_axes([0,0,1,1])
ax.plot(df['t'],df['x'] )
plt.title('Plot of X w.r.t time of entire data',  fontsize=16)
plt.xlabel('t', fontsize = 14)
plt.ylabel('x', fontsize = 14)
plt.savefig(r"C:\Users\INFO-DSK-02\Desktop\Lorentz Multi Dimension␣
↪Prediction-Phase-2\Final_Version\3D_ReservoirComputing\images\Mackey␣
↪Glass\X_with_Time.png", bbox_inches = "tight")
plt.show()
```

```
C:\Users\INFO-DSK-02\Anaconda3\lib\site-packages\ipykernel_launcher.py:9:
UserWarning: This figure includes Axes that are not compatible with
tight_layout, so results might be incorrect.
  if __name__ == '__main__':
C:\Users\INFO-DSK-02\Anaconda3\lib\site-packages\IPython\core\pylabtools.py:128:
UserWarning: This figure includes Axes that are not compatible with
```

```
tight_layout, so results might be incorrect.
  fig.canvas.print_figure(bytes_io, **kw)
```



2.0.1 Split data for training and testing and creating teaches to train ESN on Input data

[10]:
```
data_in = df[['x']]
data_T  =df['t']
```

[11]:
```
data_in = np.array(data_in)
data_t = np.array(data_T)
```

[12]:
```
train_in = np.array(data_in[0:trainLen])
train_out = np.array(data_in[0+10:trainLen+10])
test_in = np.array(data_in[trainLen:trainLen+testLen])
test_out = np.array(data_in[trainLen+10:trainLen+testLen+10])
```

[13]:
```
len(test_out)
```

[13]: 10000

[14]:
```
trainLen+10
```

[14]: 1910

[15]:
```
trainLen+testLen+10
```

[15]: 11910

[16]:
```
train_in_t = np.array(data_T[0:trainLen])
train_out_t = np.array(data_T[0+10:trainLen])
test_in_t = np.array(data_T[trainLen:trainLen+testLen])
test_out_t = np.array(data_T[trainLen+10:trainLen+testLen+10])
```

[17]:
```
len(test_in)
```

[17]: 10000

2.0.2 Modify Parameters to tune ESN for better fit

[18]:
```
n_reservoir = 500 # number of recurrent units
leak_rate = 0.3 # leaking rate (=1/time_constant_of_neurons)
spectral_radius = 1.4 # Scaling of recurrent matrix
input_scaling = 1. # Scaling of input matrix
proba_non_zero_connec_W = 0.2 # Sparsity of recurrent matrix: Perceptage of␣
 ↪non-zero connections in W matrix
proba_non_zero_connec_Win = 1. # Sparsity of input matrix
proba_non_zero_connec_Wfb = 1. # Sparsity of feedback matrix
regularization_coef =  0.01 #None # regularization coefficient, if None,␣
 ↪pseudo-inverse is use instead of ridge regression
```

[19]:
```
n_inputs = 1
input_bias = True # add a constant input to 1
n_outputs = 1
```

[20]:
```
N = n_reservoir#100
dim_inp = n_inputs #26
```

2.0.3 Generating weights for input and hidden layers

[21]:
```
### Generating random weight matrices with custom method
W = np.random.rand(N,N) - 0.5
if input_bias:
    Win = np.random.rand(N,dim_inp+1) - 0.5
else:
    Win = np.random.rand(N,dim_inp) - 0.5
Wfb = np.random.rand(N,n_outputs) - 0.5
```

[22]:
```
## delete the fraction of connections given the sparsity (i.e. proba of␣
↪non-zero connections):
mask = np.random.rand(N,N) # create a mask Uniform[0;1]
W[mask > proba_non_zero_connec_W] = 0 # set to zero some connections given by␣
↪the mask
mask = np.random.rand(N,Win.shape[1])
Win[mask > proba_non_zero_connec_Win] = 0
# mask = np.random.rand(N,Wfb.shape[1])
# Wfb[mask > proba_non_zero_connec_Wfb] = 0
```

[23]:
```
## SCALING of matrices
# scaling of input matrix
Win = Win * input_scaling
# scaling of recurrent matrix
# compute the spectral radius of these weights:
print( 'Computing spectral radius...')
original_spectral_radius = np.max(np.abs(np.linalg.eigvals(W)))
#TODO: check if this operation is quicker: max(abs(linalg.eig(W)[0])) #from␣
↪scipy import linalg
print( "default spectral radius before scaling:", original_spectral_radius)
# rescale them to reach the requested spectral radius:
W = W * (spectral_radius / original_spectral_radius)
print( "spectral radius after scaling", np.max(np.abs(np.linalg.eigvals(W))))
```

```
Computing spectral radius…
default spectral radius before scaling: 2.9980673007522407
spectral radius after scaling 1.400000000000013
```

2.1 Input data dimensions

[24]:
```
print('Dimensions of Training data: ', train_in.shape[1])
print('Dimensions of Testing data: ', test_in.shape[1])
```

```
Dimensions of Training data:  1
Dimensions of Testing data:  1
```

2.1.1 Pass Parameters to ESN

[25]:
```
reservoir = ESN.ESN(lr=leak_rate, W=W, Win=Win, input_bias=input_bias,␣
↪ridge=regularization_coef, Wfb=None, fbfunc=None)
```

2.2 Input data to reservoir model

[26]:
```
internal_trained = reservoir.train(inputs=[train_in], teachers=[train_out],␣
↪wash_nr_time_step=initLen, verbose=False)
output_pred, internal_pred = reservoir.run(inputs=[test_in,], reset_state=False)
errorLen = len(test_out[:]) #testLen #2000
```

2.3 Dimensions of the output data

[27]:
```
print('Shape of Output data Dimensions: ', output_pred[0].shape[1])
```

```
Shape of Output data Dimensions:  1
```

2.3.1 Create dataframe for predicted values and test values

[28]:
```
import pandas as pd
df_pred = pd.DataFrame(output_pred[0])
```

[29]:
```
df_pred.shape
```

[29]:
```
(10000, 1)
```

[30]:
```
test_out = pd.DataFrame(test_out)
```

[31]:
```
test_out.shape
```

[31]:
```
(10000, 1)
```

2.3.2 MSE for X

[32]:
```
## printing errors made on test set
# mse = sum( np.square( test_out[:] - output_pred[0] ) ) / errorLen
# print( 'MSE = ' + str( mse ))
mse_x = np.mean((test_out[0][:] - df_pred[0])**2) # Mean Squared Error: see␣
 ↪https://en.wikipedia.org/wiki/Mean_squared_error
rmse_x = np.sqrt(mse_x) # Root Mean Squared Error: see https://en.wikipedia.org/
 ↪wiki/Root-mean-square_deviation for more info
nmrse_mean_x = abs(rmse_x / np.mean(test_out[0][:])) # Normalised RMSE (based␣
 ↪on mean)
nmrse_maxmin_x = rmse_x / abs(np.max(test_out[0][:]) - np.min(test_out[0][:]))␣
 ↪# Normalised RMSE (based on max - min)
```

[33]:
```
print("\n**********  MSE and RMSE for Predictions on X  **********")
print("Errors computed over %d time steps" % (errorLen))
print("\nMean Squared error (MSE) for x : \t\t%.4e " % (mse_x) )
print("Root Mean Squared error (RMSE) for x : \t\t%.4e\n " % rmse_x )
print("Normalized RMSE (based on mean) for x : \t%.4e " % (nmrse_mean_x) )
print("Normalized RMSE (based on max - min) for x : \t%.4e " % (nmrse_maxmin_x)␣
 ↪)
print("*********************************************************\n")
```

```
**********  MSE and RMSE for Predictions on X  **********
Errors computed over 10000 time steps
```

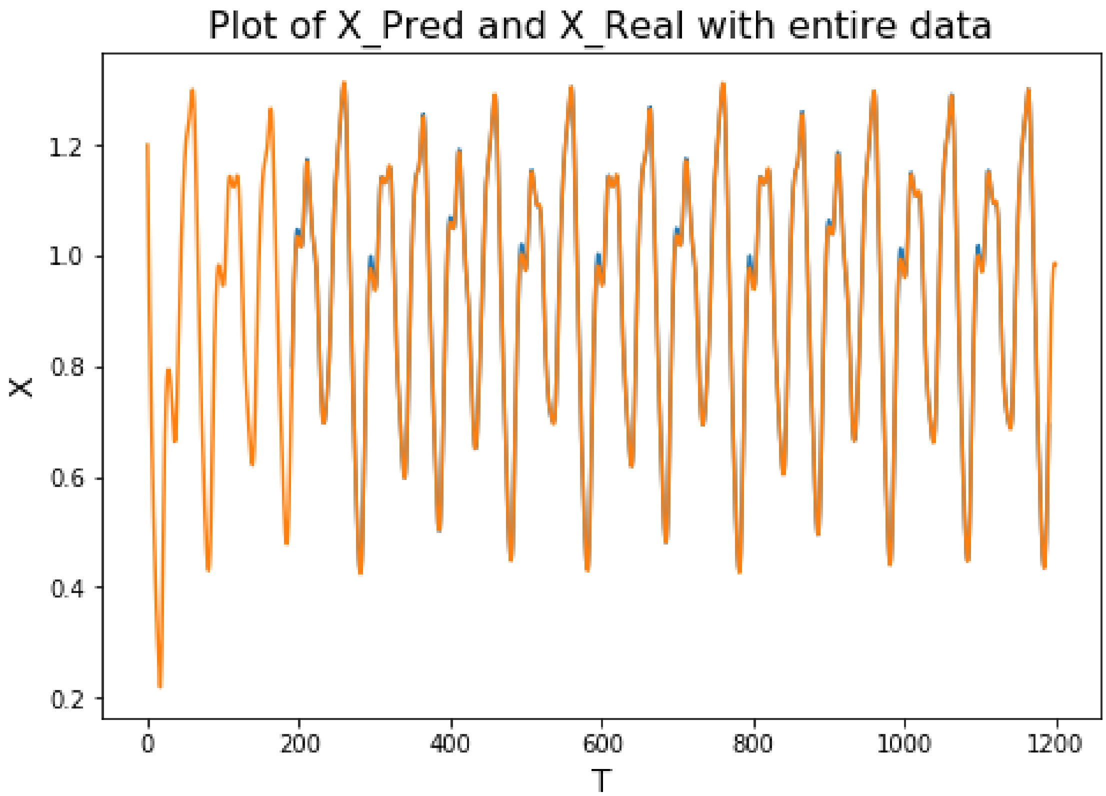
```
Mean Squared error (MSE) for x :               2.8114e-05
Root Mean Squared error (RMSE) for x :         5.3023e-03

Normalized RMSE (based on mean) for x :        5.6965e-03
Normalized RMSE (based on max - min) for x :   5.9620e-03
*********************************************************
```

[34]:

```
fig = plt.figure()
ax=fig.add_axes([0,0,1,1])
ax.plot(test_out_t,df_pred[0])
plt.title('Plot of X_Pred and X_Real with entire data',  fontsize=16)
plt.xlabel('T', fontsize = 14)
plt.ylabel('X', fontsize = 14)
ax.plot(df['t'],df['x'])
plt.savefig(r"C:\Users\INFO-DSK-02\Desktop\Lorentz Multi Dimension␣
 ↪Prediction-Phase-2\Final_Version\3D_ReservoirComputing\images\Mackey␣
 ↪Glass\X_Pred_X_Real_on Entire_data.png", bbox_inches = "tight")
plt.show()
```

```
C:\Users\INFO-DSK-02\Anaconda3\lib\site-packages\ipykernel_launcher.py:8:
UserWarning: This figure includes Axes that are not compatible with
tight_layout, so results might be incorrect.
```



[35]:

```
fig = plt.figure()
ax=fig.add_axes([0,0,1,1])
ax.plot(test_out_t,df_pred[0])
plt.title('Plot of X_Pred and X_Real with test data',  fontsize=16)
plt.xlabel('T', fontsize = 14)
plt.ylabel('X', fontsize = 14)
ax.plot(test_out_t,test_out[0])
plt.savefig(r"C:\Users\INFO-DSK-02\Desktop\Lorentz Multi Dimension␣
 ↪Prediction-Phase-2\Final_Version\3D_ReservoirComputing\images\Mackey␣
 ↪Glass\X_pred_vs_X_Real_with_time_on_Test_Data.png", bbox_inches = "tight")
plt.show()
```

```
C:\Users\INFO-DSK-02\Anaconda3\lib\site-packages\ipykernel_launcher.py:8:
UserWarning: This figure includes Axes that are not compatible with
tight_layout, so results might be incorrect.
```

3 Plotting Local Error from predicted and actual values

[36]:
```
df_local_error = pd.DataFrame()
```

[37]:
```
df_local_error['X_Local_Error'] = test_out[0] - df_pred[0]
```

[38]:
```
df_local_error.describe()
```

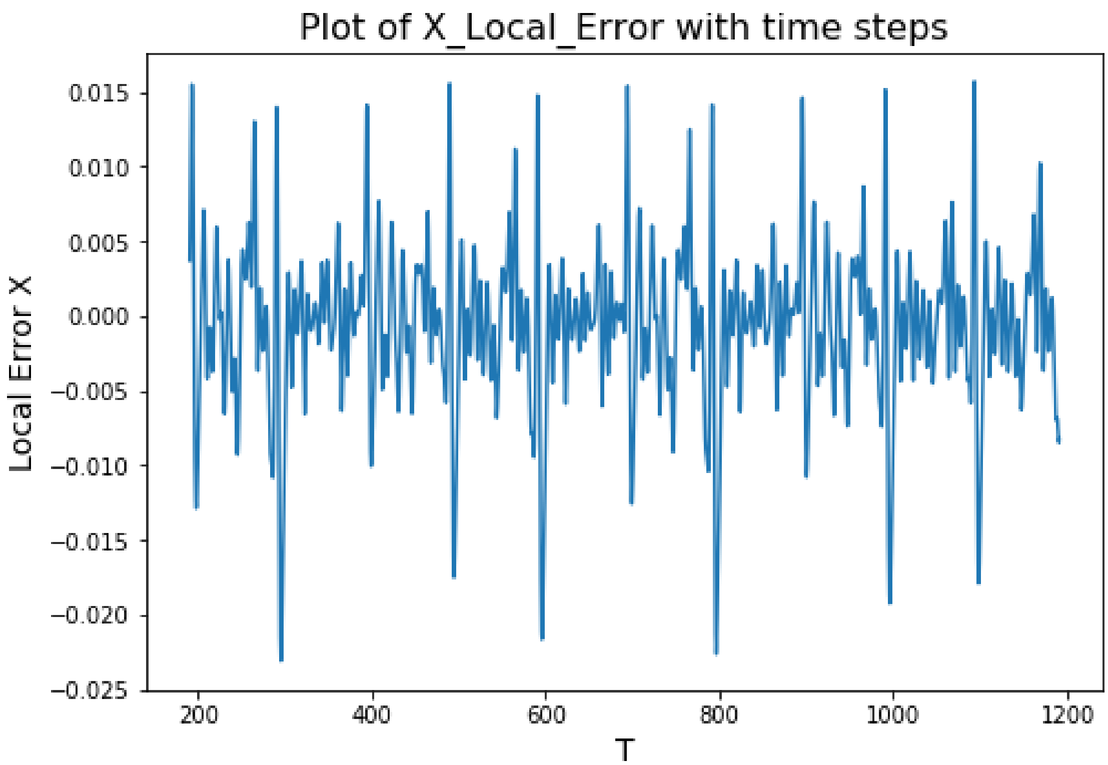
[38]:

	X_Local_Error
count	10000.000000
mean	-0.000563
std	0.005273
min	-0.023076
25%	-0.002969
50%	-0.000263
75%	0.002214
max	0.015720

[46]:
```
fig = plt.figure()
ax=fig.add_axes([0,0,1,1])
```

```
ax.plot(test_out_t,df_local_error['X_Local_Error'] )
plt.title('Plot of X_Local_Error with time steps',  fontsize=16)
plt.xlabel('T', fontsize = 14)
plt.ylabel('Local Error X', fontsize = 14)
plt.savefig(r"C:\Users\INFO-DSK-02\Desktop\Lorentz Multi Dimension␣
↪Prediction-Phase-2\Final_Version\3D_ReservoirComputing\images\Mackey␣
↪Glass\Local_Error_X.png", bbox_inches = "tight")
plt.show()
```

```
C:\Users\INFO-DSK-02\Anaconda3\lib\site-packages\ipykernel_launcher.py:7:
UserWarning: This figure includes Axes that are not compatible with
tight_layout, so results might be incorrect.
  import sys
```



[40]:
```
df_pred.columns= ['X_pred']
```

[41]:
```
df_pred.head()
```

[41]:

	X_pred
0	0.800015
1	0.805767
2	0.811425

```
3   0.816991
4   0.822466
```

[42]:
```
test_out.columns = ['X_test']
```

[43]:
```
df_out = pd.concat([df_pred, test_out], axis = 1)
```

[44]:
```
df_out['Test_T'] = test_out_t
```

[45]:
```
df_out.to_excel(r'C:\Users\INFO-DSK-02\Desktop\Lorentz Multi Dimension␣
↪Prediction-Phase-2\Final_Version\3D_ReservoirComputing\Output\MC_Data\MG_Output.
↪xlsx', index= False)
```