

## Import required modules

```
In [1]: 1 import numpy as np
2 import pandas as pd
3 import matplotlib.pyplot as plt
4 import math
5
6 from sklearn.metrics import mean_squared_error
7 from sklearn.metrics import r2_score
8
9 from mpl_toolkits.mplot3d import Axes3D
10
```

## Import obtained equations

```
In [2]: 1 eq_T1 = open('T1_model.txt','r').readlines()
2 eq_T2 = open('T2_model.txt','r').readlines()
3 eq_T3 = open('T3_model.txt','r').readlines()
4 eq_T4 = open('T4_model.txt','r').readlines()
5 eq_T5 = open('T5_model.txt','r').readlines()
6 eq_T6 = open('T6_model.txt','r').readlines()
```

## Import required python scripts

```
In [3]: 1 from clean_data_01 import *
2 from scale_data import *
3 from make_stationary import *
4 from reverse_stationary import *
5 from use_ALAMO import *
```

## Import and make data stationary

```
In [4]: 1 df = clean_data('Process 1 Bi-Weekly.xlsx')
2 df_scaled = scale_data(df.copy())
3 df_stat = make_stationary(df_scaled.copy(),True)
4 df_stat = df_stat.interpolate()
5 df_stat.dropna(inplace=True)
6
7 df_scaled=df_scaled.interpolate()
```

## Split data into train and test and isolate the test data

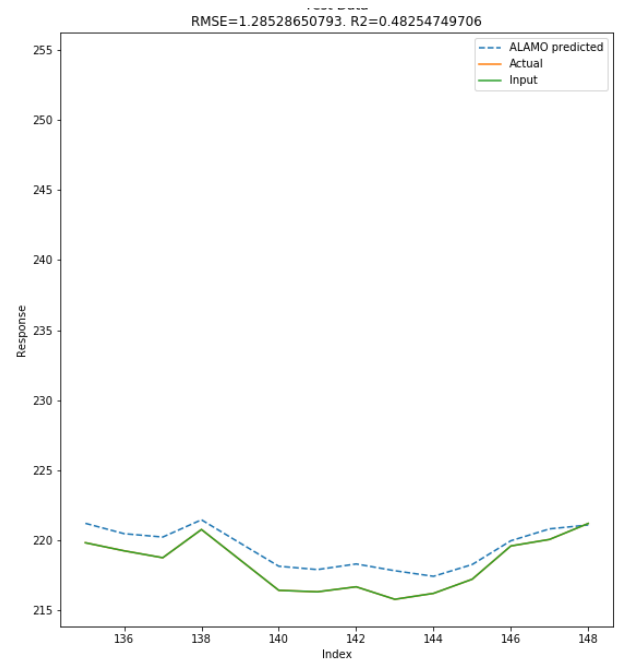
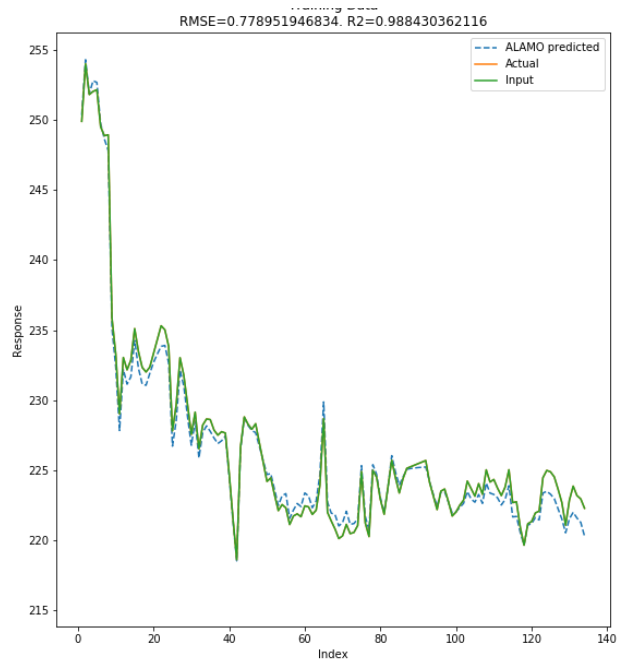
```
In [5]: 1 blind_test_fraction = 0.1
2
3 # Forming the blind_test df
4 train = df_stat[:math.ceil(0.9*len(df_stat))]
5 blind_test = df_stat[math.ceil(0.9*len(df_stat)):]
6
7 train_copy = train.copy()
8
9 # exporting the blind test df as .csv
10 blind_test.to_csv('blind_test.csv')
```

## Obtain input array using previously defined functions for ALAMO

### Simulate and forecast T1

In [6]:

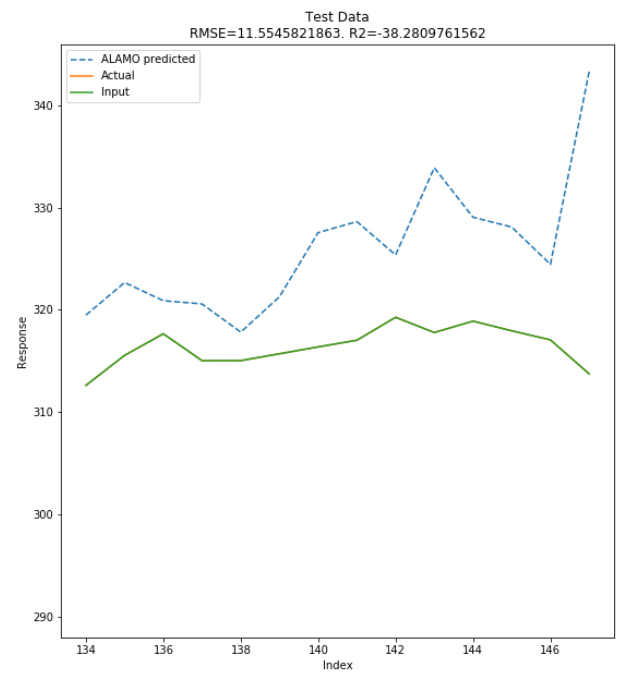
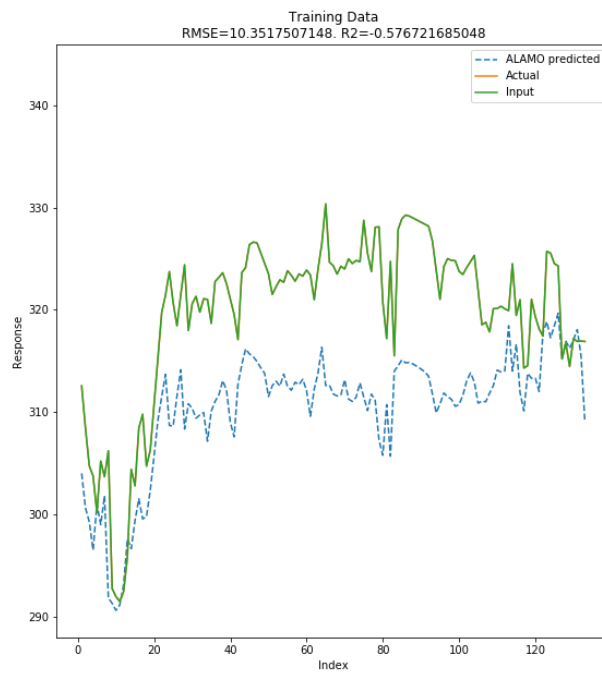
```
1 # T1
2
3 response = 'T1(C)'
4 exog_list = open('T1_exog_list.txt','r').read().splitlines()
5 orders = open('T1_exog_order.txt','r').read().splitlines()
6 orders = [int(i) for i in orders]
7 endog_order = orders[-1]
8 exog_order = orders[:-1]
9 max_ex = max(max(exog_order),endog_order)
10 req_train_data,req_test_data,train_array,test_array = get_data(train_copy,blind_test,'T1(C)',exog_list,exog_o
11
12 input_train_array = np.append(train_array,req_train_data.reshape(-1,1),axis=1)
13 input_test_array = np.append(test_array,req_test_data.reshape(-1,1),axis=1)
14
15 terms,coeff = parse_terms(eq_T1)
16 xlabel = open('T1_xlabels.txt','r').readlines()
17
18 train_predictions = np.array(calc_from_string(input_train_array,terms,coeff,xlabel[0]))
19 test_predictions = np.array(calc_from_string(input_test_array,terms,coeff,xlabel[0]))
20
21 clean_data_df = df.interpolate()
22 scale_data_df = df_scaled
23
24 resp_col_num = train.columns.get_loc(response)
25 train_end = train.shape[0]
26 test_end=train.shape[0]+blind_test.shape[0]
27
28 train_pred_unscaled = reverse_stat(train_predictions,scale_data_df[response].iloc[max_ex],min(clean_data_df[r
29 val_pred_unscaled = reverse_stat(test_predictions,scale_data_df[response].iloc[train_end],min(clean_data_df[r
30
31 plt.figure(figsize=(20,10))
32 ax1=plt.subplot(121)
33
34 plt.plot(range(max_ex,train_end),train_pred_unscaled,'--',label='ALAMO predicted')
35 actual_data=clean_data_df[response].iloc[max_ex+1:train_end+1]
36 plt.plot(range(max_ex,train_end),actual_data,label='Actual')
37
38 ### CHECK REVERSE STATIONARITY by applying reverse_stat on df_stat to get original data
39 input_val=reverse_stat(df_stat[response].iloc[max_ex:train_end],scale_data_df[response].iloc[max_ex],min(clea
40 plt.plot(range(max_ex,train_end),input_val,label='Input')
41
42
43 plt.title('Training Data'+
44         '\nRMSE='+str(np.sqrt(mean_squared_error(actual_data,train_pred_unscaled)))+
45         '. R2='+str(r2_score(actual_data,train_pred_unscaled)))
46 plt.xlabel('Index')
47 plt.ylabel('Response')
48 plt.legend()
49
50 ax2=plt.subplot(122,sharey=ax1)
51 plt.plot(range(train_end,test_end),val_pred_unscaled,'--',label='ALAMO predicted')
52 actual_data=clean_data_df[response].iloc[train_end+1:test_end+1]
53 plt.plot(range(train_end,test_end),actual_data,label='Actual')
54
55 ### CHECK REVERSE STATIONARITY by applying reverse_stat on df_stat to get original data
56 input_val=reverse_stat(df_stat[response].iloc[train_end:test_end],scale_data_df[response].iloc[train_end],min
57 plt.plot(range(train_end,test_end),input_val,label='Input')
58
59
60 plt.title('Test Data'+
61         '\nRMSE='+str(np.sqrt(mean_squared_error(actual_data,val_pred_unscaled)))+
62         '. R2='+str(r2_score(actual_data,val_pred_unscaled)))
63 plt.xlabel('Index')
64 plt.ylabel('Response')
65 # plt.ylim(min(train_pred_unscaled),max(train_pred_unscaled))
66 plt.legend()
67 plt.show()
68
69 T1_train_pred = train_pred_unscaled
70 T1_test_pred = val_pred_unscaled
71
72 train_copy['T1(C)'] = np.append(np.full((train_copy.shape[0]-train_predictions.shape[0],),np.NaN),train_predi
```



**Simulate and forecast T2**

In [7]:

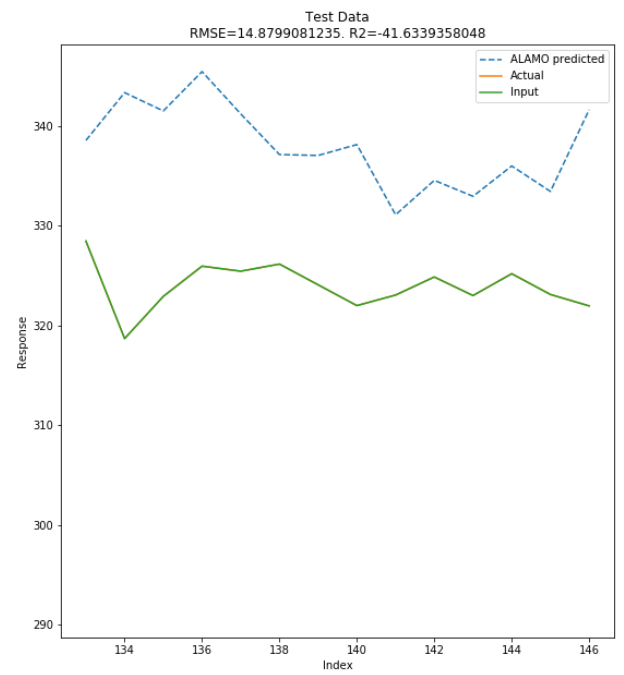
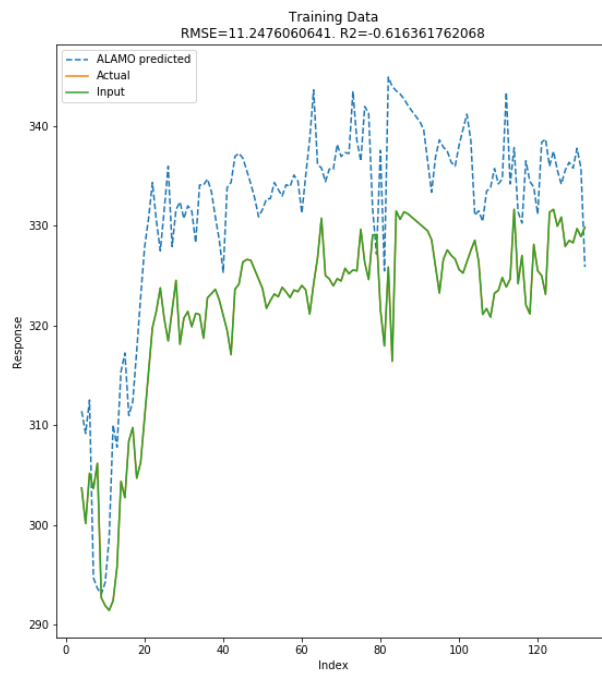
```
1 # T2
2
3 train_copy.dropna(inplace=True)
4
5 response = 'T2(C)'
6 exog_list = open('T2_exog_list.txt','r').read().splitlines()
7 orders = open('T2_exog_order.txt','r').read().splitlines()
8 orders = [int(i) for i in orders]
9 endog_order = orders[-1]
10 exog_order = orders[:-1]
11 max_ex = max(max(exog_order),endog_order)
12 req_train_data,req_test_data,train_array,test_array = get_data(train_copy,blind_test,'T2(C)',exog_list,exog_o
13
14 input_train_array = np.append(train_array,req_train_data.reshape(-1,1),axis=1)
15 input_test_array = np.append(test_array,req_test_data.reshape(-1,1),axis=1)
16
17 terms,coeff = parse_terms(eq_T2)
18 xlabels = open('T2_xlabels.txt','r').readlines()
19
20 train_predictions = np.array(calc_from_string(input_train_array,terms,coeff,xlabels[0]))
21 test_predictions = np.array(calc_from_string(input_test_array,terms,coeff,xlabels[0]))
22
23 clean_data_df = df.interpolate()
24 scale_data_df = df_scaled
25
26 resp_col_num = train.columns.get_loc(response)
27 train_end = train_copy.shape[0]
28 test_end=train_copy.shape[0]+blind_test.shape[0]
29
30 train_pred_unscaled = reverse_stat(train_predictions,scale_data_df[response].iloc[max_ex],min(clean_data_df[r
31 val_pred_unscaled = reverse_stat(test_predictions,scale_data_df[response].iloc[train_end],min(clean_data_df[r
32
33 plt.figure(figsize=(20,10))
34 ax1=plt.subplot(121)
35
36 plt.plot(range(max_ex,train_end),train_pred_unscaled,'--',label='ALAMO predicted')
37 actual_data=clean_data_df[response].iloc[max_ex+1:train_end+1]
38 plt.plot(range(max_ex,train_end),actual_data,label='Actual')
39
40 ### CHECK REVERSE STATIONARITY by applying reverse_stat on df_stat to get original data
41 input_val=reverse_stat(df_stat[response].iloc[max_ex:train_end],scale_data_df[response].iloc[max_ex],min(clea
42 plt.plot(range(max_ex,train_end),input_val,label='Input')
43
44
45 plt.title('Training Data'+
46         '\nRMSE='+str(np.sqrt(mean_squared_error(actual_data,train_pred_unscaled)))+
47         '. R2='+str(r2_score(actual_data,train_pred_unscaled)))
48 plt.xlabel('Index')
49 plt.ylabel('Response')
50 plt.legend()
51
52 ax2=plt.subplot(122,sharey=ax1)
53 plt.plot(range(train_end,test_end),val_pred_unscaled,'--',label='ALAMO predicted')
54 actual_data=clean_data_df[response].iloc[train_end+1:test_end+1]
55 plt.plot(range(train_end,test_end),actual_data,label='Actual')
56
57 ### CHECK REVERSE STATIONARITY by applying reverse_stat on df_stat to get original data
58 input_val=reverse_stat(df_stat[response].iloc[train_end:test_end],scale_data_df[response].iloc[train_end],min
59 plt.plot(range(train_end,test_end),input_val,label='Input')
60
61
62 plt.title('Test Data'+
63         '\nRMSE='+str(np.sqrt(mean_squared_error(actual_data,val_pred_unscaled)))+
64         '. R2='+str(r2_score(actual_data,val_pred_unscaled)))
65 plt.xlabel('Index')
66 plt.ylabel('Response')
67 # plt.ylim(min(train_pred_unscaled),max(train_pred_unscaled))
68 plt.legend()
69 plt.show()
70
71 T2_train_pred = train_pred_unscaled
72 T2_test_pred = val_pred_unscaled
73
74 train_copy['T2(C)'] = np.append(np.full((train_copy.shape[0]-train_predictions.shape[0]),),np.NaN),train_predi
```



**Simulate and forecast T3**

In [8]:

```
1 # T3
2
3 train_copy.dropna(inplace=True)
4
5 response = 'T3(C)'
6 exog_list = open('T3_exog_list.txt','r').read().splitlines()
7 orders = open('T3_exog_order.txt','r').read().splitlines()
8 orders = [int(i) for i in orders]
9 endog_order = orders[-1]
10 exog_order = orders[:-1]
11 max_ex = max(max(exog_order),endog_order)
12 req_train_data,req_test_data,train_array,test_array = get_data(train_copy,blind_test,'T3(C)',exog_list,exog_o
13
14 input_train_array = np.append(train_array,req_train_data.reshape(-1,1),axis=1)
15 input_test_array = np.append(test_array,req_test_data.reshape(-1,1),axis=1)
16
17 terms,coeff = parse_terms(eq_T3)
18 xlabel = open('T3_xlabels.txt','r').readlines()
19
20 train_predictions = np.array(calc_from_string(input_train_array,terms,coeff,xlabel[0]))
21 test_predictions = np.array(calc_from_string(input_test_array,terms,coeff,xlabel[0]))
22
23 clean_data_df = df.interpolate()
24 scale_data_df = df_scaled
25
26 resp_col_num = train.columns.get_loc(response)
27 train_end = train_copy.shape[0]
28 test_end=train_copy.shape[0]+blind_test.shape[0]
29
30 train_pred_unscaled = reverse_stat(train_predictions,scale_data_df[response].iloc[max_ex],min(clean_data_df[r
31 val_pred_unscaled = reverse_stat(test_predictions,scale_data_df[response].iloc[train_end],min(clean_data_df[r
32
33 plt.figure(figsize=(20,10))
34 ax1=plt.subplot(121)
35
36 plt.plot(range(max_ex,train_end),train_pred_unscaled,'--',label='ALAMO predicted')
37 actual_data=clean_data_df[response].iloc[max_ex+1:train_end+1]
38 plt.plot(range(max_ex,train_end),actual_data,label='Actual')
39
40 ### CHECK REVERSE STATIONARITY by applying reverse_stat on df_stat to get original data
41 input_val=reverse_stat(df_stat[response].iloc[max_ex:train_end],scale_data_df[response].iloc[max_ex],min(clea
42 plt.plot(range(max_ex,train_end),input_val,label='Input')
43
44
45 plt.title('Training Data'+
46         '\nRMSE='+str(np.sqrt(mean_squared_error(actual_data,train_pred_unscaled)))+
47         '. R2='+str(r2_score(actual_data,train_pred_unscaled)))
48 plt.xlabel('Index')
49 plt.ylabel('Response')
50 plt.legend()
51
52 ax2=plt.subplot(122,sharey=ax1)
53 plt.plot(range(train_end,test_end),val_pred_unscaled,'--',label='ALAMO predicted')
54 actual_data=clean_data_df[response].iloc[train_end+1:test_end+1]
55 plt.plot(range(train_end,test_end),actual_data,label='Actual')
56
57 ### CHECK REVERSE STATIONARITY by applying reverse_stat on df_stat to get original data
58 input_val=reverse_stat(df_stat[response].iloc[train_end:test_end],scale_data_df[response].iloc[train_end],min
59 plt.plot(range(train_end,test_end),input_val,label='Input')
60
61
62 plt.title('Test Data'+
63         '\nRMSE='+str(np.sqrt(mean_squared_error(actual_data,val_pred_unscaled)))+
64         '. R2='+str(r2_score(actual_data,val_pred_unscaled)))
65 plt.xlabel('Index')
66 plt.ylabel('Response')
67 # plt.ylim(min(train_pred_unscaled),max(train_pred_unscaled))
68 plt.legend()
69 plt.show()
70
71 T3_train_pred = train_pred_unscaled
72 T3_test_pred = val_pred_unscaled
73
74 train_copy['T3(C)'] = np.append(np.full((train_copy.shape[0]-train_predictions.shape[0]),),np.NaN),train_predi
```



**Simulate and forecast T4**

In [9]:

```
1 # T4
2
3 train_copy.dropna(inplace=True)
4
5 response = 'T4(C)'
6 exog_list = open('T4_exog_list.txt','r').read().splitlines()
7 orders = open('T4_exog_order.txt','r').read().splitlines()
8 orders = [int(i) for i in orders]
9 endog_order = orders[-1]
10 exog_order = orders[:-1]
11 max_ex = max(max(exog_order),endog_order)
12 req_train_data,req_test_data,train_array,test_array = get_data(train_copy,blind_test,'T4(C)',exog_list,exog_o
13
14 input_train_array = np.append(train_array,req_train_data.reshape(-1,1),axis=1)
15 input_test_array = np.append(test_array,req_test_data.reshape(-1,1),axis=1)
16
17 terms,coeff = parse_terms(eq_T4)
18 xlabel = open('T4_xlabels.txt','r').readlines()
19
20 train_predictions = np.array(calc_from_string(input_train_array,terms,coeff,xlabel[0]))
21 test_predictions = np.array(calc_from_string(input_test_array,terms,coeff,xlabel[0]))
22
23 clean_data_df = df.interpolate()
24 scale_data_df = df_scaled
25
26 resp_col_num = train.columns.get_loc(response)
27 train_end = train_copy.shape[0]
28 test_end=train_copy.shape[0]+blind_test.shape[0]
29
30 train_pred_unscaled = reverse_stat(train_predictions,scale_data_df[response].iloc[max_ex],min(clean_data_df[r
31 val_pred_unscaled = reverse_stat(test_predictions,scale_data_df[response].iloc[train_end],min(clean_data_df[r
32
33 plt.figure(figsize=(20,10))
34 ax1=plt.subplot(121)
35
36 plt.plot(range(max_ex,train_end),train_pred_unscaled,'--',label='ALAMO predicted')
37 actual_data=clean_data_df[response].iloc[max_ex+1:train_end+1]
38
39
40 print (train_predictions.shape)
41
42
43 plt.plot(range(max_ex,train_end),actual_data,label='Actual')
44
45 ### CHECK REVERSE STATIONARITY by applying reverse_stat on df_stat to get original data
46 input_val=reverse_stat(df_stat[response].iloc[max_ex:train_end],scale_data_df[response].iloc[max_ex],min(clea
47 plt.plot(range(max_ex,train_end),input_val,label='Input')
48
49
50 plt.title('Training Data'+
51           '\nRMSE='+str(np.sqrt(mean_squared_error(actual_data,train_pred_unscaled)))+
52           '. R2='+str(r2_score(actual_data,train_pred_unscaled)))
53 plt.xlabel('Index')
54 plt.ylabel('Response')
55 plt.legend()
56
57 ax2=plt.subplot(122,sharey=ax1)
58 plt.plot(range(train_end,test_end),val_pred_unscaled,'--',label='ALAMO predicted')
59 actual_data=clean_data_df[response].iloc[train_end+1:test_end+1]
60 plt.plot(range(train_end,test_end),actual_data,label='Actual')
61
62 ### CHECK REVERSE STATIONARITY by applying reverse_stat on df_stat to get original data
63 input_val=reverse_stat(df_stat[response].iloc[train_end:test_end],scale_data_df[response].iloc[train_end],min
64 plt.plot(range(train_end,test_end),input_val,label='Input')
65
66
67 plt.title('Test Data'+
68           '\nRMSE='+str(np.sqrt(mean_squared_error(actual_data,val_pred_unscaled)))+
69           '. R2='+str(r2_score(actual_data,val_pred_unscaled)))
70 plt.xlabel('Index')
71 plt.ylabel('Response')
72 # plt.ylim(min(train_pred_unscaled),max(train_pred_unscaled))
73 plt.legend()
74 plt.show()
75
76 T4_train_pred = train_pred_unscaled
```

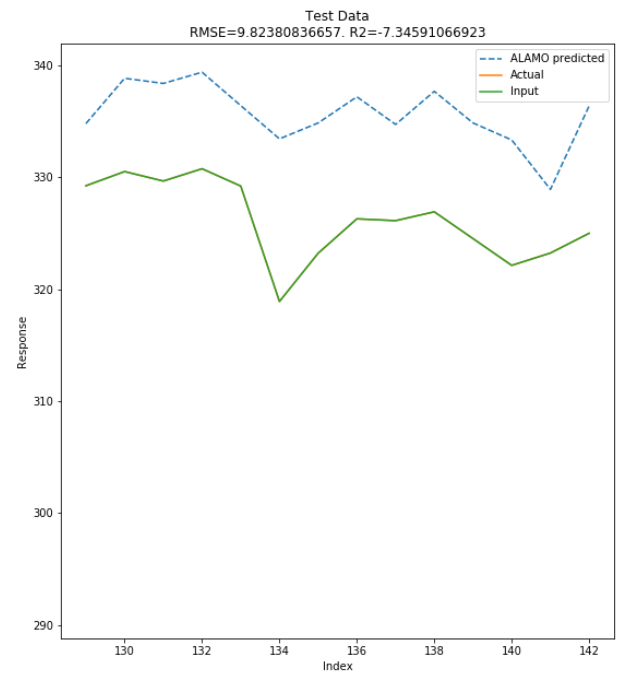
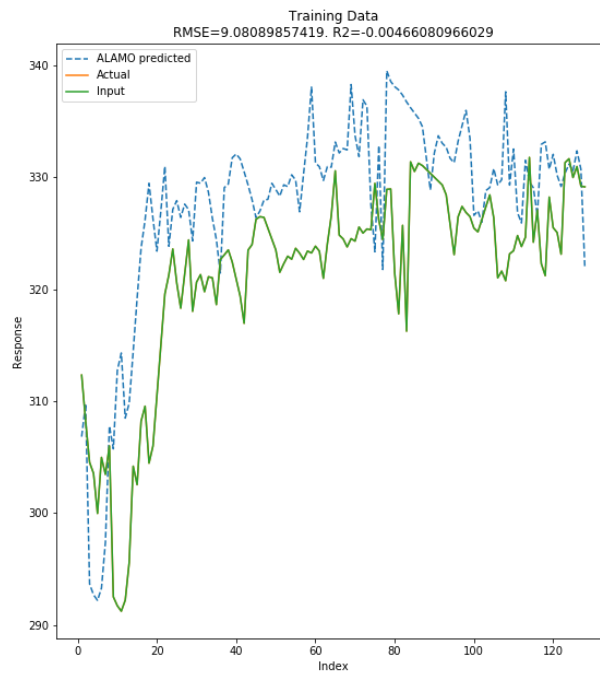


```

77 T4_test_pred = val_pred_unscaled
78
79 train_copy['T4(C)'] = np.append(np.full((train_copy.shape[0]-train_predictions.shape[0]),np.NaN),train_predi

```

(128,)



**Simulate and forecast T5**

```

In [10]: 1 # T5
2
3 train_copy.dropna(inplace=True)
4
5 response = 'T5(C)'
6 exog_list = open('T5_exog_list.txt','r').read().splitlines()
7 orders = open('T5_exog_order.txt','r').read().splitlines()
8 orders = [int(i) for i in orders]
9 endog_order = orders[-1]
10 exog_order = orders[:-1]
11 max_ex = max(max(exog_order),endog_order)
12 req_train_data,req_test_data,train_array,test_array = get_data(train_copy,blind_test,'T5(C)',exog_list,exog_o
13
14 input_train_array = np.append(train_array,req_train_data.reshape(-1,1),axis=1)
15 input_test_array = np.append(test_array,req_test_data.reshape(-1,1),axis=1)
16
17 terms,coeff = parse_terms(eq_T5)
18 xlabel = open('T5_xlabels.txt','r').readlines()
19
20 train_predictions = np.array(calc_from_string(input_train_array,terms,coeff,xlabel[0]))
21 test_predictions = np.array(calc_from_string(input_test_array,terms,coeff,xlabel[0]))
22
23 clean_data_df = df.interpolate()
24 scale_data_df = df_scaled
25
26 resp_col_num = train.columns.get_loc(response)
27 train_end = train_copy.shape[0]
28 test_end=train_copy.shape[0]+blind_test.shape[0]
29
30 train_pred_unscaled = reverse_stat(train_predictions,scale_data_df[response].iloc[max_ex],min(clean_data_df[r
31 val_pred_unscaled = reverse_stat(test_predictions,scale_data_df[response].iloc[train_end],min(clean_data_df[r
32
33 plt.figure(figsize=(20,10))
34 ax1=plt.subplot(121)
35
36 plt.plot(range(max_ex,train_end),train_pred_unscaled,'--',label='ALAMO predicted')
37 actual_data=clean_data_df[response].iloc[max_ex+1:train_end+1]
38
39
40 print (train_predictions.shape)
41
42
43 plt.plot(range(max_ex,train_end),actual_data,label='Actual')
44
45 ### CHECK REVERSE STATIONARITY by applying reverse_stat on df_stat to get original data
46 input_val=reverse_stat(df_stat[response].iloc[max_ex:train_end],scale_data_df[response].iloc[max_ex],min(clea
47 plt.plot(range(max_ex,train_end),input_val,label='Input')
48
49
50 plt.title('Training Data'+
51          '\nRMSE='+str(np.sqrt(mean_squared_error(actual_data,train_pred_unscaled)))+
52          '. R2='+str(r2_score(actual_data,train_pred_unscaled)))
53 plt.xlabel('Index')
54 plt.ylabel('Response')
55 plt.legend()
56
57 ax2=plt.subplot(122,sharey=ax1)
58 plt.plot(range(train_end,test_end),val_pred_unscaled,'--',label='ALAMO predicted')
59 actual_data=clean_data_df[response].iloc[train_end+1:test_end+1]
60 plt.plot(range(train_end,test_end),actual_data,label='Actual')
61
62 ### CHECK REVERSE STATIONARITY by applying reverse_stat on df_stat to get original data
63 input_val=reverse_stat(df_stat[response].iloc[train_end:test_end],scale_data_df[response].iloc[train_end],min
64 plt.plot(range(train_end,test_end),input_val,label='Input')
65
66
67 plt.title('Test Data'+
68          '\nRMSE='+str(np.sqrt(mean_squared_error(actual_data,val_pred_unscaled)))+
69          '. R2='+str(r2_score(actual_data,val_pred_unscaled)))
70 plt.xlabel('Index')
71 plt.ylabel('Response')
72 # plt.ylim(min(train_pred_unscaled),max(train_pred_unscaled))
73 plt.legend()
74 plt.show()
75
76 T5_train_pred = train_pred_unscaled

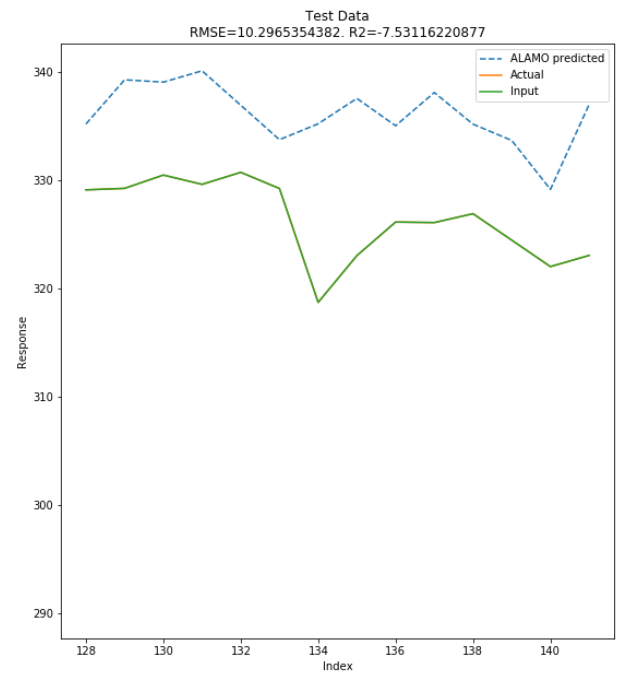
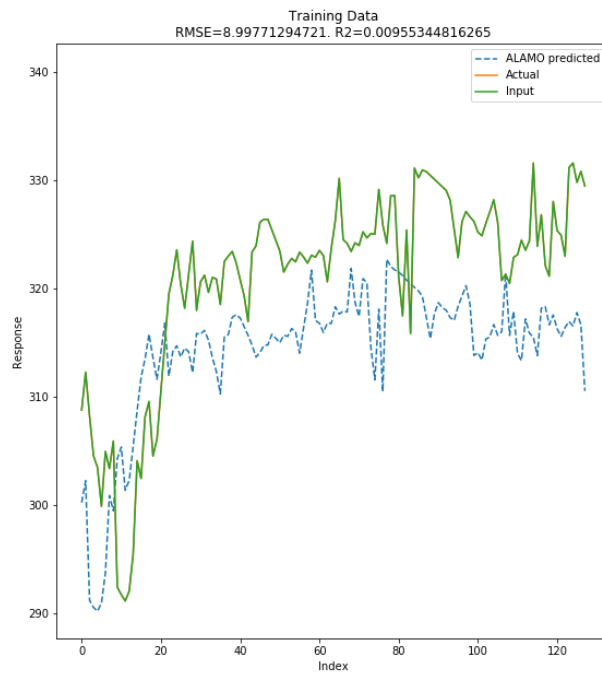
```

```

77 T5_test_pred = val_pred_unscaled
78
79 train_copy['T5(C)'] = np.append(np.full((train_copy.shape[0]-train_predictions.shape[0]),np.NaN),train_predi

```

(128,)



**Simulate and forecast T6**

```

In [11]: 1 # T6
2
3 train_copy.dropna(inplace=True)
4
5 response = 'T6(C)'
6 exog_list = open('T6_exog_list.txt','r').read().splitlines()
7 orders = open('T6_exog_order.txt','r').read().splitlines()
8 orders = [int(i) for i in orders]
9 endog_order = orders[-1]
10 exog_order = orders[:-1]
11 max_ex = max(max(exog_order),endog_order)
12 req_train_data,req_test_data,train_array,test_array = get_data(train_copy,blind_test,'T6(C)',exog_list,exog_o
13
14 input_train_array = np.append(train_array,req_train_data.reshape(-1,1),axis=1)
15 input_test_array = np.append(test_array,req_test_data.reshape(-1,1),axis=1)
16
17 terms,coeff = parse_terms(eq_T6)
18 xlabel = open('T6_xlabels.txt','r').readlines()
19
20 train_predictions = np.array(calc_from_string(input_train_array,terms,coeff,xlabel[0]))
21 test_predictions = np.array(calc_from_string(input_test_array,terms,coeff,xlabel[0]))
22
23 clean_data_df = df.interpolate()
24 scale_data_df = df_scaled
25
26 resp_col_num = train.columns.get_loc(response)
27 train_end = train_copy.shape[0]
28 test_end=train_copy.shape[0]+blind_test.shape[0]
29
30 train_pred_unscaled = reverse_stat(train_predictions,scale_data_df[response].iloc[max_ex],min(clean_data_df[r
31 val_pred_unscaled = reverse_stat(test_predictions,scale_data_df[response].iloc[train_end],min(clean_data_df[r
32
33 plt.figure(figsize=(20,10))
34 ax1=plt.subplot(121)
35
36 plt.plot(range(max_ex,train_end),train_pred_unscaled,'--',label='ALAMO predicted')
37 actual_data=clean_data_df[response].iloc[max_ex+1:train_end+1]
38
39
40 print (train_predictions.shape)
41
42
43 plt.plot(range(max_ex,train_end),actual_data,label='Actual')
44
45 ### CHECK REVERSE STATIONARITY by applying reverse_stat on df_stat to get original data
46 input_val=reverse_stat(df_stat[response].iloc[max_ex:train_end],scale_data_df[response].iloc[max_ex],min(clea
47 plt.plot(range(max_ex,train_end),input_val,label='Input')
48
49
50 plt.title('Training Data'+
51           '\nRMSE='+str(np.sqrt(mean_squared_error(actual_data,train_pred_unscaled)))+
52           '. R2='+str(r2_score(actual_data,train_pred_unscaled)))
53 plt.xlabel('Index')
54 plt.ylabel('Response')
55 plt.legend()
56
57 ax2=plt.subplot(122,sharey=ax1)
58 plt.plot(range(train_end,test_end),val_pred_unscaled,'--',label='ALAMO predicted')
59 actual_data=clean_data_df[response].iloc[train_end+1:test_end+1]
60 plt.plot(range(train_end,test_end),actual_data,label='Actual')
61
62 ### CHECK REVERSE STATIONARITY by applying reverse_stat on df_stat to get original data
63 input_val=reverse_stat(df_stat[response].iloc[train_end:test_end],scale_data_df[response].iloc[train_end],min
64 plt.plot(range(train_end,test_end),input_val,label='Input')
65
66
67 plt.title('Test Data'+
68           '\nRMSE='+str(np.sqrt(mean_squared_error(actual_data,val_pred_unscaled)))+
69           '. R2='+str(r2_score(actual_data,val_pred_unscaled)))
70 plt.xlabel('Index')
71 plt.ylabel('Response')
72 # plt.ylim(min(train_pred_unscaled),max(train_pred_unscaled))
73 plt.legend()
74 plt.show()
75
76 T6_train_pred = train_pred_unscaled

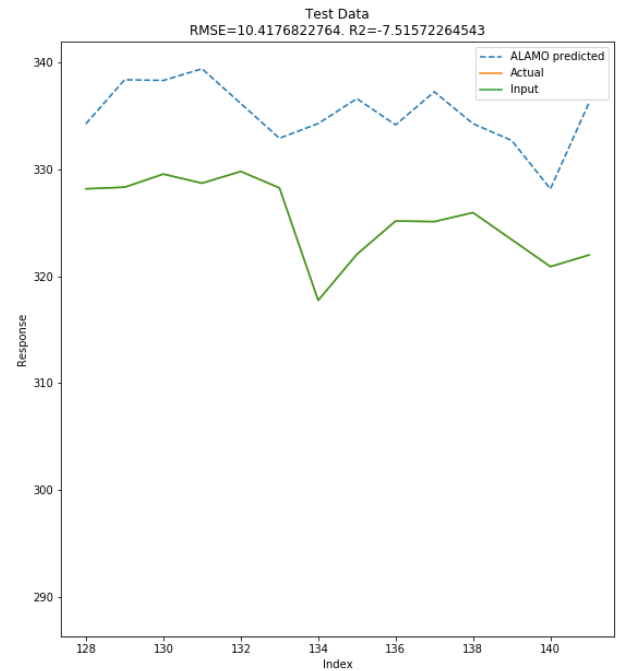
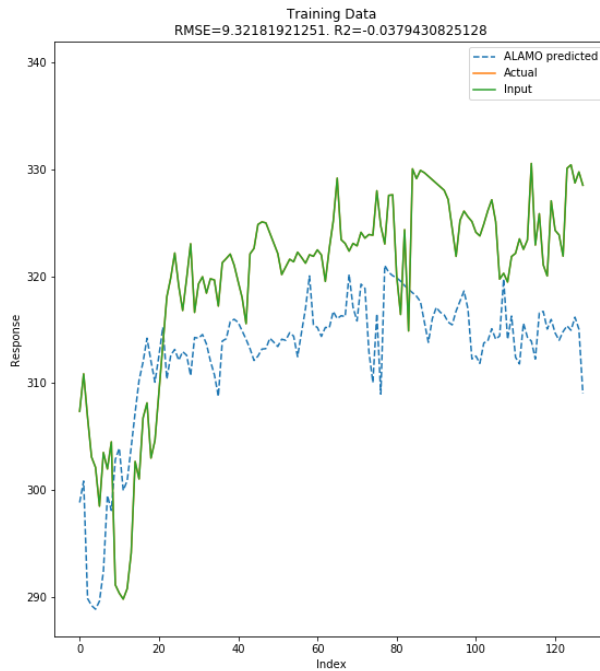
```

```

77 T6_test_pred = val_pred_unscaled
78
79 train_copy['T6(C)'] = np.append(np.full((train_copy.shape[0]-train_predictions.shape[0]),np.NaN),train_predi

```

(128,)



## Combine all results

```

In [12]: 1 # Replace the top rows with NaNs to ensure consistency in the shape
2
3 train_pred_unscaled_T1 = np.append(np.full((train.shape[0]-T1_train_pred.shape[0]),np.NaN),T1_train_pred,axis=0)
4 train_pred_unscaled_T2 = np.append(np.full((train.shape[0]-T2_train_pred.shape[0]),np.NaN),T2_train_pred,axis=0)
5 train_pred_unscaled_T3 = np.append(np.full((train.shape[0]-T3_train_pred.shape[0]),np.NaN),T3_train_pred,axis=0)
6 train_pred_unscaled_T4 = np.append(np.full((train.shape[0]-T4_train_pred.shape[0]),np.NaN),T4_train_pred,axis=0)
7 train_pred_unscaled_T5 = np.append(np.full((train.shape[0]-T5_train_pred.shape[0]),np.NaN),T5_train_pred,axis=0)
8 train_pred_unscaled_T6 = np.append(np.full((train.shape[0]-T6_train_pred.shape[0]),np.NaN),T6_train_pred,axis=0)

```

```

In [13]: 1 # Combine train and test predictions in a single array
2 all_data_T1 = np.append(train_pred_unscaled_T1,T1_test_pred,axis=0)
3 all_data_T2 = np.append(train_pred_unscaled_T2,T2_test_pred,axis=0)
4 all_data_T3 = np.append(train_pred_unscaled_T3,T3_test_pred,axis=0)
5 all_data_T4 = np.append(train_pred_unscaled_T4,T4_test_pred,axis=0)
6 all_data_T5 = np.append(train_pred_unscaled_T5,T5_test_pred,axis=0)
7 all_data_T6 = np.append(train_pred_unscaled_T6,T6_test_pred,axis=0)

```

```

In [14]: 1 # Combine all predictions in a single dataframe
2 predict = pd.DataFrame()
3 predict['T1(C)'] = all_data_T1
4 predict['T2(C)'] = all_data_T2
5 predict['T3(C)'] = all_data_T3
6 predict['T4(C)'] = all_data_T4
7 predict['T5(C)'] = all_data_T5
8 predict['T6(C)'] = all_data_T6

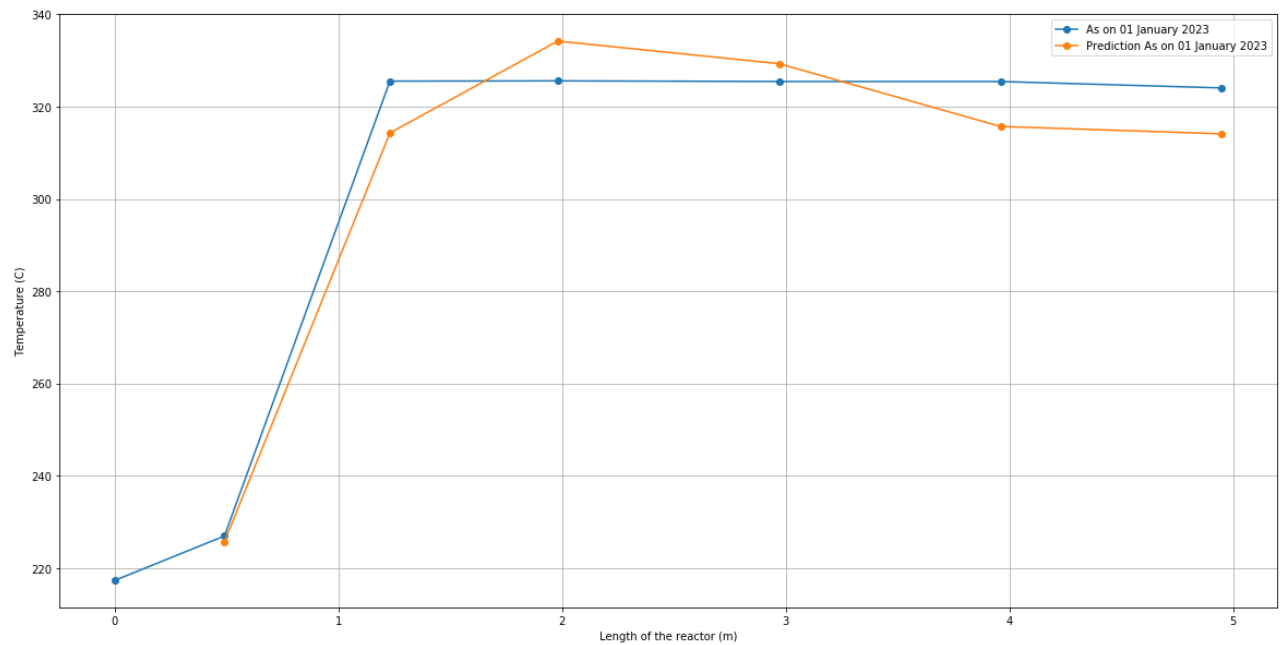
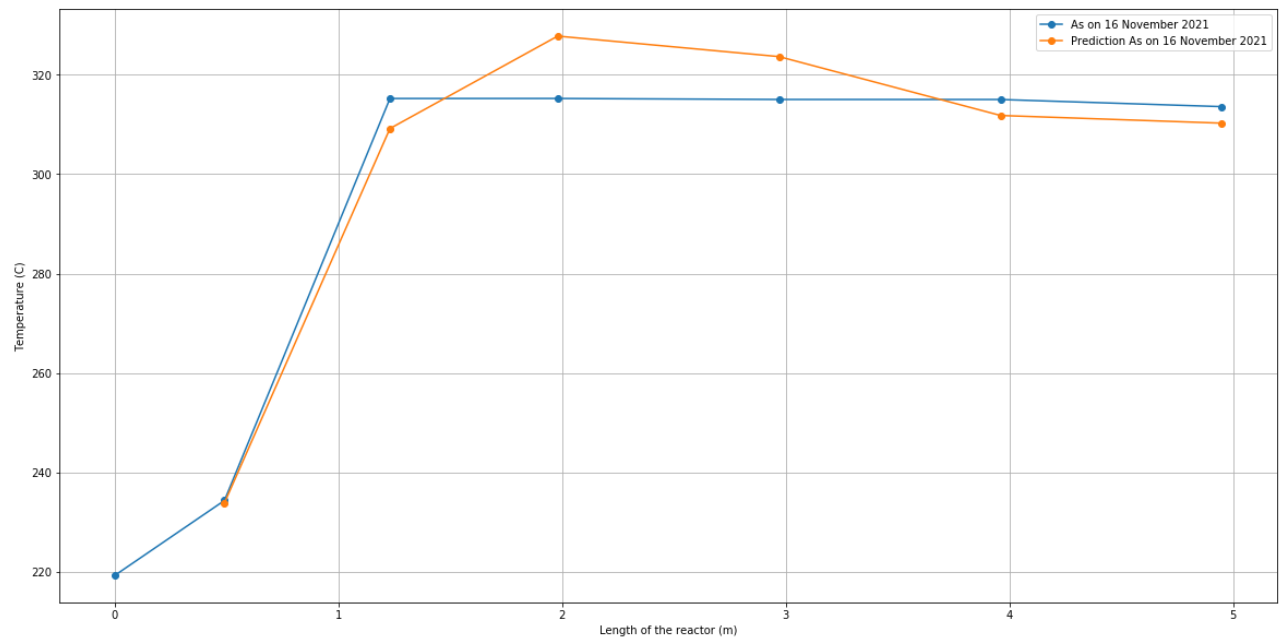
```

## Visualize all predictions

```
In [15]: 1 #Define lengths along the reactor
2 length = [0,0.49,1.23,1.98,2.97,3.96,4.95]
3 length_pred = [0.49,1.23,1.98,2.97,3.96,4.95]
4
5 # Generate an interpolated copy of the original dataframe
6 df2=df.copy().interpolate()
7 df2.dropna(inplace=True)
8
9 predict_df = predict.copy()
10 predict_df['Day']=df2['Day']
11 predict_df.dropna(inplace=True)
```

In [16]:

```
1  #Set figure size
2  fig_size = plt.rcParams["figure.figsize"]
3  fig_size[0] = 20
4  fig_size[1] = 10
5
6  #Randomize seed for constant set of random numbers
7  np.random.seed(100)
8
9  #Randomly pick 4 time points from the data and store in a List
10 random_index_list = [22]
11
12 #Plot the randomly selected points against reactor length
13 for h in sorted(random_index_list):
14     plt.plot(length,[df2.loc[h,'T0(C)'],df2.loc[h,'T1(C)'],df2.loc[h,'T2(C)'],df2.loc[h,'T3(C)'],
15 df2.loc[h,'T4(C)'],df2.loc[h,'T5(C)'],df2.loc[h,'T6(C)']],'-o',
16             label='As on '+str(df2.iloc[h,0].strftime('%d %B %Y')))
17     plt.plot(length_pred,[predict_df.loc[h,'T1(C)'],predict_df.loc[h,'T2(C)'],predict_df.loc[h,'T3(C)'],
18 predict_df.loc[h,'T4(C)'],predict_df.loc[h,'T5(C)'],predict_df.loc[h,'T6(C)']],'-o',
19             label='Prediction As on '+str(df2.iloc[h,0].strftime('%d %B %Y')))
20
21 plt.grid()
22
23 plt.xlabel('Length of the reactor (m)')
24 plt.ylabel('Temperature (C)')
25 # plt.savefig('tempvslength.png')
26
27 plt.legend()
28 plt.show()
29
30
31
32 random_index_list = [49]
33
34 #Plot the randomly selected points against reactor length
35 for h in sorted(random_index_list):
36     plt.plot(length,[df2.loc[h,'T0(C)'],df2.loc[h,'T1(C)'],df2.loc[h,'T2(C)'],df2.loc[h,'T3(C)'],
37 df2.loc[h,'T4(C)'],df2.loc[h,'T5(C)'],df2.loc[h,'T6(C)']],'-o',
38             label='As on '+str(df2.iloc[h,0].strftime('%d %B %Y')))
39     plt.plot(length_pred,[predict_df.loc[h,'T1(C)'],predict_df.loc[h,'T2(C)'],predict_df.loc[h,'T3(C)'],
40 predict_df.loc[h,'T4(C)'],predict_df.loc[h,'T5(C)'],predict_df.loc[h,'T6(C)']],'-o',
41             label='Prediction As on '+str(df2.iloc[h,0].strftime('%d %B %Y')))
42
43 plt.grid()
44
45 plt.xlabel('Length of the reactor (m)')
46 plt.ylabel('Temperature (C)')
47 # plt.savefig('tempvslength.png')
48
49 plt.legend()
50 plt.show()
```



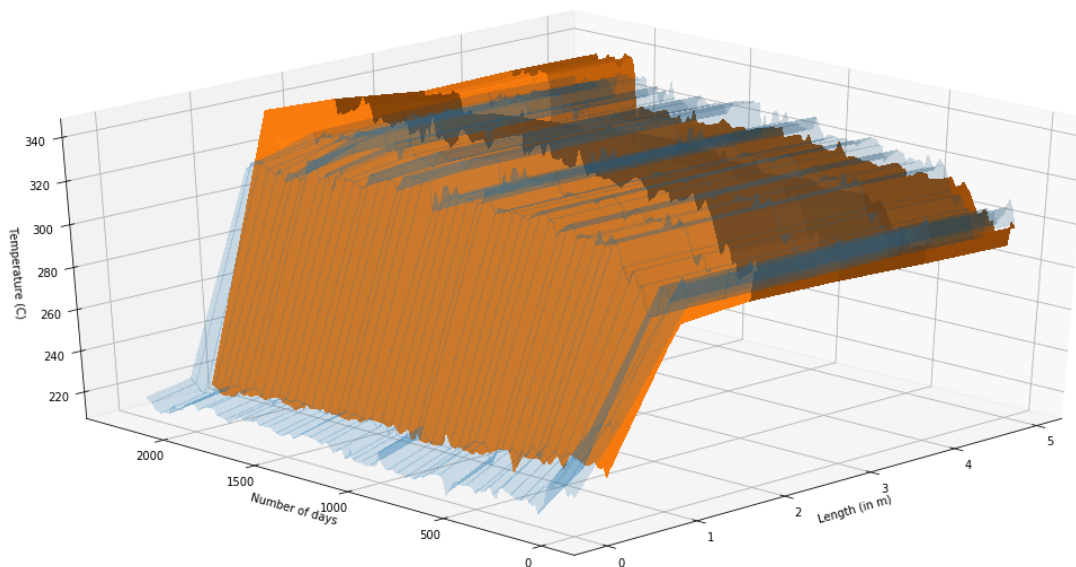
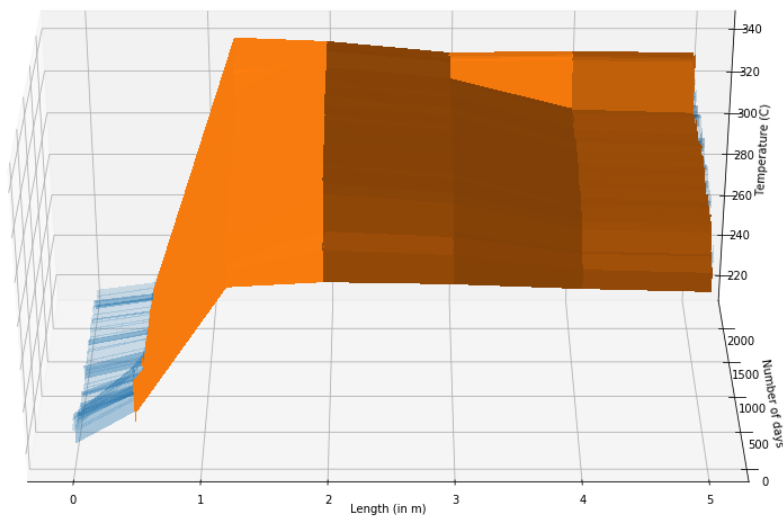
```
In [17]: 1 X,Y = np.meshgrid(length,(df2['Day']-df2['Day'].iloc[0]).dt.days)
2         Z = df2.iloc[:,12:19].values
3
4         X2,Y2 = np.meshgrid(length_pred,(predict_df['Day']-predict_df['Day'].iloc[0]).dt.days)
5         Z2=predict_df.iloc[:,0:6].values
```

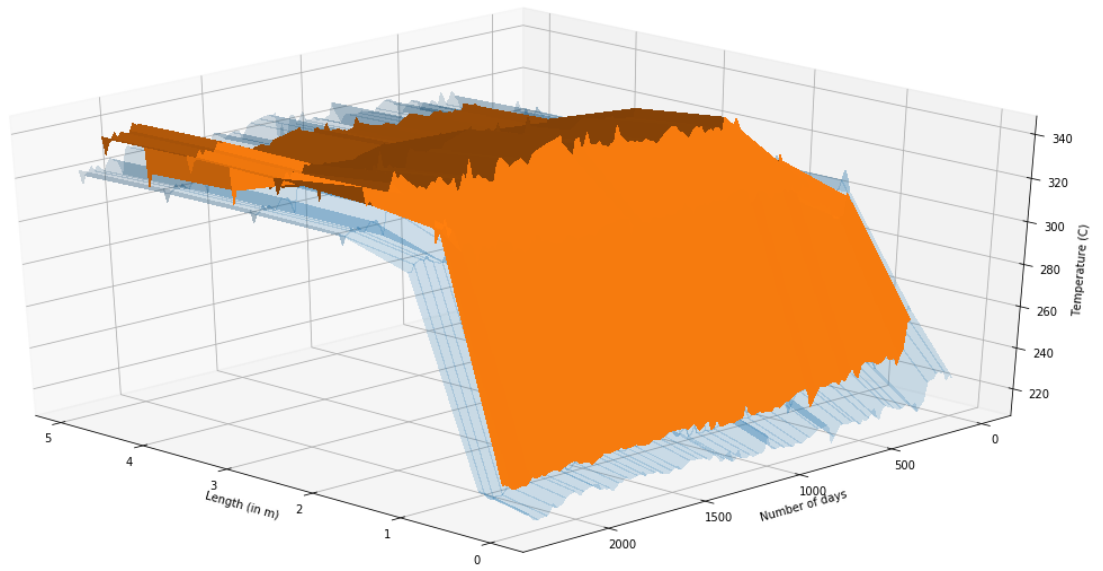
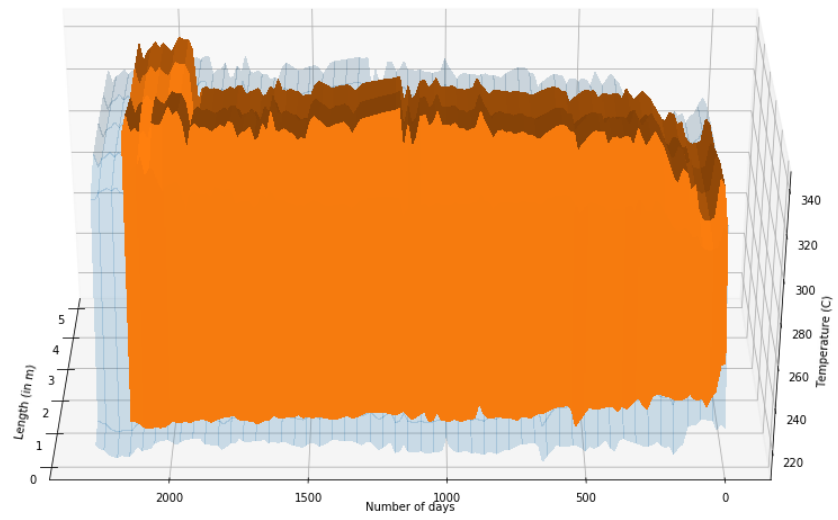


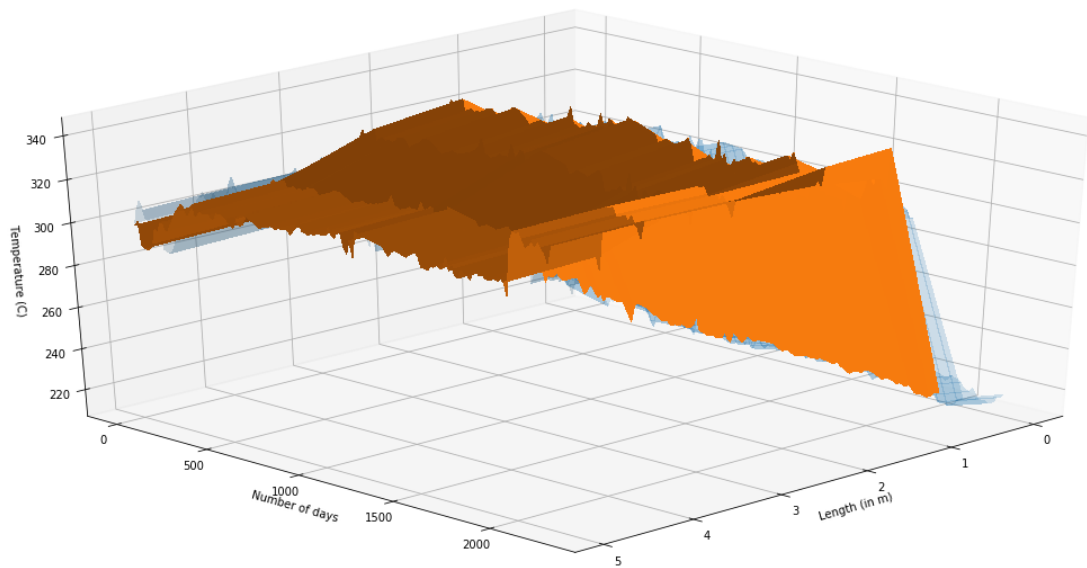
```

In [18]: 1 # Plot 3D plots
2
3 fig_size = plt.rcParams["figure.figsize"]
4 fig_size[0] = 20
5 fig_size[1] = 10
6
7 for angle in range(270,0,-45):
8     ax = plt.figure()
9     fig = ax.gca(projection='3d')
10    fig.set_zlabel('Temperature (C)')
11    fig.set_xlabel('Length (in m)')
12    fig.set_ylabel('Number of days')
13    fig.plot_surface(X,Y,Z,linewidth=0,antialiased=False,alpha=0.2)
14    fig.plot_surface(X2,Y2,Z2,linewidth=0,antialiased=False)
15
16    fig.view_init(30,angle)
17    plt.show()

```



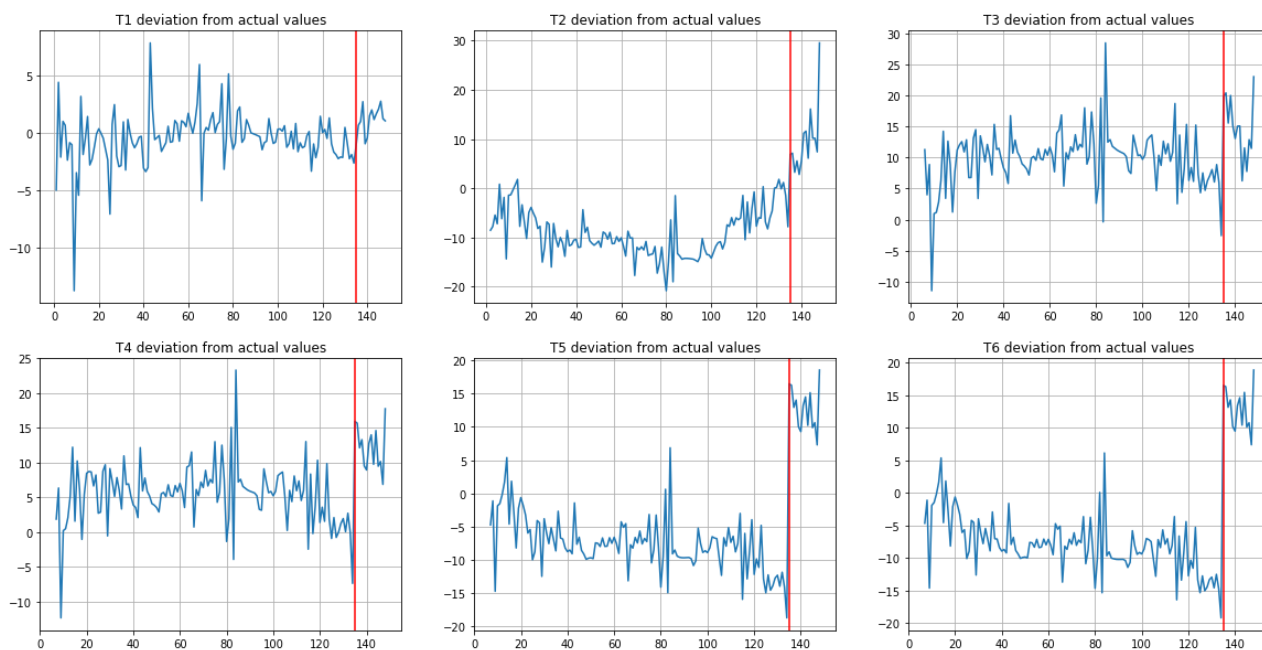




```
In [19]: 1 # Calculate deviations from original values
2
3 T1_deviation = predict['T1(C)'].interpolate() - df['T1(C)'].interpolate()
4 T2_deviation = predict['T2(C)'].interpolate() - df['T2(C)'].interpolate()
5 T3_deviation = predict['T3(C)'].interpolate() - df['T3(C)'].interpolate()
6 T4_deviation = predict['T4(C)'].interpolate() - df['T4(C)'].interpolate()
7 T5_deviation = predict['T5(C)'].interpolate() - df['T5(C)'].interpolate()
8 T6_deviation = predict['T6(C)'].interpolate() - df['T6(C)'].interpolate()
```

In [20]:

```
1 # Plot deviations
2
3 plt.figure(figsize=(20,10))
4
5 plt.subplot(231)
6 plt.plot(T1_deviation)
7 plt.axvline(x=len(train),color='r')
8 plt.title('T1 deviation from actual values')
9 plt.grid()
10
11 plt.subplot(232)
12 plt.plot(T2_deviation)
13 plt.axvline(x=len(train),color='r')
14 plt.title('T2 deviation from actual values')
15 plt.grid()
16
17 plt.subplot(233)
18 plt.plot(T3_deviation)
19 plt.axvline(x=len(train),color='r')
20 plt.title('T3 deviation from actual values')
21 plt.grid()
22
23 plt.subplot(234)
24 plt.plot(T4_deviation)
25 plt.axvline(x=len(train),color='r')
26 plt.title('T4 deviation from actual values')
27 plt.grid()
28
29 plt.subplot(235)
30 plt.plot(T5_deviation)
31 plt.axvline(x=len(train),color='r')
32 plt.title('T5 deviation from actual values')
33 plt.grid()
34
35 plt.subplot(236)
36 plt.plot(T6_deviation)
37 plt.axvline(x=len(train),color='r')
38 plt.title('T6 deviation from actual values')
39 plt.grid()
40
41 plt.show()
```



**Variation of RMSE with time**

```

In [21]: 1 # Define an empty list to track RMSE with time
2
3 RMSE_time = []
4
5 for h in range(predict_df.index[0],149):
6     actual = [df2.loc[h, 'T1(C)'],df2.loc[h, 'T2(C)'],df2.loc[h, 'T3(C)'],df2.loc[h, 'T4(C)'],df2.loc[h, 'T5(C)'],
7     pred = [predict_df.loc[h, 'T1(C)'],predict_df.loc[h, 'T2(C)'],predict_df.loc[h, 'T3(C)'],predict_df.loc[h, 'T
8     RMSE_time.append(np.sqrt(mean_squared_error(actual,pred)))

```

```

In [22]: 1 plt.plot(df2['Day'][predict_df.index[0]:-1],RMSE_time)
2 plt.ylabel('RMSE')
3 plt.xlabel('Time')
4 plt.title('Variation of RMSE with Time')
5 plt.axvline(x=df2['Day'].loc[len(train)],color='r')
6 plt.grid()
7 plt.show()

```

