Import required modules

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import math

from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score

from mpl_toolkits.mplot3d import Axes3D

```

Import obtained equations

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

Import required python scripts

In [3]:
```
from clean_data_01 import *
from scale_data import *
from make_stationary import *
from reverse_stationary import *
from use_ALAMO import *
```

Import and make data stationary

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

df_scaled=df_scaled.interpolate()
```

Split data into train and test and isolate the test data

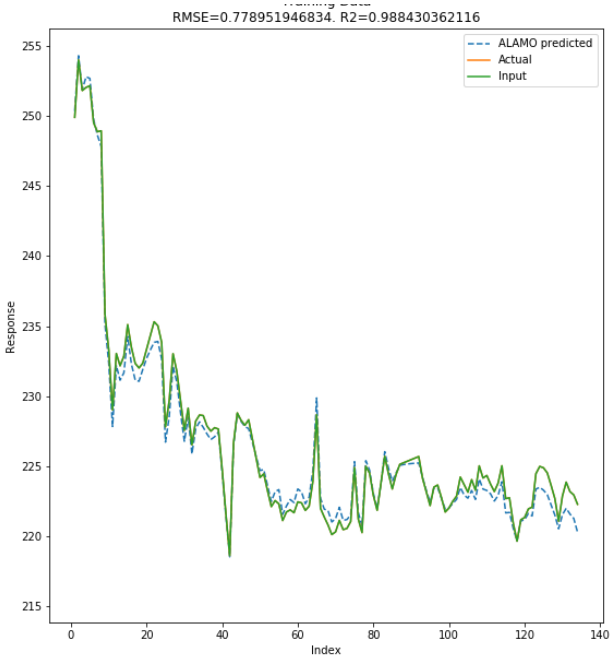
In [5]:
```
blind_test_fraction = 0.1

# Forming the blind_test df
train = df_stat[:math.ceil(0.9*len(df_stat))]
blind_test = df_stat[math.ceil(0.9*len(df_stat)):]

train_copy = train.copy()

# exporting the blind test df as .csv
blind_test.to_csv('blind_test.csv')
```

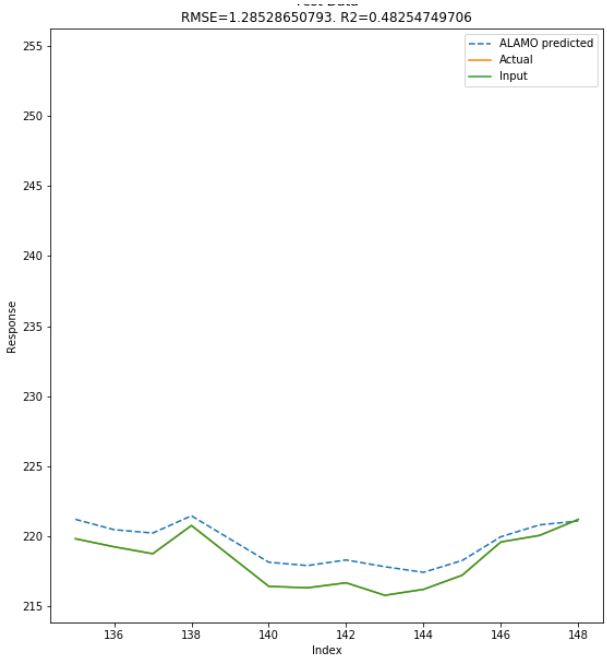
Obtain input array using previously defined functions for ALAMO

Simulate and forecast T1

In [6]:

```
# T1

response = 'T1(C)'
exog_list = open('T1_exog_list.txt','r').read().splitlines()
orders = open('T1_exog_order.txt','r').read().splitlines()
orders = [int(i) for i in orders]
endog_order = orders[-1]
exog_order = orders[:-1]
max_ex = max(max(exog_order),endog_order)
req_train_data,req_test_data,train_array,test_array = get_data(train_copy,blind_test,'T1(C)',exog_list,exog_o

input_train_array = np.append(train_array,req_train_data.reshape(-1,1),axis=1)
input_test_array = np.append(test_array,req_test_data.reshape(-1,1),axis=1)

terms,coeff = parse_terms(eq_T1)
xlabels = open('T1_xlabels.txt','r').readlines()

train_predictions = np.array(calc_from_string(input_train_array,terms,coeff,xlabels[0]))
test_predictions = np.array(calc_from_string(input_test_array,terms,coeff,xlabels[0]))

clean_data_df = df.interpolate()
scale_data_df = df_scaled

resp_col_num = train.columns.get_loc(response)
train_end = train.shape[0]
test_end=train.shape[0]+blind_test.shape[0]

train_pred_unscaled = reverse_stat(train_predictions,scale_data_df[response].iloc[max_ex],min(clean_data_df[r
val_pred_unscaled = reverse_stat(test_predictions,scale_data_df[response].iloc[train_end],min(clean_data_df[r

plt.figure(figsize=(20,10))
ax1=plt.subplot(121)

plt.plot(range(max_ex,train_end),train_pred_unscaled,'--',label='ALAMO predicted')
actual_data=clean_data_df[response].iloc[max_ex+1:train_end+1]
plt.plot(range(max_ex,train_end),actual_data,label='Actual')

### CHECK REVERSE STATIONARITY by applying reverse_stat on df_stat to get original data
input_val=reverse_stat(df_stat[response].iloc[max_ex:train_end],scale_data_df[response].iloc[max_ex],min(clea
plt.plot(range(max_ex,train_end),input_val,label='Input')


plt.title('Training Data'+
          '\nRMSE='+str(np.sqrt(mean_squared_error(actual_data,train_pred_unscaled)))+
          '. R2='+str(r2_score(actual_data,train_pred_unscaled)))
plt.xlabel('Index')
plt.ylabel('Response')
plt.legend()

ax2=plt.subplot(122,sharey=ax1)
plt.plot(range(train_end,test_end),val_pred_unscaled,'--',label='ALAMO predicted')
actual_data=clean_data_df[response].iloc[train_end+1:test_end+1]
plt.plot(range(train_end,test_end),actual_data,label='Actual')

### CHECK REVERSE STATIONARITY by applying reverse_stat on df_stat to get original data
input_val=reverse_stat(df_stat[response].iloc[train_end:test_end],scale_data_df[response].iloc[train_end],min
plt.plot(range(train_end,test_end),input_val,label='Input')


plt.title('Test Data'+
          '\nRMSE='+str(np.sqrt(mean_squared_error(actual_data,val_pred_unscaled)))+
          '. R2='+str(r2_score(actual_data,val_pred_unscaled)))
plt.xlabel('Index')
plt.ylabel('Response')
# plt.ylim(min(train_pred_unscaled),max(train_pred_unscaled))
plt.legend()
plt.show()

T1_train_pred = train_pred_unscaled
T1_test_pred = val_pred_unscaled

train_copy['T1(C)'] = np.append(np.full((train_copy.shape[0]-train_predictions.shape[0],),np.NaN),train_predi
```

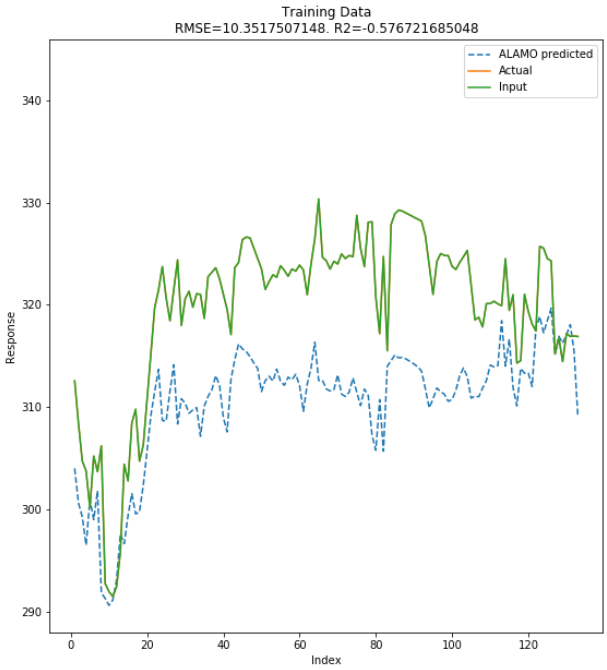


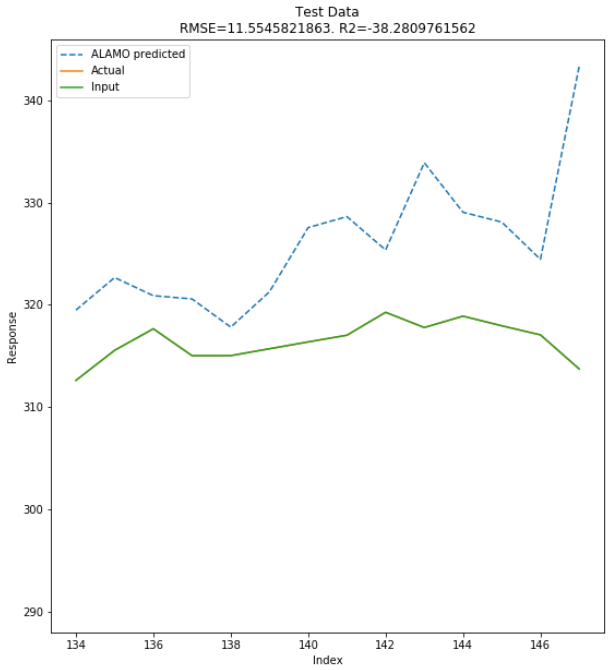


Simulate and forecast T2

In [7]:

```
# T2

train_copy.dropna(inplace=True)

response = 'T2(C)'
exog_list = open('T2_exog_list.txt','r').read().splitlines()
orders = open('T2_exog_order.txt','r').read().splitlines()
orders = [int(i) for i in orders]
endog_order = orders[-1]
exog_order = orders[:-1]
max_ex = max(max(exog_order),endog_order)
req_train_data,req_test_data,train_array,test_array = get_data(train_copy,blind_test,'T2(C)',exog_list,exog_o

input_train_array = np.append(train_array,req_train_data.reshape(-1,1),axis=1)
input_test_array = np.append(test_array,req_test_data.reshape(-1,1),axis=1)

terms,coeff = parse_terms(eq_T2)
xlabels = open('T2_xlabels.txt','r').readlines()

train_predictions = np.array(calc_from_string(input_train_array,terms,coeff,xlabels[0]))
test_predictions = np.array(calc_from_string(input_test_array,terms,coeff,xlabels[0]))

clean_data_df = df.interpolate()
scale_data_df = df_scaled

resp_col_num = train.columns.get_loc(response)
train_end = train_copy.shape[0]
test_end=train_copy.shape[0]+blind_test.shape[0]

train_pred_unscaled = reverse_stat(train_predictions,scale_data_df[response].iloc[max_ex],min(clean_data_df[r
val_pred_unscaled = reverse_stat(test_predictions,scale_data_df[response].iloc[train_end],min(clean_data_df[r

plt.figure(figsize=(20,10))
ax1=plt.subplot(121)

plt.plot(range(max_ex,train_end),train_pred_unscaled,'--',label='ALAMO predicted')
actual_data=clean_data_df[response].iloc[max_ex+1:train_end+1]
plt.plot(range(max_ex,train_end),actual_data,label='Actual')

### CHECK REVERSE STATIONARITY by applying reverse_stat on df_stat to get original data
input_val=reverse_stat(df_stat[response].iloc[max_ex:train_end],scale_data_df[response].iloc[max_ex],min(clea
plt.plot(range(max_ex,train_end),input_val,label='Input')


plt.title('Training Data'+
          '\nRMSE='+str(np.sqrt(mean_squared_error(actual_data,train_pred_unscaled)))+
          '. R2='+str(r2_score(actual_data,train_pred_unscaled)))
plt.xlabel('Index')
plt.ylabel('Response')
plt.legend()

ax2=plt.subplot(122,sharey=ax1)
plt.plot(range(train_end,test_end),val_pred_unscaled,'--',label='ALAMO predicted')
actual_data=clean_data_df[response].iloc[train_end+1:test_end+1]
plt.plot(range(train_end,test_end),actual_data,label='Actual')

### CHECK REVERSE STATIONARITY by applying reverse_stat on df_stat to get original data
input_val=reverse_stat(df_stat[response].iloc[train_end:test_end],scale_data_df[response].iloc[train_end],min
plt.plot(range(train_end,test_end),input_val,label='Input')


plt.title('Test Data'+
          '\nRMSE='+str(np.sqrt(mean_squared_error(actual_data,val_pred_unscaled)))+
          '. R2='+str(r2_score(actual_data,val_pred_unscaled)))
plt.xlabel('Index')
plt.ylabel('Response')
# plt.ylim(min(train_pred_unscaled),max(train_pred_unscaled))
plt.legend()
plt.show()

T2_train_pred = train_pred_unscaled
T2_test_pred = val_pred_unscaled

train_copy['T2(C)'] = np.append(np.full((train_copy.shape[0]-train_predictions.shape[0],),np.NaN),train_predi
```

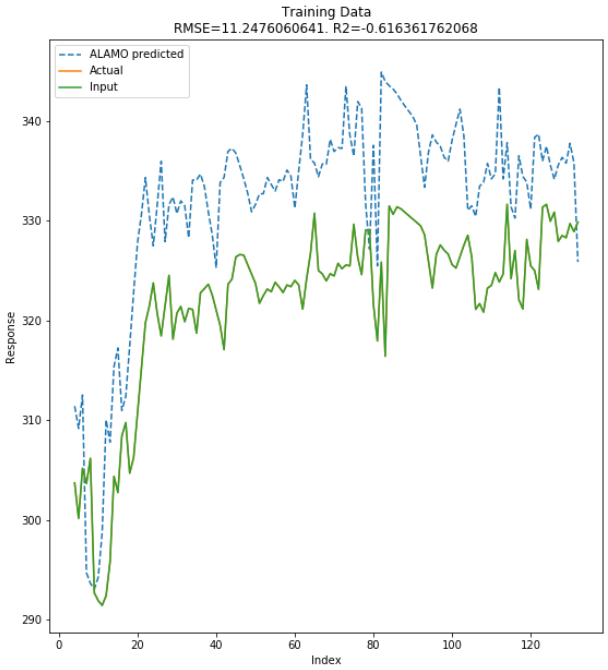


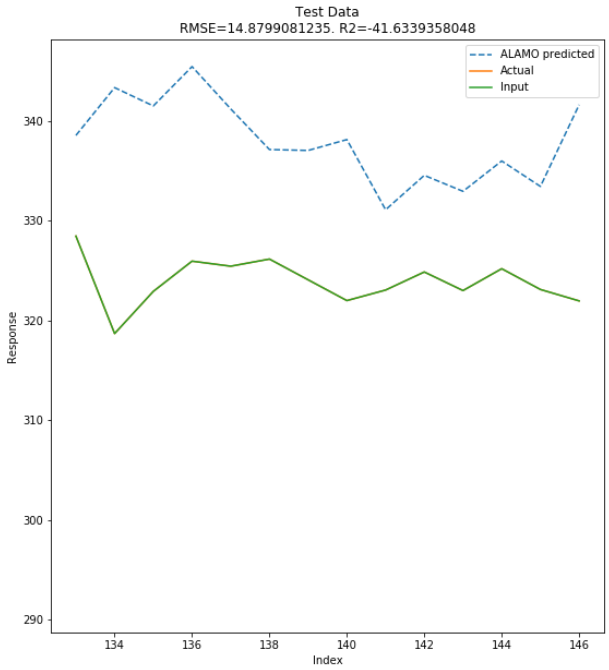


Simulate and forecast T3

In [8]:

```
# T3

train_copy.dropna(inplace=True)

response = 'T3(C)'
exog_list = open('T3_exog_list.txt','r').read().splitlines()
orders = open('T3_exog_order.txt','r').read().splitlines()
orders = [int(i) for i in orders]
endog_order = orders[-1]
exog_order = orders[:-1]
max_ex = max(max(exog_order),endog_order)
req_train_data,req_test_data,train_array,test_array = get_data(train_copy,blind_test,'T3(C)',exog_list,exog_o

input_train_array = np.append(train_array,req_train_data.reshape(-1,1),axis=1)
input_test_array = np.append(test_array,req_test_data.reshape(-1,1),axis=1)

terms,coeff = parse_terms(eq_T3)
xlabels = open('T3_xlabels.txt','r').readlines()

train_predictions = np.array(calc_from_string(input_train_array,terms,coeff,xlabels[0]))
test_predictions = np.array(calc_from_string(input_test_array,terms,coeff,xlabels[0]))

clean_data_df = df.interpolate()
scale_data_df = df_scaled

resp_col_num = train.columns.get_loc(response)
train_end = train_copy.shape[0]
test_end=train_copy.shape[0]+blind_test.shape[0]

train_pred_unscaled = reverse_stat(train_predictions,scale_data_df[response].iloc[max_ex],min(clean_data_df[r
val_pred_unscaled = reverse_stat(test_predictions,scale_data_df[response].iloc[train_end],min(clean_data_df[r

plt.figure(figsize=(20,10))
ax1=plt.subplot(121)

plt.plot(range(max_ex,train_end),train_pred_unscaled,'--',label='ALAMO predicted')
actual_data=clean_data_df[response].iloc[max_ex+1:train_end+1]
plt.plot(range(max_ex,train_end),actual_data,label='Actual')

### CHECK REVERSE STATIONARITY by applying reverse_stat on df_stat to get original data
input_val=reverse_stat(df_stat[response].iloc[max_ex:train_end],scale_data_df[response].iloc[max_ex],min(clea
plt.plot(range(max_ex,train_end),input_val,label='Input')


plt.title('Training Data'+
          '\nRMSE='+str(np.sqrt(mean_squared_error(actual_data,train_pred_unscaled)))+
          '. R2='+str(r2_score(actual_data,train_pred_unscaled)))
plt.xlabel('Index')
plt.ylabel('Response')
plt.legend()

ax2=plt.subplot(122,sharey=ax1)
plt.plot(range(train_end,test_end),val_pred_unscaled,'--',label='ALAMO predicted')
actual_data=clean_data_df[response].iloc[train_end+1:test_end+1]
plt.plot(range(train_end,test_end),actual_data,label='Actual')

### CHECK REVERSE STATIONARITY by applying reverse_stat on df_stat to get original data
input_val=reverse_stat(df_stat[response].iloc[train_end:test_end],scale_data_df[response].iloc[train_end],min
plt.plot(range(train_end,test_end),input_val,label='Input')


plt.title('Test Data'+
          '\nRMSE='+str(np.sqrt(mean_squared_error(actual_data,val_pred_unscaled)))+
          '. R2='+str(r2_score(actual_data,val_pred_unscaled)))
plt.xlabel('Index')
plt.ylabel('Response')
# plt.ylim(min(train_pred_unscaled),max(train_pred_unscaled))
plt.legend()
plt.show()

T3_train_pred = train_pred_unscaled
T3_test_pred = val_pred_unscaled

train_copy['T3(C)'] = np.append(np.full((train_copy.shape[0]-train_predictions.shape[0],),np.NaN),train_predi
```





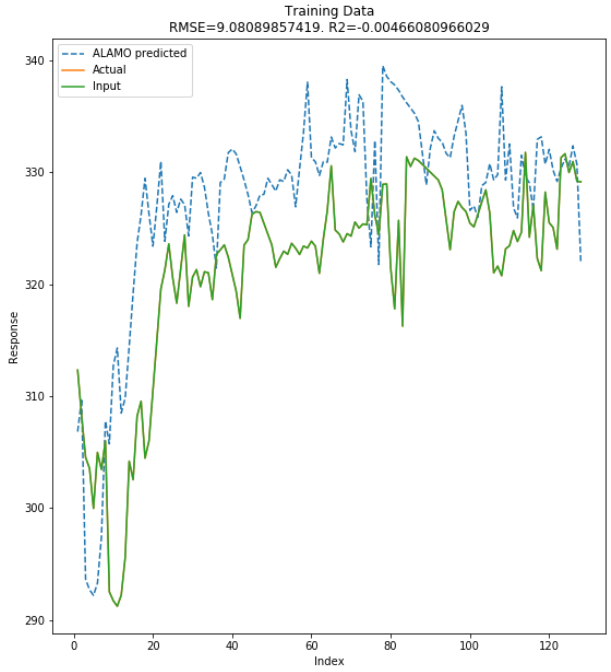
Simulate and forecast T4

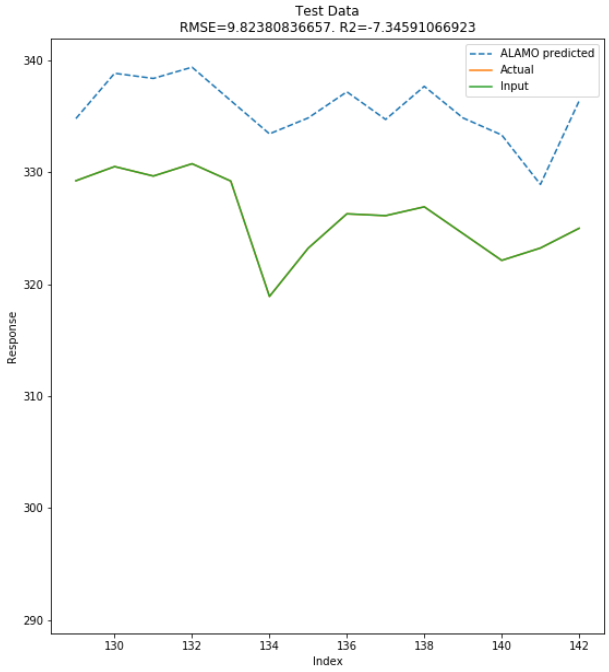
In [9]:

```
# T4

train_copy.dropna(inplace=True)

response = 'T4(C)'
exog_list = open('T4_exog_list.txt','r').read().splitlines()
orders = open('T4_exog_order.txt','r').read().splitlines()
orders = [int(i) for i in orders]
endog_order = orders[-1]
exog_order = orders[:-1]
max_ex = max(max(exog_order),endog_order)
req_train_data,req_test_data,train_array,test_array = get_data(train_copy,blind_test,'T4(C)',exog_list,exog_o

input_train_array = np.append(train_array,req_train_data.reshape(-1,1),axis=1)
input_test_array = np.append(test_array,req_test_data.reshape(-1,1),axis=1)

terms,coeff = parse_terms(eq_T4)
xlabels = open('T4_xlabels.txt','r').readlines()

train_predictions = np.array(calc_from_string(input_train_array,terms,coeff,xlabels[0]))
test_predictions = np.array(calc_from_string(input_test_array,terms,coeff,xlabels[0]))

clean_data_df = df.interpolate()
scale_data_df = df_scaled

resp_col_num = train.columns.get_loc(response)
train_end = train_copy.shape[0]
test_end=train_copy.shape[0]+blind_test.shape[0]

train_pred_unscaled = reverse_stat(train_predictions,scale_data_df[response].iloc[max_ex],min(clean_data_df[r
val_pred_unscaled = reverse_stat(test_predictions,scale_data_df[response].iloc[train_end],min(clean_data_df[r

plt.figure(figsize=(20,10))
ax1=plt.subplot(121)

plt.plot(range(max_ex,train_end),train_pred_unscaled,'--',label='ALAMO predicted')
actual_data=clean_data_df[response].iloc[max_ex+1:train_end+1]


print (train_predictions.shape)


plt.plot(range(max_ex,train_end),actual_data,label='Actual')

### CHECK REVERSE STATIONARITY by applying reverse_stat on df_stat to get original data
input_val=reverse_stat(df_stat[response].iloc[max_ex:train_end],scale_data_df[response].iloc[max_ex],min(clea
plt.plot(range(max_ex,train_end),input_val,label='Input')


plt.title('Training Data'+
          '\nRMSE='+str(np.sqrt(mean_squared_error(actual_data,train_pred_unscaled)))+
          '. R2='+str(r2_score(actual_data,train_pred_unscaled)))
plt.xlabel('Index')
plt.ylabel('Response')
plt.legend()

ax2=plt.subplot(122,sharey=ax1)
plt.plot(range(train_end,test_end),val_pred_unscaled,'--',label='ALAMO predicted')
actual_data=clean_data_df[response].iloc[train_end+1:test_end+1]
plt.plot(range(train_end,test_end),actual_data,label='Actual')

### CHECK REVERSE STATIONARITY by applying reverse_stat on df_stat to get original data
input_val=reverse_stat(df_stat[response].iloc[train_end:test_end],scale_data_df[response].iloc[train_end],min
plt.plot(range(train_end,test_end),input_val,label='Input')


plt.title('Test Data'+
          '\nRMSE='+str(np.sqrt(mean_squared_error(actual_data,val_pred_unscaled)))+
          '. R2='+str(r2_score(actual_data,val_pred_unscaled)))
plt.xlabel('Index')
plt.ylabel('Response')
# plt.ylim(min(train_pred_unscaled),max(train_pred_unscaled))
plt.legend()
plt.show()

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,)
```





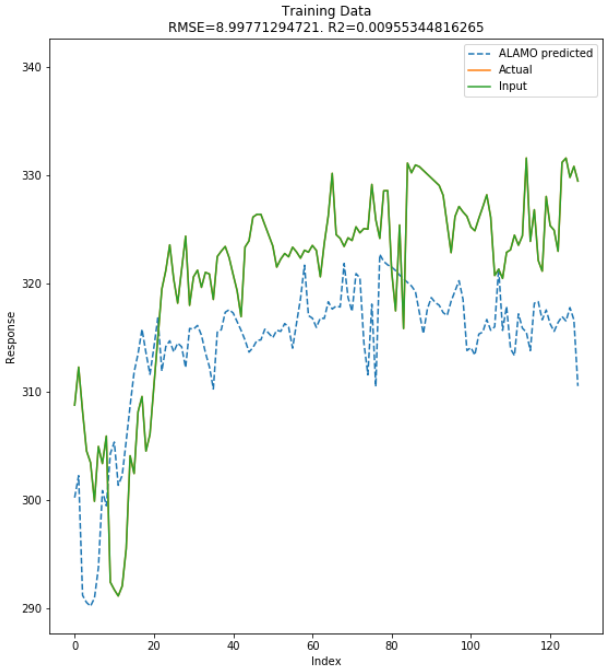
Simulate and forecast T5

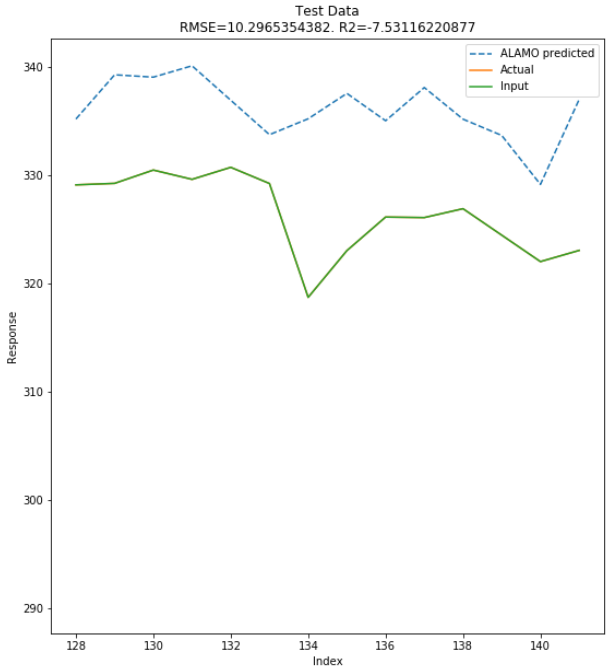
In [10]:

```
# T5

train_copy.dropna(inplace=True)

response = 'T5(C)'
exog_list = open('T5_exog_list.txt','r').read().splitlines()
orders = open('T5_exog_order.txt','r').read().splitlines()
orders = [int(i) for i in orders]
endog_order = orders[-1]
exog_order = orders[:-1]
max_ex = max(max(exog_order),endog_order)
req_train_data,req_test_data,train_array,test_array = get_data(train_copy,blind_test,'T5(C)',exog_list,exog_o

input_train_array = np.append(train_array,req_train_data.reshape(-1,1),axis=1)
input_test_array = np.append(test_array,req_test_data.reshape(-1,1),axis=1)

terms,coeff = parse_terms(eq_T5)
xlabels = open('T5_xlabels.txt','r').readlines()

train_predictions = np.array(calc_from_string(input_train_array,terms,coeff,xlabels[0]))
test_predictions = np.array(calc_from_string(input_test_array,terms,coeff,xlabels[0]))

clean_data_df = df.interpolate()
scale_data_df = df_scaled

resp_col_num = train.columns.get_loc(response)
train_end = train_copy.shape[0]
test_end=train_copy.shape[0]+blind_test.shape[0]

train_pred_unscaled = reverse_stat(train_predictions,scale_data_df[response].iloc[max_ex],min(clean_data_df[r
val_pred_unscaled = reverse_stat(test_predictions,scale_data_df[response].iloc[train_end],min(clean_data_df[r

plt.figure(figsize=(20,10))
ax1=plt.subplot(121)

plt.plot(range(max_ex,train_end),train_pred_unscaled,'--',label='ALAMO predicted')
actual_data=clean_data_df[response].iloc[max_ex+1:train_end+1]


print (train_predictions.shape)


plt.plot(range(max_ex,train_end),actual_data,label='Actual')

### CHECK REVERSE STATIONARITY by applying reverse_stat on df_stat to get original data
input_val=reverse_stat(df_stat[response].iloc[max_ex:train_end],scale_data_df[response].iloc[max_ex],min(clea
plt.plot(range(max_ex,train_end),input_val,label='Input')


plt.title('Training Data'+
          '\nRMSE='+str(np.sqrt(mean_squared_error(actual_data,train_pred_unscaled)))+
          '. R2='+str(r2_score(actual_data,train_pred_unscaled)))
plt.xlabel('Index')
plt.ylabel('Response')
plt.legend()

ax2=plt.subplot(122,sharey=ax1)
plt.plot(range(train_end,test_end),val_pred_unscaled,'--',label='ALAMO predicted')
actual_data=clean_data_df[response].iloc[train_end+1:test_end+1]
plt.plot(range(train_end,test_end),actual_data,label='Actual')

### CHECK REVERSE STATIONARITY by applying reverse_stat on df_stat to get original data
input_val=reverse_stat(df_stat[response].iloc[train_end:test_end],scale_data_df[response].iloc[train_end],min
plt.plot(range(train_end,test_end),input_val,label='Input')


plt.title('Test Data'+
          '\nRMSE='+str(np.sqrt(mean_squared_error(actual_data,val_pred_unscaled)))+
          '. R2='+str(r2_score(actual_data,val_pred_unscaled)))
plt.xlabel('Index')
plt.ylabel('Response')
# plt.ylim(min(train_pred_unscaled),max(train_pred_unscaled))
plt.legend()
plt.show()

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,)
```





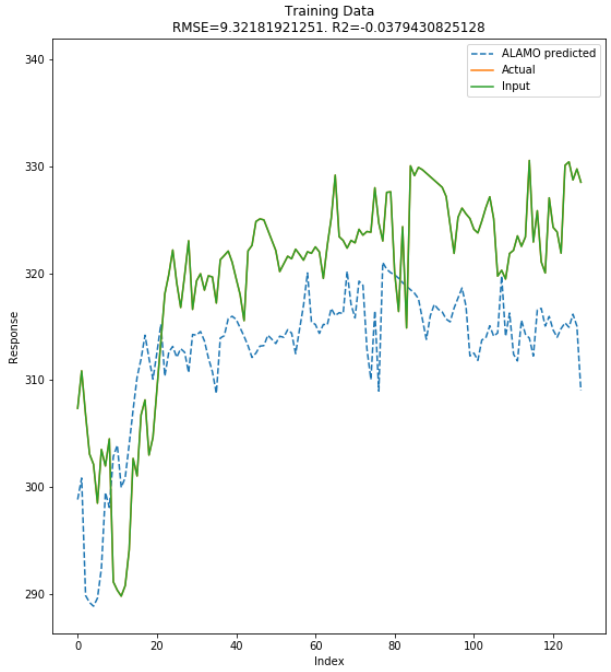
Simulate and forecast T6

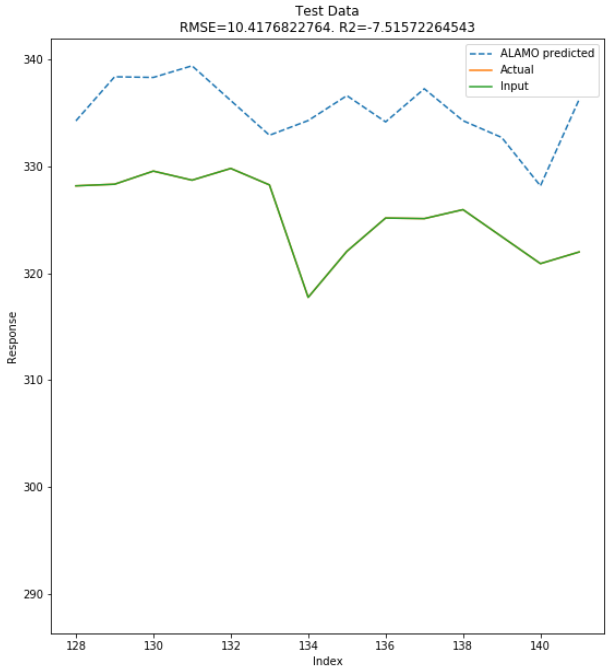
In [11]:

```
# T6

train_copy.dropna(inplace=True)

response = 'T6(C)'
exog_list = open('T6_exog_list.txt','r').read().splitlines()
orders = open('T6_exog_order.txt','r').read().splitlines()
orders = [int(i) for i in orders]
endog_order = orders[-1]
exog_order = orders[:-1]
max_ex = max(max(exog_order),endog_order)
req_train_data,req_test_data,train_array,test_array = get_data(train_copy,blind_test,'T6(C)',exog_list,exog_o

input_train_array = np.append(train_array,req_train_data.reshape(-1,1),axis=1)
input_test_array = np.append(test_array,req_test_data.reshape(-1,1),axis=1)

terms,coeff = parse_terms(eq_T6)
xlabels = open('T6_xlabels.txt','r').readlines()

train_predictions = np.array(calc_from_string(input_train_array,terms,coeff,xlabels[0]))
test_predictions = np.array(calc_from_string(input_test_array,terms,coeff,xlabels[0]))

clean_data_df = df.interpolate()
scale_data_df = df_scaled

resp_col_num = train.columns.get_loc(response)
train_end = train_copy.shape[0]
test_end=train_copy.shape[0]+blind_test.shape[0]

train_pred_unscaled = reverse_stat(train_predictions,scale_data_df[response].iloc[max_ex],min(clean_data_df[r
val_pred_unscaled = reverse_stat(test_predictions,scale_data_df[response].iloc[train_end],min(clean_data_df[r

plt.figure(figsize=(20,10))
ax1=plt.subplot(121)

plt.plot(range(max_ex,train_end),train_pred_unscaled,'--',label='ALAMO predicted')
actual_data=clean_data_df[response].iloc[max_ex+1:train_end+1]


print (train_predictions.shape)


plt.plot(range(max_ex,train_end),actual_data,label='Actual')

### CHECK REVERSE STATIONARITY by applying reverse_stat on df_stat to get original data
input_val=reverse_stat(df_stat[response].iloc[max_ex:train_end],scale_data_df[response].iloc[max_ex],min(clea
plt.plot(range(max_ex,train_end),input_val,label='Input')


plt.title('Training Data'+
          '\nRMSE='+str(np.sqrt(mean_squared_error(actual_data,train_pred_unscaled)))+
          '. R2='+str(r2_score(actual_data,train_pred_unscaled)))
plt.xlabel('Index')
plt.ylabel('Response')
plt.legend()

ax2=plt.subplot(122,sharey=ax1)
plt.plot(range(train_end,test_end),val_pred_unscaled,'--',label='ALAMO predicted')
actual_data=clean_data_df[response].iloc[train_end+1:test_end+1]
plt.plot(range(train_end,test_end),actual_data,label='Actual')

### CHECK REVERSE STATIONARITY by applying reverse_stat on df_stat to get original data
input_val=reverse_stat(df_stat[response].iloc[train_end:test_end],scale_data_df[response].iloc[train_end],min
plt.plot(range(train_end,test_end),input_val,label='Input')


plt.title('Test Data'+
          '\nRMSE='+str(np.sqrt(mean_squared_error(actual_data,val_pred_unscaled)))+
          '. R2='+str(r2_score(actual_data,val_pred_unscaled)))
plt.xlabel('Index')
plt.ylabel('Response')
# plt.ylim(min(train_pred_unscaled),max(train_pred_unscaled))
plt.legend()
plt.show()

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,axi
4 train_pred_unscaled_T2 = np.append(np.full((train.shape[0]-T2_train_pred.shape[0],),np.NaN),T2_train_pred,axi
5 train_pred_unscaled_T3 = np.append(np.full((train.shape[0]-T3_train_pred.shape[0],),np.NaN),T3_train_pred,axi
6 train_pred_unscaled_T4 = np.append(np.full((train.shape[0]-T4_train_pred.shape[0],),np.NaN),T4_train_pred,axi
7 train_pred_unscaled_T5 = np.append(np.full((train.shape[0]-T5_train_pred.shape[0],),np.NaN),T5_train_pred,axi
8 train_pred_unscaled_T6 = np.append(np.full((train.shape[0]-T6_train_pred.shape[0],),np.NaN),T6_train_pred,axi
```

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
```

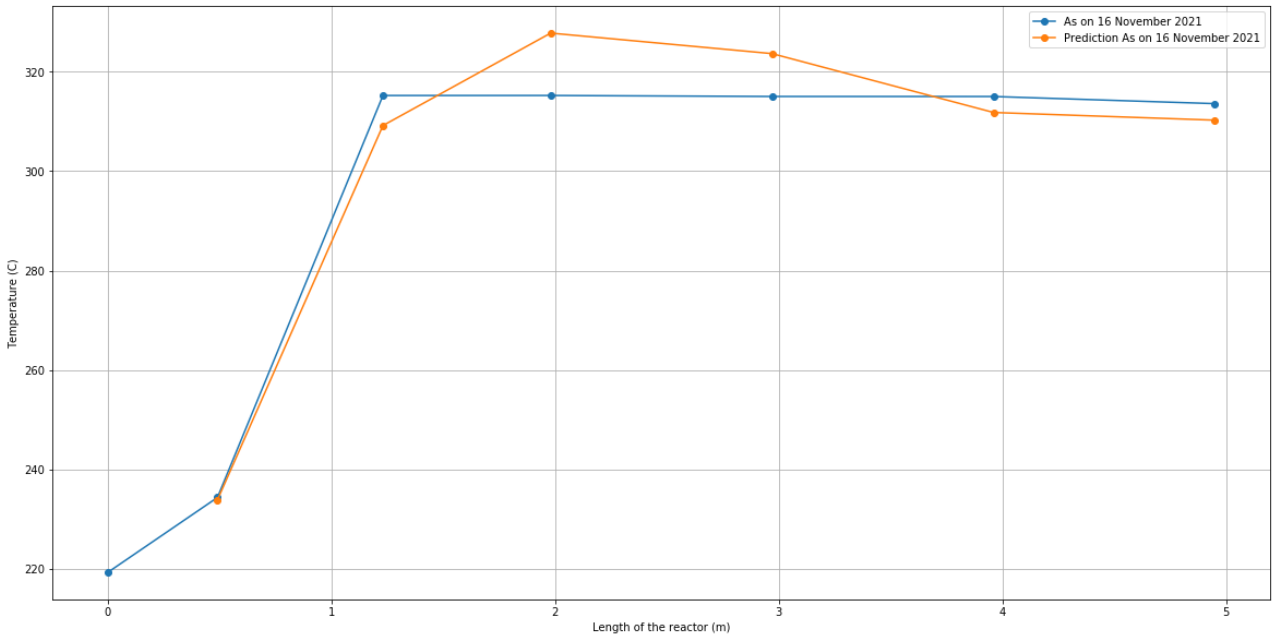
Visualize all predictions

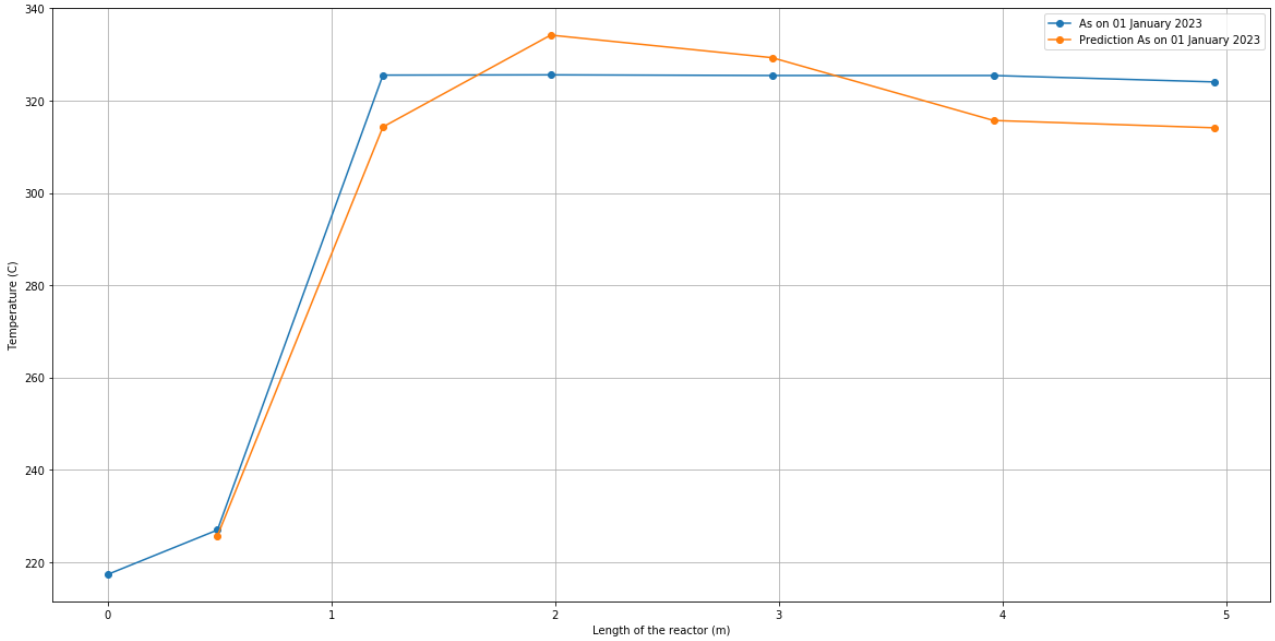
In [15]:

```
#Define lengths along the reactor
length = [0,0.49,1.23,1.98,2.97,3.96,4.95]
length_pred = [0.49,1.23,1.98,2.97,3.96,4.95]

# Generate an interpolated copy of the original dataframe
df2=df.copy().interpolate()
df2.dropna(inplace=True)

predict_df = predict.copy()
predict_df['Day']=df2['Day']
predict_df.dropna(inplace=True)
```

In [16]:

```
#Set figure size
fig_size = plt.rcParams["figure.figsize"]
fig_size[0] = 20
fig_size[1] = 10

#Randomize seet for constant set of random numbers
np.random.seed(100)

#Randomly pick 4 time points from the data and store in a list
random_index_list = [22]

#Plot the randomly selected points against reactor length
for h in sorted(random_index_list):
    plt.plot(length,[df2.loc[h,'T0(C)'],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)'],df2.loc[h,'T6(C)']],'-o',
             label='As on '+str(df2.iloc[h,0].strftime('%d %B %Y')))
    plt.plot(length_pred,[predict_df.loc[h,'T1(C)'],predict_df.loc[h,'T2(C)'],predict_df.loc[h,'T3(C)'],
                          predict_df.loc[h,'T4(C)'],predict_df.loc[h,'T5(C)'],predict_df.loc[h,'T6(C)']],'-o'
             label='Prediction As on '+str(df2.iloc[h,0].strftime('%d %B %Y')))

plt.grid()

plt.xlabel('Length of the reactor (m)')
plt.ylabel('Temperature (C)')
# plt.savefig('tempvslength.png')

plt.legend()
plt.show()


random_index_list = [49]

#Plot the randomly selected points against reactor length
for h in sorted(random_index_list):
    plt.plot(length,[df2.loc[h,'T0(C)'],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)'],df2.loc[h,'T6(C)']],'-o',
             label='As on '+str(df2.iloc[h,0].strftime('%d %B %Y')))
    plt.plot(length_pred,[predict_df.loc[h,'T1(C)'],predict_df.loc[h,'T2(C)'],predict_df.loc[h,'T3(C)'],

predict_df.loc[h,'T4(C)'],predict_df.loc[h,'T5(C)'],predict_df.loc[h,'T6(C)']],'-o',
             label='Prediction As on '+str(df2.iloc[h,0].strftime('%d %B %Y')))

plt.grid()

plt.xlabel('Length of the reactor (m)')
plt.ylabel('Temperature (C)')
# plt.savefig('tempvslength.png')

plt.legend()
plt.show()
```



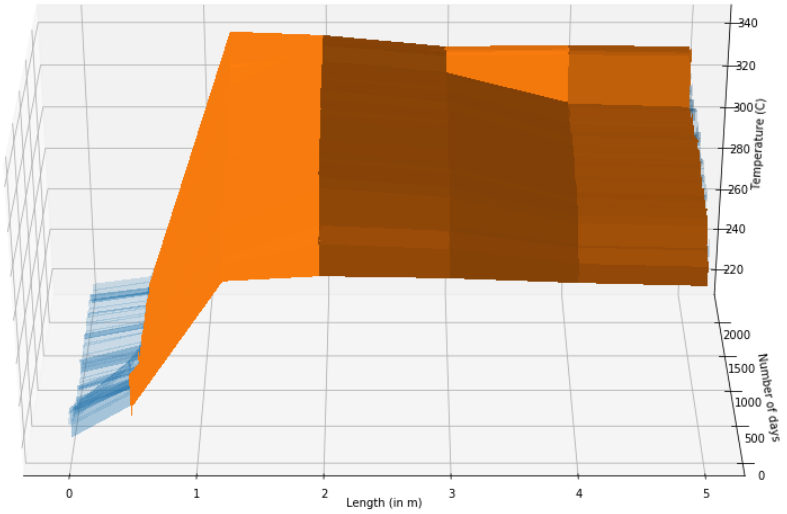


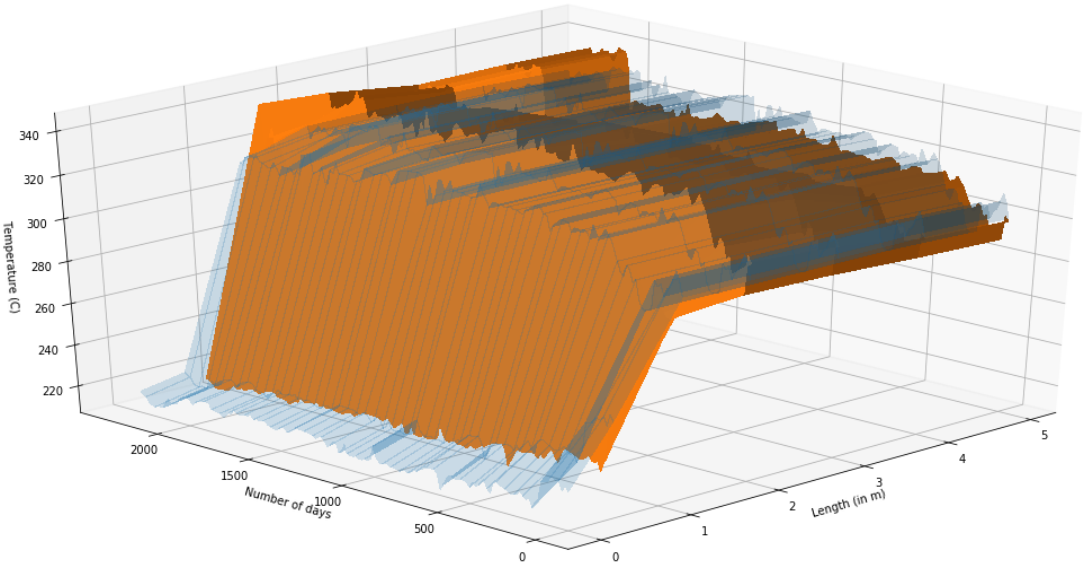
In [17]:

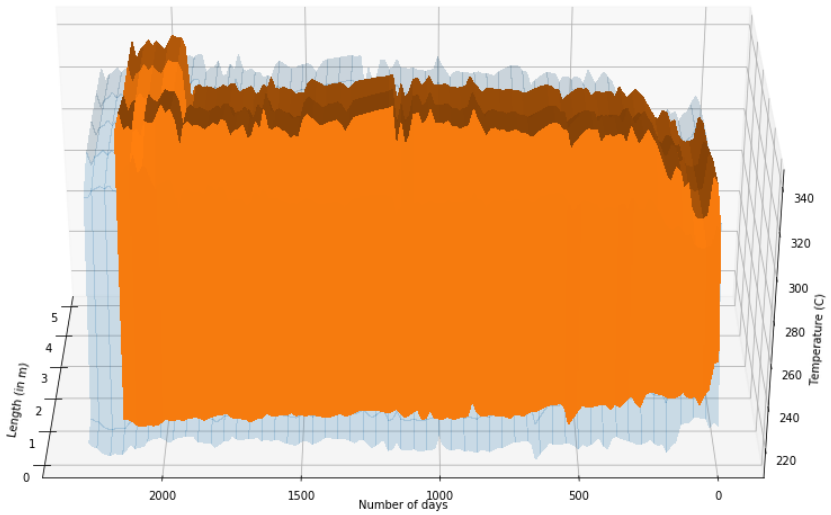
```
X,Y = np.meshgrid(length,(df2['Day']-df2['Day'].iloc[0]).dt.days)
Z = df2.iloc[:,12:19].values

X2,Y2 = np.meshgrid(length_pred,(predict_df['Day']-predict_df['Day'].iloc[0]).dt.days)
Z2=predict_df.iloc[:,0:6].values
```

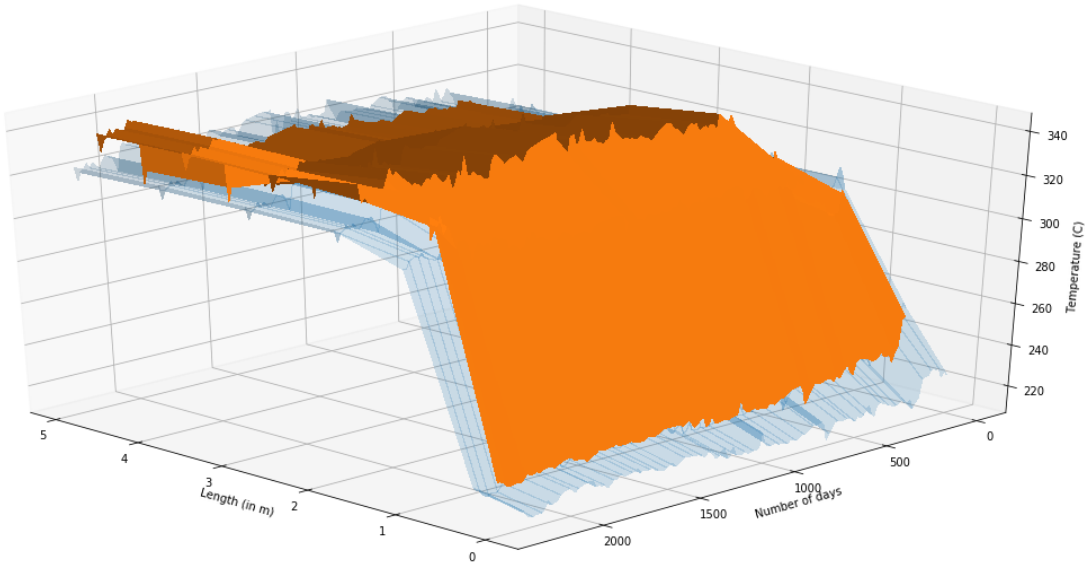
In [18]:

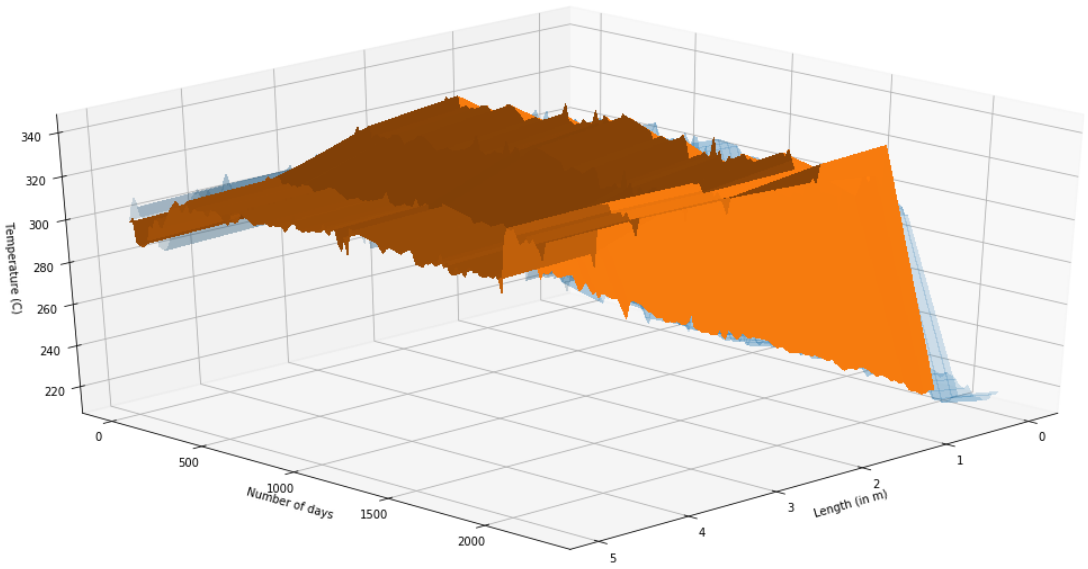
```
# Plot 3D plots

fig_size = plt.rcParams["figure.figsize"]
fig_size[0] = 20
fig_size[1] = 10

for angle in range(270,0,-45):
    ax = plt.figure()
    fig = ax.gca(projection='3d')
    fig.set_zlabel('Temperature (C)')
    fig.set_xlabel('Length (in m)')
    fig.set_ylabel('Number of days')
    fig.plot_surface(X,Y,Z,linewidth=0,antialiased=False,alpha=0.2)
    fig.plot_surface(X2,Y2,Z2,linewidth=0,antialiased=False)

    fig.view_init(30,angle)
    plt.show()
```






Length (in m)
0
1
2
3
4
5
2000
1500
1000
500
0
Number of days
340
320
300
280
260
240
220
Temperature (C)


5
4
3
Length (in m)
2
1
0
2000
1500
1000
500
0
Number of days
340
320
300
280
260
240
220
Temperature (C)

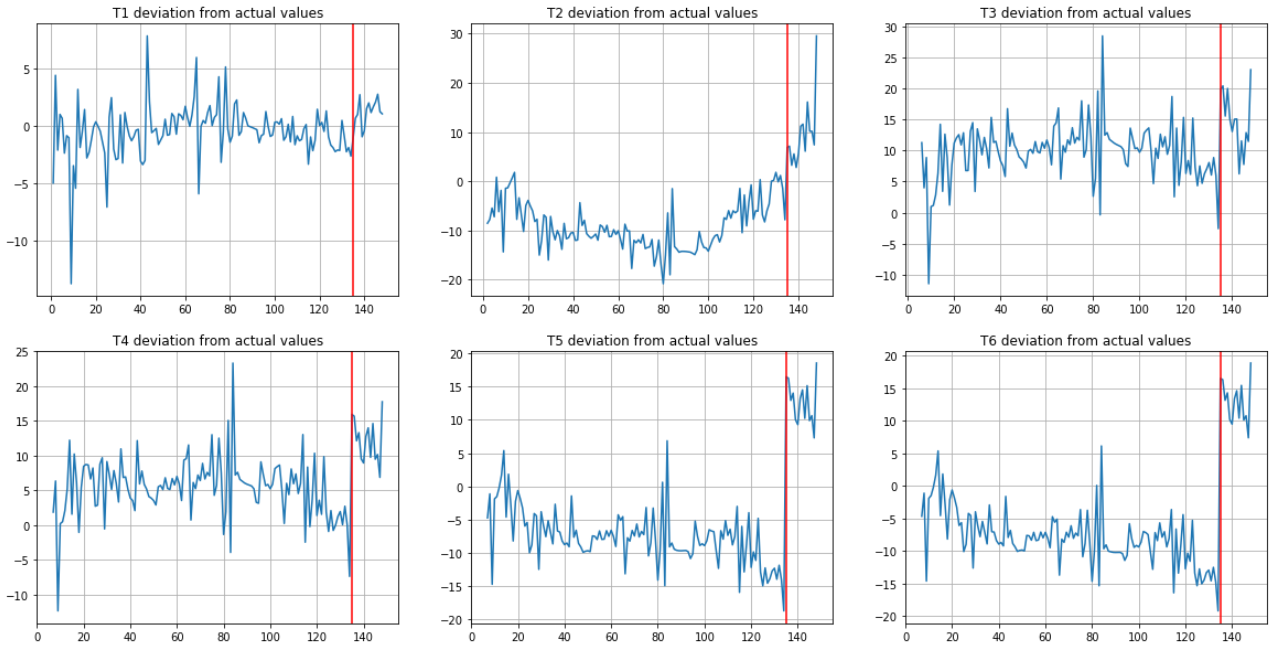


In [19]:

```
# Calculate deviations from original values

T1_deviation = predict['T1(C)'].interpolate() - df['T1(C)'].interpolate()
T2_deviation = predict['T2(C)'].interpolate() - df['T2(C)'].interpolate()
T3_deviation = predict['T3(C)'].interpolate() - df['T3(C)'].interpolate()
T4_deviation = predict['T4(C)'].interpolate() - df['T4(C)'].interpolate()
T5_deviation = predict['T5(C)'].interpolate() - df['T5(C)'].interpolate()
T6_deviation = predict['T6(C)'].interpolate() - df['T6(C)'].interpolate()
```

In [20]:

```
# Plot deviations

plt.figure(figsize=(20,10))

plt.subplot(231)
plt.plot(T1_deviation)
plt.axvline(x=len(train),color='r')
plt.title('T1 deviation from actual values')
plt.grid()

plt.subplot(232)
plt.plot(T2_deviation)
plt.axvline(x=len(train),color='r')
plt.title('T2 deviation from actual values')
plt.grid()

plt.subplot(233)
plt.plot(T3_deviation)
plt.axvline(x=len(train),color='r')
plt.title('T3 deviation from actual values')
plt.grid()

plt.subplot(234)
plt.plot(T4_deviation)
plt.axvline(x=len(train),color='r')
plt.title('T4 deviation from actual values')
plt.grid()

plt.subplot(235)
plt.plot(T5_deviation)
plt.axvline(x=len(train),color='r')
plt.title('T5 deviation from actual values')
plt.grid()

plt.subplot(236)
plt.plot(T6_deviation)
plt.axvline(x=len(train),color='r')
plt.title('T6 deviation from actual values')
plt.grid()

plt.show()
```



Variation of RMSE with time

In [21]:

```
# Define an empty list to track RMSE with time

RMSE_time = []

for h in range(predict_df.index[0],149):
    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)'],
    pred = [predict_df.loc[h,'T1(C)'],predict_df.loc[h,'T2(C)'],predict_df.loc[h,'T3(C)'],predict_df.loc[h,'T
    RMSE_time.append(np.sqrt(mean_squared_error(actual,pred)))
```

In [22]:

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