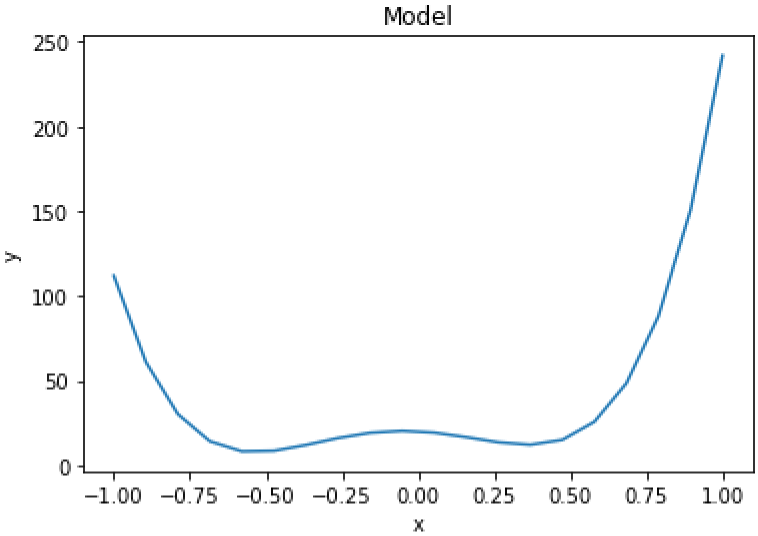
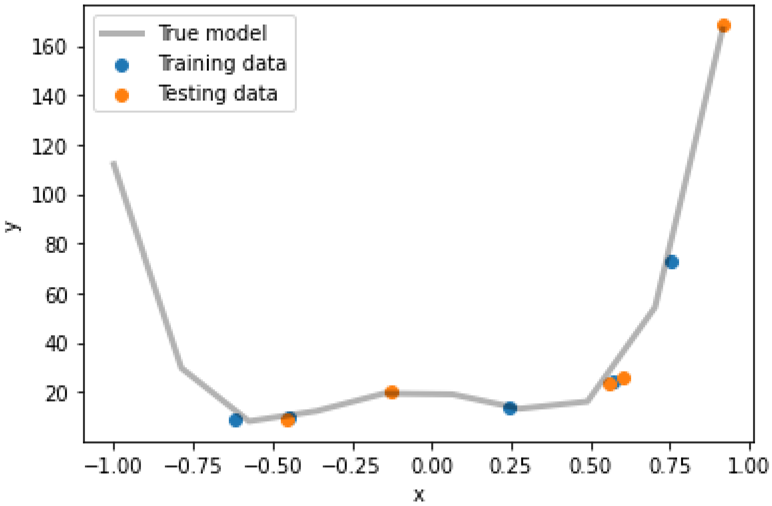
In [10]:
```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import norm, laplace
import math
import pandas as pd
```

In [2]:
```
def arbitrary_poly(params):
    poly_model = lambda x: sum([p*(x**i) for i, p in enumerate(params)])
    return poly_model

# params: [theta_0, theta_1, ... , theta_n], where n = model order and theta_
true_params = [20,-10,-93,75,250]
y_model = arbitrary_poly(true_params)

# Plot true model
x = np.linspace(start=-1, stop=1, num=20)
plt.figure()
plt.plot(x, y_model(x))
plt.xlabel("x")
plt.ylabel("y")
plt.title("Model");
```



In [5]:
```
# Hyperparameters for the type of noise-generating distribution.
loc = 0           # location (mean) parameter
scale = 1         # scaling (std dev) parameter
magnitude = 1.2   # noise magnitude
N = 10          # number of samples

np.random.seed(1234)  # Non-random generation between code executions. Commen

# Generate data points
range_low = -1
range_high = 1
u = np.sort(np.random.uniform(range_low,range_high,N))
y_true = y_model(u)

# Generate noisez

pdf = laplace.pdf

normVariance = 1 # Input as the scale parameter in the normal distribution
```

```
laplaceVariance = 1

alfa = 0

gamma = 0.1

noiseNorm = magnitude * np.random.normal(loc, normVariance, int(alfa * N))

noiseLaplace = magnitude * np.random.laplace(loc, laplaceVariance, int((1-alf
y = y_true + noiseLaplace

train_u = u[::2]
test_u = u[1::2]
train_y = y[::2]
test_y = y[1::2]

plt.scatter(train_u, train_y, label=r"Training data")

plt.scatter(test_u, test_y, label=r"Testing data")

u0 = np.linspace(-1, max(u), N)
plt.plot(u0, y_model(u0), "k", alpha=0.3, lw=3, label="True model")
plt.legend()
plt.xlabel("x")
plt.ylabel("y");
```

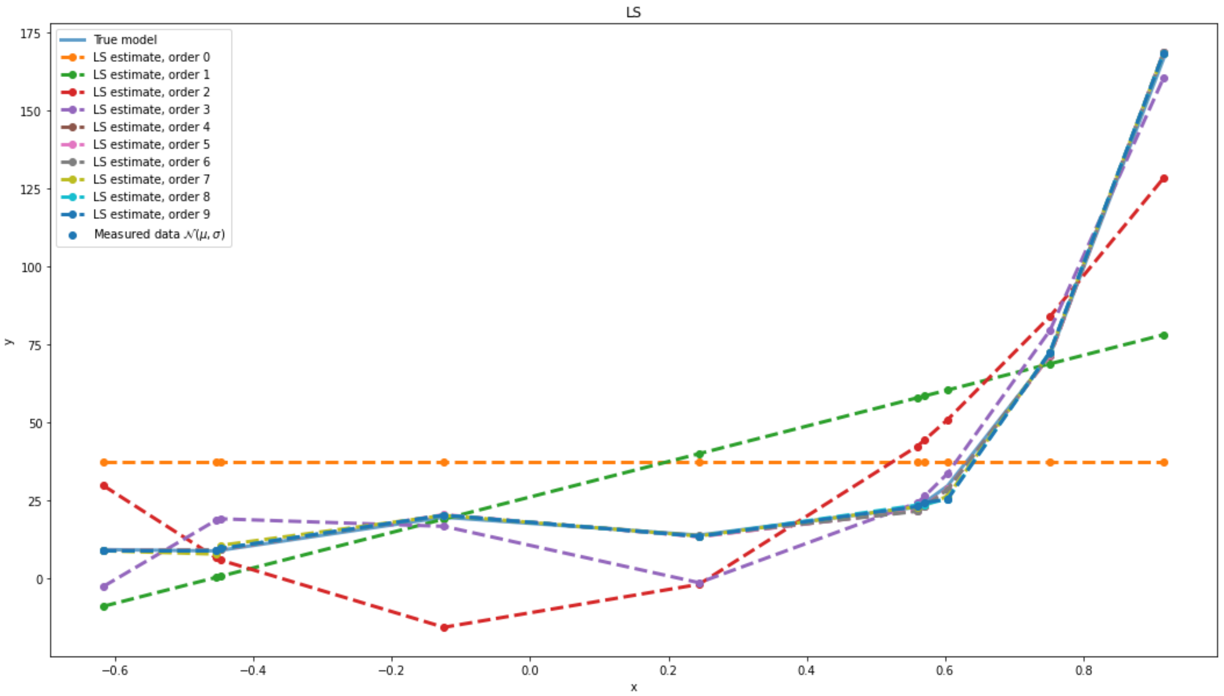


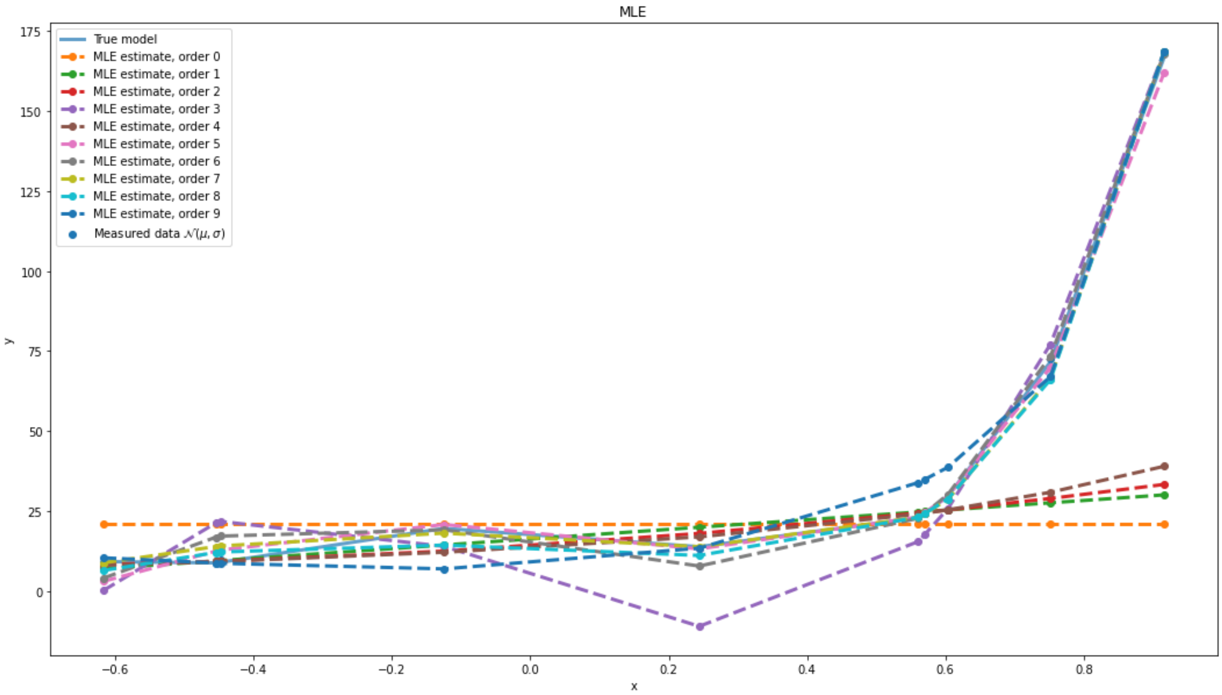
In [83]:

```
# Matrix form
def LSorderFunc(order, u, y, N):
    u_tensor_0 = np.reshape(u,(N,1))
    #print(u_tensor_0)
    ones_vec = np.ones((N,1))

    u_tensor = ones_vec
    #print(ones_vec)

    #print(u_tensor)

    for i in range(1,order):
        u_tensor = np.append(u_tensor, np.power(u_tensor_0, i) ,axis=1)
    #print(u_tensor1)

    #-------------
    u_transpose_dot_u = np.dot(u_tensor.T,u_tensor)  # calculating dot produc
    u_transpose_dot_u_inv = np.linalg.inv(u_transpose_dot_u) #calculating inv

    u_transpose_dot_y = np.dot(u_tensor.T,y)  # calculating dot product
```

```
    LS_params = np.dot(u_transpose_dot_u_inv,u_transpose_dot_y)
    LS_params_rounded = ["{:.2f}".format(round(i, 2)) for i in LS_params.toli
    #print(f"LS parameters:         {LS_params_rounded}")
   # print(f"True model parameters: {true_params}")

    #diffParams = []
    #for i in range(0, order):
    #    diffParams.append(float(true_params[i] - float(LS_params_rounded[i])

    #print("The differnence between the estimated theata and the real Theta i
    #print(diffParams)

    # Recreate model based on LS estimate:
    LS_params = LS_params.tolist()
    LS_estimate = arbitrary_poly(LS_params)
    return LS_params, LS_estimate

def log_lik(par_vec, y, x):
      # Use the distribution class chosen earlier
    # If the standard deviation parameter is negative, return a large value:
    if par_vec[-1] < 0:
        return(1e8)
    # The likelihood function values:
    lik = pdf(y,
              loc = sum([p*(x**i) for i, p in enumerate(par_vec[:-1])]),
              scale = par_vec[-1])

    #This is similar to calculating the likelihood for Y - XB
    # res = y - par_vec[0] - par_vec[1] * x
    # lik = norm.pdf(res, loc = 0, sd = par_vec[2])

    # If all logarithms are zero, return a large value
    if all(v == 0 for v in lik):
        return(1e8)
    # Logarithm of zero = -Inf
    return(-sum(np.log(lik[np.nonzero(lik)])))

def MLEfunction(order,u, y, N):

    import scipy.optimize as optimize

    # The likelihood function includes the scale (std dev) parameter which is
    # therefore the initial guess verctor has length n+2 [theta_0_hat, theta_
    init_guess = np.zeros(order+1)
    init_guess[-1] = N

    # Do Maximum Likelihood Estimation:
    opt_res = optimize.minimize(fun = log_lik,
                                x0 = init_guess,
                                options={'disp': False},
                                args = (y, u))

    MLE_params = opt_res.x[:-1]
    MLE_estimate = arbitrary_poly(MLE_params)

    MLE_params_rounded = ["{:.2f}".format(round(i, 2)) for i in MLE_params.to
    #print(f"\nMLE parameters of order :      {MLE_params_rounded}")

    return MLE_params, MLE_estimate



def plotSeveral(orderArray):
    plt.figure(1, figsize=(18,10))
```

```
    plt.scatter(u, y,  label=r"Measured data $\mathcal{N}(\mu, \sigma)$")
    u0 = np.linspace(min(u), max(u), N)
    plt.plot(u, y_model(u), alpha=0.7, lw=3, label="True model")

    for i in orderArray:
        LS_params, LS_estimate = LSorderFunc(i, u, y, N)
        plt.plot(u, LS_estimate(u),"o--",  lw=3, label=r"LS estimate, order "
    plt.title("LS")
    plt.legend()
    plt.xlabel("x")
    plt.ylabel("y");

    plt.figure(2, figsize=(18,10))
    plt.scatter(u, y,  label=r"Measured data $\mathcal{N}(\mu, \sigma)$")
    u0 = np.linspace(min(u), max(u), N)
    plt.plot(u, y_model(u), alpha=0.7, lw=3, label="True model")

    for i in orderArray:
        MLE_params, MLE_estimate = MLEfunction(i, u, y, N)
        plt.plot(u, MLE_estimate(u),"o--",  lw=3, label=r"MLE estimate, order

    plt.title("MLE")
    plt.legend()
    plt.xlabel("x")
    plt.ylabel("y");


plotSeveral([1,2,3,4,5,6,7,8,9,10])
```





In [11]:

```
def RMSE(y_t, y_hat):
    return np.sqrt(np.mean((y_t-y_hat) ** 2))

def RSS(y_t, y_hat):
    return np.sum((y_t-y_hat) ** 2)

def FUV(y_t, y_hat):
    resid_sum_of_squares = RSS(y_t, y_hat)
    est_mean = np.sum(y_t) / len(y_t)
    variance = np.sum((y_t - est_mean) ** 2)
    perf_index = resid_sum_of_squares / variance
    return perf_index

def R2(y_t, y_hat):
    return 1 - FUV(y_t, y_hat)

def FIT(y_t, y_hat):
    sqrt_FUV = np.sqrt(FUV(y_t, y_hat))
    perf_index = 100 * (1 - sqrt_FUV)
    return perf_index

def MAD(y_t, y_hat):
    return np.mean(np.abs(y_t - y_hat))

def select_model(performance):
    return performance.index(min(performance))
```

In [23]:

```
def get_models(orderArray):
    LS_param_array=[]
    LS_estimate_array=[]
    MLE_param_array=[]
    MLE_estimate_array = []

    for i in orderArray:
        LS_params, LS_estimate = LSorderFunc(i, u, y, N)
        LS_param_array.append(LS_params)
        LS_estimate_array.append(LS_estimate)

        MLE_params, MLE_estimate = MLEfunction(i, u, y, N)
        MLE_param_array.append(MLE_params)
```

```
            MLE_estimate_array.append(MLE_estimate)

        return LS_param_array, LS_estimate_array, MLE_param_array, MLE_estimate_a

orderArray = [1,2,3,4,5,6,7,8,9,10]
LS_param_array, LS_estimate_array, MLE_param_array, MLE_estimate_array = get_
```

Out[23]: <function __main__.arbitrary_poly.<locals>.<lambda>(x)>

In [52]:

```
LS_train = {"Order" : orderArray,
                            "RMSE" : np.zeros(10),
                            "RSS" : np.zeros(10),
                            "FUV" : np.zeros(10),
                            "R2" : np.zeros(10),
                            "FIT" : np.zeros(10),
                            "MAD" : np.zeros(10)



}
LS_test = {"Order" : orderArray,
                            "RMSE" : np.zeros(10),
                            "RSS" : np.zeros(10),
                            "FUV" : np.zeros(10),
                            "R2" : np.zeros(10),
                            "FIT" : np.zeros(10),
                            "MAD" : np.zeros(10)



}

MLE_train = {"Order" : orderArray,
                            "RMSE" : np.zeros(10),
                            "RSS" : np.zeros(10),
                            "FUV" : np.zeros(10),
                            "R2" : np.zeros(10),
                            "FIT" : np.zeros(10),
                            "MAD" : np.zeros(10)



}
MLE_test = {"Order" : orderArray,
                            "RMSE" : np.zeros(10),
                            "RSS" : np.zeros(10),
                            "FUV" : np.zeros(10),
                            "R2" : np.zeros(10),
                            "FIT" : np.zeros(10),
                            "MAD" : np.zeros(10)



}

for i in range(10):
    train_y_hat = LS_estimate_array[i](train_u)
    LS_train["RMSE"][i] = RMSE(train_y, train_y_hat)
    LS_train["RSS"][i] = RSS(train_y, train_y_hat)
    LS_train["FUV"][i] = FUV(train_y, train_y_hat)
    LS_train["R2"][i] = R2(train_y, train_y_hat)
    LS_train["FIT"][i] = FIT(train_y, train_y_hat)
    LS_train["MAD"][i] = MAD(train_y, train_y_hat)

    test_y_hat = LS_estimate_array[i](test_u)
    LS_test["RMSE"][i] = RMSE(test_y, test_y_hat)
    LS_test["RSS"][i] = RSS(test_y, test_y_hat)
    LS_test["FUV"][i] = FUV(test_y, test_y_hat)
```

```
    LS_test["R2"][i] = R2(test_y, test_y_hat)
    LS_test["FIT"][i] = FIT(test_y, test_y_hat)
    LS_test["MAD"][i] = MAD(test_y, test_y_hat)

    train_y_hat = MLE_estimate_array[i](train_u)
    MLE_train["RMSE"][i] = RMSE(train_y, train_y_hat)
    MLE_train["RSS"][i] = RSS(train_y, train_y_hat)
    MLE_train["FUV"][i] = FUV(train_y, train_y_hat)
    MLE_train["R2"][i] = R2(train_y, train_y_hat)
    MLE_train["FIT"][i] = FIT(train_y, train_y_hat)
    MLE_train["MAD"][i] = MAD(train_y, train_y_hat)

    test_y_hat = MLE_estimate_array[i](test_u)
    MLE_test["RMSE"][i] = RMSE(test_y, test_y_hat)
    MLE_test["RSS"][i] = RSS(test_y, test_y_hat)
    MLE_test["FUV"][i] = FUV(test_y, test_y_hat)
    MLE_test["R2"][i] = R2(test_y, test_y_hat)
    MLE_test["FIT"][i] = FIT(test_y, test_y_hat)
    MLE_test["MAD"][i] = MAD(test_y, test_y_hat)



LS_train_df = pd.DataFrame(data=LS_train)
LS_test_df = pd.DataFrame(data=LS_test)
MLE_train_df = pd.DataFrame(data=MLE_train)
MLE_test_df = pd.DataFrame(data=MLE_test)
import tabulate as tab
print('LS training models:')
print(tab.tabulate(LS_train_df, headers='keys', tablefmt='psql', showindex=Fa
print('\nML training models:')
print(tab.tabulate(MLE_train_df, headers='keys', tablefmt='psql', showindex=F

print('\nLS testing models:')
print(tab.tabulate(LS_test_df, headers='keys', tablefmt='psql', showindex=Fal
print('\nML testing models:')
print(tab.tabulate(MLE_test_df, headers='keys', tablefmt='psql', showindex=Fa
```

LS training models:

Order	RMSE	RSS	FUV	R2	FIT	MAD
1	26.7121	3567.68	1.2371	-0.237097	-11.2249	25.7121
2	21.3089	2270.34	0.787245	0.212755	11.2732	18.1847
3	15.6047	1217.53	0.422181	0.577819	35.0246	14.2851
4	9.93219	493.242	0.171032	0.828968	58.6439	8.9468
5	0.72423	2.62255	0.000909371	0.999091	96.9844	0.518555
6	0.694553	2.41202	0.00083637	0.999164	97.108	0.530325
7	0.661486	2.18782	0.000758628	0.999241	97.2457	0.476956
8	0.599647	1.79789	0.000623419	0.999377	97.5032	0.421888
9	0.284662	0.405162	0.00014049	0.99986	98.8147	0.130333
10	0.000124639	7.76738e-08	2.69335e-11	1	99.9995	9.98725e-05

ML training models:

Order	RMSE	RSS	FUV	R2	FIT	MAD
1	24.4843	2997.41	1.03936	-0.0393569	-1.94885	17.1782
2	20.3147	2063.44	0.715502	0.284498	15.4127	10.7447
3	19.5887	1918.59	0.665275	0.334725	18.4356	9.81031
4	13.2924	883.434	0.306332	0.693668	44.6527	11.2591
5	18.69	1746.58	0.605627	0.394373	22.1779	9.22845
6	3.20527	51.3686	0.0178121	0.982188	86.6538	2.4445
7	4.75568	113.082	0.0392114	0.960789	80.1981	3.75813
8	3.38067	57.1448	0.019815	0.980185	85.9234	2.21292
9	3.47062	60.2261	0.0208835	0.979117	85.5489	2.78217
10	5.40567	146.107	0.0506626	0.949337	77.4916	3.73624

LS testing models:

Order	RMSE	RSS	FUV	R2	FIT	MAD
1	60.9704	18586.9	1.03819	-0.0381925	-1.89173	40.6385
2	46.1565	10652.1	0.594986	0.405014	22.8647	33.9616
3	27.9973	3919.25	0.218914	0.781086	53.2118	24.5694
4	6.9169	239.218	0.0133618	0.986638	88.4407	6.12967
5	1.58424	12.549	0.00070094	0.999299	97.3525	1.01612
6	1.59134	12.6618	0.000707236	0.999293	97.3406	1.03228
7	1.57264	12.366	0.000690718	0.999309	97.3718	1.01804
8	0.623458	1.9435	0.000108556	0.999891	98.9581	0.417658
9	0.225561	0.254389	1.42091e-05	0.999986	99.623	0.130333
10	0.000108848	5.92391e-08	3.30886e-12	1	99.9998	9.33618e-05

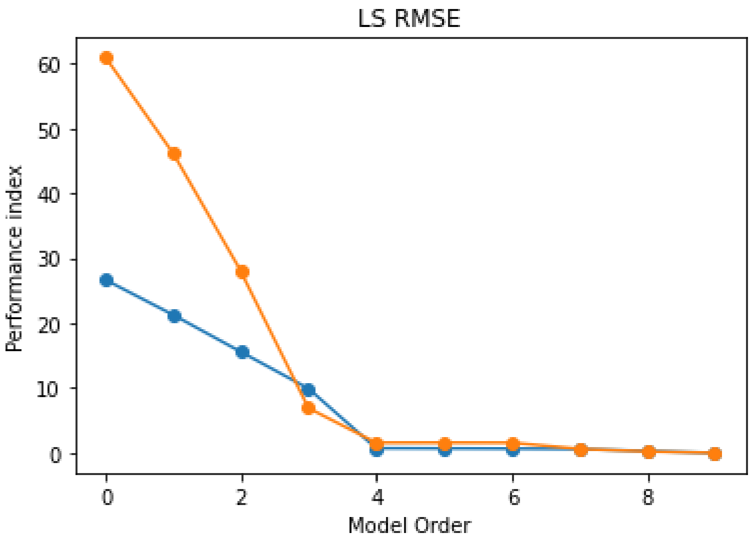
ML testing models:

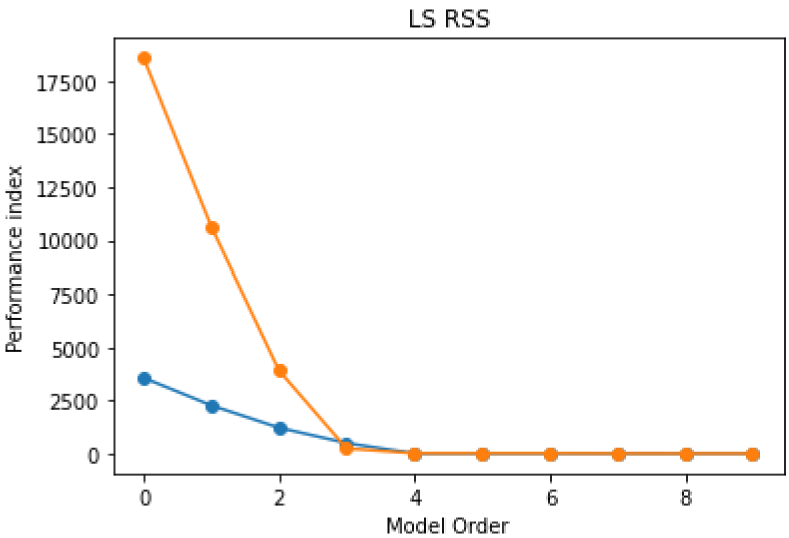
Order	RMSE	RSS	FUV	R2	FIT	MAD
1	66.1303	21866.1	1.22135	-0.221352	-10.5148	33.3141
2	61.885	19148.8	1.06957	-0.069575	-3.42026	29.2777
3	60.4707	18283.5	1.02124	-0.0212448	-1.05666	28.9104
4	7.09941	252.008	0.0140762	0.985924	88.1357	5.34827
5	57.9521	16792.2	0.937947	0.0620528	3.15233	27.7718
6	3.85328	74.2387	0.00414668	0.995853	93.5605	3.01185
7	4.22264	89.1533	0.00497975	0.99502	92.9433	2.9734
8	2.88439	41.5985	0.00232353	0.997676	95.1797	2.11295
9	3.37286	56.8808	0.00317714	0.996823	94.3634	2.51879
10	9.70074	470.522	0.0262815	0.973719	83.7884	7.48567

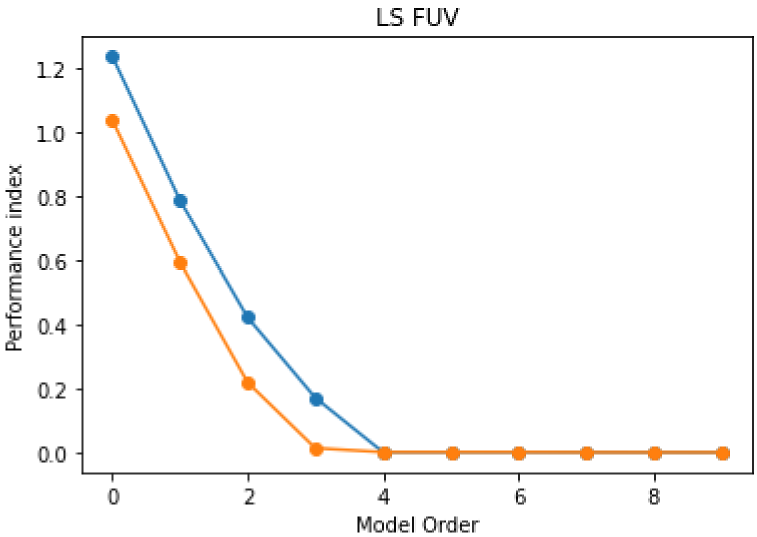
In [55]:

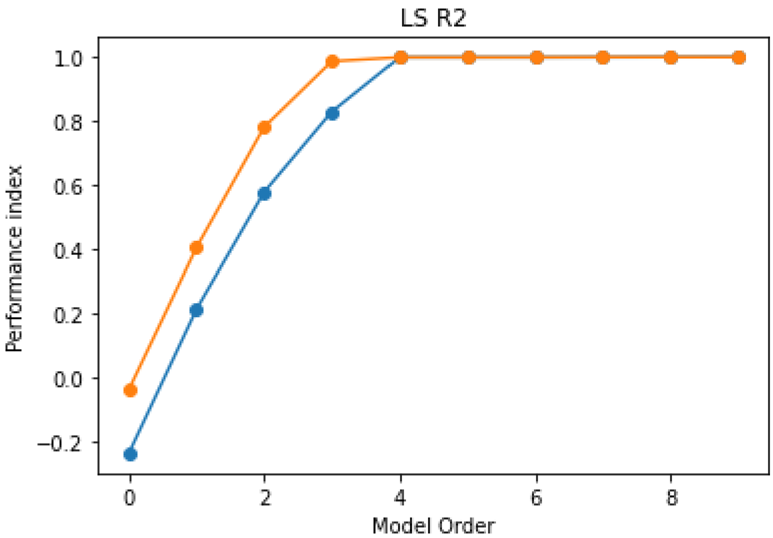
```
for i,key in enumerate(LS_train_df.columns):
    if(key != "Order"):
        plt.figure(num=i+1)
        plt.plot(np.array(orderArray)-1,LS_train_df[key],"o-")
        plt.plot(np.array(orderArray)-1,LS_test_df[key],"o-")
        plt.xlabel("Model Order")
```

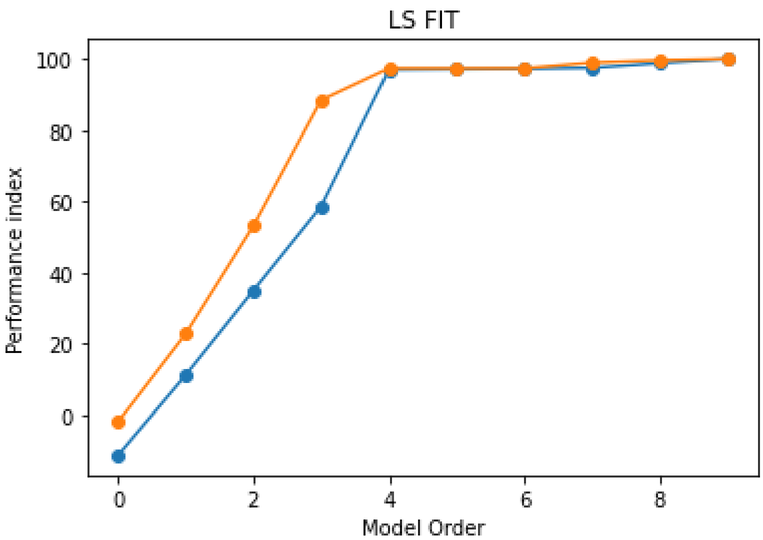
```
        plt.ylabel("Performance index")
        plt.title(f"LS {key}")



for i,key in enumerate(LS_train_df.columns):
    if(key != "Order"):
        plt.figure(num=i+10)
        plt.plot(np.array(orderArray)-1,MLE_train_df[key],"o-")
        plt.plot(np.array(orderArray)-1,MLE_test_df[key],"o-")
        plt.xlabel("Model Order")
        plt.ylabel("Performance index")
        plt.title(f"MLE {key}")
```

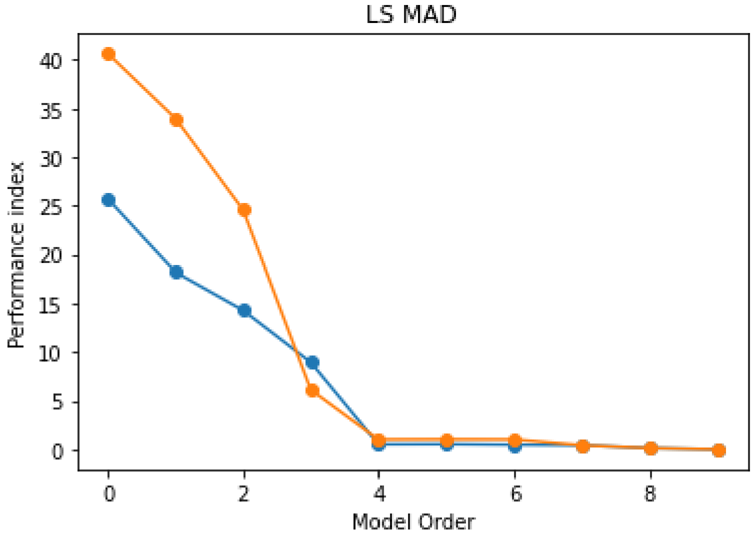


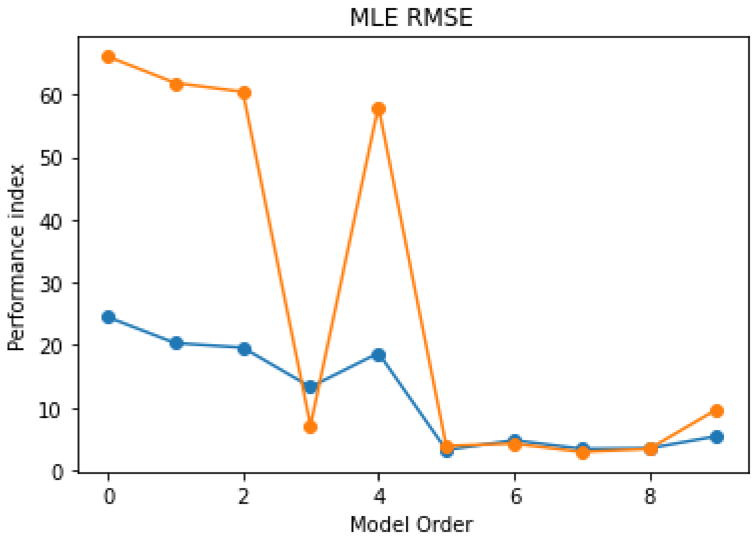


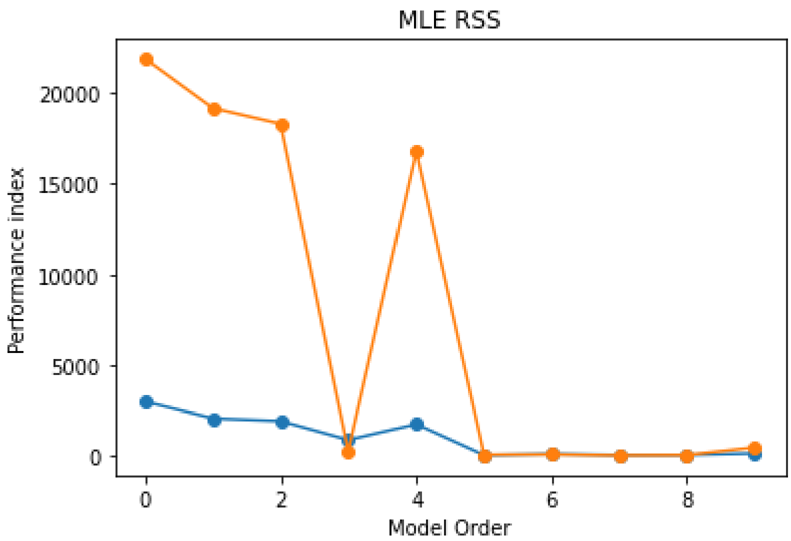

LS FUV
Performance index
Model Order
1.2
1.0
0.8
0.6
0.4
0.2
0.0
0
2
4
6
8

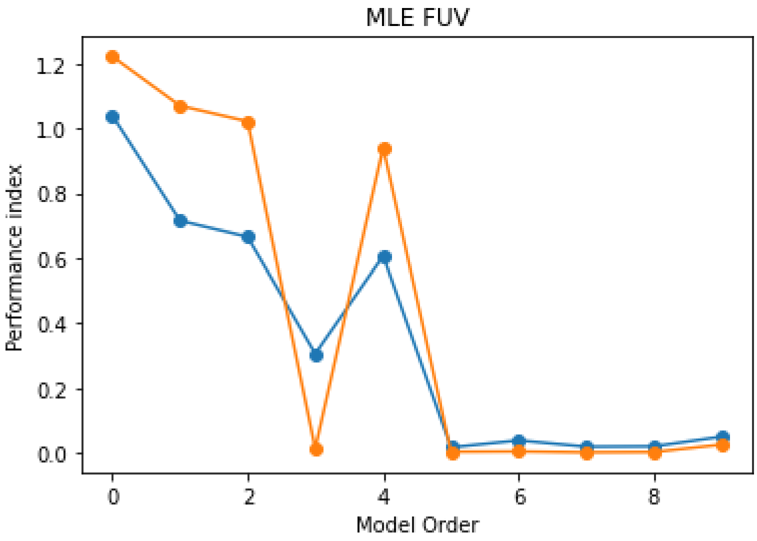

LS R2
Performance index
Model Order
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
0
2
4
6
8

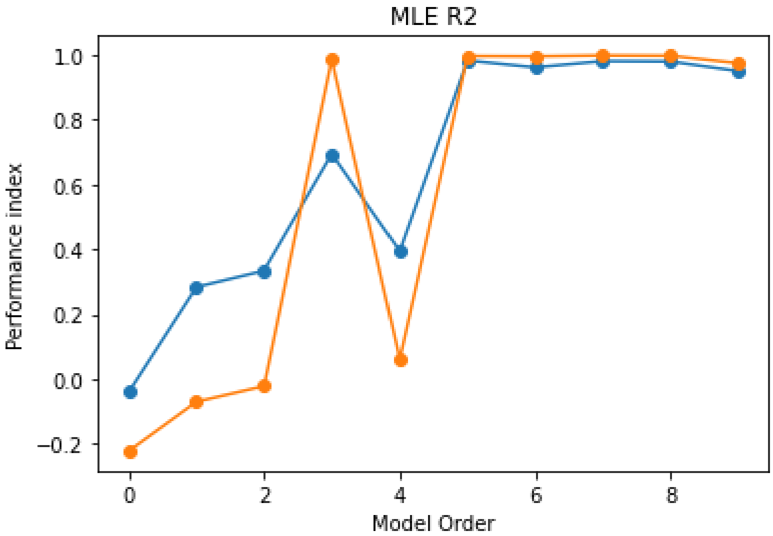

LS FIT
Performance index
Model Order
100
80
60
40
20
0
0
2
4
6
8

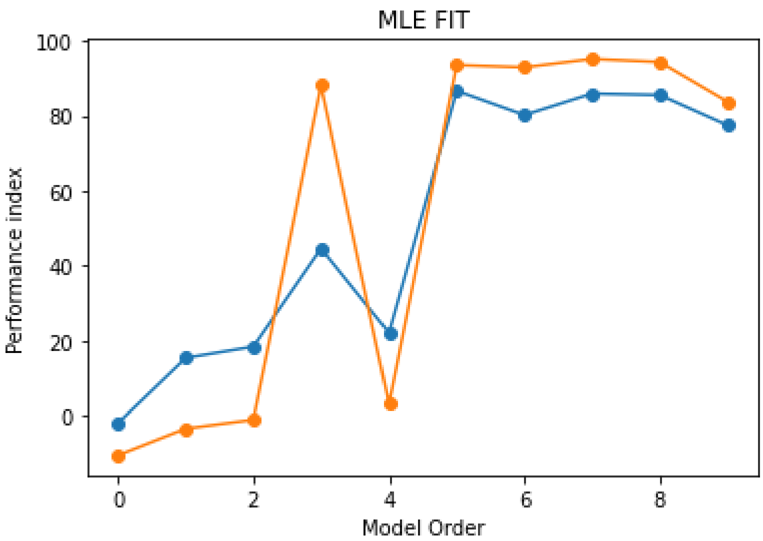

LS MAD
Performance index
Model Order
0
5
10
15
20
25
30
35
40
0
2
4
6
8

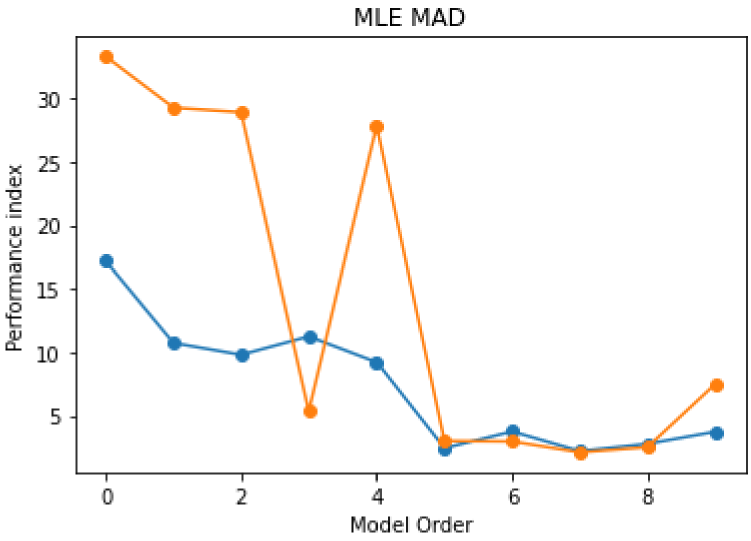

MLE RMSE
Performance index
Model Order
0
10
20
30
40
50
60
0
2
4
6
8


MLE RSS
Performance index
Model Order
0
5000
10000
15000
20000
0
2
4
6
8


MLE FUV
Performance index
Model Order
1.2
1.0
0.8
0.6
0.4
0.2
0.0
0
2
4
6
8


MLE R2
Performance index
Model Order
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
0
2
4
6
8


MLE FIT
Performance index
Model Order
100
80
60
40
20
0
0
2
4
6
8



We can observe from the plots above that the LS estimates performs somewhat better when increasing the order. However the increase in performance is marginal when the order is higher than 4. So a 4th order model would be the best in this case. However as we have limited training and testing data it is hard to make solid conclusions concerning order size.

In the ML case, one can observe that the orders 3 and 5 generally gives the best performance. The main difference from the LS case is that ML performs worse when increasing the order above 5.

Underfitting and overfitting is also an important aspect when choosing model order. Underfitting tends to happen when the model order is lower than the order of the true model and overfitting tends to happen when the model order is higher. An underfit model tends to lose structure in the underlying model and and overfit model tends to follow noise and variations too much and losing the essence of the underlying model.

Concerning the bias variance trade-off, a low order model would give high bias and low variance and a high order model would give the opposite

In [ ]: