Declaring some variables and pacakges

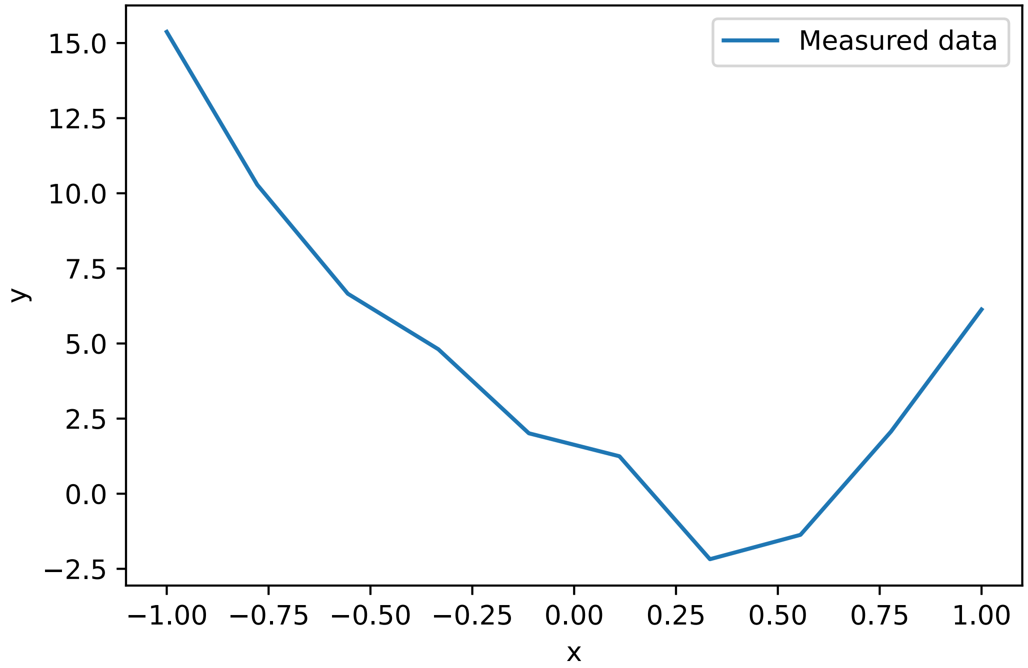
In [79]:
```
# matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import norm, laplace, bernoulli
import scipy.optimize as optimize
import math
from prettytable import PrettyTable

N = 10
magnitude = 1.2
mu = 0  # loc
sigma = 1  # beta
alpha = 0
gamma = 0.1
x = np.linspace(-1, 1, N)
params = [1, -8, 4, 3, 5]
model_order = np.linspace(0,9,10)
```

Function for creating the true model and functino for adding noise

In [80]:
```
def create_y_model(params):
    def y(x): return sum([p*(x**i) for i, p in enumerate(params)])
    return y

def add_noise_to_true_model(x,N, params, magnitude, alpha, mu, sigma):
    y = create_y_model(params)
    y_true = y(x)
    y_added_noise = y_true + magnitude * \
        (alpha*np.random.normal(mu, sigma, N) +
         (1-alpha)*np.random.laplace(mu, sigma, N))
    return y_added_noise
```

Creating our measurment data and plotting it

In [81]:
```
y_measured_data = add_noise_to_true_model(x,N, params, magnitude, alpha
, mu, sigma)
plt.figure()
plt.plot(x, y_measured_data, label="Measured data")
plt.legend()
plt.xlabel("x")
plt.ylabel("y")
plt.show()
```



Splitting up the data into training and testing set

In [82]:
```
y_traning_set = y_measured_data[0:math.floor(len(y_measured_data)*1/2)]
x_traning_set = x[0:math.floor(len(x)/2)]
```

```
y_testing_set = y_measured_data[math.floor(len(y_measured_data)*1/2):]
x_testing_set = x[math.floor(len(x)*1/2):]
```

Function for estimating the LS parameters for each mode

In [83]:
```
def LS_estimator_given_mode(mode, x, y):
    u_tensor_0 = np.reshape(x, (len(x), 1))


    ones_vec = np.ones((len(x), 1))
    u_tensor = np.append(ones_vec, u_tensor_0, axis=1)

    if mode ==1:
        u_tensor = ones_vec
    for i in range(2, mode):
        u_tensor = np.append(u_tensor, np.power(u_tensor_0, i), axis=1)

    u_transpose_dot_u = np.dot(u_tensor.T, u_tensor)  # calculating dot 
product
    u_transpose_dot_u_inv = np.linalg.inv(
        u_transpose_dot_u)  # calculating inverse
    u_transpose_dot_y = np.dot(u_tensor.T, y)  # calculating dot produc
t

    LS_params = np.dot(u_transpose_dot_u_inv, u_transpose_dot_y)
    # Recreate model based on LS estimate:
    LS_params = LS_params.tolist()
    return LS_params
```

Function for estimating ML parameters for each mode

In [84]:
```
def log_lik(par_vec,x,y):
    pdf = laplace.pdf
    # If the standard deviation parameter is negative, return a large v
alue:
    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])

    if all(v == 0 for v in lik):
        return(1e8)
    # Logarithm of zero = -Inf
    return(-sum(np.log(lik[np.nonzero(lik)])))


def ML_estimator_given_mode(mode,x,y):
    init_guess = np.zeros(mode+1)
    init_guess[-1] = len(x)

    opt_res = optimize.minimize(fun=log_lik,
                                x0=init_guess,
                                options={'disp': False},
                                args=(x,y))
    MLE_params = opt_res.x[:-1]
    MLE_params = MLE_params.tolist()
    return MLE_params
```

Creating a function wich takes in data, model orders as a vector and the type of model (LS or ML) as an argument. Function return the models with the estimated parameters based on the given data

In [85]:
```
def creating_different_model(x_data, y_data,data,type):
    if type == "ML":
        for i in range(len(data)):
            data[i] = ML_estimator_given_mode(i + 1,x_data,y_data)
            data[i]= create_y_model(data[i])
    if type == "LS":
        for i in range(len(data)):
            data[i] = LS_estimator_given_mode(i + 1,x_data,y_data)
            data[i]= create_y_model(data[i])

    return data
```

Declaring and estimating the parameters for the 10 LS and ML models

In [86]:
```
y_hat_ML_10_models = [0] *10
y_hat_LS_10_models = [0] *10

creating_different_model(x, y_measured_data,y_hat_LS_10_models,"LS")
creating_different_model(x_traning_set, y_traning_set,y_hat_ML_10_model
s,"ML")
```

Out[86]:
```
[<function __main__.create_y_model.<locals>.y(x)>,
 <function __main__.create_y_model.<locals>.y(x)>,
 <function __main__.create_y_model.<locals>.y(x)>,
 <function __main__.create_y_model.<locals>.y(x)>,
 <function __main__.create_y_model.<locals>.y(x)>,
 <function __main__.create_y_model.<locals>.y(x)>,
 <function __main__.create_y_model.<locals>.y(x)>,
 <function __main__.create_y_model.<locals>.y(x)>,
 <function __main__.create_y_model.<locals>.y(x)>,
 <function __main__.create_y_model.<locals>.y(x)>]
```

Function for computing the RMSE performance index

In [87]:
```
def rmse_performance_index(x_data,y_data,y_hat_models):
    performance_vector = [0]*len(y_hat_models)
    for i in range(len(y_hat_models)):
        y_hat = y_hat_models[i](x_data)
        for j in range(len(y_data)):
            performance_vector[i] += (abs(y_data[j] - y_hat[j]))**2
        performance_vector[i] = (performance_vector[i]/len(y_data))**(1
/2)
        #print(performance_vector[i])
    return performance_vector
```

Function for computing the RSS performance index

In [88]:
```
def rss_performance_index(x_data,y_data,y_hat_models):
```

```
    performance_vector = [0]*len(y_hat_models)
    for i in range(len(y_hat_models)):
        y_hat = y_hat_models[i](x_data)
        for j in range(len(y_data)):
            performance_vector[i] += (abs(y_data[j] - y_hat[j]))**2
        #print(performance_vector[i])
    return performance_vector
```

Function for computing mean

In [89]:
```
def mean(x_data,y_data,y_hat_models):
    mean = [0]*len(y_hat_models)
    for i in range(len(y_hat_models)):
        y_hat = y_hat_models[i](x_data)
        for j in range(len(y_hat)):
            mean[i] += y_hat[j]
        mean[i] = mean[i]/len(y_hat)
    return mean
```

Function for computing the FVU performance index

In [90]:
```
def fvu_performance_index(x_data,y_data,y_hat_models):
    mean_vector = mean(x_data,y_data,y_hat_models)
    rss_performance_vector = rss_performance_index(x_data,y_data,y_hat_
models)
    fvu_performance_vector = [0]*len(y_hat_models)

    variance_vector = [0]*len(y_hat_models)
    for i in range(len(y_hat_models)):
        for j in range(len(y_data)):
            variance_vector[i] += (abs(y_data[j] - mean_vector[i]))**2
        fvu_performance_vector[i] = rss_performance_vector[i]/variance_
vector[i]
    return fvu_performance_vector
```

Function for computing the R^2 performance index

In [91]:
```
def rr_performance_index(x_data,y_data,y_hat_models):
    fvu_performance_vector = fvu_performance_index(x_data,y_data,y_hat_
models)
    rr_performance_vector = [0]*len(y_hat_models)

    for i in range(len(y_hat_models)):
        rr_performance_vector[i] = 1 - fvu_performance_vector[i]

    return rr_performance_vector
```

Function for computing FIT performance index

In [92]:
```
def fit_performance_index(x_data,y_data,y_hat_models):
    fvu_performance_vector = fvu_performance_index(x_data,y_data,y_hat_
models)
    fit_performance_vector = [0]*len(y_hat_models)

    for i in range(len(y_hat_models)):
        fit_performance_vector[i] = 100*(1 - (fvu_performance_vector[i
])**(1/2))
    return fit_performance_vector
```

Creating a functino wich takes in the data set as an argument. It will then calculate the performance index for all the models and print it as a table

In [93]:
```
def make_table(x_data,y_data):
    ML_rmse_performance_vector = rmse_performance_index(x_data,y_data,y
_hat_ML_10_models)
    ML_rss_performance_vector = rss_performance_index(x_data,y_data,y_h
at_ML_10_models)
    ML_fvu_performance_vector = fvu_performance_index(x_data,y_data,y_h
at_ML_10_models)
    ML_rr_performance_vector  = rr_performance_index(x_data,y_data,y_ha
t_ML_10_models)
    ML_fit_performance_vector = fit_performance_index(x_data,y_data,y_h
```

```
at_ML_10_models)

    LS_rmse_performance_vector = rmse_performance_index(x_data,y_data,y
_hat_LS_10_models)
    LS_rss_performance_vector = rss_performance_index(x_data,y_data,y_h
at_LS_10_models)
    LS_fvu_performance_vector = fvu_performance_index(x_data,y_data,y_h
at_LS_10_models)
    LS_rr_performance_vector  = rr_performance_index(x_data,y_data,y_ha
t_LS_10_models)
    LS_fit_performance_vector = fit_performance_index(x_data,y_data,y_h
at_LS_10_models)

    x = PrettyTable()
    x.add_column("Paramter order(MLE)",model_order)
    x.add_column("RMSE",ML_rmse_performance_vector)
    x.add_column("RSS",ML_rss_performance_vector)
    x.add_column("FVU",ML_fvu_performance_vector)
    x.add_column("RR",ML_rr_performance_vector)
    x.add_column("FIT",ML_fit_performance_vector)
    print(x)

    t = PrettyTable()
    t.add_column("Paramter order(LS)",model_order)
    t.add_column("RMSE",LS_rmse_performance_vector)
    t.add_column("RSS",LS_rss_performance_vector)
    t.add_column("FVU",LS_fvu_performance_vector)
    t.add_column("RR",LS_rr_performance_vector)
    t.add_column("FIT",LS_fit_performance_vector)
    print(t)
```

Makeing a table for the traning data set, a table of the model orders and their computed performance indexes

In [94]:
```
make_table(x_traning_set,y_traning_set)
```

```
+--------------------+--------------------+--------------------+----
------------------+--------------------+------------------+
```

Paramter order(MLE)	RMSE	RSS	FVU	RR	FIT
0.0	4.777466479016286	114.12092979062132	1.0	0.0	0.0
1.0	1.1061735215349675	6.1180992987253555	0.05680475736428353	0.9431952426357164	76.16625137241655
2.0	0.6339258742229281	2.0093100700465185	0.018547446076393158	0.9814525539236069	86.3810991352484
3.0	0.3619828479942683	0.6551579112102077	0.006100226521964934	0.993899773478035	92.18960530961147
4.0	0.3333560963003287	0.55563143470297	0.005178914264367435	0.9948210857356325	92.80353262748491
5.0	0.35873686036609503	0.6434606749266158	0.005991601659433062	0.994008398340567	92.25945631145134
6.0	0.28819712729963315	0.41528792091880473	0.0038695649664841425	0.9961304350335158	93.77941725681256
7.0	0.6171388089874399	1.904301547792179	0.017691909127063254	0.9823080908729367	86.69890638817121
8.0	0.7989538700007153	3.191636431945599	0.029680111256880495	0.9703198887431195	82.77208333637509
9.0	1.362820953665232	9.286404758745064	0.08326330784608969	0.9167366921539103	71.14461785973201

Paramter order(LS)	RMSE	RSS	FVU	RR	FIT
0.0	5.700886327872605	162.50052461662398	1.0	0.0	0.0
1.0	2.954072656253969	43.632726292136894	0.4054787261138132	0.5945212738861868	36.32278852573605
2.0	0.7534644060468675	2.8385430558977935	0.026378567651686374	0.9736214323483137	83.75851987912235
3.0	0.6982166411238399	2.4375323897112846	0.022720159971794645	0.9772798400282053	84.9267919898269

```
|         4.0          |   0.2659013994121197    |  0.35351777104661813  |
0.0032951276237206395 | 0.9967048723762794 | 94.25967977921036 |
|         5.0          |   0.3105881578264924    |  0.48232501891027074  |
0.004493939380286912 | 0.9955060606197131 | 93.29631490873346 |
|         6.0          |  0.34682394761067387    |   0.6014342531812573  |
0.005603709053142751 | 0.9943962909468572 | 92.51420742129281 |
|         7.0          |   0.3219421405063864    |   0.5182337091691692  |
0.00482918299464844  | 0.9951708170053516 | 93.05076767214648 |
|         8.0          |  0.38217365976084333    |   0.7302835310749842  |
0.006805178333908577 | 0.9931948216660914 | 91.75064952016913 |
|         9.0          | 3.9390532921624515e-10 | 7.758070419247922e-19 |
7.231374934301327e-21 |        1.0         | 99.99999999149625 |
+---------------------+------------------------+-----------------------+
----------------------+--------------------+-------------------+
```

Makeing a table for the testing data set, a table of the model orders and their computed performance indexes

In [95]:
```
make_table(x_testing_set,y_testing_set)
```

```
+---------------------+--------------------+--------------------+------
--------------+---------------------+---------------------+
| Paramter order(MLE) |        RMSE        |        RSS         |
FVU          |          RR          |         FIT          |
+---------------------+--------------------+--------------------+------
--------------+---------------------+---------------------+
|         0.0         | 6.214294125804777  | 193.08725741005878 |
1.0          |         0.0          |         0.0          |
|         1.0         | 9.746211779819818  | 474.94322028549294 | 1.478
8690369766566 | -0.4788690369766566  |  -21.60875942861422 |
|         2.0         | 2.2087099350977226 | 24.391997886996933 | 0.565
0090387004066 | 0.43499096129959336  |  24.832916865132603 |
|         3.0         | 8.286122988575908  | 343.29917090903075 | 1.825
2058422370545 | -0.8252058422370545  |  -35.1001792092466  |
|         4.0         | 3.761617967557474  |  70.7488486692561  |  1.26
300290191739  | -0.26300290191739006 | -12.383401884681788 |
|         5.0         | 4.322726628090211  | 93.42982750600085  | 1.341
2523066656916 | -0.3412523066656916  | -15.812447805306817 |
```

```
|         6.0         | 8.738122771285836  | 381.7739478303203  | 1.648
5858636466184 | -0.6485858636466184  |  -28.39726880454343 |
|         7.0         | 2.5062109580715064 | 31.405466831788488 | 0.644
5654872895434 | 0.35543451271045656  |  19.715164116158057 |
|         8.0         | 3.359925971654846  | 56.44551267500381  | 1.217
9648828804337 | -0.2179648828804337  | -10.361446297175437 |
|         9.0         | 3.798973218645618  | 72.16098757993323  | 1.207
453280738429  | -0.20745328073842906 |   -9.8841790586083  |
+---------------------+--------------------+--------------------+------
--------------+--------------------+--------------------+
+--------------------+-----------------------+-----------------------+
-----------------------+---------------------+--------------------+
| Paramter order(LS) |          RMSE         |          RSS          |
FVU           |          RR          |         FIT          |
+--------------------+-----------------------+-----------------------+
-----------------------+---------------------+--------------------+
|        0.0         |   4.434424985759466    |   98.32062477163922   |
1.0           |         0.0          |         0.0          |
|        1.0         |   4.310621113363217    |   92.90727191486368   |
2.139338856182609    | -1.1393388561826092 | -46.2647892071981   |
|        2.0         |   1.4482903017646591   |   10.487723990927835  |
0.241496655582252    |  0.758503344417748  | 50.857690776454966 |
|        3.0         |   0.9520065594230637   |   4.531582445922696   |
0.10512868092311341  |  0.8948713190768866 |  67.5764466902356   |
|        4.0         |   0.8552142083949194   |   3.656956711202743   |
0.08483814204629163  |  0.9151618579537084 | 70.87301216289407   |
|        5.0         |   0.6897145584645149   |   2.3785308607895037  |
0.05512495498304014  |  0.9448750450169598 | 76.52129582301438   |
|        6.0         |   0.4485694829263983   |   1.0060729050642816  |
0.023316797993074966 |  0.976683202006925  | 84.73016110331383   |
|        7.0         |  0.44717056830030766   |   0.9998075857701005  |
0.02317963533134174  |  0.9768203646686583 | 84.77514028591996   |
|        8.0         |  0.38217365975423717   |   0.7302835310497371  |
0.016930963696558606 |  0.9830690363034414 |  86.9880963358321   |
|        9.0         | 3.8436053225871353e-10 | 7.386650937910079e-19 |
1.713697918482738e-20 |         1.0         | 99.99999998690917  |
+--------------------+-----------------------+-----------------------+
-----------------------+---------------------+--------------------+
```

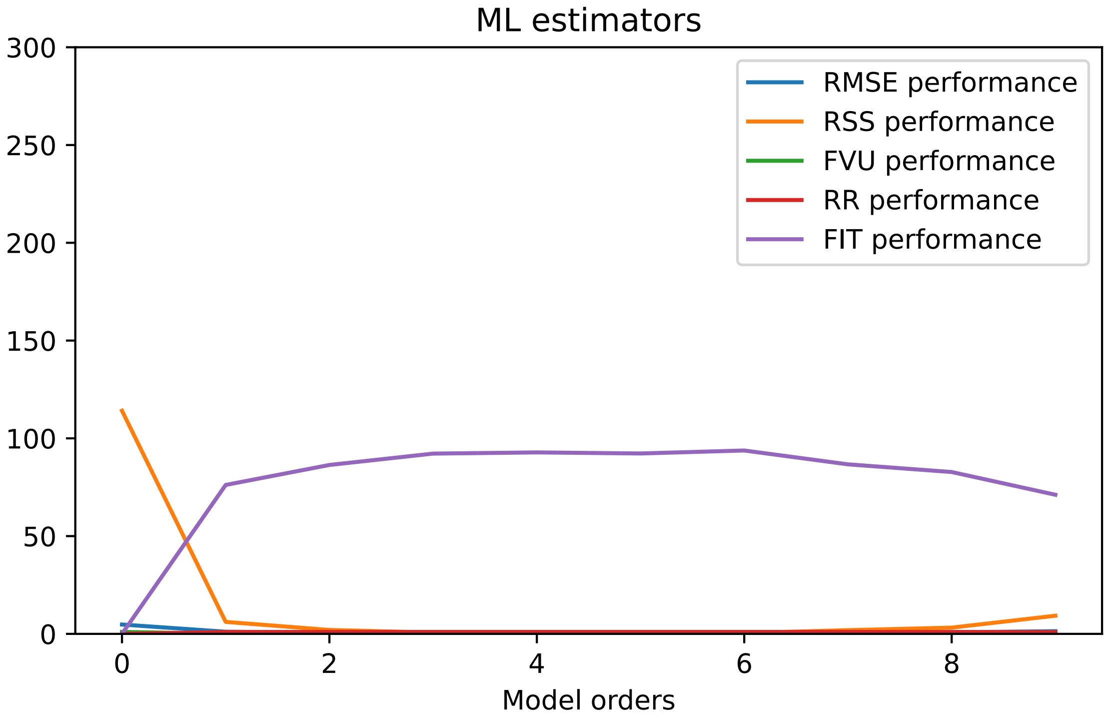
Making a function that plots all the performance indexes as a function of the models orders given the type of data

In [96]:
```
def plotting(x_data,y_data):
    ML_rmse_performance_vector = rmse_performance_index(x_data,y_data,y
_hat_ML_10_models)
    ML_rss_performance_vector = rss_performance_index(x_data,y_data,y_h
at_ML_10_models)
    ML_fvu_performance_vector = fvu_performance_index(x_data,y_data,y_h
at_ML_10_models)
    ML_rr_performance_vector  = rr_performance_index(x_data,y_data,y_ha
t_ML_10_models)
    ML_fit_performance_vector = fit_performance_index(x_data,y_data,y_h
at_ML_10_models)

    LS_rmse_performance_vector = rmse_performance_index(x_data,y_data,y
_hat_LS_10_models)
    LS_rss_performance_vector = rss_performance_index(x_data,y_data,y_h
at_LS_10_models)
    LS_fvu_performance_vector = fvu_performance_index(x_data,y_data,y_h
at_LS_10_models)
    LS_rr_performance_vector  = rr_performance_index(x_data,y_data,y_ha
t_LS_10_models)
    LS_fit_performance_vector = fit_performance_index(x_data,y_data,y_h
at_LS_10_models)

    plt.figure()
    plt.title("ML estimators")
    plt.xlabel("Model orders")
    plt.plot(model_order, ML_rmse_performance_vector, label= "RMSE perf
ormance")
    plt.plot(model_order, ML_rss_performance_vector, label= "RSS perfor
mance")
    plt.plot(model_order, ML_fvu_performance_vector, label= "FVU perfor
mance")
    plt.plot(model_order, ML_rr_performance_vector, label= "RR performa
nce")
    plt.plot(model_order, ML_fit_performance_vector, label= "FIT perfor
mance")
```

```
    plt.ylim(0, 300)
    plt.legend()
    plt.tight_layout()

    plt.figure()
    plt.title("LS estimators")
    plt.xlabel("Model orders")
    plt.plot(model_order, LS_rmse_performance_vector, label= "RMSE perf
ormance")
    plt.plot(model_order, LS_rss_performance_vector, label= "RSS perfor
mance")
    plt.plot(model_order, LS_fvu_performance_vector, label= "FVU perfor
mance")
    plt.plot(model_order, LS_rr_performance_vector, label= "RR performa
nce")
    plt.plot(model_order, LS_fit_performance_vector, label= "FIT perfor
mance")
    plt.ylim(0, 300)
    plt.legend()
    plt.tight_layout()
    plt.show()
```

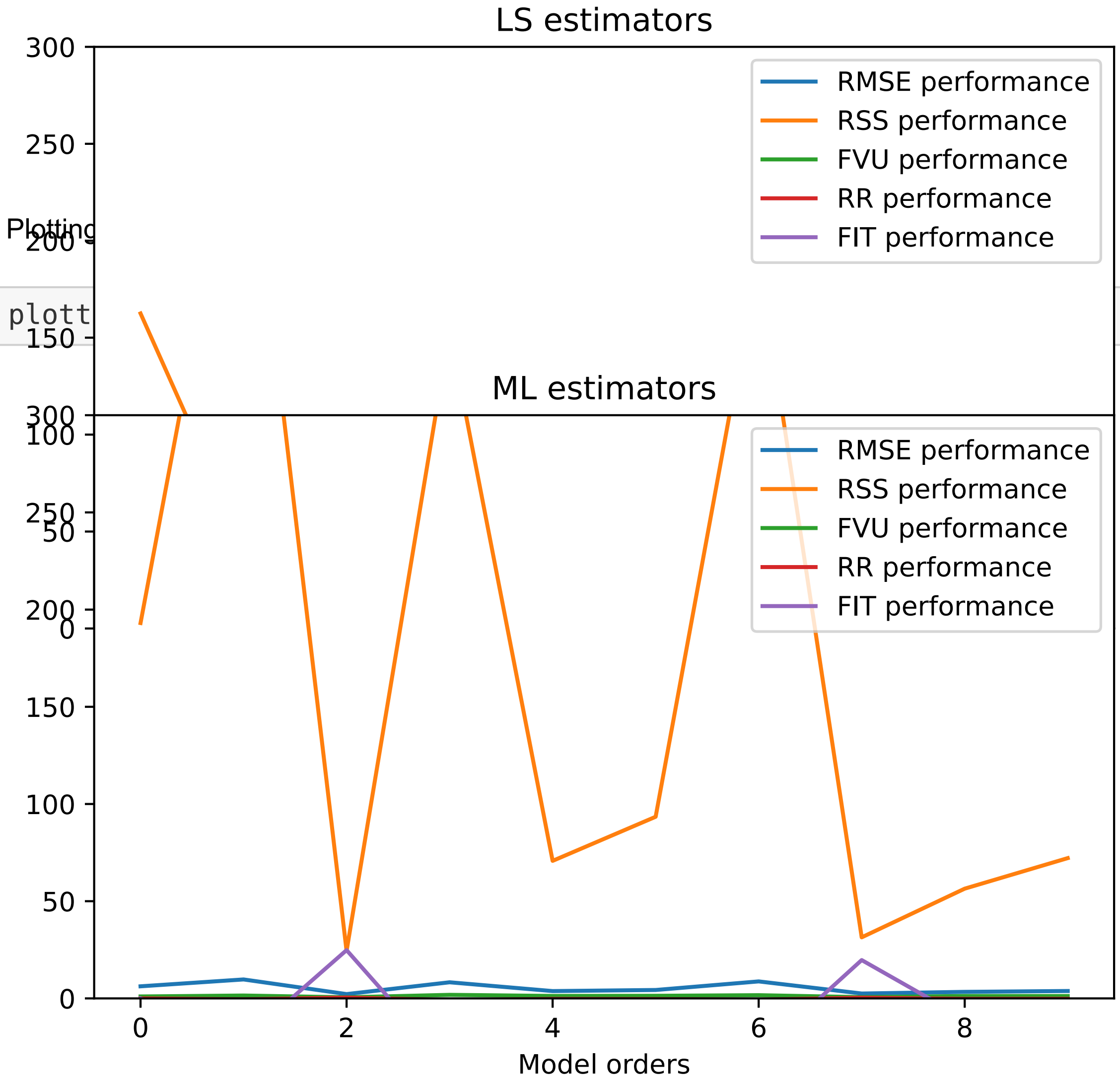
Plotting the performance indexes as a function of the models orders on the training set

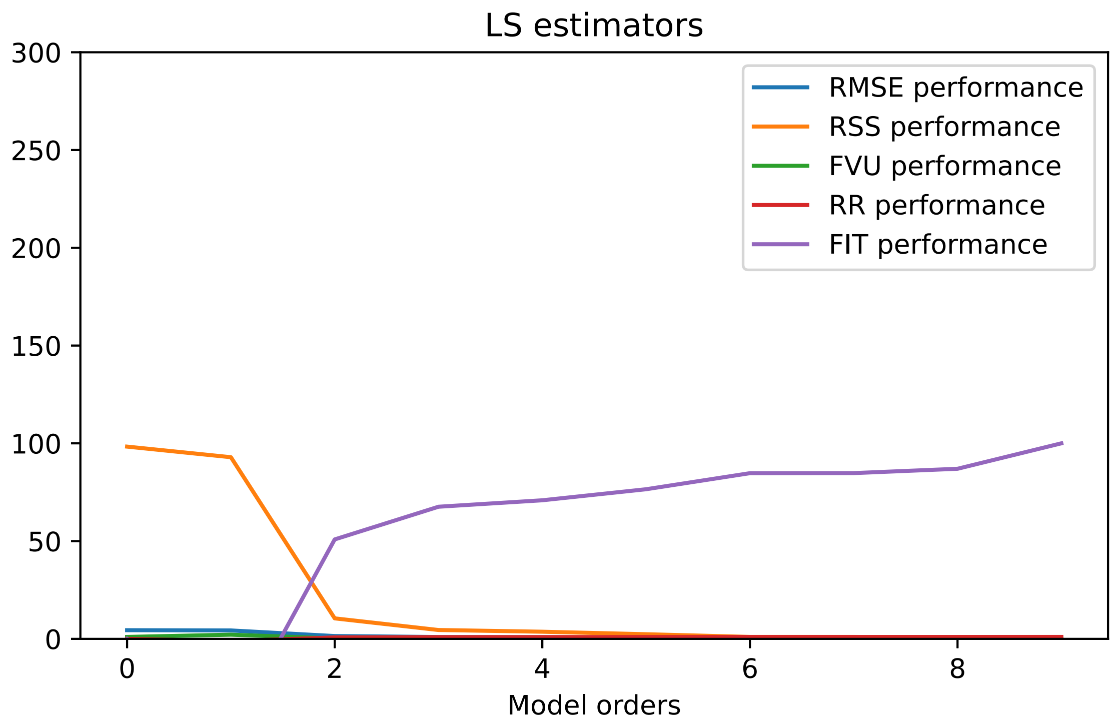
In [97]:
```
plotting(x_traning_set,y_traning_set)
```


ML estimators
300
250
200
150
100
50
0
0
2
4
6
8
Model orders
RMSE performance
RSS performance
FVU performance
RR performance
FIT performance

Plotting

In [98]: `plott`





Interpriting the data For the traing set we can clearly see that as the model order increase, all the performance indexes reduce excpet for the FIT performance index. For the ML models, the FIT performance index increases til model order 1 and then stays more or less constant. For the LS models, the FIT performance index keeps increasing as the models orders also increase. This indicates that the higher level model orders are very good at a exactly modeling the data. To get an indication of whether this is overfitting the data, we have to take a look at how well the performance indexes performaned on the test set. For the test set the plots look quite different. For the ML models, the RMSE, RSS and the FIT performance index are quite terrible and not nearly as low as on the training set. This indicates that the ML models have been overfitted for the higher models orders. For the LS estimatros, it is hard to tell the plots apart because it seems like the all the performance idexes behaves the same way for both data set.

I know that data that have been overfitted have lower bias because it makes a perfect model intersects all the points. Concequently when the model is used for a test set, it will not peform well because it is not generalized anymore and therefore it will also have high variance. On the other hand, data that are underfitted will have the opposite effect. It will have higher error on the trainig set because the model is too general, but it will also perform better on the test set as well. Therefore underfitted data will have higher bias but lower variance.

We can see that for the lower order ML and LS models, the performance indexes performs more or less the same for both the training and test set. This could indicate that the data are underfitted.

In [ ]:

```

```