In [1]:

```
import numpy as np
import pandas as pd
%matplotlib inline
import matplotlib as mpl
import matplotlib.pyplot as plt
from scipy import special
```

In [264]:

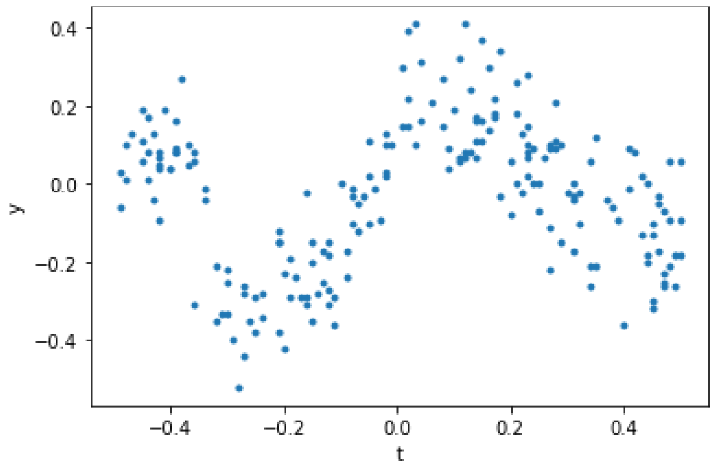
```
t = np.array([-0.49,  0.48, -0.19,  0.24, -0.08,  0.49, -0.21, -0.16, -0.13,
        0.5 ,  0.09,  0.14, -0.27,  0.01,  0.37, -0.42, -0.45, -0.14,
        0.17,  0.14, -0.38,  0.18, -0.45, -0.3 , -0.12,  0.02, -0.11,
        0.38, -0.01,  0.12,  0.44,  0.32,  0.2 , -0.26,  0.04,  0.23,
        0.18,  0.14,  0.15, -0.21, -0.41,  0.06, -0.25, -0.31, -0.42,
       -0.09,  0.34, -0.05,  0.21,  0.26,  0.27,  0.27, -0.28, -0.49,
        0.01,  0.29, -0.15,  0.34,  0.49,  0.03, -0.12,  0.15, -0.02,
       -0.02,  0.24,  0.47, -0.06,  0.34, -0.29, -0.43, -0.09,  0.14,
        0.23,  0.23,  0.08,  0.12, -0.42, -0.12, -0.19,  0.45, -0.36,
       -0.08,  0.13, -0.16,  0.23,  0.44,  0.5 , -0.3 ,  0.45, -0.24,
       -0.42,  0.28,  0.11, -0.47,  0.11, -0.27,  0.41,  0.46, -0.36,
       -0.32, -0.05,  0.45, -0.08, -0.39, -0.34, -0.37,  0.32, -0.24,
       -0.44, -0.03, -0.4 ,  0.47,  0.27, -0.32, -0.2 , -0.37, -0.13,
       -0.07,  0.08, -0.44, -0.44,  0.43, -0.04,  0.09,  0.35, -0.02,
       -0.12,  0.42, -0.39, -0.25, -0.45,  0.44, -0.02,  0.31, -0.48,
        0.23, -0.07, -0.17,  0.4 ,  0.35, -0.2 ,  0.45, -0.1 ,  0.17,
        0.21, -0.11, -0.3 ,  0.48,  0.31,  0.21,  0.2 ,  0.27,  0.47,
       -0.39,  0.46,  0.31,  0.23,  0.5 , -0.43,  0.13, -0.34,  0.02,
       -0.27,  0.3 , -0.18,  0.41,  0.17, -0.15,  0.29, -0.05, -0.39,
        0.25,  0.28,  0.02, -0.21,  0.11,  0.16,  0.47,  0.1 ,  0.03,
        0.22,  0.43,  0.16, -0.21, -0.16,  0.39,  0.25,  0.04, -0.4 ,
        0.22,  0.46,  0.12,  0.15,  0.48, -0.48, -0.15, -0.42,  0.28,
       -0.36,  0.31])

y = np.array([-0.06, -0.09, -0.19,  0.  , -0.03, -0.18, -0.12, -0.31, -0.17,
       -0.18,  0.09,  0.16, -0.28,  0.15, -0.04,  0.04,  0.06, -0.28,
        0.18,  0.11,  0.27,  0.34,  0.11, -0.22, -0.18,  0.39, -0.29,
       -0.06,  0.1 ,  0.08, -0.  , -0.02,  0.06, -0.35,  0.16,  0.28,
       -0.03,  0.17,  0.37, -0.15,  0.19,  0.21, -0.29, -0.33,  0.08,
       -0.24,  0.06,  0.02,  0.26,  0.07,  0.1 ,  0.09, -0.52,  0.03,
        0.3 ,  0.1 , -0.15, -0.26, -0.26,  0.1 , -0.27,  0.11,  0.02,
        0.1 ,  0.09, -0.23, -0.03, -0.21, -0.4 , -0.04, -0.17,  0.07,
        0.08,  0.15,  0.27,  0.41,  0.07, -0.31, -0.29, -0.3 ,  0.06,
       -0.1 ,  0.08, -0.29,  0.07, -0.2 ,  0.06, -0.25, -0.13, -0.34,
        0.05,  0.21,  0.07,  0.13,  0.32, -0.26, -0.01, -0.17, -0.31,
       -0.21,  0.11, -0.1 , -0.01,  0.16, -0.01,  0.1 , -0.1 , -0.28,
        0.08, -0.09,  0.04, -0.07, -0.11, -0.35, -0.23,  0.05, -0.25,
       -0.12,  0.15,  0.01,  0.17,  0.02, -0.01,  0.04, -0.21,  0.03,
       -0.15,  0.08,  0.08, -0.38,  0.19, -0.18,  0.13, -0.03,  0.1 ,
        0.1 , -0.05, -0.29, -0.36,  0.12, -0.42, -0.32, -0.  ,  0.22,
        0.  , -0.36, -0.33,  0.06, -0.04,  0.18, -0.08, -0.22, -0.25,
        0.09, -0.03, -0.17,  0.02, -0.09,  0.13,  0.24, -0.04,  0.15,
       -0.44, -0.02, -0.24,  0.09,  0.17, -0.35, -0.15, -0.1 ,  0.08,
        0.  ,  0.09,  0.22, -0.38,  0.06,  0.14, -0.26,  0.19,  0.41,
        0.13, -0.13,  0.3 , -0.15, -0.02, -0.09, -0.07,  0.31,  0.04,
       -0.02, -0.05,  0.07,  0.16, -0.21,  0.01, -0.2 , -0.09,  0.11,
        0.08, -0.  ])
```

In [265]:

```
plt.plot(t, y, '.')
plt.xlabel('t')
plt.ylabel('y')
```

Out[265]:

```
Text(0, 0.5, 'y')
```



In [266]:

```
def fourier(t, p):
    n = t.size
    return np.stack(
        [np.cos(f * 2*np.pi*t) for f in range(0, (p+1)//2)] +
        [np.sin(f * 2*np.pi*t) for f in range(1, (p+2)//2)])
```

In [267]:

```
def legendre(t, p):
    n = t.size
    return np.stack(
        [special.legendre(d)(t) for d in range(p)])
```

Compute performance measure (cost function - MSE)

Measure average cross-validated MSE.

In [268]:

```
def find_ground_truth(feature,p,n_folds):

    mse_val = 0
    mse_val_total = 0
    arr_mse = [];

    dict = {};  # Store index of MSE values using dictionary

    for j in range(1,p+1):   #Iterate through Features
        mse_val = 0;
        X = feature(t,j+1)     #Get X for given model
        for train_index, test_index in folds.split(X.T,y):    #Iterate through folds

            X_train, X_test, y_train, y_test = X[:,train_index], X[:,test_index], \
                                               y[train_index], y[test_index]

            theta_hat=np.linalg.inv(X_train @ X_train.T) @ X_train @ y_train   # fit-trai
n model
            y_predict = theta_hat @ X_test
            mse_val += ((y_test - y_predict)**2).mean()   # compute cost function


        mse_val_total = mse_val/n_folds;

        dict.update({mse_val_total:j})
        arr_mse.append(mse_val_total)

        plt.plot(t,theta_hat @ X, '.',label=str(j))
        plt.legend(fontsize=7, loc="upper right")
    return arr_mse,dict;
```

Use 10-fold cross-validation to compare models.

In [269]:

```
n_of_splits = 10
n_of_features = 5
```

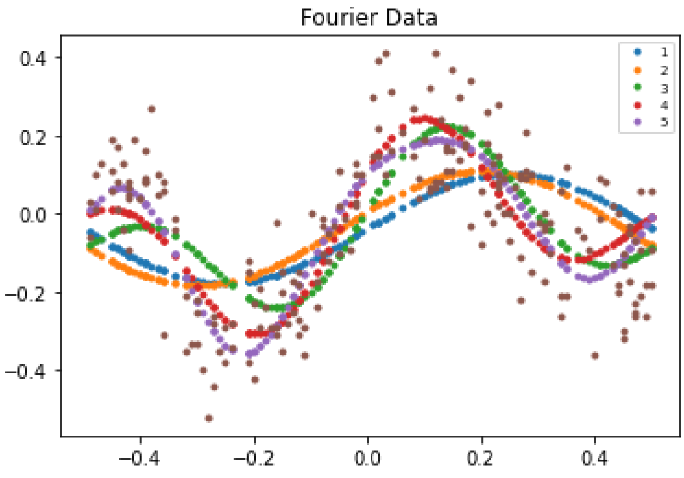
In [270]:

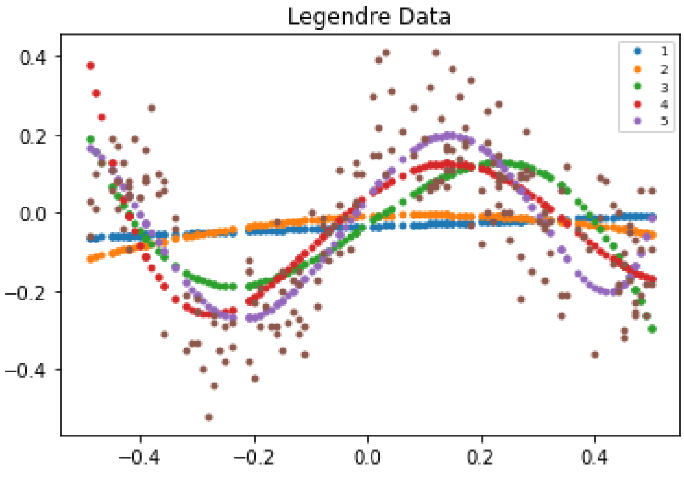
```
from sklearn.model_selection import KFold
folds = KFold(n_splits=n_of_splits)
```

Output: best model and best p.

In [279]:

```
fourier_mse,fourier_dict = find_ground_truth(fourier,n_of_features,n_of_splits)
fourier_min_mse=min(fourier_mse);

plt.plot(t, y, '.')
plt.title("Fourier Data")
plt.show()

legendre_mse,legendre_dict = find_ground_truth(legendre,n_of_features,n_of_splits)
legendre_min_mse=min(legendre_mse);

plt.plot(t, y, '.')
plt.title("Legendre Data")
plt.show()


if(fourier_min_mse < legendre_min_mse):
    print('Data Source : Fourier\nNumber of Feature(s): ',fourier_dict.get(fourier_min_m
se), ' Min MSE:',fourier_min_mse)
else:
    print('Data Source : Legendre\nNumber of Feature(s): ', legendre_dict.get(legendre_m
in_mse), ' Min MSE:',legendre_min_mse)
```





```
Data Source : Fourier
Number of Feature(s):  5  Min MSE: 0.014137183578989809
```

Cross Validation Score for different Models

In [75]:

```
from sklearn.model_selection import cross_val_score
from sklearn.metrics import accuracy_score
```

> **Logistic, RandomForest and SV Models will not be useful since this is not classification problem.**

LinearRegression Model

In [291]:

```
from sklearn.linear_model import LinearRegression
```

In [292]:

```
np.mean(cross_val_score(LinearRegression(), legendre(t,20).T, y, scoring='r2', cv=10))
```

Out[292]:

```
0.625961574846807
```

In [293]:

```
np.mean(cross_val_score(LinearRegression(), fourier(t,20).T, y, scoring='r2', cv=10))
```

Out[293]:

```
0.6425652690915111
```

SVR Model

In [306]:

```
from sklearn.svm import SVR
from sklearn.metrics import mean_squared_error
```

In [307]:

```
np.mean(cross_val_score(SVR(gamma='auto'), legendre(t,20).T, y,cv=10))
```

Out[307]:

```
0.6002013521027897
```

In [308]:

```
np.mean(cross_val_score(SVR(gamma='auto'), fourier(t,20).T, y,cv=10))
```

Out[308]:

```
0.6000705718453085
```

Putting eveything togather

> **10 K Folds, 20 features**

In [309]:

```
folds=10
p=20
```

In [310]:

```
def check_accuracy(features,model):
    arr_svr = [];
    dict = {};
    for j in range(1,p+1):   #Iterate through p features
        if(model== "SVM"):
            score=np.mean(cross_val_score(SVR(gamma='auto'), features(t,j).T, y,cv=folds
))
        else:
            score=np.mean(cross_val_score(LinearRegression(), features(t,j).T, y, scorin
g='r2', cv=10))
        arr_svr.append(score)
        dict.update({score:j})
    return dict,arr_svr;
```

In [311]:

```
def check_models(model):
    dict_fourier,arr_f_score = check_accuracy(fourier,model)
    dict_legendre,arr_l_score = check_accuracy(legendre,model)

    max_f_score=max(arr_f_score)
    max_l_score=max(arr_l_score)

    if(max_f_score>max_l_score):
        print('\nModel:',model,'\nSource with high score: Fourier \nAccuracy: ',max_f_sc
ore, 'Features:',dict_fourier.get(max_f_score))
    else:
        print('\nModel:',model,'\nSource with high score: Legendre \nAccuracy: ',max_l_s
core, 'Features:',dict_legendre.get(max_l_score))
    return;
```

In [312]:

```
check_models("Linear Regression")
```

```
Model: Linear Regression
Source with high score: Fourier
Accuracy:  0.6466266724987295 Features: 16
```

In [313]:

```
check_models("SVM")
```

```
Model: SVM
Source with high score: Fourier
Accuracy:  0.6493367018369034 Features: 9
```

Linear Regression vs SVR Model

Score: Higher is better

> **Winner: SVR Model. Data: Fourier**

Hyperparameter Tuning Using Grid Search Cross-Validation for Linear Regression Model

A common use of cross-validation is for tuning hyperparameters of a model. The most common technique is what is called grid search cross-validation

---

> **GridSearchCV should be used to find the optimal parameters to train your final model. Typically, you should run GridSearchCV then look at the parameters that gave the model with the best score. You should then take these parameters and train your final model on all of the data.**

In [297]:

```
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LinearRegression
from sklearn import linear_model
```

> **10 K Folds, 20 features**

In [315]:

```
folds=10
p=20
```

> **Ignore future version warnings**

In [316]:

```
from warnings import simplefilter
simplefilter(action='ignore', category=FutureWarning)
simplefilter(action='ignore', category=DeprecationWarning)
```

In [317]:

```
def grid_cv(features):

    X =features(t,p+1)

    X_train, X_test, y_train, y_test = train_test_split(X.T, y, test_size=folds/100)

    model = linear_model.LinearRegression()
    parameters = {'fit_intercept':[True,False], 'normalize':[True,False], 'copy_X':[True
, False]}

    grid = GridSearchCV(model,parameters, cv=None)
    grid.fit(X_train, y_train)
    print('R2 / variance : ', grid.best_score_)
    print('Residual sum of squares : %.2f' % np.mean((grid.predict(X_test) - y_test) **
2))
    print('Best Params : ',grid.best_params_)

    #df_cv_results = pd.DataFrame(grid.cv_results_)
    #df_cv_results

    return;
```

In [318]:

```
grid_cv(legendre)
```

```
R2 / variance :  0.6208548813318361
Residual sum of squares : 0.01
Best Params :  {'copy_X': True, 'fit_intercept': True, 'normalize': False}
```

In [319]:

```
grid_cv(fourier)
```

```
R2 / variance :  0.6639939935277696
Residual sum of squares : 0.03
Best Params :  {'copy_X': True, 'fit_intercept': True, 'normalize': True}
```

Linear Regression - R2 variance. Lower is better.

> **Lower MSE is observed for Legendre Data**

Fit with a different model and a different p and Visualize Fit.

MODEL - SVR P = 20

In [89]:

```
from sklearn.svm import SVR
from sklearn.metrics import mean_squared_error
svm = SVR(gamma='auto')
```

In [303]:

```
p = 20
```

In [304]:

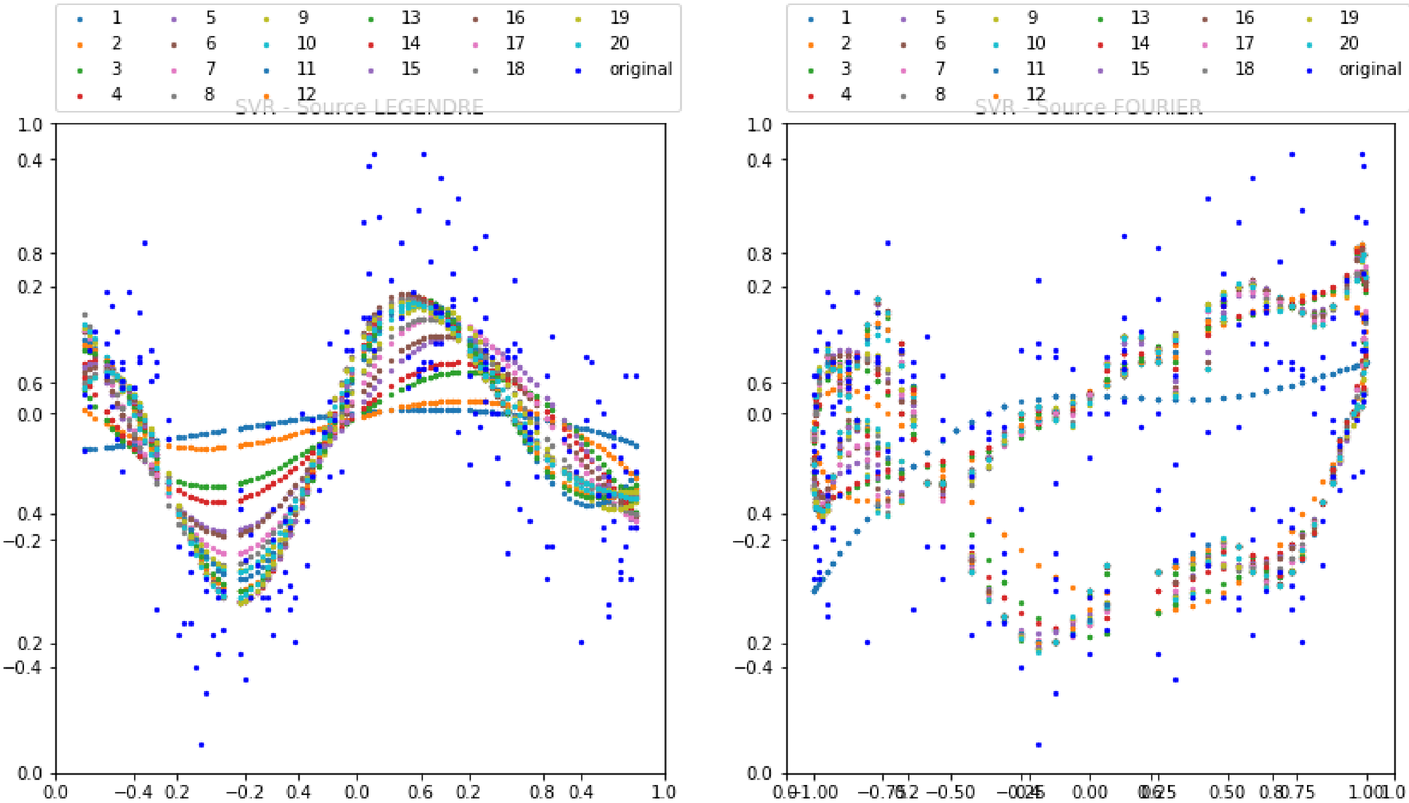
```
def find_ground_truth_using_SVR(feature,ax):
    ##ITERATE
    min_mse=0;
    score = 0
    mse = 0;
    arr_mse = [];
    dict = {};

    for j in range(1,p+1): #Iterate through Features
        mse_val = 0;

        X = feature(t,j+1)
        svr = svm.fit(X.T, y)
        yfit = svr.predict(X.T)

        score = svr.score(X.T,y)
        mse = mean_squared_error(y, yfit)
        #print("P:",j,"R-squared:", score,"MSE:", mean_squared_error(y, yfit))
        dict.update({mse:j})
        arr_mse.append(mse)
        ax.scatter(X.T[:,1], yfit,s=5, label=str(j))

    min_mse=min(arr_mse);
    print('Min MSE : ', min_mse, 'Features: ',dict.get(min_mse))
    ax.scatter(X.T[:,1], y,s=5, color="blue", label="original")
    ax.set_frame_on(False)
    ax.legend(bbox_to_anchor=(0,1.02),loc="lower left", borderaxespad=0.,ncol=6,fontsize
=10)

    return;
```

In [305]:

```
fig, ax = plt.subplots(1, 2, figsize=(12, 4))
fig.set_size_inches(14, 7, forward=True)
```

```
ax1 = fig.add_subplot(121)
ax2 = fig.add_subplot(122)

print('LEGENDRE')
ax1.set_title('SVR - Source LEGENDRE')
find_ground_truth_using_SVR(legendre,ax1)


print('FOURIER')
ax2.set_title('SVR - Source FOURIER')
find_ground_truth_using_SVR(fourier,ax2)
```

```
LEGENDRE
Min MSE :  0.01244189380268154 Features:  18
FOURIER
Min MSE :  0.009643013480620395 Features:  20
```



SVM Model MSE.

> **Lower is better. Lower MSE is observed for Fourier Data**

In [ ]: