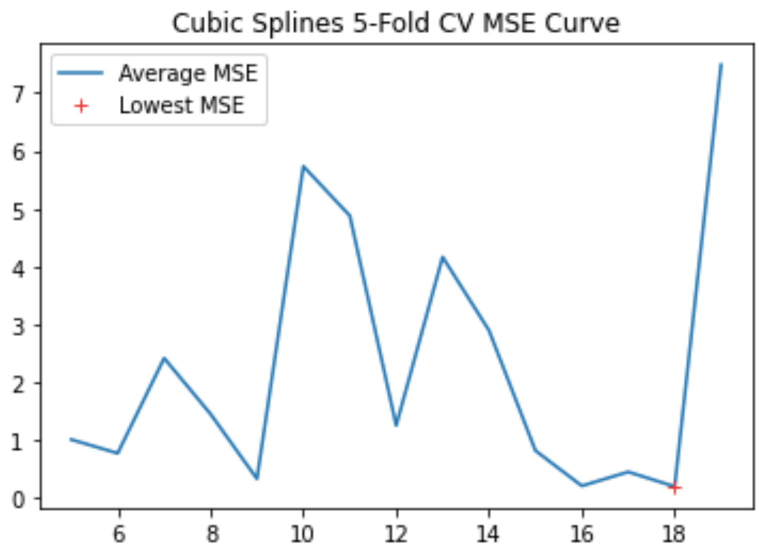
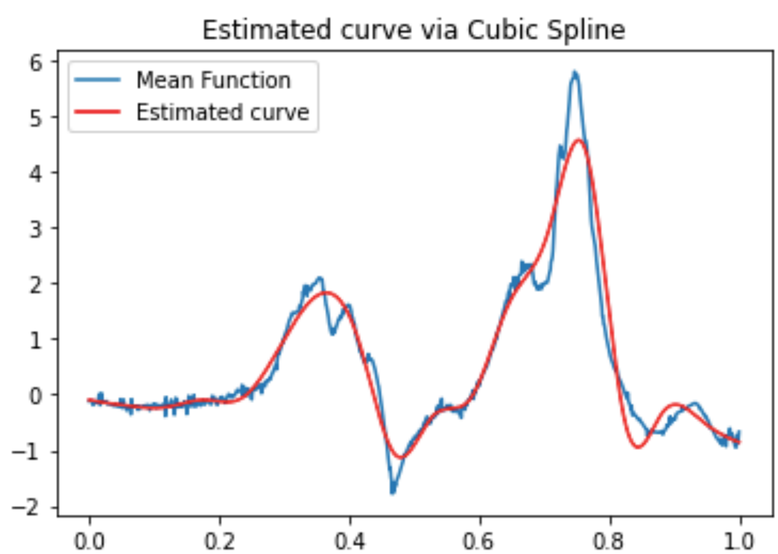
Q3

(a)

Optimal number of knots for Cubic Spline is 18, the corresponding MSE is 0.197.

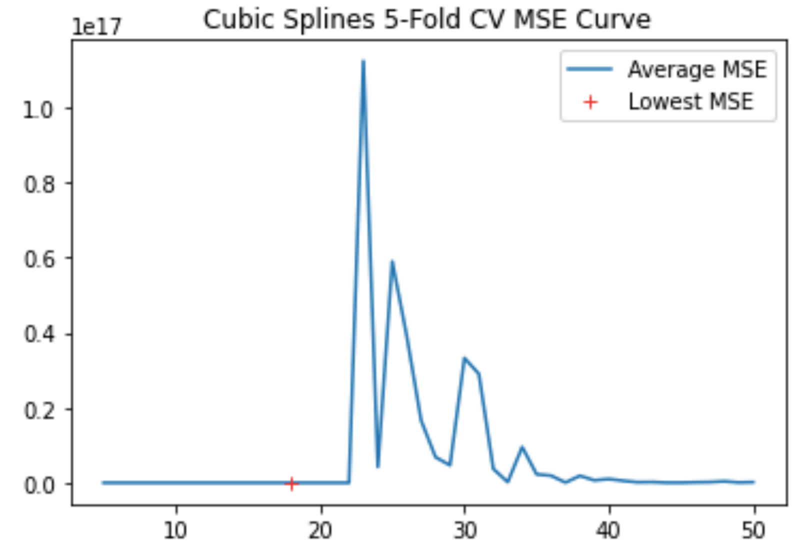
When I iterate number of knots from 5 to 19, the cross validation MSE curve and the fitted curve(along with the mean fucntion) are shown below.

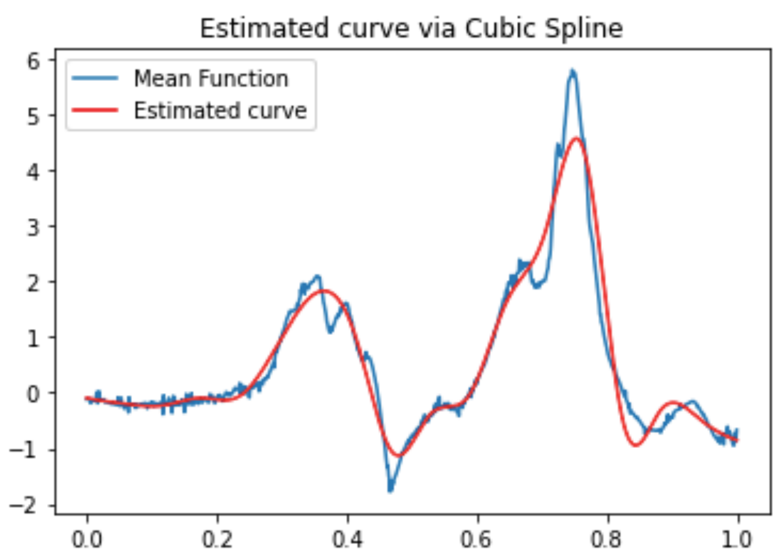





```
Optimal number of knots is 18
The corresponding CV MSE is 0.19710463598402977
```

When I iterate number of knots from 5 to 50, the cross validation MSE curve and the fitted curve(along with the mean fucntion) are shown below. Please scroll down to view the code or check out the code file uploaded separately.



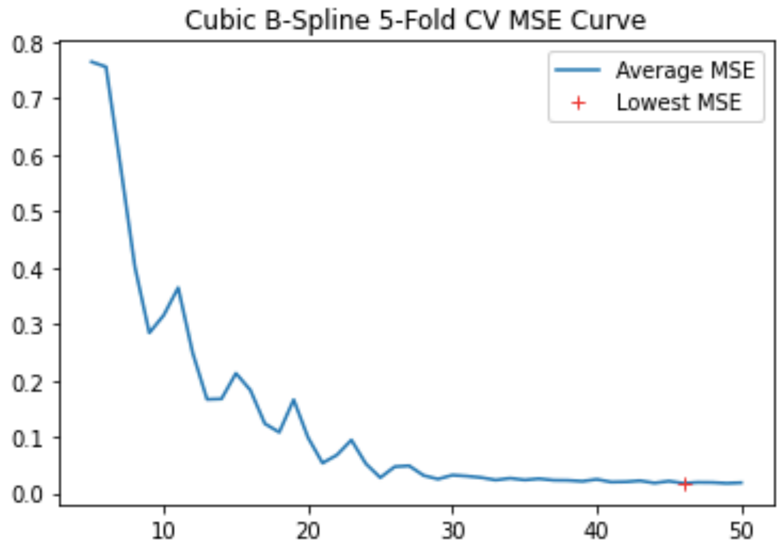


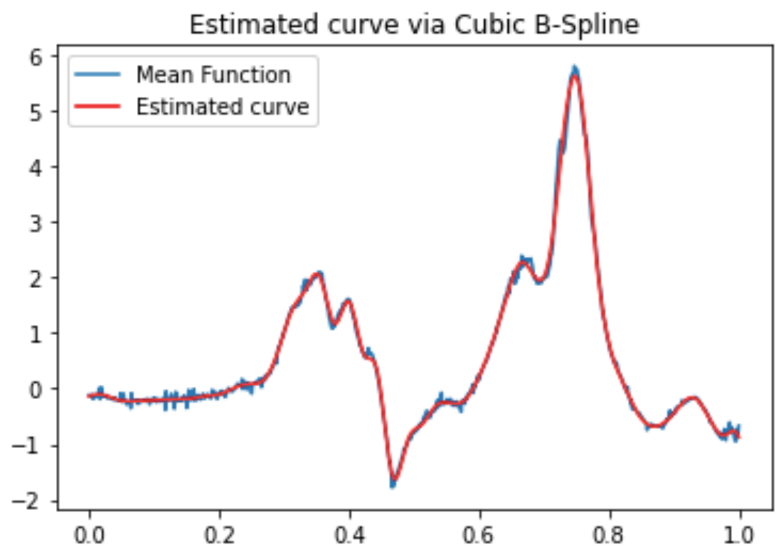

```
Optimal number of knots is 18
The corresponding CV MSE is 0.19710463598402977
```

(b)

Optimal number of knots for Cubic B-Spline is 46, the corresponding MSE is 0.0178.

Cross validation MSE curve and the fitted curve(along with the mean fucntion) are shown below. Please scroll down to view the code or check out the code file uploaded separately.

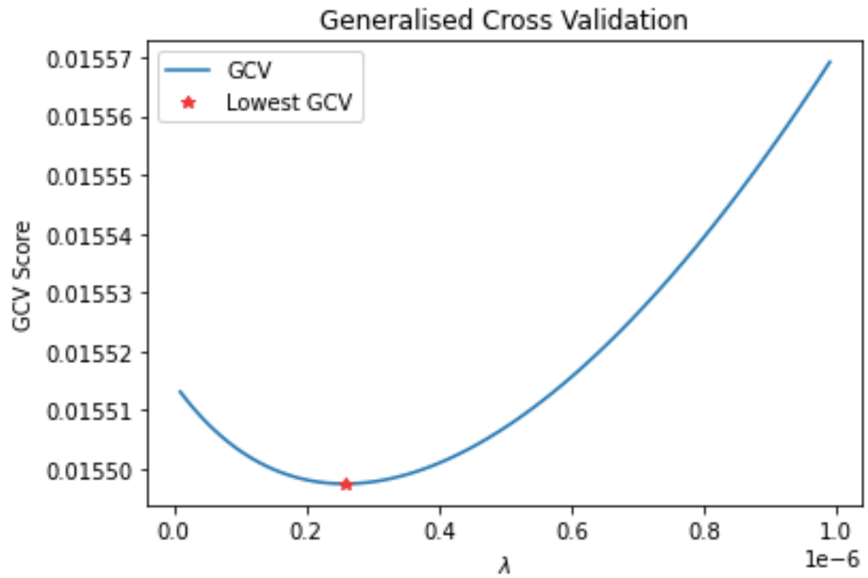


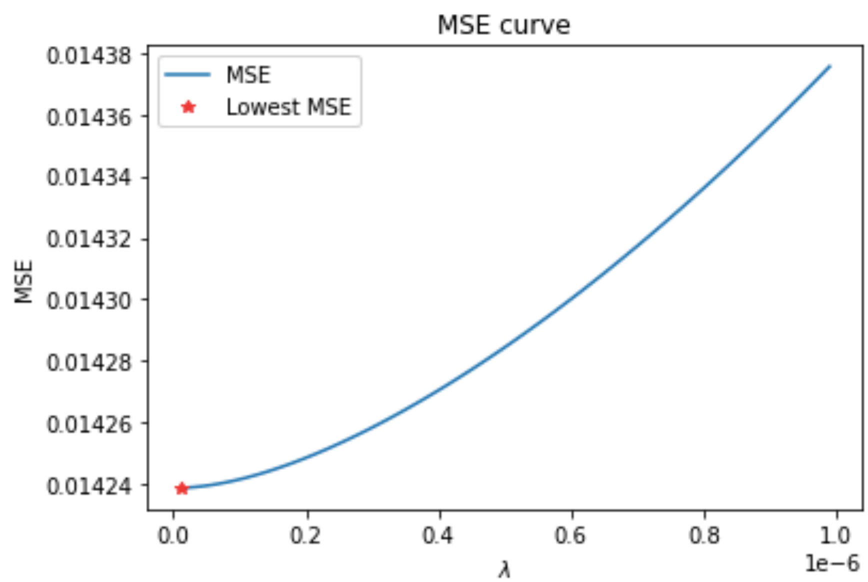



```
Optimal number of knots is 46
The corresponding CV MSE is 0.01780512925878195
```

(c)

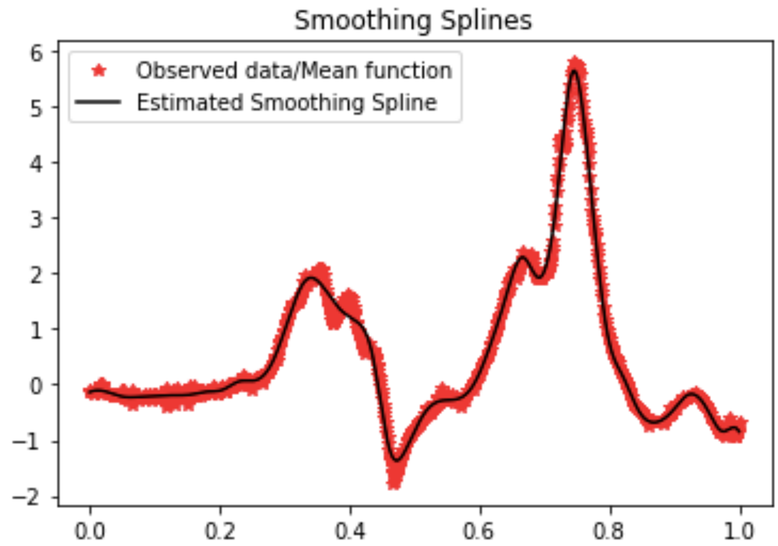
Optimal lambda for Smoothing Spline is 2.6e-07, and the corresponding GCV score is approximately 0.0155. The GCV curve and CV MSE curve are shown as follows.





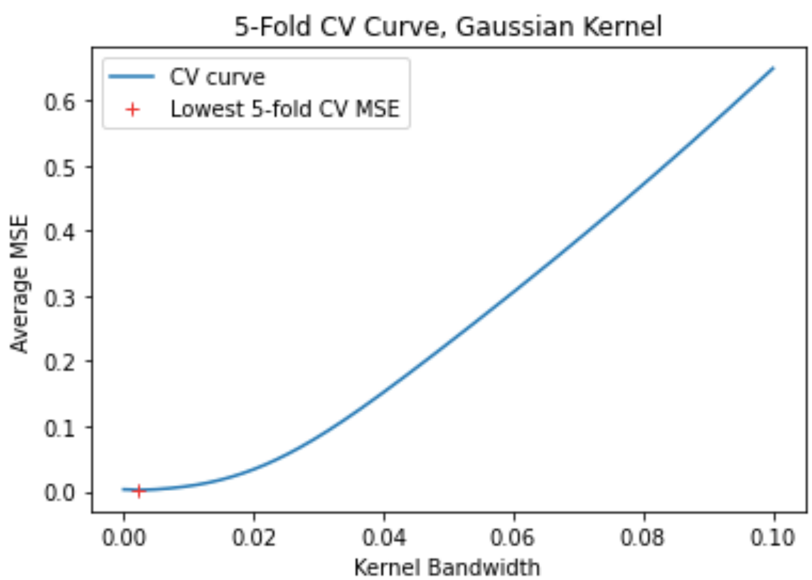

```
Optimal lambda is 2.6e-07
The corresponding GCV score is0.01549751340298802
```

The fitted curve via Smoothing Spline, along with the mean function is shown below.

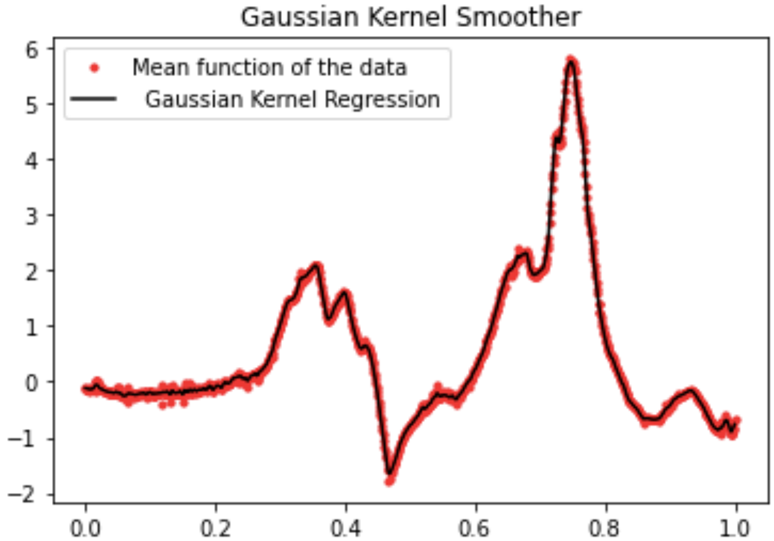


(d)

For the Gaussian kernel, the optimal bandwidth is 0.00228, the corresponding CV MSE is 0.0022. The CV MSE curve and the fitted curve, along with the mean function, are shown below.




```
Optimal bandwidth:0.0022800000000000003
CV MSE:0.00224218106868602
```


Gaussian Kernel Smoother
Mean function of the data
Gaussian Kernel Regression
6
5
4
3
2
1
0
−1
−2
0.0
0.2
0.4
0.6
0.8
1.0

In [170]:

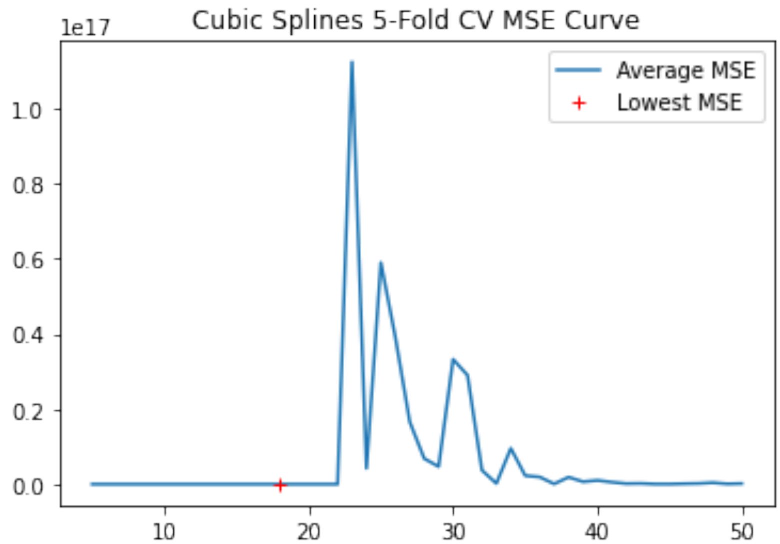
```
from sklearn.model_selection import KFold
from sklearn.model_selection import LeaveOneOut
import pandas as pd
import numpy as np
from scipy.interpolate import BSpline
import matplotlib.pyplot as plt
%matplotlib inline

#helper function: Gaussian kernel
def kerf(z): #Gaussian kernel
    return np.exp(-z**2)*(2*np.pi)**-0.5

#helper function: B-SPLINES
def BSplineBasis(x: np.array, knots: np.array, degree: int) -> np.arra
y:
    '''Return B-Spline basis. Python equivalent to bs in R or the spma
k/spval combination in MATLAB.
    This function acts like the R command bs(x,knots=knots,degree=degr
ee, intercept=False)
    Arguments:
        x: Points to evaluate spline on, sorted increasing
        knots: Spline knots, sorted increasing
        degree: Spline degree.
    Returns:
        B: Array of shape (x.shape[0], len(knots)+degree+1).
    Note that a spline has len(knots)+degree coefficients. However, be
cause the intercept is missing
    you will need to remove the last 2 columns. It's being kept this w
ay to retain compatibility with
    both the matlab spmak way and how R's bs works.

    If K = length(knots) (includes boundary knots)
    Mapping this to R's bs: (Props to Nate Bartlett )
    bs(x,knots,degree,intercept=T)[,2:K+degree] is same as BSplineBasi
s(x,knots,degree)[:,:-2]
    BF = bs(x,knots,degree,intercept=F) drops the first column so BF[,
1:K+degree] == BSplineBasis(x,knots,degree)[:,:-2]
    '''
    nKnots = knots.shape[0]
    lo = min(x[0], knots[0])
    hi = max(x[-1], knots[-1])
    augmented_knots = np.append(
        np.append([lo]*degree, knots), [hi]*degree)
    DOF = nKnots + degree +1 # DOF = K+M, M = degree+1
    spline = BSpline(augmented_knots, np.eye(DOF),
                     degree, extrapolate=False)
    B = spline(x)
    return B
```

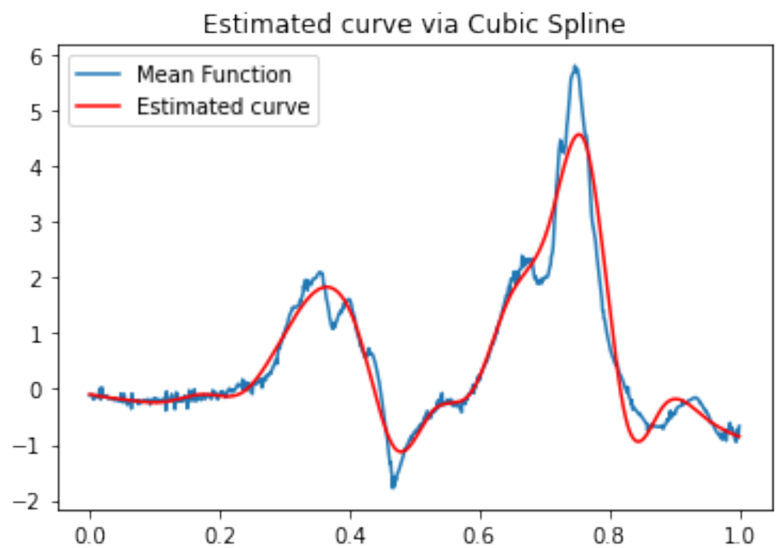
In [167]:

```
#Cubic Spline M=4
# Define knots and basis

data = pd.read_csv('Question3.csv',header=None).to_numpy()
mean_func = data.mean(0) #observed y
X = np.linspace(0,1,1000)

num_k = np.arange(5,51) #varying number of knots from 5 to 50
avg_mse = [] #track avg cv error from every number of knots
for i in num_k:
    kf = KFold(shuffle=True,random_state=25) #123,876543
    MSE = [] #MSEs from all five folds
    for tr,ts in kf.split(X):   #for every number of knots, calculate
avg mse from 5 eval folds
        k = np.linspace(np.min(X[tr]), np.max(X[tr]), i) #evenly sprea
d i knots
        H = []
        H.append(np.ones((X[tr].shape[0], 1)))
        H.append(X[tr].reshape(len(tr), -1))
        H.append(X[tr].reshape(len(tr), -1)**2)
        H.append(X[tr].reshape(len(tr), -1)**3)
        for kk in k:
            H.append(np.maximum((X[tr]-kk)**3, 0).reshape(len(tr), -1)
)
        H = np.hstack(H) #basis matrix of shape(len(tr),M+K)
        # Least square estimates
        # "Correct" way
        B_hat = np.linalg.lstsq(H, mean_func[tr])[0] #estimated coef f
rom the 4 folds
        #eval fold to get mse
        #basis matrix for eval fold
        k = np.linspace(np.min(X[ts]), np.max(X[ts]), i) #evenly sprea
d i knots
        H = []
        H.append(np.ones((X[ts].shape[0], 1)))
        H.append(X[ts].reshape(len(ts), -1))
        H.append(X[ts].reshape(len(ts), -1)**2)
        H.append(X[ts].reshape(len(ts), -1)**3)
        for kk in k:
            H.append(np.maximum((X[ts]-kk)**3, 0).reshape(len(ts), -1)
)
        H = np.hstack(H) #basis matrix of shape(len(ts),K+M)
        y_hat = H@B_hat #estimate for eval fold
        mse = np.mean((y_hat - mean_func[ts])**2)
        MSE.append(mse)
    avg_mse.append(np.mean(MSE))

op_num = num_k[np.argmin(avg_mse)]
min_mse = np.min(avg_mse)
```

```
#use optimal number of knots to estimate mean function
k = np.linspace(np.min(X), np.max(X), op_num) #evenly spread i knots
H = []
H.append(np.ones((X.shape[0], 1)))
H.append(X.reshape(1000, -1))
H.append(X.reshape(1000, -1)**2)
H.append(X.reshape(1000, -1)**3)
for kk in k:
    H.append(np.maximum((X-kk)**3, 0).reshape(1000, -1))
H = np.hstack(H) #basis matrix of shape(1000,K+M)
# Least square estimates
# "Correct" way
B_hat = np.linalg.lstsq(H, mean_func)[0]
y_hat = H@B_hat

plt.plot(num_k,avg_mse,label='Average MSE')
plt.plot(op_num,min_mse,'r+',label='Lowest MSE')
plt.title('Cubic Splines 5-Fold CV MSE Curve')
plt.legend()
plt.show()

plt.plot(X, mean_func, label='Mean Function')
plt.plot(X, y_hat, 'r', label='Estimated curve')
plt.title('Estimated curve via Cubic Spline')
plt.legend()
plt.show()

print(f'Optimal number of knots is {op_num}\nThe corresponding CV MSE is {min_mse}')
```

```
/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:25: FutureWarning: `rcond` parameter will change to the default of machine precision times ``max(M, N)`` where M and N are the input matrix dimensions.
To use the future default and silence this warning we advise to pass `rcond=None`, to keep using the old, explicitly pass `rcond=-1`.
/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:57: FutureWarning: `rcond` parameter will change to the default of machine precision times ``max(M, N)`` where M and N are the input matrix dimensions.
To use the future default and silence this warning we advise to pass `rcond=None`, to keep using the old, explicitly pass `rcond=-1`.
```






```
Optimal number of knots is 18
The corresponding CV MSE is 0.19710463598402977
```

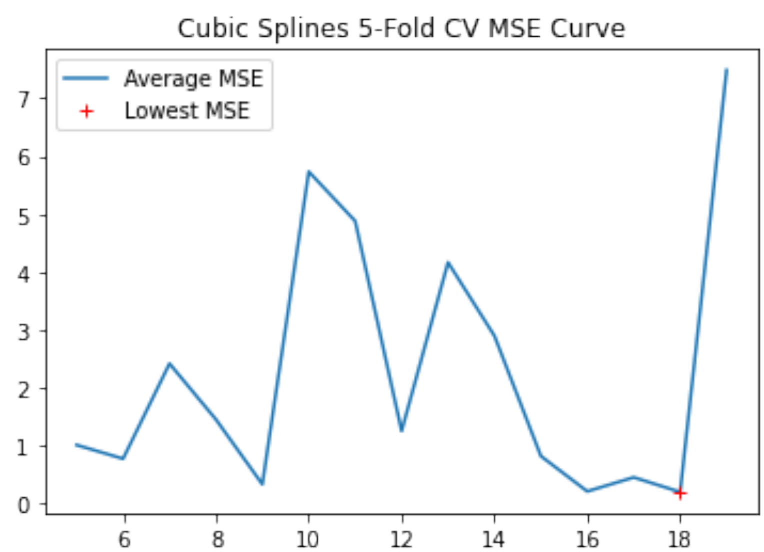
In [171]:

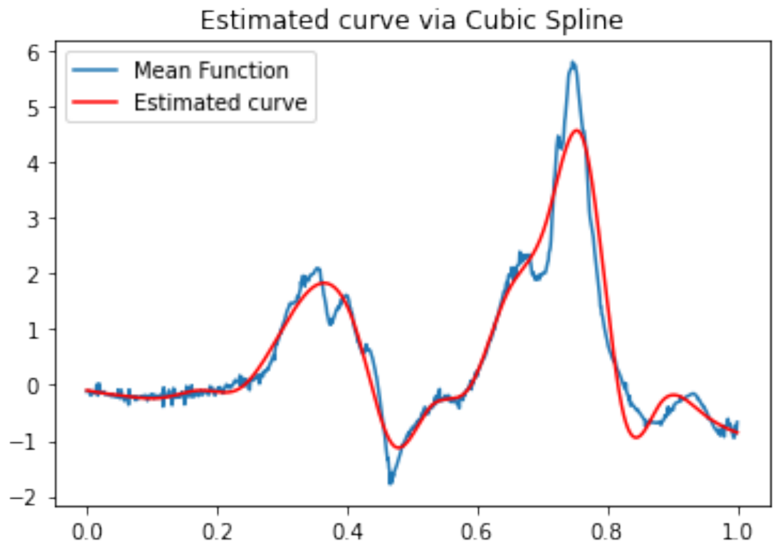
```
#Cubic Spline M=4
# Define knots and basis

data = pd.read_csv('Question3.csv',header=None).to_numpy()
mean_func = data.mean(0) #observed y
X = np.linspace(0,1,1000)

num_k = np.arange(5,20) #varying number of knots from 5 to 50
avg_mse = [] #track avg cv error from every number of knots
for i in num_k:
    kf = KFold(shuffle=True,random_state=25) #123,876543
    MSE = [] #MSEs from all five folds
    for tr,ts in kf.split(X):    #for every number of knots, calculate
avg mse from 5 eval folds
```

```
        k = np.linspace(np.min(X[tr]), np.max(X[tr]), i) #evenly sprea
d i knots
        H = []
        H.append(np.ones((X[tr].shape[0], 1)))
        H.append(X[tr].reshape(len(tr), -1))
        H.append(X[tr].reshape(len(tr), -1)**2)
        H.append(X[tr].reshape(len(tr), -1)**3)
        for kk in k:
            H.append(np.maximum((X[tr]-kk)**3, 0).reshape(len(tr), -1)
)
        H = np.hstack(H) #basis matrix of shape(len(tr),M+K)
        # Least square estimates
        # "Correct" way
        B_hat = np.linalg.lstsq(H, mean_func[tr])[0] #estimated coef f
rom the 4 folds
        #eval fold to get mse
        #basis matrix for eval fold
        k = np.linspace(np.min(X[ts]), np.max(X[ts]), i) #evenly sprea
d i knots
        H = []
        H.append(np.ones((X[ts].shape[0], 1)))
        H.append(X[ts].reshape(len(ts), -1))
        H.append(X[ts].reshape(len(ts), -1)**2)
        H.append(X[ts].reshape(len(ts), -1)**3)
        for kk in k:
            H.append(np.maximum((X[ts]-kk)**3, 0).reshape(len(ts), -1)
)
        H = np.hstack(H) #basis matrix of shape(len(ts),K+M)
        y_hat = H@B_hat #estimate for eval fold
        mse = np.mean((y_hat - mean_func[ts])**2)
        MSE.append(mse)
    avg_mse.append(np.mean(MSE))

op_num = num_k[np.argmin(avg_mse)]
min_mse = np.min(avg_mse)

#use optimal number of knots to estimate mean function
k = np.linspace(np.min(X), np.max(X), op_num) #evenly spread i knots
H = []
H.append(np.ones((X.shape[0], 1)))
H.append(X.reshape(1000, -1))
H.append(X.reshape(1000, -1)**2)
H.append(X.reshape(1000, -1)**3)
for kk in k:
    H.append(np.maximum((X-kk)**3, 0).reshape(1000, -1))
H = np.hstack(H) #basis matrix of shape(1000,K+M)
# Least square estimates
# "Correct" way
B_hat = np.linalg.lstsq(H, mean_func)[0]
y_hat = H@B_hat
```

```
plt.plot(num_k,avg_mse,label='Average MSE')
plt.plot(op_num,min_mse,'r+',label='Lowest MSE')
plt.title('Cubic Splines 5-Fold CV MSE Curve')
plt.legend()
plt.show()

plt.plot(X, mean_func, label='Mean Function')
plt.plot(X, y_hat, 'r', label='Estimated curve')
plt.title('Estimated curve via Cubic Spline')
plt.legend()
plt.show()

print(f'Optimal number of knots is {op_num}\nThe corresponding CV MSE
is {min_mse}')
```

```
/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:25: Fut
ureWarning: `rcond` parameter will change to the default of machine
precision times ``max(M, N)`` where M and N are the input matrix dim
ensions.
To use the future default and silence this warning we advise to pass
`rcond=None`, to keep using the old, explicitly pass `rcond=-1`.
/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:57: Fut
ureWarning: `rcond` parameter will change to the default of machine
precision times ``max(M, N)`` where M and N are the input matrix dim
ensions.
To use the future default and silence this warning we advise to pass
`rcond=None`, to keep using the old, explicitly pass `rcond=-1`.
```





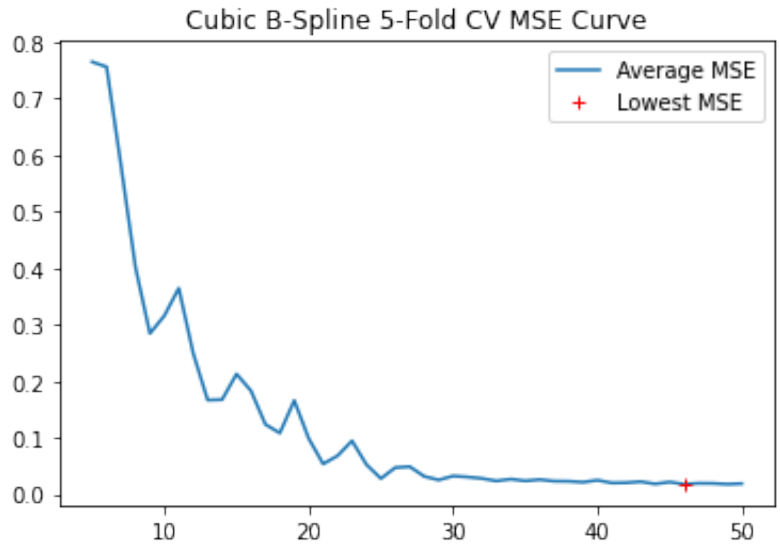

```
Optimal number of knots is 18
The corresponding CV MSE is 0.19710463598402977
```

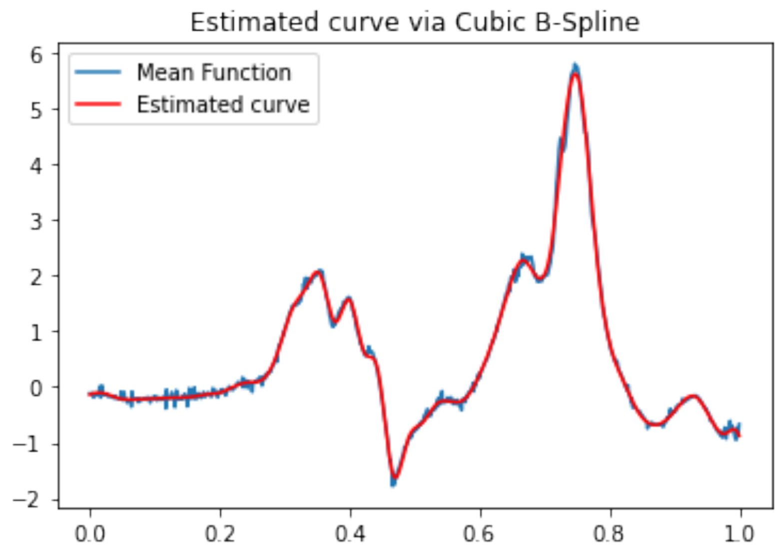
In [168]:

```
#Cubic B spline
data = pd.read_csv('Question3.csv',header=None).to_numpy()
mean_func = data.mean(0) #true y
X = np.linspace(0,1,1000)

num_k = np.arange(5,51) #varying number of knots from 5 to 50
avg_mse = [] #track avg cv error from every number of knots
for i in num_k:
    kf = KFold(shuffle=True,random_state=25) #123,876543
    MSE = [] #MSEs from all five folds
    for tr,ts in kf.split(X):   #for every number of knots, calculate avg mse from 5 eval folds
        k = np.linspace(np.min(X[tr]), np.max(X[tr]), i) #evenly spread i knots
        H = BSplineBasis(X[tr], k, 3)[:,:-2]#basis matrix of shape(len(tr),K+2m-M,where m is num of augmented knots on each side))
        # Least square estimates
        # "Correct" way
        B_hat = np.linalg.lstsq(H, mean_func[tr])[0] #estimated coef from the 4 folds
        #eval fold to get mse
        #basis matrix for eval fold
        k = np.linspace(np.min(X[ts]), np.max(X[ts]), i) #evenly spread i knots
        H = BSplineBasis(X[ts], k, 3)[:,:-2]
        y_hat = H@B_hat #estimate for eval fold
        mse = np.mean((y_hat - mean_func[ts])**2)
        MSE.append(mse)
    avg_mse.append(np.mean(MSE))
```

```
op_num = num_k[np.argmin(avg_mse)]
min_mse = np.min(avg_mse)

#use optimal number of knots to estimate mean function
k = np.linspace(np.min(X), np.max(X), op_num) #evenly spread i knots
H = BSplineBasis(X, k, 3)[:,:-2]
B_hat = np.linalg.lstsq(H, mean_func)[0]
y_hat = H@B_hat

plt.plot(num_k,avg_mse,label='Average MSE')
plt.plot(op_num,min_mse,'r+',label='Lowest MSE')
plt.title('Cubic B-Spline 5-Fold CV MSE Curve')
plt.legend()
plt.show()

plt.plot(X, mean_func, label='Mean Function')
plt.plot(X, y_hat, 'r', label='Estimated curve')
plt.title('Estimated curve via Cubic B-Spline')
plt.legend()
plt.show()

print(f'Optimal number of knots is {op_num}\nThe corresponding CV MSE 
is {min_mse}')
```

```
/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:16: Fut
ureWarning: `rcond` parameter will change to the default of machine
precision times ``max(M, N)`` where M and N are the input matrix dim
ensions.
To use the future default and silence this warning we advise to pass
`rcond=None`, to keep using the old, explicitly pass `rcond=-1`.
  app.launch_new_instance()
/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:32: Fut
ureWarning: `rcond` parameter will change to the default of machine
precision times ``max(M, N)`` where M and N are the input matrix dim
ensions.
To use the future default and silence this warning we advise to pass
`rcond=None`, to keep using the old, explicitly pass `rcond=-1`.
```






```
Optimal number of knots is 46
The corresponding CV MSE is 0.01780512925878195
```

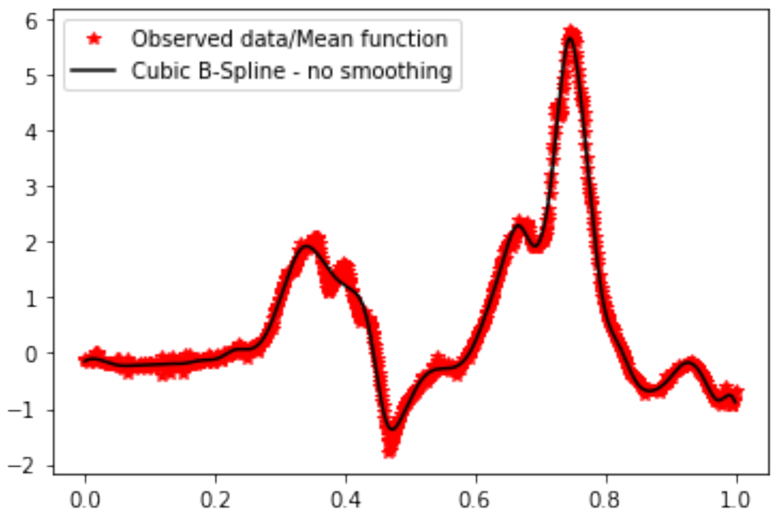
In [169]:

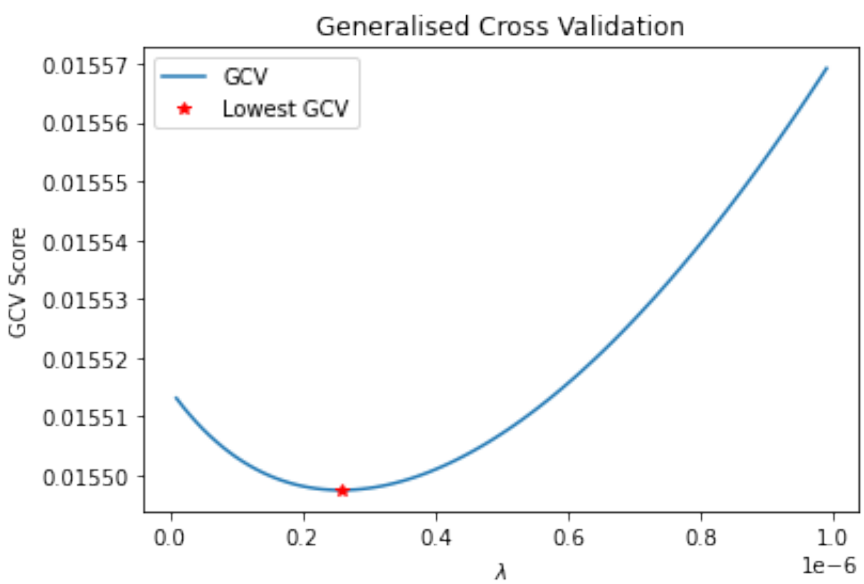
```
#Smoothing spline
#lams = np.arange(0.00000001, 0.000001,0.00000001)
data = pd.read_csv('Question3.csv',header=None).to_numpy()
mean_func = data.mean(0) #true y
n = 1000 #number of explanatory variables
D = np.linspace(0, 1, n)
k = 40
# Generate B-spline basis:
knots = np.linspace(0, 1, k)
B = BSplineBasis(D, knots, 3)[:,:-2]
# Least Square Estimation:
yhat = B@np.linalg.lstsq(B, mean_func)[0]

plt.plot(D, mean_func, 'r*', label='Observed data/Mean function')
```

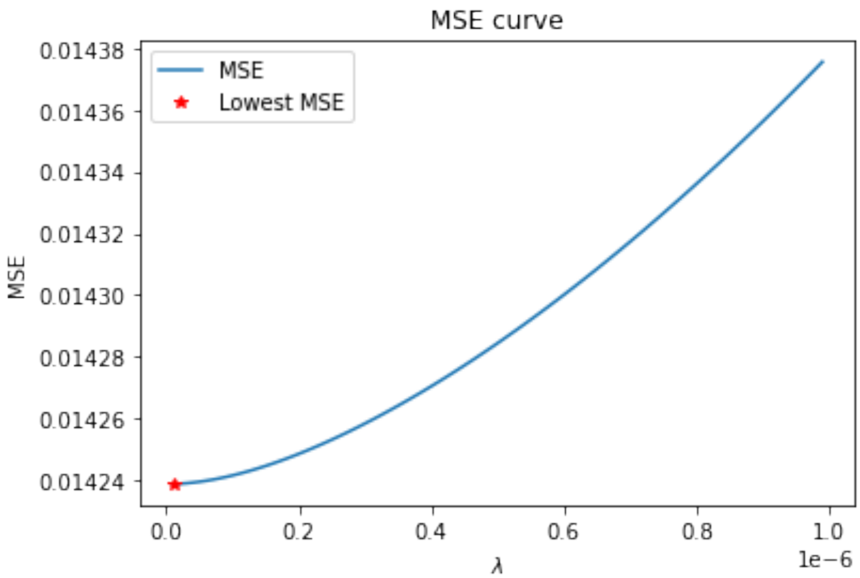
```
plt.plot(D, yhat, 'k-', label='Cubic B-Spline - no smoothing')
plt.legend()
plt.show()

# Smoothing Penalty
# Different lambda selection
B2 = np.diff(B, axis=0, n=2)*(n-1)**2  # Numerical derivative
omega = B2.T.dot(B2)/(n-2)
lams = np.arange(0.00000001, 0.000001,0.00000001)
p = len(lams)
MSE = []
RSS = []
df = []
for lam in lams:
    S = B@np.linalg.inv(B.T@B+lam*omega)@B.T  # Not great but we still
need to get the trace of S
    yhat = S.dot(mean_func)
    MSE.append(((yhat-mean_func)**2).mean())
    RSS.append(((yhat-mean_func)**2).sum())
    df.append(np.trace(S))
RSS = np.array(RSS)
df = np.array(df)
# GCV criterion
GCV = (RSS/n)/(1-df/n)**2
i = np.argmin(GCV)
m = GCV[i]
plt.plot(lams, GCV, label='GCV')
plt.plot(lams[i], m, 'r*', label='Lowest GCV')
plt.title("Generalised Cross Validation")
plt.xlabel('$\lambda$')
plt.ylabel('GCV Score')
plt.legend()
plt.show()

plt.plot(lams,MSE,label='MSE')
plt.plot(lams[np.argmin(MSE)], np.min(MSE), 'r*', label='Lowest MSE')
plt.title("MSE curve")
plt.xlabel('$\lambda$')
plt.ylabel('MSE')
plt.legend()
plt.show()

print(f'Optimal lambda is {lams[i]}\nThe corresponding GCV score is{m}
')

S = B@np.linalg.inv(B.T@B+lams[i]*omega)@B.T
yhat = S@mean_func
plt.plot(D, mean_func, 'r*', label='Observed data/Mean function')
plt.plot(D, yhat, 'k-', label='Estimated Smoothing Spline')
plt.legend()
```

```
plt.title("Smoothing Splines")
plt.show()
```

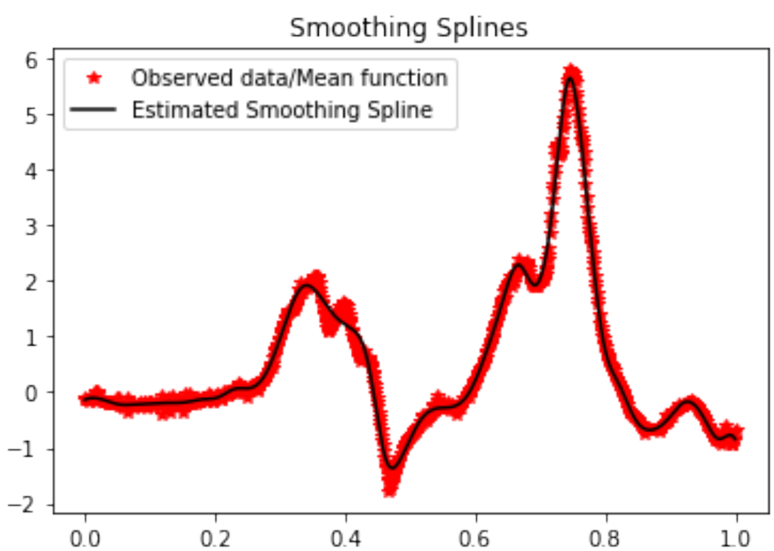
```
/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:12: Fut
ureWarning: `rcond` parameter will change to the default of machine
precision times ``max(M, N)`` where M and N are the input matrix dim
ensions.
To use the future default and silence this warning we advise to pass
`rcond=None`, to keep using the old, explicitly pass `rcond=-1`.
  if sys.path[0] == '':
```





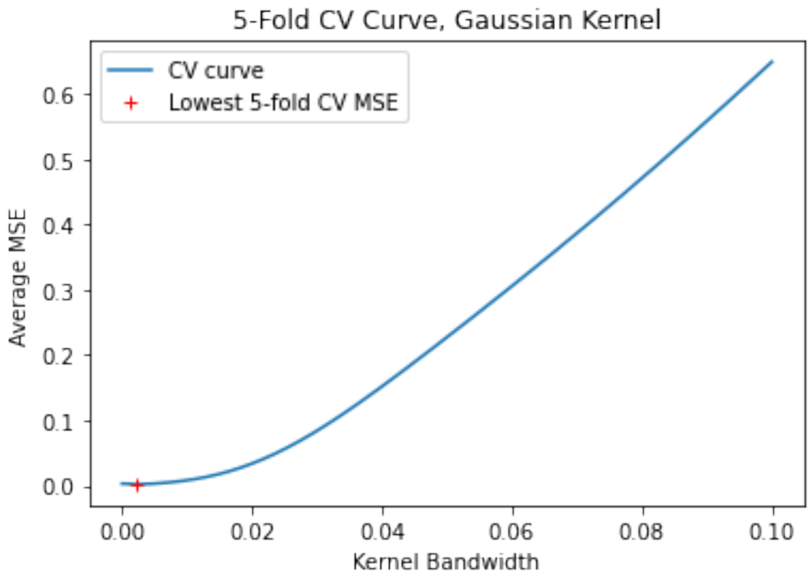



```
Optimal lambda is 2.6e-07
The corresponding GCV score is0.01549751340298802
```



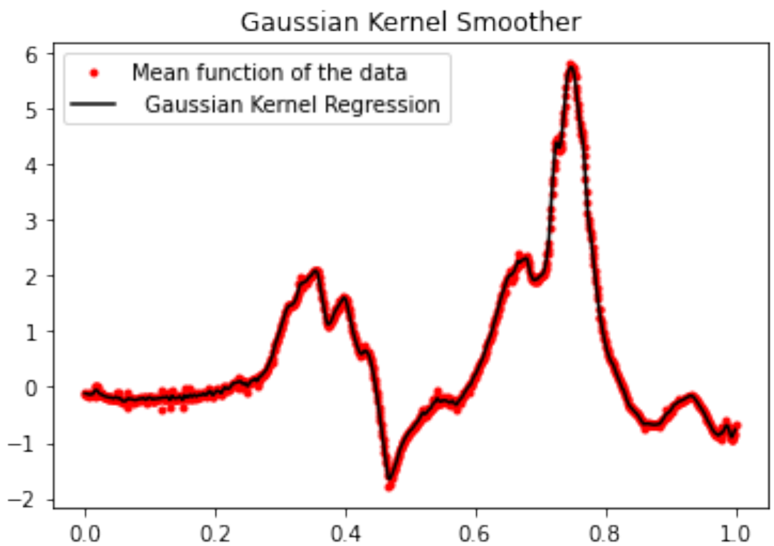
In [120]:

```
#Gaussian kernel
data = pd.read_csv('Question3.csv',header=None).to_numpy()
mean_func = data.mean(0) #true y
x = np.linspace(0,1,1000)

#5 fold cv
bandwidths = np.arange(0.00008,0.1,0.0001)
MSEs = [] #track average MSE of each bandwidth
for w in bandwidths:
    MSE = []
    kf = KFold(shuffle=True,random_state=20) #20,25,99,39
    for tr,ts in kf.split(x):
        err = 0 #track rss for one evaluation fold
        for i in ts:
            k = kerf((x[tr]-x[i])/w)
            y_hat = np.average(mean_func[tr],weights=k)
            err += (y_hat - mean_func[i])**2
        mse = err/len(ts) #get mse for one eval fold
        MSE.append(mse)
    MSEs.append(np.mean(MSE))

w_star = bandwidths[np.argmin(MSEs)]

plt.plot(bandwidths,MSEs,label='CV curve')
plt.plot(w_star, min(MSEs),'r+', label='Lowest 5-fold CV MSE')
plt.title('5-Fold CV Curve, Gaussian Kernel')
plt.xlabel('Kernel Bandwidth')
plt.ylabel('Average MSE')
plt.legend()
plt.show()
print(f'Optimal bandwidth:{w_star}\nCV MSE:{np.min(MSEs)}')
```




```
Optimal bandwidth:0.0022800000000000003
CV MSE:0.00224218106868602
```

In [122]:
```
# Interpolation for N values
N = 1000
xx = np.linspace(min(x), max(x), N)
yy = []
for xx_ in xx:
    z = kerf((xx_-x)/w_star)
    yy.append(np.average(mean_func, weights=z))

plt.plot(x, mean_func, 'r.', label='Mean function of the data')
plt.plot(xx, yy, 'k', label='  Gaussian Kernel Regression')
plt.legend()
plt.title('Gaussian Kernel Smoother')
plt.show()
```



In [ ]:

In [ ]:

In [ ]: