Question 4

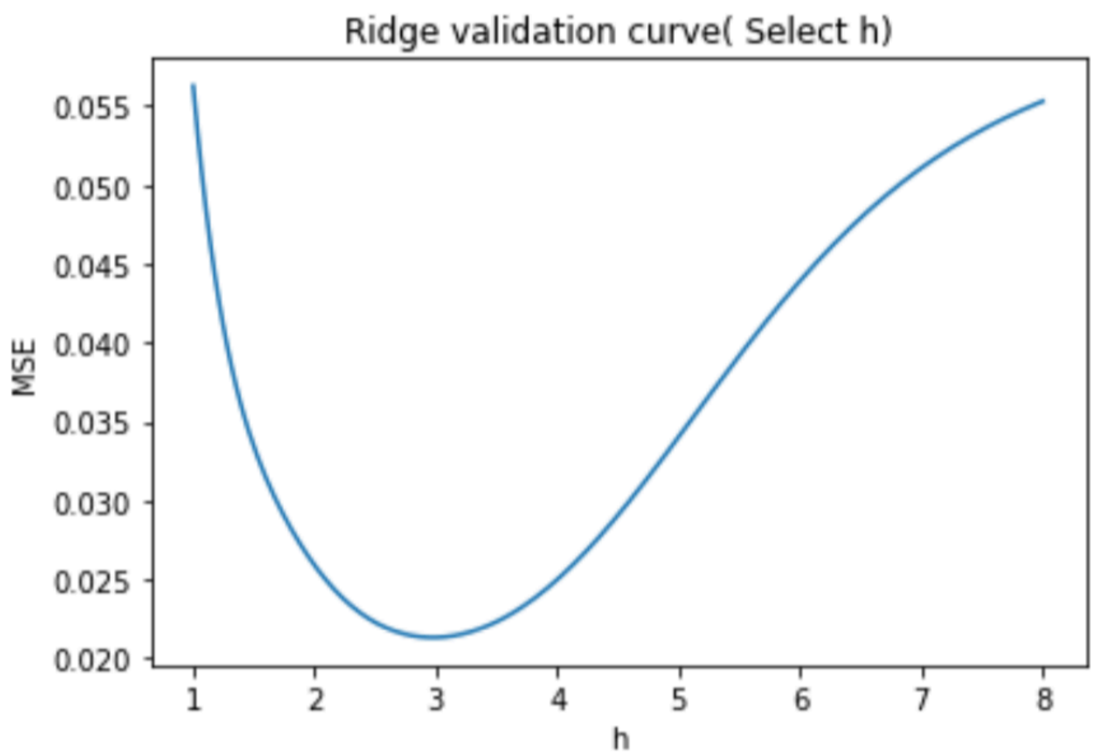
Standardizing both X and y I got results as follows.

The magnitude of MSE changes if I left response variable y unscaled.

Please see the code for details.

(a)

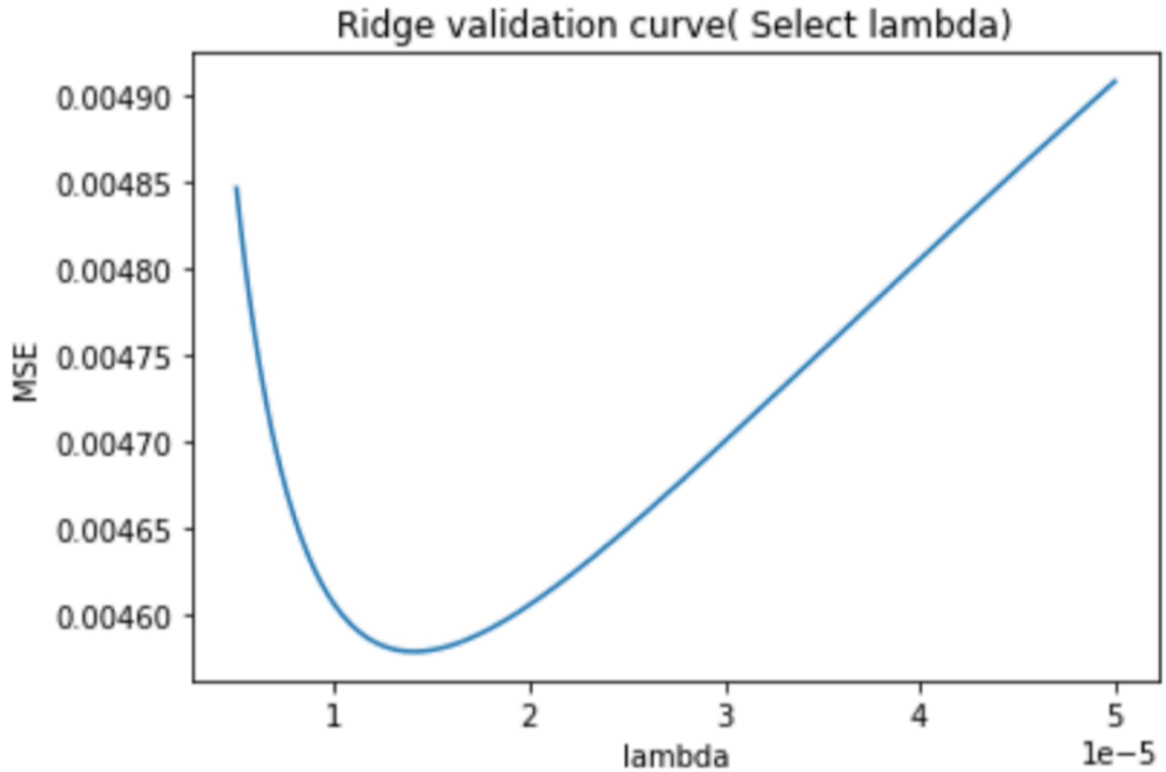
The optimal kernel bandwidth h is 2.969849246231156 and the corresponding MSE is 0.021320227689634673.




```
Optimal h is 2.969849246231156
Optimal MSE is 0.021320227689634673
```

(b)

The optimal lambda is 1.4045226130653267e-05 and the corresponding MSE is 0.004579308935070984.




```
Optimal lambda is 1.4045226130653267e-05
Optimal MSE is 0.004579308935070984
```

(c)

The mean-squared error on the test set is 0.00492169562453762

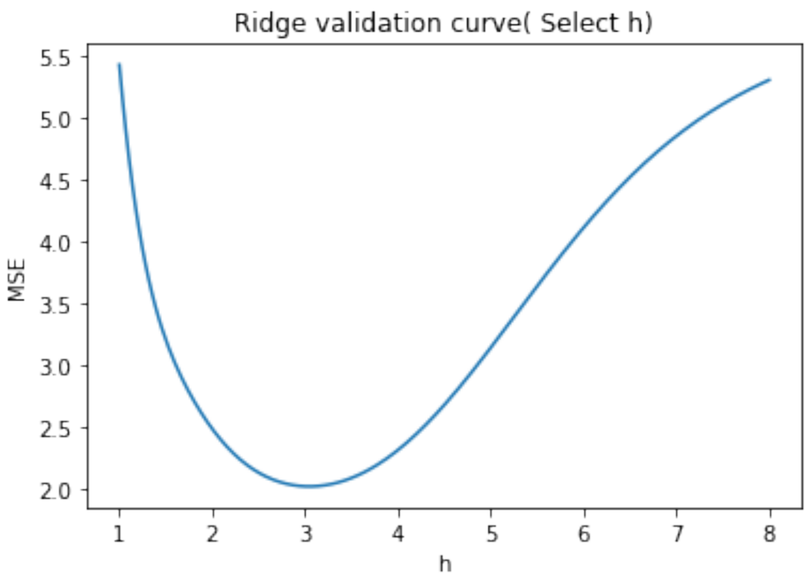
```
MSE on test is 0.00492169562453762
```

In [53]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import Ridge
from scipy.spatial.distance import cdist
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
```

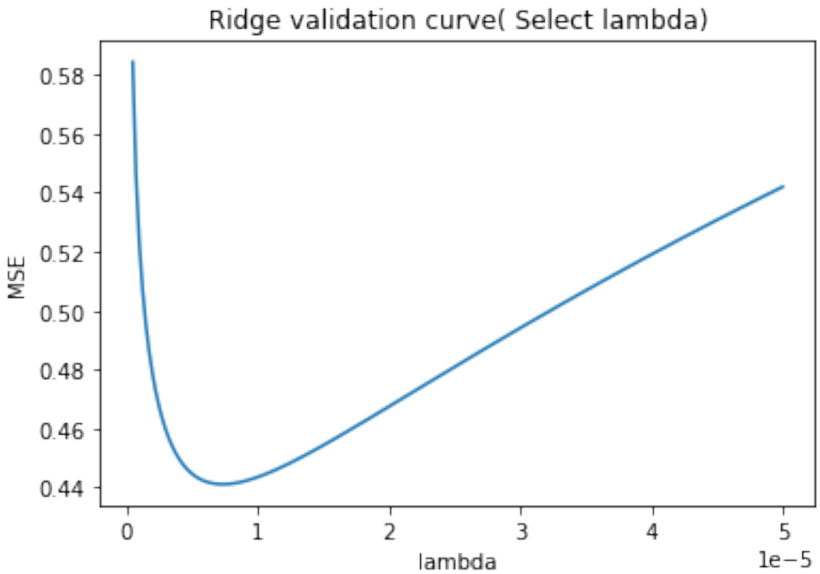
In [50]:
```
data = pd.read_csv('Question4-3.csv',header=None).to_numpy()

X_train, y_train = data[:600,:8], data[:600,8][:,np.newaxis]
```

```
X_test, y_test = data[600:,:8], data[600:,8][:,np.newaxis]

scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

X_tr, X_val, y_tr, y_val = train_test_split(
    X_train, y_train, test_size=0.2, random_state=42)

#r2 score on standardized X and y
#beta = np.linalg.lstsq(X_tr,y_tr,rcond=None)[0]
#y_pred = X_tr@beta
#print(r2_score(y_tr, y_pred))

H = np.linspace(1,8,200)
lam = 0.01
mse = []
for h in H:
    reg = Ridge(alpha=lam)
    kernel1 = np.exp(-cdist(X_tr,X_tr)**2 / h) #train kernel
    kernel2 = np.exp(-cdist(X_tr,X_val)**2 / h) # val kernel
    reg.fit(kernel1,y_tr)
    yhat = reg.predict(kernel2.T)
    mse.append(mean_squared_error(y_val,yhat))

plt.figure()
plt.plot(H,mse)
plt.xlabel('h')
plt.ylabel('MSE')
plt.title('Ridge validation curve( Select h)')
plt.show()
op_h = H[np.argmin(mse)]
print(f'Optimal h is {op_h}')
print(f'Optimal MSE is {mse[np.argmin(mse)]}')

X_tr, X_val, y_tr, y_val = train_test_split(
    X_train, y_train, test_size=0.2, random_state=69)#99

kernel0 = np.exp(-cdist(X_train,X_train)**2 / op_h) #train kernel
kernel1 = np.exp(-cdist(X_tr,X_tr)**2 / op_h) #train kernel
kernel2 = np.exp(-cdist(X_tr,X_val)**2 / op_h) # val kernel
kernel3 = np.exp(-cdist(X_train,X_test)**2 / op_h) # test kernel

lam = np.linspace(0.0000005,0.00005,200)
mse = []
for l in lam:
    reg = Ridge(alpha=l)
    reg.fit(kernel1,y_tr)
    yhat = reg.predict(kernel2.T)
    mse.append(mean_squared_error(y_val,yhat))
```

```
plt.figure()
plt.plot(lam,mse)
plt.xlabel('lambda')
plt.ylabel('MSE')
plt.title('Ridge validation curve( Select lambda)')
plt.show()
op_l = lam[np.argmin(mse)]
print(f'Optimal lambda is {op_l}')
print(f'Optimal MSE is {mse[np.argmin(mse)]}')

reg = Ridge(alpha=op_l)
reg.fit(kernel0,y_train)
yhat = reg.predict(kernel3.T)
print('')
print(f'MSE on test is {mean_squared_error(y_test,yhat)}')
```




```
Optimal h is 3.040201005025126
Optimal MSE is 2.0188514477860084
```



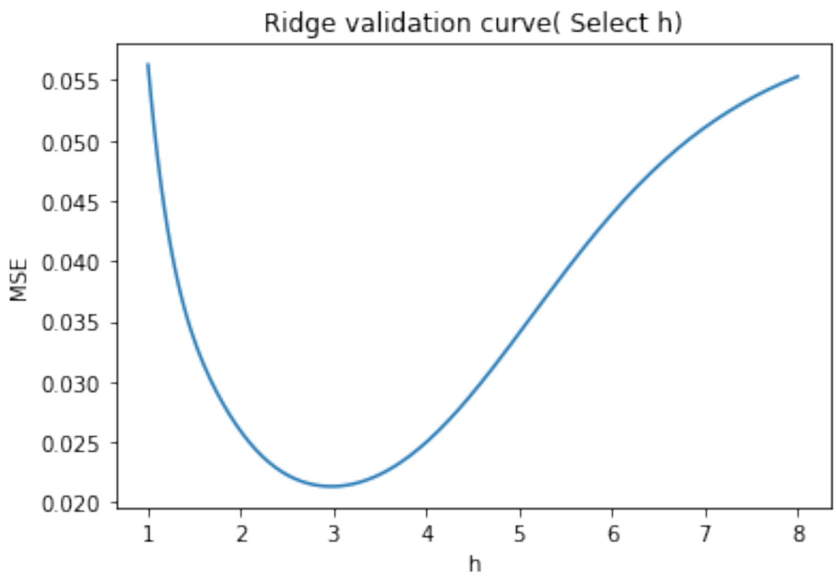

```
Optimal lambda is 7.464824120603017e-06
Optimal MSE is 0.4411426749398487

MSE on test is 0.32426427910524225
```

In [46]:

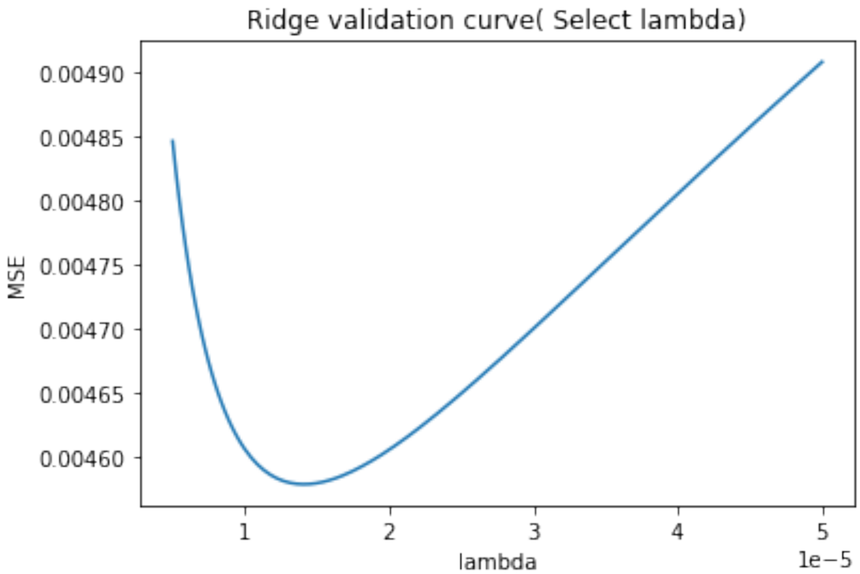
```
data = pd.read_csv('Question4-3.csv',header=None).to_numpy()

X_train, y_train = data[:600,:8], data[:600,8]
X_test, y_test = data[600:,:8], data[600:,8]

X_tr, X_val, y_tr, y_val = train_test_split(
    X_train, y_train, test_size=0.2, random_state=42)
y_tr = y_tr[:,np.newaxis]
y_val = y_val[:,np.newaxis]
y_test = y_test[:,np.newaxis]

scaler = StandardScaler()
X_tr = scaler.fit_transform(X_tr)
X_val = scaler.transform(X_val)
X_test = scaler.transform(X_test)

scaler = StandardScaler()
y_tr = scaler.fit_transform(y_tr)
y_val = scaler.transform(y_val)
y_test = scaler.transform(y_test)

#r2 score on standardized X and y
#beta = np.linalg.lstsq(X_tr,y_tr,rcond=None)[0]
#y_pred = X_tr@beta
#print(r2_score(y_tr, y_pred))

H = np.linspace(1,8,200)
```

```
lam = 0.01
mse = []
for h in H:
    reg = Ridge(alpha=lam)
    kernel1 = np.exp(-cdist(X_tr,X_tr)**2 / h) #train kernel
    kernel2 = np.exp(-cdist(X_tr,X_val)**2 / h) # val kernel
    reg.fit(kernel1,y_tr)
    yhat = reg.predict(kernel2.T)
    mse.append(mean_squared_error(y_val,yhat))

plt.figure()
plt.plot(H,mse)
plt.xlabel('h')
plt.ylabel('MSE')
plt.title('Ridge validation curve( Select h)')
plt.show()
op_h = H[np.argmin(mse)]
print(f'Optimal h is {op_h}')
print(f'Optimal MSE is {mse[np.argmin(mse)]}')

X_train, y_train = data[:600,:8], data[:600,8]
X_test, y_test = data[600:,:8], data[600:,8]
X_tr, X_val, y_tr, y_val = train_test_split(
    X_train, y_train, test_size=0.2, random_state=99)
y_tr = y_tr[:,np.newaxis]
y_val = y_val[:,np.newaxis]
y_test = y_test[:,np.newaxis]

scaler = StandardScaler()
X_tr = scaler.fit_transform(X_tr)
X_val = scaler.transform(X_val)
X_test = scaler.transform(X_test)

scaler = StandardScaler()
y_tr = scaler.fit_transform(y_tr)
y_val = scaler.transform(y_val)
y_test = scaler.transform(y_test)

kernel1 = np.exp(-cdist(X_tr,X_tr)**2 / op_h) #train kernel
kernel2 = np.exp(-cdist(X_tr,X_val)**2 / op_h) # val kernel
kernel3 = np.exp(-cdist(X_tr,X_test)**2 / op_h) # test kernel

lam = np.linspace(0.000005,0.00005,200)
mse = []
for l in lam:
    reg = Ridge(alpha=l)
    reg.fit(kernel1,y_tr)
    yhat = reg.predict(kernel2.T)
    mse.append(mean_squared_error(y_val,yhat))
```

```
plt.figure()
plt.plot(lam,mse)
plt.xlabel('lambda')
plt.ylabel('MSE')
plt.title('Ridge validation curve( Select lambda)')
plt.show()
op_l = lam[np.argmin(mse)]
print(f'Optimal lambda is {op_l}')
print(f'Optimal MSE is {mse[np.argmin(mse)]}')

reg = Ridge(alpha=op_l)
reg.fit(kernel1,y_tr)
yhat = reg.predict(kernel3.T)
print('')
print(f'MSE on test is {mean_squared_error(y_test,yhat)}')
```




```
Optimal h is 2.969849246231156
Optimal MSE is 0.021320227689634673
```

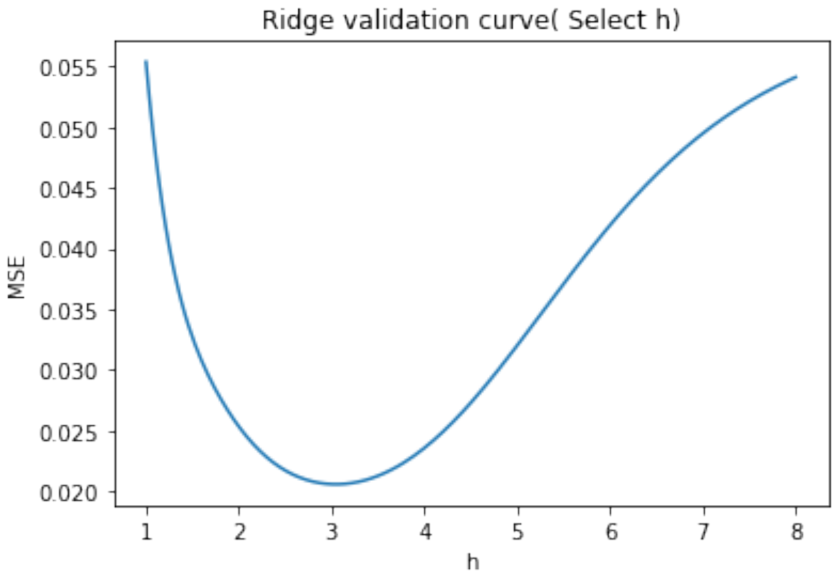



```
Optimal lambda is 1.4045226130653267e-05
Optimal MSE is 0.004579308935070984

MSE on test is 0.00492169562453762
```

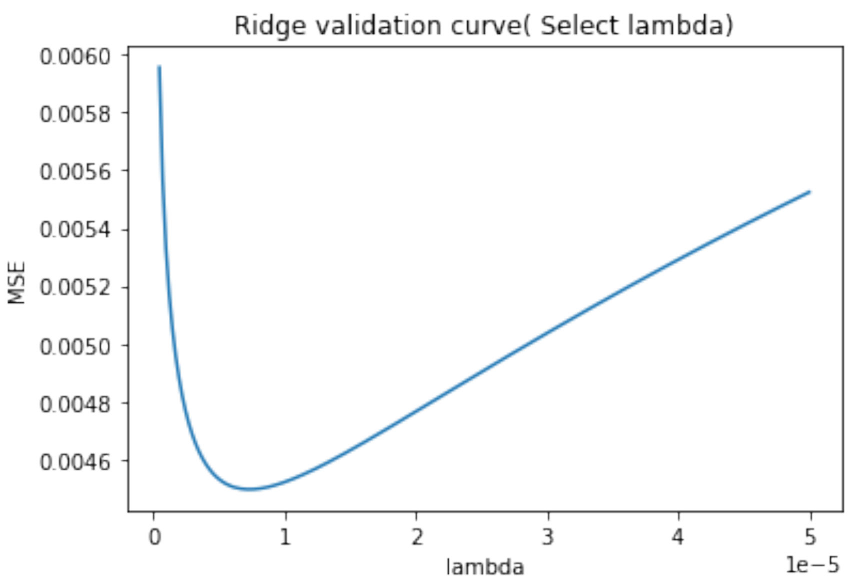
In [48]:

```python
data = pd.read_csv('Question4-3.csv',header=None).to_numpy()

X_train, y_train = data[:600,:8], data[:600,8][:,np.newaxis]
X_test, y_test = data[600:,:8], data[600:,8][:,np.newaxis]

scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

scaler = StandardScaler()
y_train = scaler.fit_transform(y_train)
y_test = scaler.transform(y_test)

X_tr, X_val, y_tr, y_val = train_test_split(
    X_train, y_train, test_size=0.2, random_state=42)

#r2 score on standardized X and y
#beta = np.linalg.lstsq(X_tr,y_tr,rcond=None)[0]
#y_pred = X_tr@beta
#print(r2_score(y_tr, y_pred))

H = np.linspace(1,8,200)
lam = 0.01
mse = []
for h in H:
    reg = Ridge(alpha=lam)
    kernel1 = np.exp(-cdist(X_tr,X_tr)**2 / h) #train kernel
```

```
    kernel2 = np.exp(-cdist(X_tr,X_val)**2 / h) # val kernel
    reg.fit(kernel1,y_tr)
    yhat = reg.predict(kernel2.T)
    mse.append(mean_squared_error(y_val,yhat))

plt.figure()
plt.plot(H,mse)
plt.xlabel('h')
plt.ylabel('MSE')
plt.title('Ridge validation curve( Select h)')
plt.show()
op_h = H[np.argmin(mse)]
print(f'Optimal h is {op_h}')
print(f'Optimal MSE is {mse[np.argmin(mse)]}')

X_tr, X_val, y_tr, y_val = train_test_split(
    X_train, y_train, test_size=0.2, random_state=69)#99

kernel0 = np.exp(-cdist(X_train,X_train)**2 / op_h) #train kernel
kernel1 = np.exp(-cdist(X_tr,X_tr)**2 / op_h) #train kernel
kernel2 = np.exp(-cdist(X_tr,X_val)**2 / op_h) # val kernel
kernel3 = np.exp(-cdist(X_train,X_test)**2 / op_h) # test kernel

lam = np.linspace(0.0000005,0.00005,200)
mse = []
for l in lam:
    reg = Ridge(alpha=l)
    reg.fit(kernel1,y_tr)
    yhat = reg.predict(kernel2.T)
    mse.append(mean_squared_error(y_val,yhat))

plt.figure()
plt.plot(lam,mse)
plt.xlabel('lambda')
plt.ylabel('MSE')
plt.title('Ridge validation curve( Select lambda)')
plt.show()
op_l = lam[np.argmin(mse)]
print(f'Optimal lambda is {op_l}')
print(f'Optimal MSE is {mse[np.argmin(mse)]}')

reg = Ridge(alpha=op_l)
reg.fit(kernel0,y_train)
yhat = reg.predict(kernel3.T)
print('')
print(f'MSE on test is {mean_squared_error(y_test,yhat)}')
```




```
Optimal h is 3.040201005025126
Optimal MSE is 0.020578345958342646
```




```
Optimal lambda is 7.464824120603017e-06
Optimal MSE is 0.004496609491424424

MSE on test is 0.0033052568205720083
```

In [ ]: