

1a)

$\lambda = 0.01$, $\sigma = 0.04$

```
In [1]: import numpy as np
import matplotlib.pyplot as plt

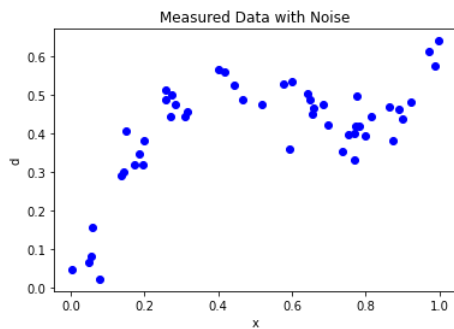
np.random.seed(1024) # ensure same noise for each run

# number of training points
n = 50

# sample n random points between 0 and 1
x = np.random.rand(n,1)

# set d = x^2 + .4 sin(1.5 pi x) + noise
d = x*x + 0.4*np.sin(1.5*np.pi*x) + 0.04*np.random.randn(n,1)

# plot result
plt.plot(x,d,'bo')
plt.xlabel('x')
plt.ylabel('d')
plt.title('Measured Data with Noise')
plt.show()
```



```
In [2]: sigma = 0.04 #defines Gaussian kernel width
p = 100 #number of points on x-axis

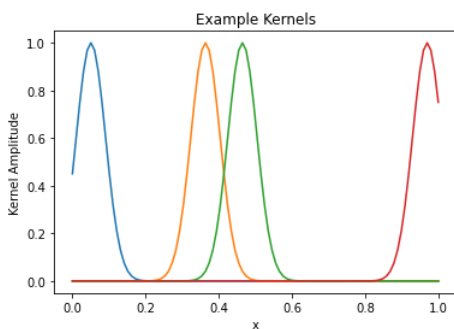
# Display examples of the kernels
x_test = np.linspace(0,1.00,p) # uniformly sample interval [0,1]
j_list = [5, 36, 46, 96] #list of indices for example kernels

Kdisplay = np.zeros((p,len(j_list)),dtype=float)

for i in range(p):
    for j in range(len(j_list)):
        Kdisplay[i,j] = np.exp(-(x_test[i]-x_test[j_list[j]])**2/(2*sigma**2))

print('Sigma = ',sigma)
plt.plot(x_test, Kdisplay)
plt.title('Example Kernels')
plt.xlabel('x')
plt.ylabel('Kernel Amplitude')
plt.show()
```

Sigma = 0.04



```
In [3]: # Kernel fitting to data

lam = 0.01 #ridge regression parameter

distsq=np.zeros((n,n),dtype=float)

for i in range(0,n):
    for j in range(0,n):
        distsq[i,j]=(x[i]-x[j])**2

K = np.exp(-distsq/(2*sigma**2))

alpha = np.linalg.inv(K+lam*np.identity(n))@d
```

```
In [4]: # Generate smooth curve corresponding to data fit

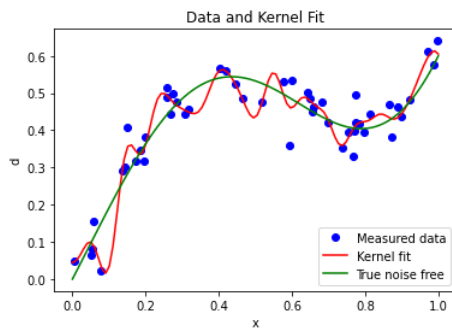
distsq_x_test = np.zeros((p,n),dtype=float)
for i in range(0,p):
    for j in range(0,n):
        distsq_x_test[i,j] = (x_test[i]-x[j])**2

dtest = np.exp(-distsq_x_test/(2*sigma**2))@alpha

dtrue = x_test*x_test + 0.4*np.sin(1.5*np.pi*x_test) # noise free data for comparison

print('Sigma = ',sigma)
print('Lambda = ',lam)
plt.plot(x,d,'bo',label='Measured data')
plt.plot(x_test,dtest,'r',label='Kernel fit')
plt.plot(x_test,dtrue,'g',label='True noise free')
plt.title('Data and Kernel Fit')
plt.legend(loc='lower right')
plt.xlabel('x')
plt.ylabel('d')
plt.show()

Sigma = 0.04
Lambda = 0.01
```



The third peak from the left has an xi value of roughly 0.45. The peak is located at the value of xi and the curve drops down on either side this xi value. The sigma determines the smoothness of the curve. The higher the sigma the smoother the curve is i.e. the less the curve factors in finer details about the dataset in question. Conversely, the smaller sigma is, the more the curve tends to overfit the data.

1b)

lamda = 0.01, sigma = 0.2

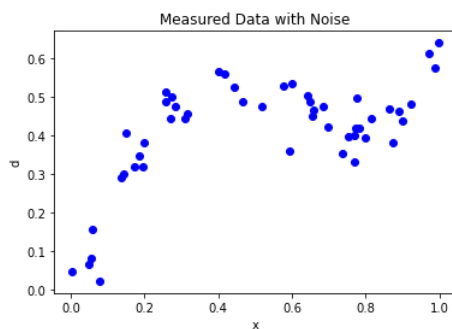
```
In [5]: np.random.seed(1024) # ensure same noise for each run

# number of training points
n = 50

# sample n random points between 0 and 1
x = np.random.rand(n,1)

# set d = x^2 + .4 sin(1.5 pi x) + noise
d = x*x + 0.4*np.sin(1.5*np.pi*x) +0.04*np.random.randn(n,1)

# plot result
plt.plot(x,d,'bo')
plt.xlabel('x')
plt.ylabel('d')
plt.title('Measured Data with Noise')
plt.show()
```



```
In [6]: sigma = 0.2 #defines Gaussian kernel width
p = 100 #number of points on x-axis

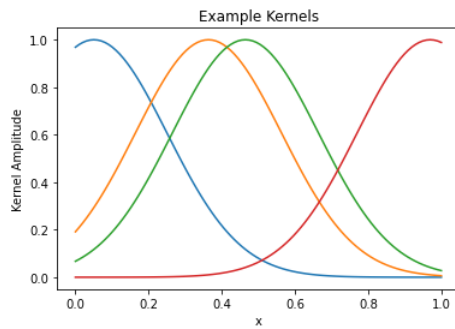
# Display examples of the kernels
x_test = np.linspace(0,1.00,p) # uniformly sample interval [0,1]
j_list = [5, 36, 46, 96] #list of indices for example kernels

Kdisplay = np.zeros((p,len(j_list)),dtype=float)

for i in range(p):
    for j in range(len(j_list)):
        Kdisplay[i,j]= np.exp(-(x_test[i]-x_test[j_list[j]])**2/(2*sigma**2))

print('Sigma = ',sigma)
plt.plot(x_test, Kdisplay)
plt.title('Example Kernels')
plt.xlabel('x')
plt.ylabel('Kernel Amplitude')
plt.show()
```

Sigma = 0.2



```
In [7]: # Kernel fitting to data

lam = 0.01 #ridge regression parameter

distsq=np.zeros((n,n),dtype=float)

for i in range(0,n):
    for j in range(0,n):
        distsq[i,j]=(x[i]-x[j])**2

K = np.exp(-distsq/(2*sigma**2))

alpha = np.linalg.inv(K+lam*np.identity(n))@d
```

In [8]: *# Generate smooth curve corresponding to data fit*

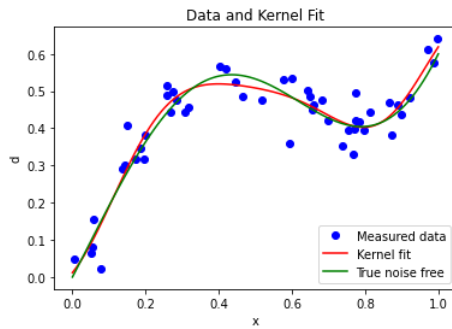
```
distsq_xtest = np.zeros((p,n),dtype=float)
for i in range(0,p):
    for j in range(0,n):
        distsq_xtest[i,j] = (x_test[i]-x[j])**2

dtest = np.exp(-distsq_xtest/(2*sigma**2))@alpha

dtrue = x_test*x_test + 0.4*np.sin(1.5*np.pi*x_test) # noise free data for comparison

print('Sigma = ',sigma)
print('Lambda = ',lam)
plt.plot(x,d,'bo',label='Measured data')
plt.plot(x_test,dtest,'r',label='Kernel fit')
plt.plot(x_test,dtrue,'g',label='True noise free')
plt.title('Data and Kernel Fit')
plt.legend(loc='lower right')
plt.xlabel('x')
plt.ylabel('d')
plt.show()

Sigma = 0.2
Lambda = 0.01
```



In []:

lamda = 0.01, sigma = 1

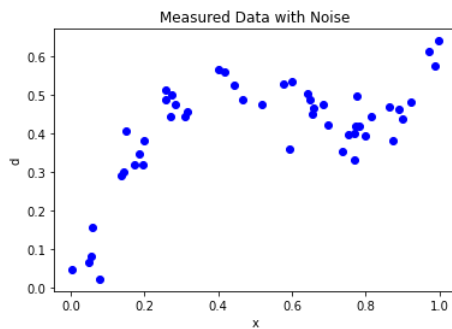
In [9]: *np.random.seed(1024) # ensure same noise for each run*

```
# number of training points
n = 50

# sample n random points between 0 and 1
x = np.random.rand(n,1)

# set d = x^2 + .4 sin(1.5 pi x) + noise
d = x*x + 0.4*np.sin(1.5*np.pi*x) +0.04*np.random.randn(n,1)

# plot result
plt.plot(x,d,'bo')
plt.xlabel('x')
plt.ylabel('d')
plt.title('Measured Data with Noise')
plt.show()
```



```
In [10]: sigma = 1 #defines Gaussian kernel width
p = 100 #number of points on x-axis

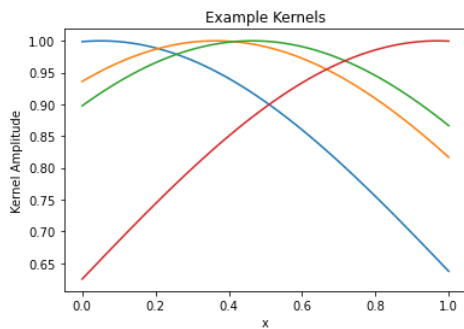
# Display examples of the kernels
x_test = np.linspace(0,1.00,p) # uniformly sample interval [0,1]
j_list = [5, 36, 46, 96] #list of indices for example kernels

Kdisplay = np.zeros((p,len(j_list)),dtype=float)

for i in range(p):
    for j in range(len(j_list)):
        Kdisplay[i,j]= np.exp(-(x_test[i]-x_test[j_list[j]])**2/(2*sigma**2))

print('Sigma = ',sigma)
plt.plot(x_test, Kdisplay)
plt.title('Example Kernels')
plt.xlabel('x')
plt.ylabel('Kernel Amplitude')
plt.show()
```

Sigma = 1



```
In [11]: # Kernel fitting to data

lam = 0.01 #ridge regression parameter

distsq=np.zeros((n,n),dtype=float)

for i in range(0,n):
    for j in range(0,n):
        distsq[i,j]=(x[i]-x[j])**2

K = np.exp(-distsq/(2*sigma**2))

alpha = np.linalg.inv(K+lam*np.identity(n))@d
```

```
In [12]: # Generate smooth curve corresponding to data fit

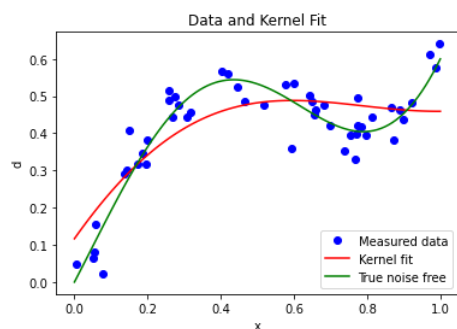
distsq_xtest = np.zeros((p,n),dtype=float)
for i in range(0,p):
    for j in range(0,n):
        distsq_xtest[i,j] = (x_test[i]-x[j])**2

dtest = np.exp(-distsq_xtest/(2*sigma**2))@alpha

dtrue = x_test*x_test + 0.4*np.sin(1.5*np.pi*x_test) # noise free data for comparison

print('Sigma = ',sigma)
print('Lambda = ',lam)
plt.plot(x,d,'bo',label='Measured data')
plt.plot(x_test,dtest,'r',label='Kernel fit')
plt.plot(x_test,dtrue,'g',label='True noise free')
plt.title('Data and Kernel Fit')
plt.legend(loc='lower right')
plt.xlabel('x')
plt.ylabel('d')
plt.show()
```

Sigma = 1
Lambda = 0.01



lamda = 1, sigma = 0.04

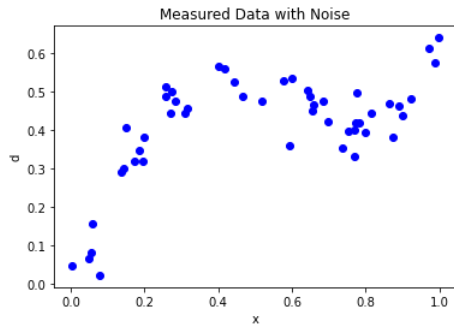
```
In [13]: np.random.seed(1024) # ensure same noise for each run

# number of training points
n = 50

# sample n random points between 0 and 1
x = np.random.rand(n,1)

# set  $d = x^2 + .4 \sin(1.5 \pi x) + \text{noise}$ 
d = x*x + 0.4*np.sin(1.5*np.pi*x) + 0.04*np.random.randn(n,1)

# plot result
plt.plot(x,d,'bo')
plt.xlabel('x')
plt.ylabel('d')
plt.title('Measured Data with Noise')
plt.show()
```



```
In [14]: sigma = 0.04 #defines Gaussian kernel width
p = 100 #number of points on x-axis

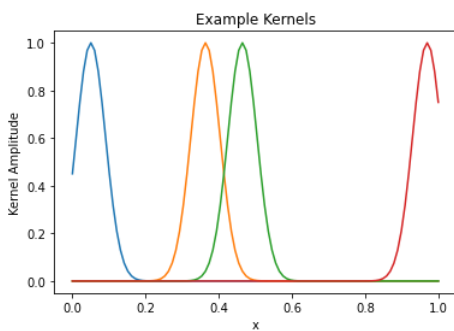
# Display examples of the kernels
x_test = np.linspace(0,1.00,p) # uniformly sample interval [0,1]
j_list = [5, 36, 46, 96] #List of indices for example kernels

Kdisplay = np.zeros((p,len(j_list)),dtype=float)

for i in range(p):
    for j in range(len(j_list)):
        Kdisplay[i,j]= np.exp(-(x_test[i]-x_test[j_list[j]])**2/(2*sigma**2))

print('Sigma = ',sigma)
plt.plot(x_test, Kdisplay)
plt.title('Example Kernels')
plt.xlabel('x')
plt.ylabel('Kernel Amplitude')
plt.show()
```

Sigma = 0.04



```
In [15]: # Kernel fitting to data

lam = 1 #ridge regression parameter

distsq=np.zeros((n,n),dtype=float)

for i in range(0,n):
    for j in range(0,n):
        distsq[i,j]=(x[i]-x[j])**2

K = np.exp(-distsq/(2*sigma**2))

alpha = np.linalg.inv(K+lam*np.identity(n))@d
```

In [16]: *# Generate smooth curve corresponding to data fit*

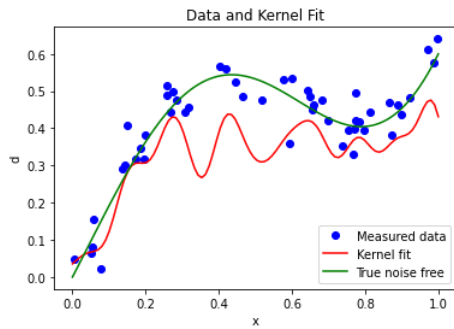
```
distsq_xtest = np.zeros((p,n),dtype=float)
for i in range(0,p):
    for j in range(0,n):
        distsq_xtest[i,j] = (x_test[i]-x[j])**2

dtest = np.exp(-distsq_xtest/(2*sigma**2))@alpha

dtrue = x_test*x_test + 0.4*np.sin(1.5*np.pi*x_test) # noise free data for comparison

print('Sigma = ',sigma)
print('Lambda = ',lam)
plt.plot(x,d,'bo',label='Measured data')
plt.plot(x_test,dtest,'r',label='Kernel fit')
plt.plot(x_test,dtrue,'g',label='True noise free')
plt.title('Data and Kernel Fit')
plt.legend(loc='lower right')
plt.xlabel('x')
plt.ylabel('d')
plt.show()
```

Sigma = 0.04
Lambda = 1



lamda = 1, sigma = 0.2

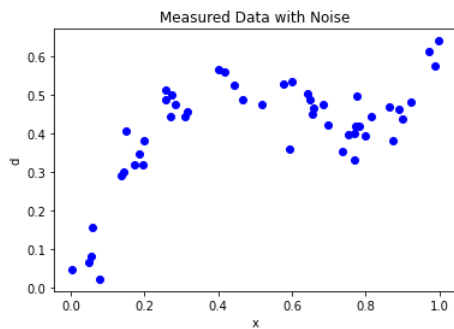
In [17]: *np.random.seed(1024) # ensure same noise for each run*

```
# number of training points
n = 50

# sample n random points between 0 and 1
x = np.random.rand(n,1)

# set d = x^2 + .4 sin(1.5 pi x) + noise
d = x*x + 0.4*np.sin(1.5*np.pi*x) + 0.04*np.random.randn(n,1)

# plot result
plt.plot(x,d,'bo')
plt.xlabel('x')
plt.ylabel('d')
plt.title('Measured Data with Noise')
plt.show()
```



```

In [18]: sigma = 0.2 #defines Gaussian kernel width
p = 100 #number of points on x-axis

# Display examples of the kernels
x_test = np.linspace(0,1.00,p) # uniformly sample interval [0,1]
j_list = [5, 36, 46, 96] #list of indices for example kernels

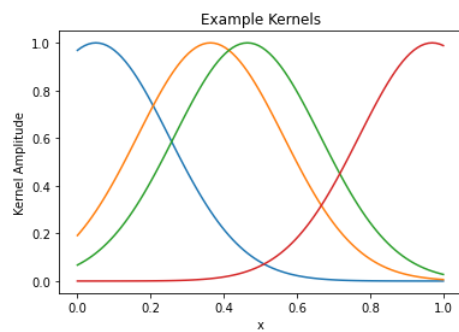
Kdisplay = np.zeros((p,len(j_list)),dtype=float)

for i in range(p):
    for j in range(len(j_list)):
        Kdisplay[i,j]= np.exp(-(x_test[i]-x_test[j_list[j]])**2/(2*sigma**2))

print('Sigma = ',sigma)
plt.plot(x_test, Kdisplay)
plt.title('Example Kernels')
plt.xlabel('x')
plt.ylabel('Kernel Amplitude')
plt.show()

```

Sigma = 0.2



```

In [19]: # Kernel fitting to data

lam = 1 #ridge regression parameter

distsq=np.zeros((n,n),dtype=float)

for i in range(0,n):
    for j in range(0,n):
        distsq[i,j]=(x[i]-x[j])**2

K = np.exp(-distsq/(2*sigma**2))

alpha = np.linalg.inv(K+lam*np.identity(n))@d

```



```
In [20]: # Generate smooth curve corresponding to data fit

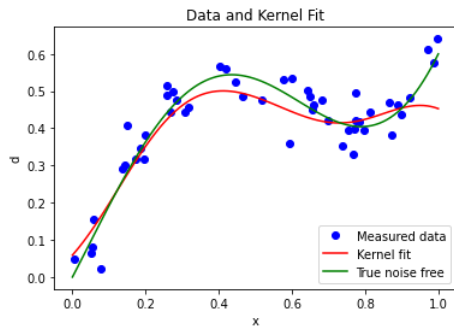
distsq_xtest = np.zeros((p,n),dtype=float)
for i in range(0,p):
    for j in range(0,n):
        distsq_xtest[i,j] = (x_test[i]-x[j])**2

dtest = np.exp(-distsq_xtest/(2*sigma**2))@alpha

dtrue = x_test*x_test + 0.4*np.sin(1.5*np.pi*x_test) # noise free data for comparison

print('Sigma = ',sigma)
print('Lambda = ',lam)
plt.plot(x,d,'bo',label='Measured data')
plt.plot(x_test,dtest,'r',label='Kernel fit')
plt.plot(x_test,dtrue,'g',label='True noise free')
plt.title('Data and Kernel Fit')
plt.legend(loc='lower right')
plt.xlabel('x')
plt.ylabel('d')
plt.show()

Sigma = 0.2
Lambda = 1
```



A low sigma value will lead to function having more bumps and being very erratic. Higher values of sigma will lead to the curve being very smooth and ultimately end up approximating the dataset more generally than if the sigma was lower. Lower sigma vals will try to catch every little perturbation in the dataset.

High values of lamda often overshoot the dataset and make very error prone curve fits. You essentially underfit the data. Low values of lamda do the exact opposite.

1c)

We could apply the principle of cross validation to select appropriate paramters for lamda and sigma. We could split the data into test and train subsets and train the classifier on the training dataset and after training apply the test dataset to the model to see how it performs. This deals with the issue of overfitting and underfitting.

In []: