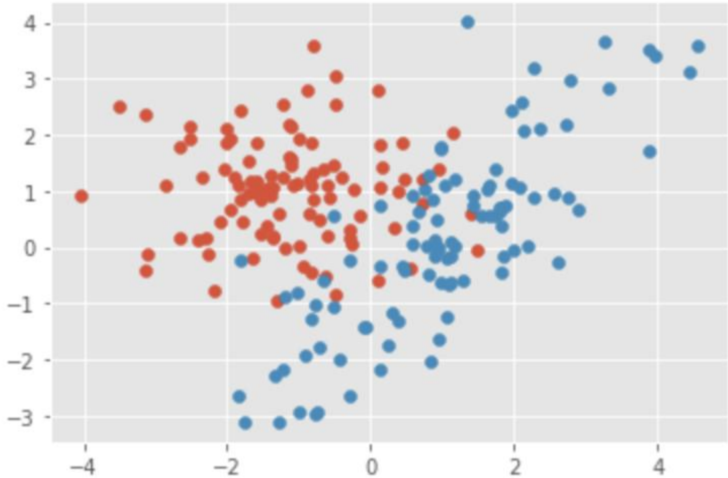
In [9]:
```python
import numpy as np
import matplotlib.pyplot as plt
from sympy import symbols,simplify
from sympy.solvers import solve
from sympy.plotting import plot
from sympy import lambdify
from IPython.display import Image
import warnings
warnings.filterwarnings("ignore")
```

In [10]:
```python
# class-conditional density

## w1
M1=[-1,1]
cov1=[[1,0],[0,1]]

## w2
M2=[1,0]
cov2=[[2,2],[2,3]]

## Generate 100 random training samples from each of the two densities

a1,b1=np.random.multivariate_normal(M1,cov1,100).T
a2,b2=np.random.multivariate_normal(M2,cov2,100).T
```

In [11]:
```python
## (a) plot these samples
plt.style.use('ggplot')
plt.scatter(a1,b1)
plt.scatter(a2,b2)
plt.show()
```

In [11]:
```
## (a) plot these samples
plt.style.use('ggplot')
plt.scatter(a1,b1)
plt.scatter(a2,b2)
plt.show()
```



In [12]:
```
## (b) Find the MLE

M1_est=[np.mean(a1),np.mean(b1)]
M2_est=[np.mean(a2),np.mean(b2)]

cov1_est=np.cov(a1,b1)
cov2_est=np.cov(a2,b2)

print('M1_estimate: \n',M1_est)
print('M2_estimate: \n',M2_est)
print('\n')
print('cov1_estimate: \n',cov1_est)
print('cov2_estimate: \n',cov2_est)
```

```
M1_estimate:
 [-1.1272347637303202, 1.0196935131399545]
M2_estimate:
 [1.0176072137767807, 0.15860527676387393]


cov1_estimate:
 [[ 1.14050164 -0.02655455]
 [-0.02655455  0.84406118]]
cov2_estimate:
 [[1.92566157 1.83208381]
 [1.83208381 2.73209582]]
```

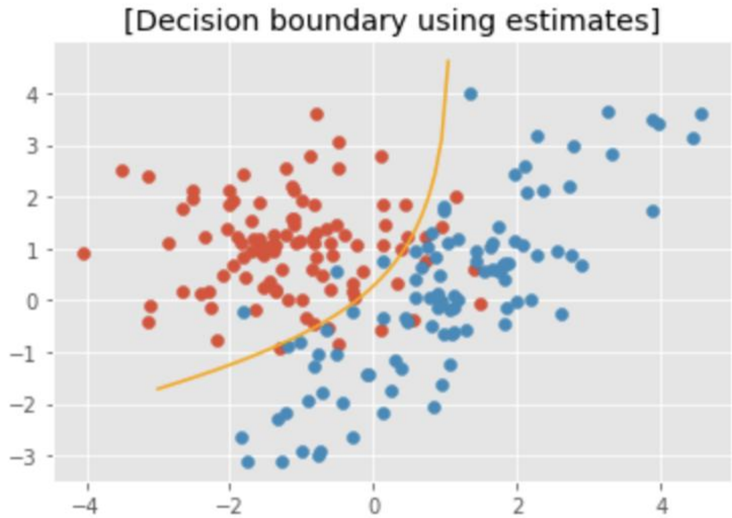
In [13]:
```
# Decision boundary formulation
Image("decision boundary.png")
```

Out[13]:

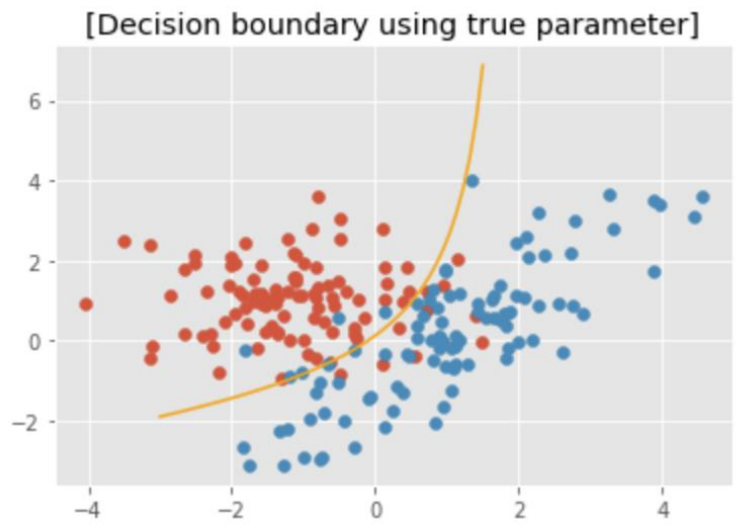
$$g(\mathbf{x}) \equiv g_1(\mathbf{x}) - g_2(\mathbf{x})$$

In [14]:
```
# Define decision boundary func

def Decision_boundary(m1,sig1,m2,sig2):
    m1=np.array(m1)
    sig1=np.array(sig1)
    m2=np.array(m2)
    sig2=np.array(sig2)
    return lambda a1,a2:(-1/2)*np.matmul(np.matmul((np.array([a1,a2])-m1).T,np.linalg.inv(sig
```

In [15]:
```
## (c) Decision boundary using estimates

x1,y1=symbols('x1 y1',real=True)
get_var=Decision_boundary(M1_est,cov1_est,M2_est,cov2_est)

F=get_var(x1,y1)
F=simplify(F)

result=lambdify(x1,solve(F,y1)[0])

x_vals=np.linspace(-3,1.5)
y_vals=result(x_vals)

plt.title('[Decision boundary using estimates]')
plt.scatter(a1,b1)
plt.scatter(a2,b2)
plt.plot(x_vals,y_vals,color='orange')
plt.show()
```

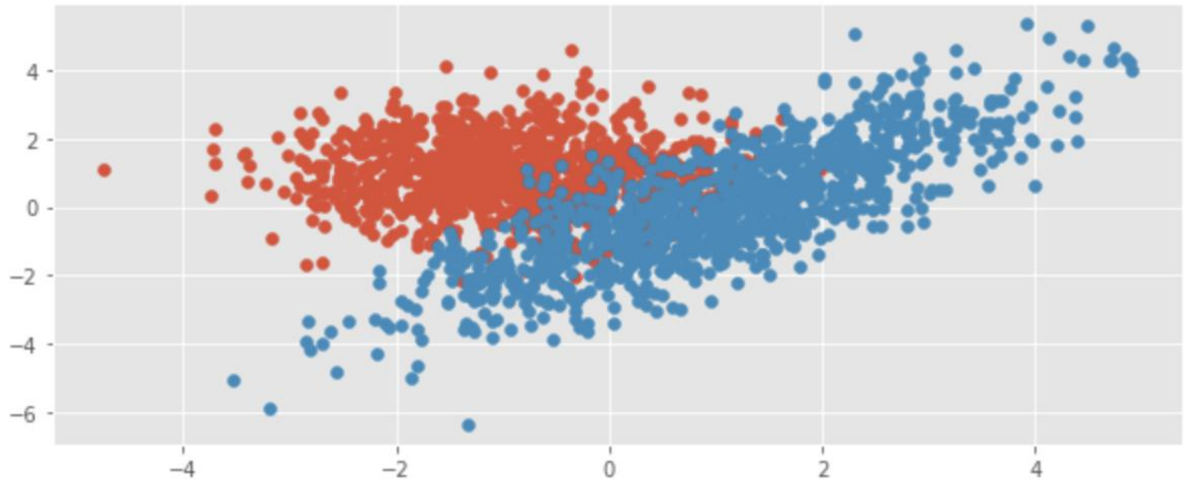


In [16]:
```python
## (d) Decision boundary using true parameter

x1,y1=symbols('x1 y1',real=True)
get_var=Decision_boundary(M1,cov1,M2,cov2)

F=simplify(get_var(x1,y1))

result=lambdify(x1,solve(F,y1)[0])

x_vals=np.linspace(-3,1.5)
y_vals=result(x_vals)

plt.title('[Decision boundary using true parameter]')
plt.scatter(a1,b1)
plt.scatter(a2,b2)
plt.plot(x_vals,y_vals,color='orange')
plt.show()
```



In [31]:
```
## (e) Draw an independent 1000 test samples

test_a1,test_b1=np.random.multivariate_normal(M1,cov1,1000).T
test_a2,test_b2=np.random.multivariate_normal(M2,cov2,1000).T


plt.rcParams["figure.figsize"] = (10,4)
plt.scatter(test_a1,test_b1)
plt.scatter(test_a2,test_b2)
plt.show()

### Input test samples class

test_classify=[]
for i,j in zip(test_a1,test_b1):
    test_classify.append([i,j,0])

for i,j in zip(test_a2,test_b2):
    test_classify.append([i,j,1])

test_classify=np.array(test_classify)

## Error rate for 'plug-in' boundary

plug_result=Decision_boundary(M1_est,cov1_est,M2_est,cov2_est)

plug_predict=[]

for i in test_classify:
    if plug_result(i[0],i[1])>0:
        plug_predict.append([i[0],i[1],0])
    else:
        plug_predict.append([i[0],i[1],1])

plug_predict=np.array(plug_predict)

error_rate1=1-(np.sum(plug_predict[:,2]==test_classify[:,2])/plug_predict.shape[0])
```

```python
print('1.\"Plug-in\" error rate : %.5f '%error_rate1)

## Error rate for true decision boundary

true_result=Decision_boundary(M1,cov1,M2,cov2)

true_predict=[]

for i in test_classify:
    if true_result(i[0],i[1])>0:
        true_predict.append([i[0],i[1],0])
    else:
        true_predict.append([i[0],i[1],1])

true_predict=np.array(true_predict)

error_rate2=1-(np.sum(true_predict[:,2]==test_classify[:,2])/true_predict.shape[0])

print('2. True boundary error rate : %.5f '%error_rate2)
```




```
1."Plug-in" error rate : 0.09150
2. True boundary error rate : 0.09050
```

In [32]:
```
# (f) Repeat parts (a)-(e)

a1,b1=np.random.multivariate_normal(M1,cov1,100).T
a2,b2=np.random.multivariate_normal(M2,cov2,100).T

# keep the same test set
a1=np.array(list(a1)+list(test_a1))
b1=np.array(list(b1)+list(test_b1))
a2=np.array(list(a2)+list(test_a2))
b2=np.array(list(b2)+list(test_b2))

M1_est=[np.mean(a1),np.mean(b1)]
M2_est=[np.mean(a2),np.mean(b2)]

cov1_est=np.cov(a1,b1)
cov2_est=np.cov(a2,b2)


### Input test samples class

test_classify=[]
for i,j in zip(a1,b1):
    test_classify.append([i,j,0])

for i,j in zip(a2,b2):
    test_classify.append([i,j,1])

test_classify=np.array(test_classify)

## Error rate for 'plug-in' boundary

plug_result=Decision_boundary(M1_est,cov1_est,M2_est,cov2_est)

plug_predict=[]
```

```
for i in test_classify:
    if plug_result(i[0],i[1])>0:
        plug_predict.append([i[0],i[1],0])
    else:
        plug_predict.append([i[0],i[1],1])

plug_predict=np.array(plug_predict)

error_rate1=1-(np.sum(plug_predict[:,2]==test_classify[:,2])/plug_predict.shape[0])

print('1.\"Plug-in\" error rate : %.5f '%error_rate1)

## Error rate for true decision boundary

true_result=Decision_boundary(M1,cov1,M2,cov2)

true_predict=[]

for i in test_classify:
    if true_result(i[0],i[1])>0:
        true_predict.append([i[0],i[1],0])
    else:
        true_predict.append([i[0],i[1],1])

true_predict=np.array(true_predict)

error_rate2=1-(np.sum(true_predict[:,2]==test_classify[:,2])/true_predict.shape[0])

print('2. True boundary error rate : %.5f '%error_rate2)
```

```
1."Plug-in" error rate : 0.09273
2. True boundary error rate : 0.09364
```

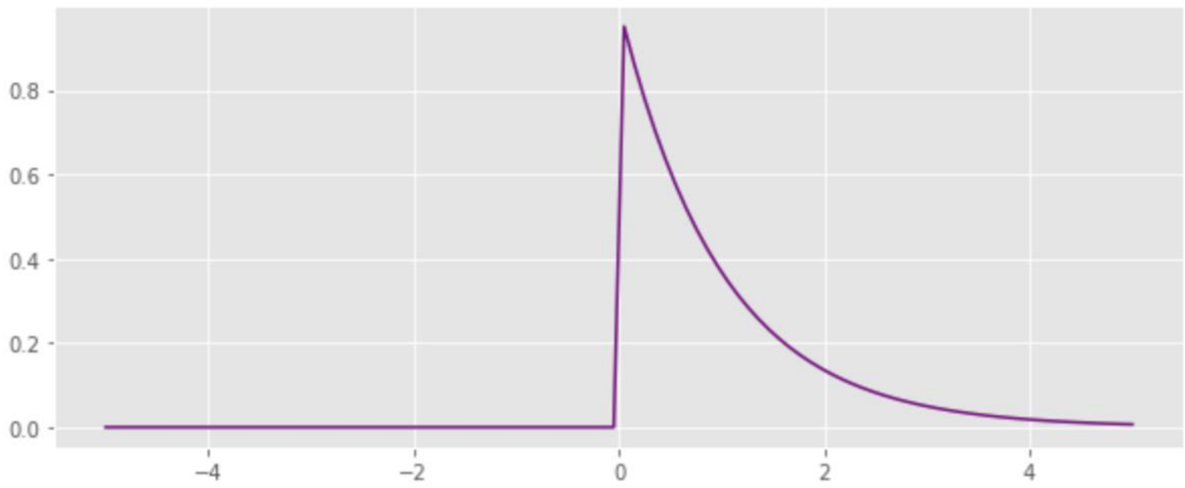
Two error rates are different because they are using the different parameters.

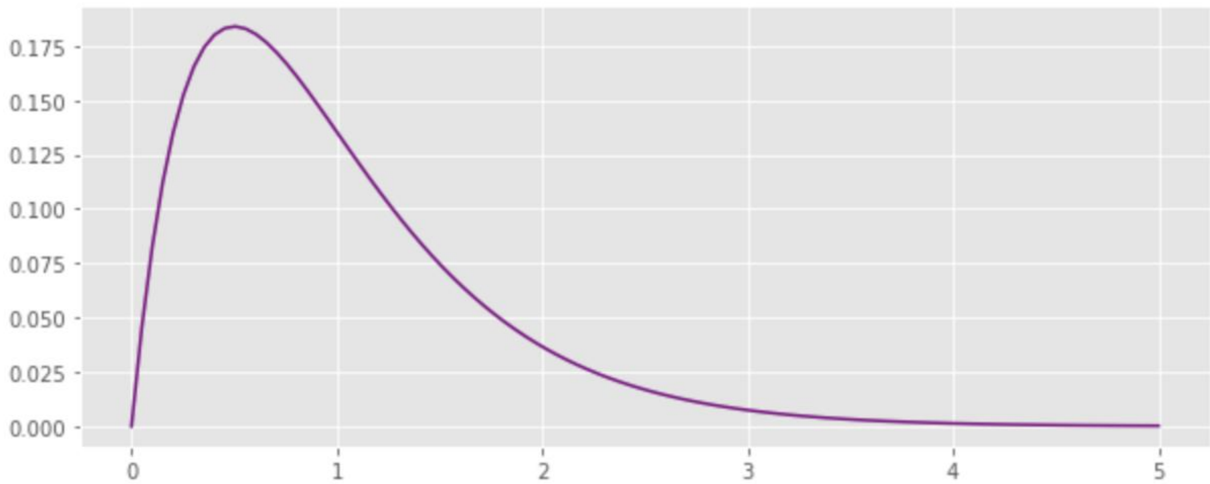
In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
import math
def exp_density(x,theta,versus=True):
    result=[]
    if versus:
        for i in x:
            if i>=0:
                result.append(theta*math.exp(-theta*i))
            else:
                result.append(0)
        result=np.array(result)
    else:
        for i in theta:
            if x<0:
                result.append(0)
            else:
                result.append(i*math.exp(-i*x))
    return result
```

```
    return result
```

In [2]:

```
## (a) Plot the two graphs

## case 1

x=np.linspace(-5,5,100)
theta=1
y=exp_density(x,theta)
plt.rcParams["figure.figsize"] = (10,4)
plt.style.use('ggplot')
plt.plot(x,y,color='purple')
plt.show()

## case 2

x=2
theta=np.linspace(0,5,100)
y=exp_density(x,theta,versus=False)
plt.plot(theta,y,color='purple')
plt.show()
```


0.8
0.6
0.4
0.2
0.0
−4
−2
0
2
4


0.175
0.150
0.125
0.100
0.075
0.050
0.025
0.000
0
1
2
3
4
5

(b) $L(\theta) = \prod_{i=1}^{n} p(x_i|\theta) = \theta^n \cdot e^{-\theta(x_1+x_2+\dots+x_n)} = \theta^n \cdot e^{-\theta \Sigma x_i}$

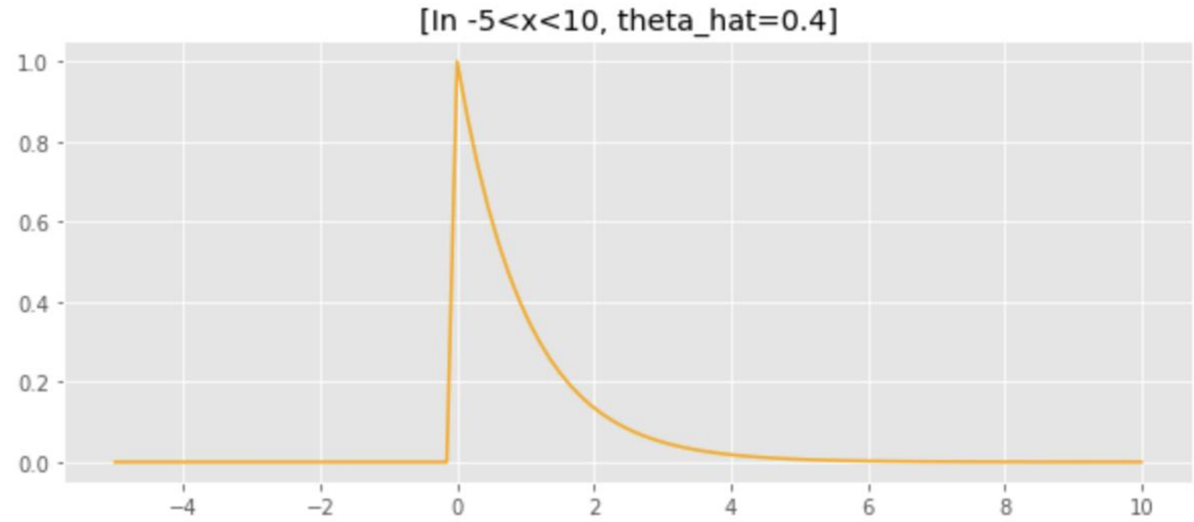
$\log(L\theta) = \log\left(\theta^n \cdot e^{-\theta \Sigma x_i}\right) = \ln \theta^n + \ln e^{-\theta \Sigma x_i}$

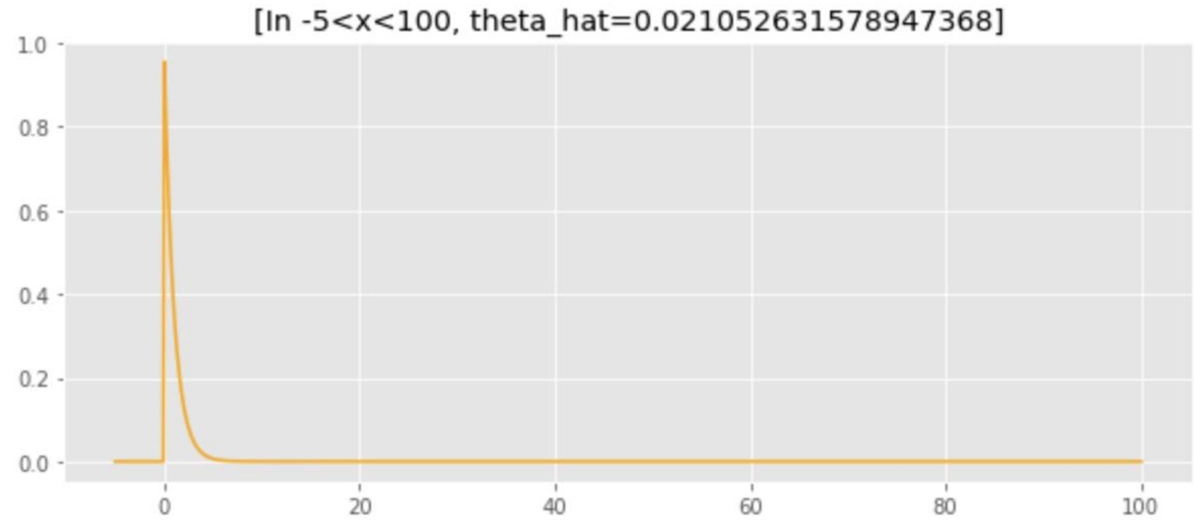
$= n \ln \theta + (-\theta \Sigma x_i) \ln e$

$\dfrac{\partial \ln(L\theta))}{\partial \theta} = n \cdot \dfrac{1}{\theta} - \Sigma x_i = 0$

$\therefore \hat{\theta} = \dfrac{1}{\frac{1}{n} \cdot \sum_{i=1}^{n} x_k}$

In [3]:
```
#(c) Mark the MLE

## graph 1
x=np.linspace(-5,10,100)
theta=1
y=exp_density(x,theta)
theta_hat=len(x)/np.sum(x)

plt.title('[In -5<x<10, theta_hat='+str(theta_hat)+']')
plt.plot(x,y,color='orange')
plt.show()

## graph 2
wide_x=np.linspace(-5,100,1000)
y=exp_density(wide_x,theta)
theta_hat=len(wide_x)/np.sum(wide_x)

plt.title('[In -5<x<100, theta_hat='+str(theta_hat)+']')
plt.plot(wide_x,y,color='orange')
plt.show()
#print('theta_hat tends to converge to 0 when we take larger range of x')
```





When x has large ranges, theta_hat tends to convergence to zero.