Question 1-a

$$\hat{y}(x) \;=\; \text{sign}\big(\sum_{j=1}^{N} \alpha_j\, K(x, x_j)\big)$$

Question 1-b

α_{100} and α_{101} are the support vectors

$$\hat{y}(x) \;=\; \text{sign}\big(\alpha_{100}\, K(x, x_{100}) \;+\; \alpha_{101}\, K(x, x_{101})\big).$$

Question 2

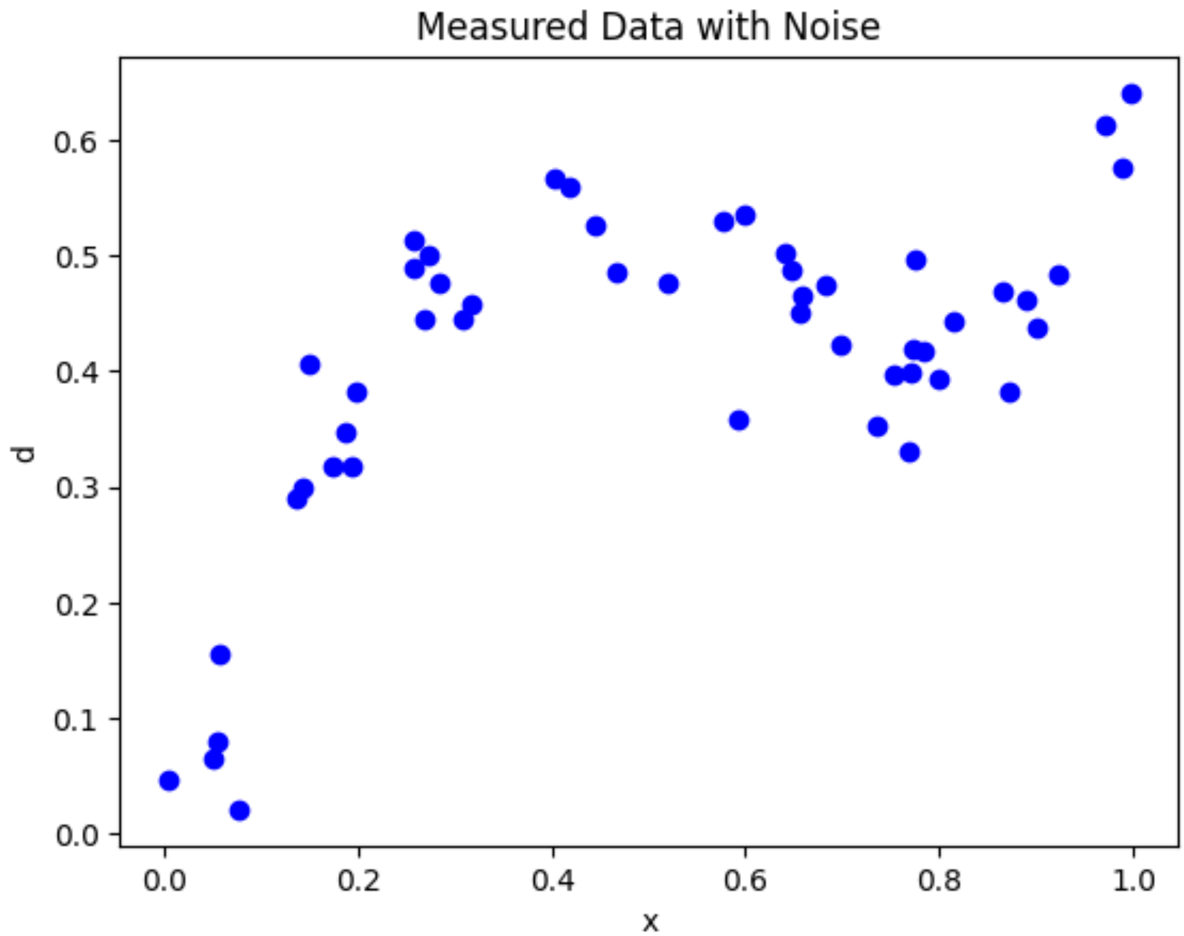
$$\frac{\partial L}{\partial w_j} = \frac{\partial L}{\partial f(x_i)} \cdot \frac{\partial f(x_i)}{\partial w_j} = \sum_{i=1}^{N} 2\big(f(x_i) - y_i\big)\, \frac{\partial f(x_i)}{\partial w_j} = 2 \sum_{i=1}^{N} \big(f(x_i) - y_i\big)\, v_j\, \mathbf{1}_{\{w_j^T x_i > 0\}}$$

Question 3

$$\max_x \big|f(x) - f_r(x)\big| \;\le\; \|v\|_2\, \varepsilon$$

Question 4

In [5]:
```
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()
```



Question 4-a

x_i determines the kernal value

σ determines the kernal width

Question 4-b

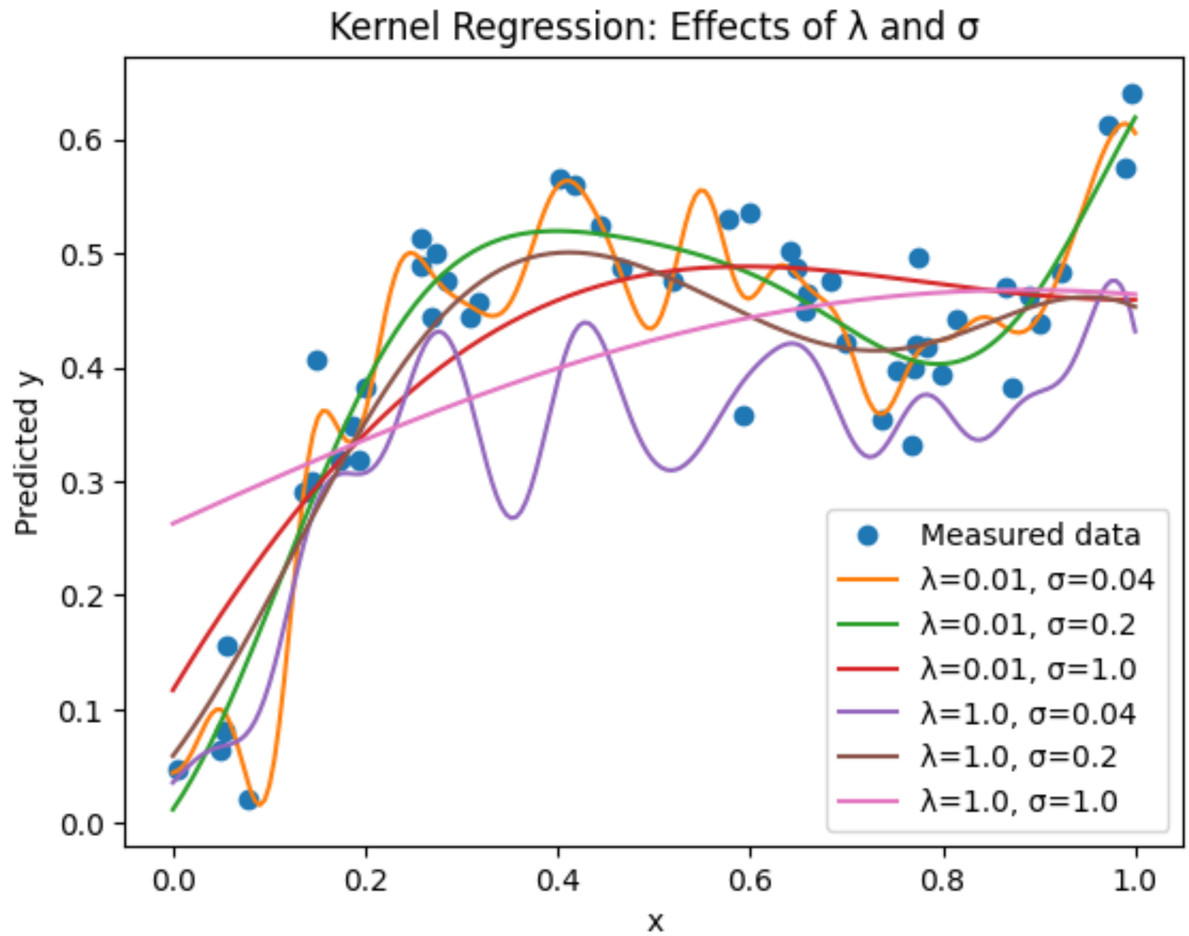
small σ makes the line overfit the data
large λ makes it smoother

Question 4-c

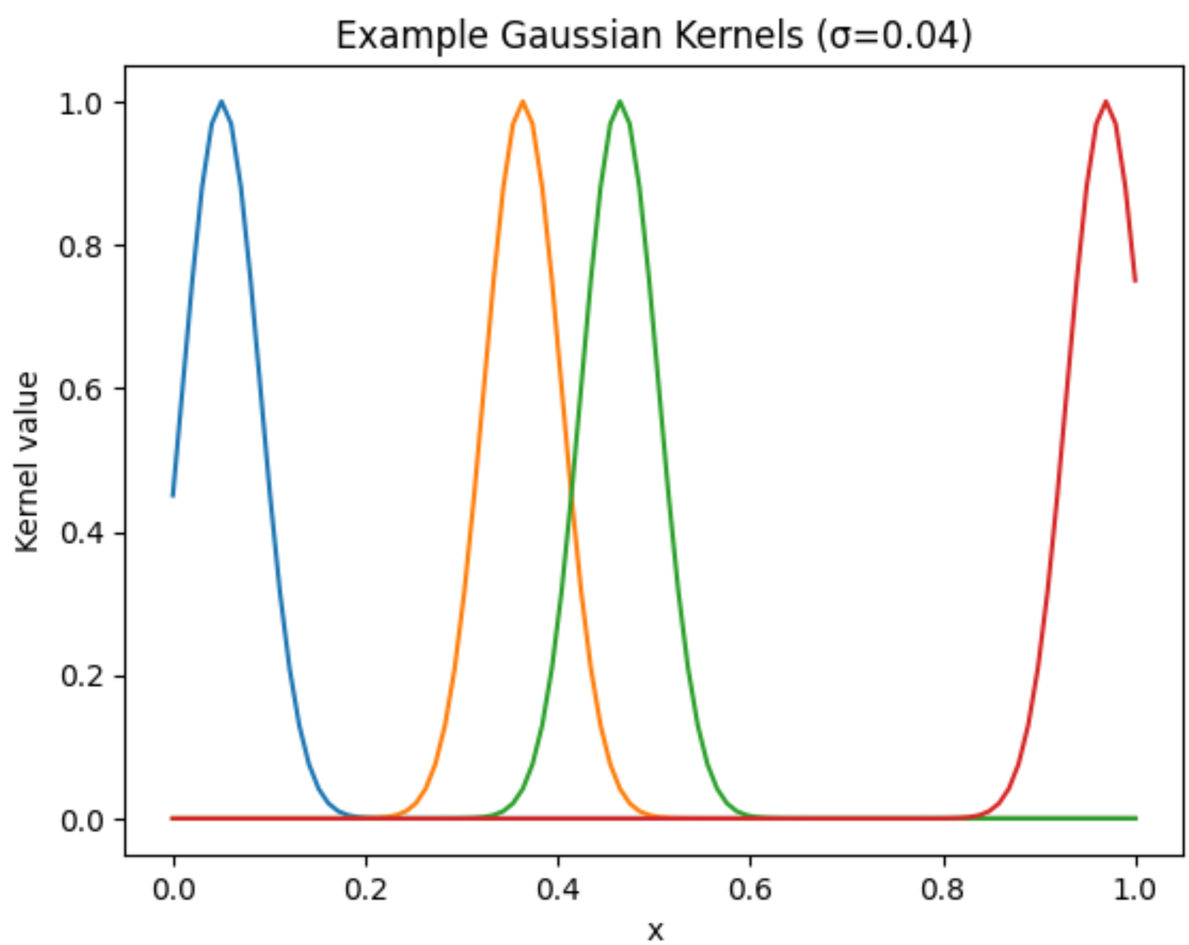
Least Squares

In [4]:
```
import numpy as np
import matplotlib.pyplot as plt

# Part A and B: Data generation
np.random.seed(1024)
n = 50
x = np.random.rand(n, 1)
d = x*x + 0.4*np.sin(1.5*np.pi*x) + 0.04*np.random.randn(n, 1)
```

```
# Part B: Kernel regression for various λ and σ
sigma_values = [0.04, 0.2, 1.0]
lambda_values = [0.01, 1.0]
x_test = np.linspace(0, 1, 200)[:, None]

def kernel_matrix(x1, x2, sigma):
    return np.exp(-((x1 - x2.T)**2) / (2 * sigma**2))

plt.figure()
plt.plot(x, d, 'o', label='Measured data')
for lam in lambda_values:
    for sigma in sigma_values:
        K = kernel_matrix(x, x, sigma)
        alpha = np.linalg.solve(K + lam * np.eye(n), d)
        y_pred = kernel_matrix(x_test, x, sigma).dot(alpha)
        plt.plot(x_test, y_pred, label=f'λ={lam}, σ={sigma}')
plt.xlabel('x')
plt.ylabel('Predicted y')
plt.title('Kernel Regression: Effects of λ and σ')
plt.legend()
plt.show()

# Part A: Example kernels
p = 100
x_test2 = np.linspace(0, 1, p)
j_list = [5, 36, 46, 96]
sigma = 0.04
Kdisplay = np.array([
    [np.exp(-(x_test2[i] - x_test2[idx])**2 / (2 * sigma**2)) for idx in j_l
    for i in range(p)
])

plt.figure()
plt.plot(x_test2, Kdisplay)
plt.xlabel('x')
plt.ylabel('Kernel value')
plt.title('Example Gaussian Kernels (σ=0.04)')
plt.show()

# Show the xi for the third peak
print("Value of x_test at index 46 (third peak):", x_test2[46])
```



Kernel Regression: Effects of λ and σ
Predicted y
x
Measured data
λ=0.01, σ=0.04
λ=0.01, σ=0.2
λ=0.01, σ=1.0
λ=1.0, σ=0.04
λ=1.0, σ=0.2
λ=1.0, σ=1.0
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.0
0.2
0.4
0.6
0.8
1.0



Example Gaussian Kernels (σ=0.04)
Kernel value
x
0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0

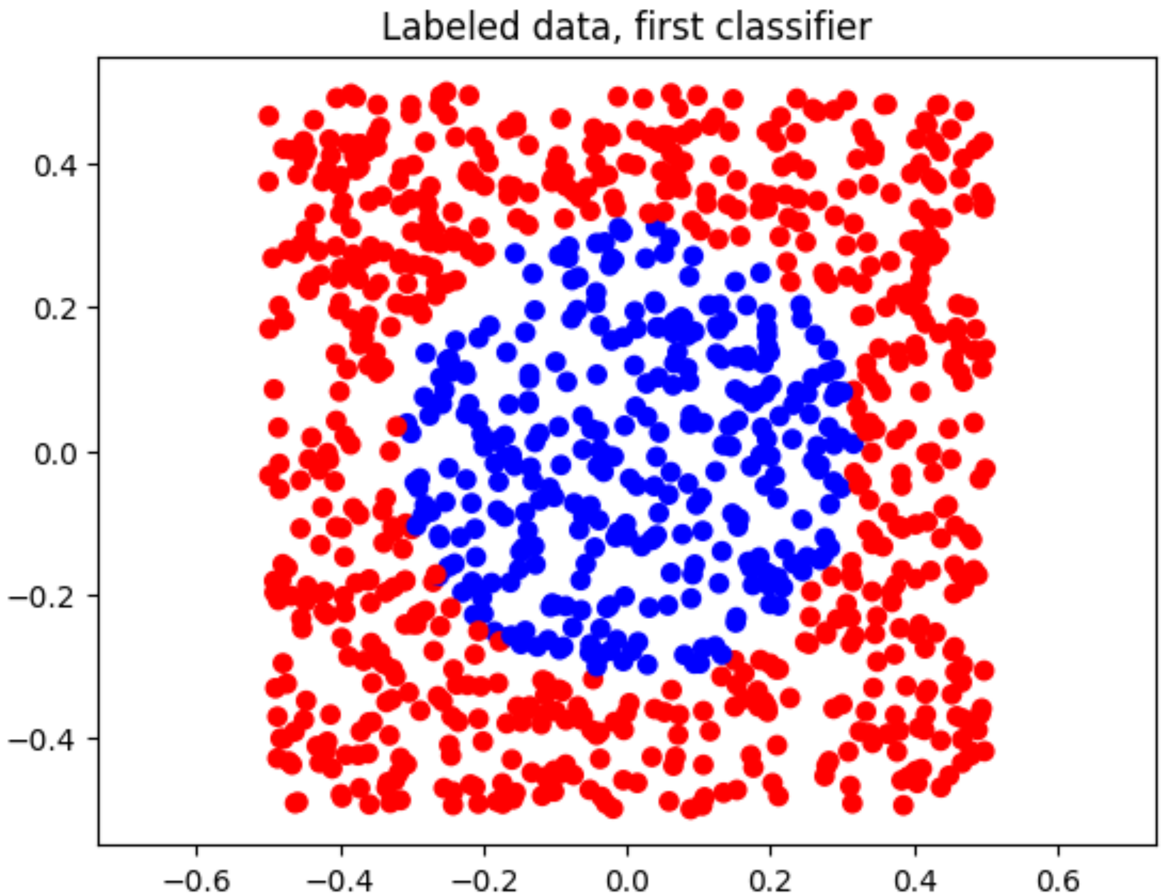
```
Value of x_test at index 46 (third peak): 0.4646464646464647
```

Question 5

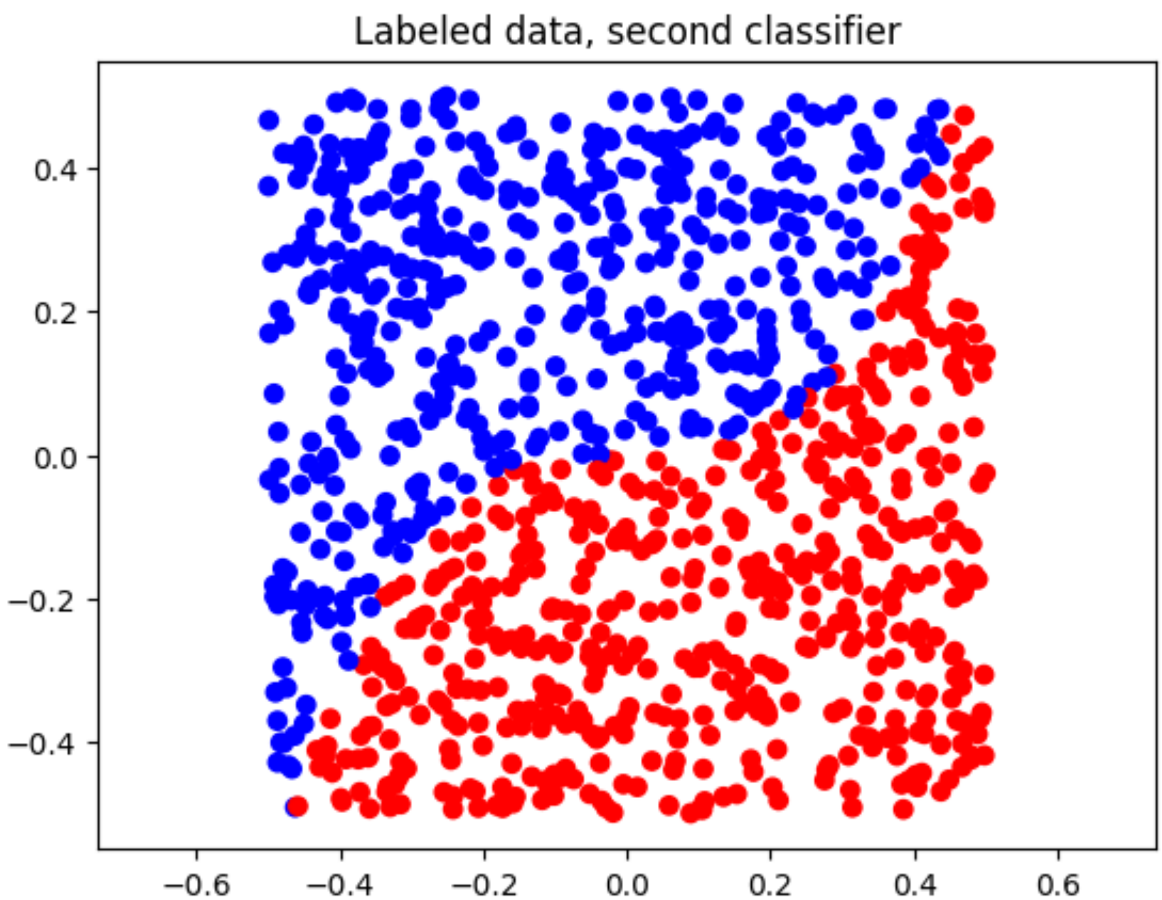
When you choose small σ the model overfits

In [9]:
```python
import numpy as np
import matplotlib.pyplot as plt

p = int(2) #features
n = int(1000) #examples

## generate training data
X = np.random.rand(n,p)-0.5
Y1 = np.sign(np.sum(X**2,1)-.1).reshape((-1, 1))

Y2 = np.sign(5*X[:,[0]]**3-X[:,[1]])
Y = np.hstack((Y1, Y2))
# Plot training data for first classification problem
plt.scatter(X[:,0], X[:,1], color=['b' if i==-1 else 'r' for i in Y1[:,0]])
plt.axis('equal')
plt.title('Labeled data, first classifier')
plt.show()
# Plot training data for second classification problem
plt.scatter(X[:,0], X[:,1], color=['b' if i==-1 else 'r' for i in Y2[:,0]])
plt.title('Labeled data, second classifier')
plt.axis('equal')
plt.show()

for sigma in [5, 0.5, 0.005]:
  # Train Classifier 1
  lam = 0.01

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

  for i in range(0,n):
      for j in range(0,n):
          d = np.linalg.norm(X[i,:]-X[j,:])
          distsq[i,j]=d**2

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

  alpha = np.linalg.inv(K+lam*np.identity(n))@Y1


  # Predict labels on a grid of points

  X_grid = []
  Y_hat_grid = []

  g = 100 #number of grid points
  Y_hat_grid = np.zeros((g,g))
```

```
x1_grid = np.linspace(-.5,.5,g)
x2_grid = np.linspace(-.5,.5,g)

for i,x1 in enumerate(x1_grid):
    for j,x2 in enumerate(x2_grid):
        Y_hat_grid[i,j] = np.exp(-np.linalg.norm(X - np.array([x1,x2]), ax

plt.contour(x1_grid, x2_grid, Y_hat_grid, np.linspace(-2,2,20))
plt.colorbar()
plt.title('Prediction before thresholding, sigma = '+ str(sigma))
plt.show()

plt.contour(x1_grid, x2_grid, np.sign(Y_hat_grid), np.linspace(-2,2,20))
plt.colorbar()
plt.title('Prediction after thresholding, sigma = '+ str(sigma))

# Train Classifier 2
lam = 0.01

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

for i in range(0,n):
    for j in range(0,n):
        d = np.linalg.norm(X[i,:]-X[j,:])
        distsq[i,j]=d**2

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

alpha = np.linalg.inv(K+lam*np.identity(n))@Y2


# Predict labels on a grid of points
X_grid = []
Y_hat_grid = []

g = 100 #number of grid points
Y_hat_grid = np.zeros((g,g))

x1_grid = np.linspace(-.5,.5,g)
x2_grid = np.linspace(-.5,.5,g)

for i,x1 in enumerate(x1_grid):
    for j,x2 in enumerate(x2_grid):
        Y_hat_grid[i,j] = np.exp(-np.linalg.norm(X - np.array([x1,x2]), ax

plt.contour(x1_grid, x2_grid, Y_hat_grid, np.linspace(-2,2,20))
plt.colorbar()
plt.title('Prediction before thresholding, sigma = '+ str(sigma))
plt.show()

plt.contour(x1_grid, x2_grid, np.sign(Y_hat_grid), np.linspace(-2,2,20))
plt.colorbar()
plt.title('Prediction after thresholding, sigma = '+ str(sigma))
```

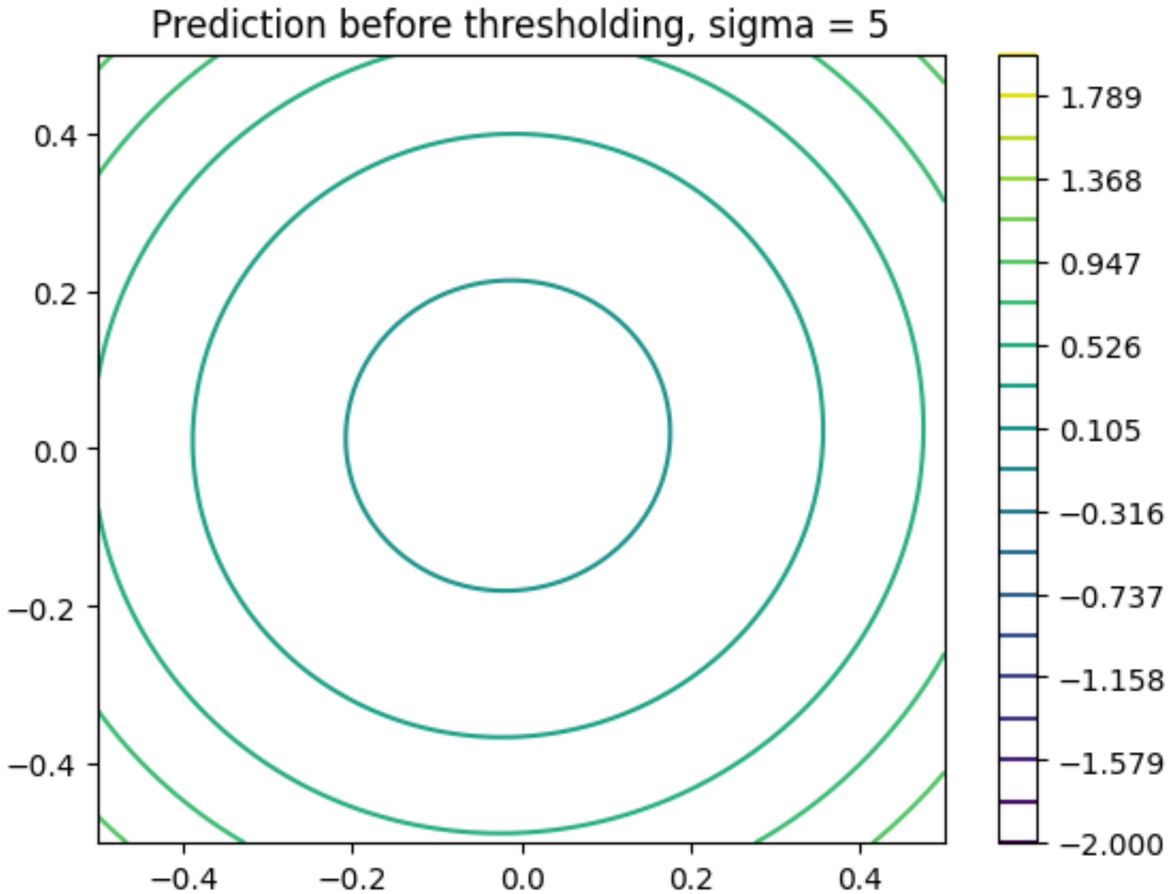


Labeled data, first classifier
0.4
0.2
0.0
−0.2
−0.4
−0.6
−0.4
−0.2
0.0
0.2
0.4
0.6



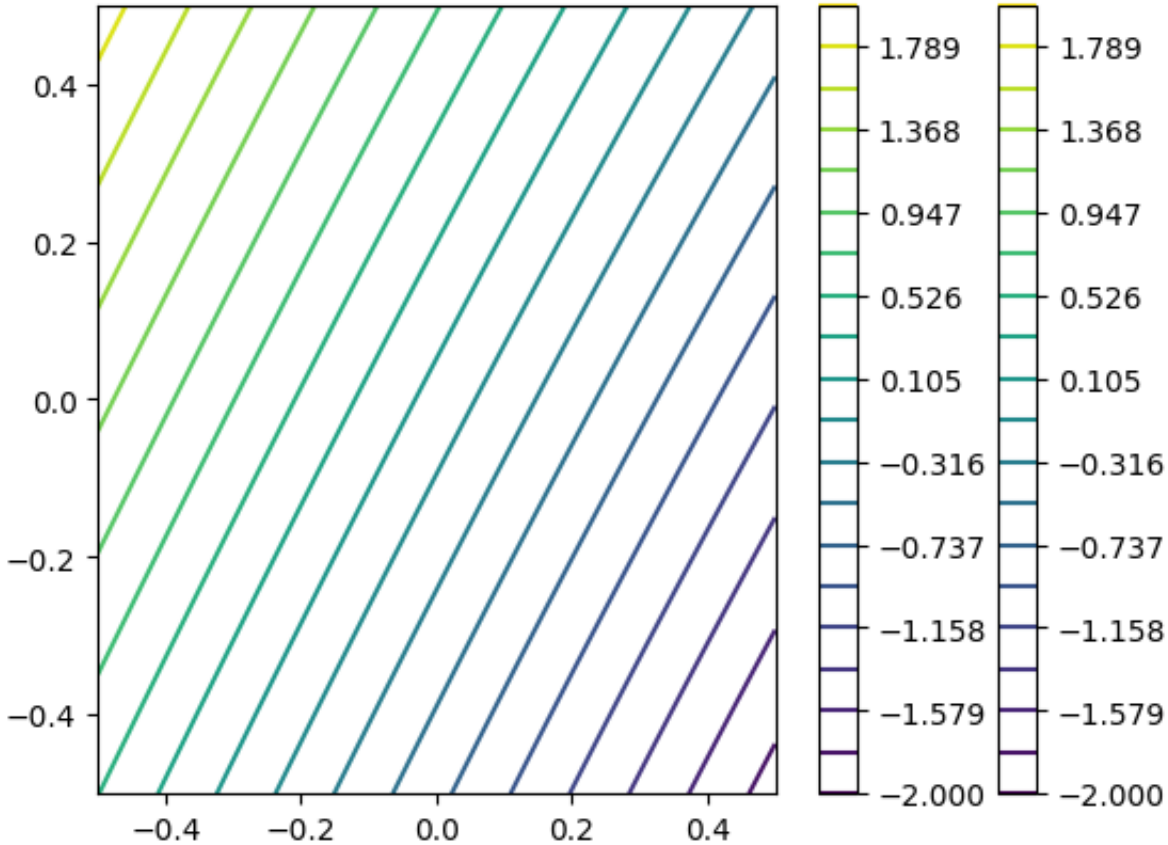
Labeled data, second classifier
0.4
0.2
0.0
−0.2
−0.4
−0.6
−0.4
−0.2
0.0
0.2
0.4
0.6

```
/var/folders/tn/v9tpvrrs4qgdbw0xd1q0l8qh0000gn/T/ipykernel_27404/802802170.p
y:53: DeprecationWarning: Conversion of an array with ndim > 0 to a scalar i
s deprecated, and will error in future. Ensure you extract a single element
from your array before performing this operation. (Deprecated NumPy 1.25.)
  Y_hat_grid[i,j] = np.exp(-np.linalg.norm(X - np.array([x1,x2]), axis =
1)**2/(2*sigma**2))@alpha
```

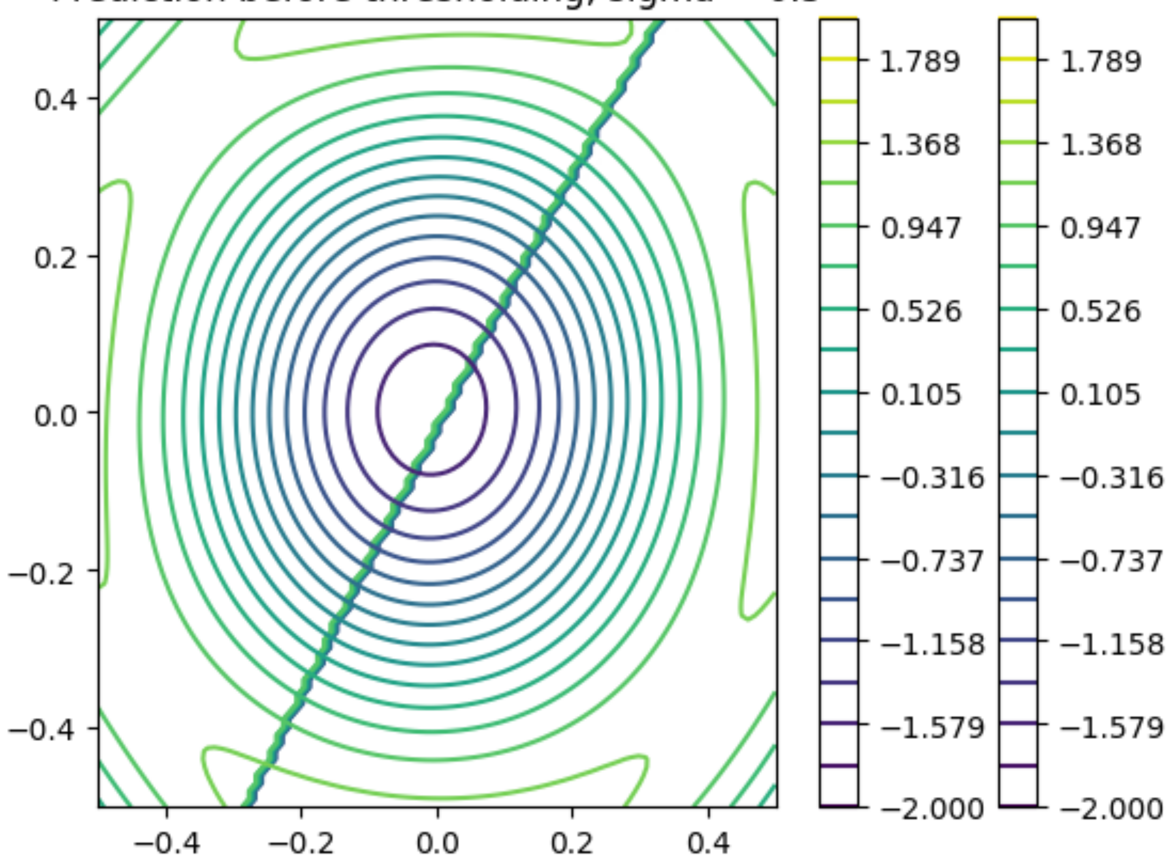



```
/var/folders/tn/v9tpvrrs4qgdbw0xd1q0l8qh0000gn/T/ipykernel_27404/802802170.p
y:91: DeprecationWarning: Conversion of an array with ndim > 0 to a scalar i
s deprecated, and will error in future. Ensure you extract a single element
from your array before performing this operation. (Deprecated NumPy 1.25.)
  Y_hat_grid[i,j] = np.exp(-np.linalg.norm(X - np.array([x1,x2]), axis =
1)**2/(2*sigma**2))@alpha
```

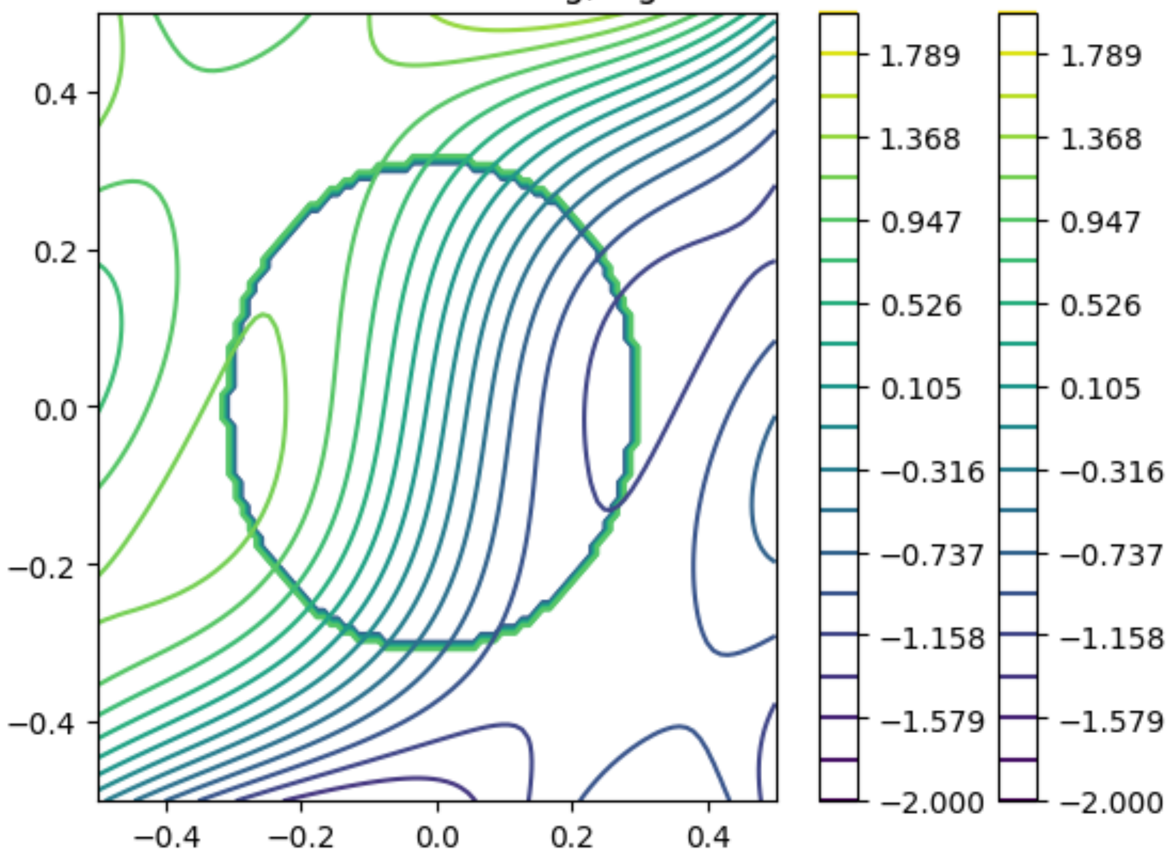
Prediction before thresholding, sigma = 5



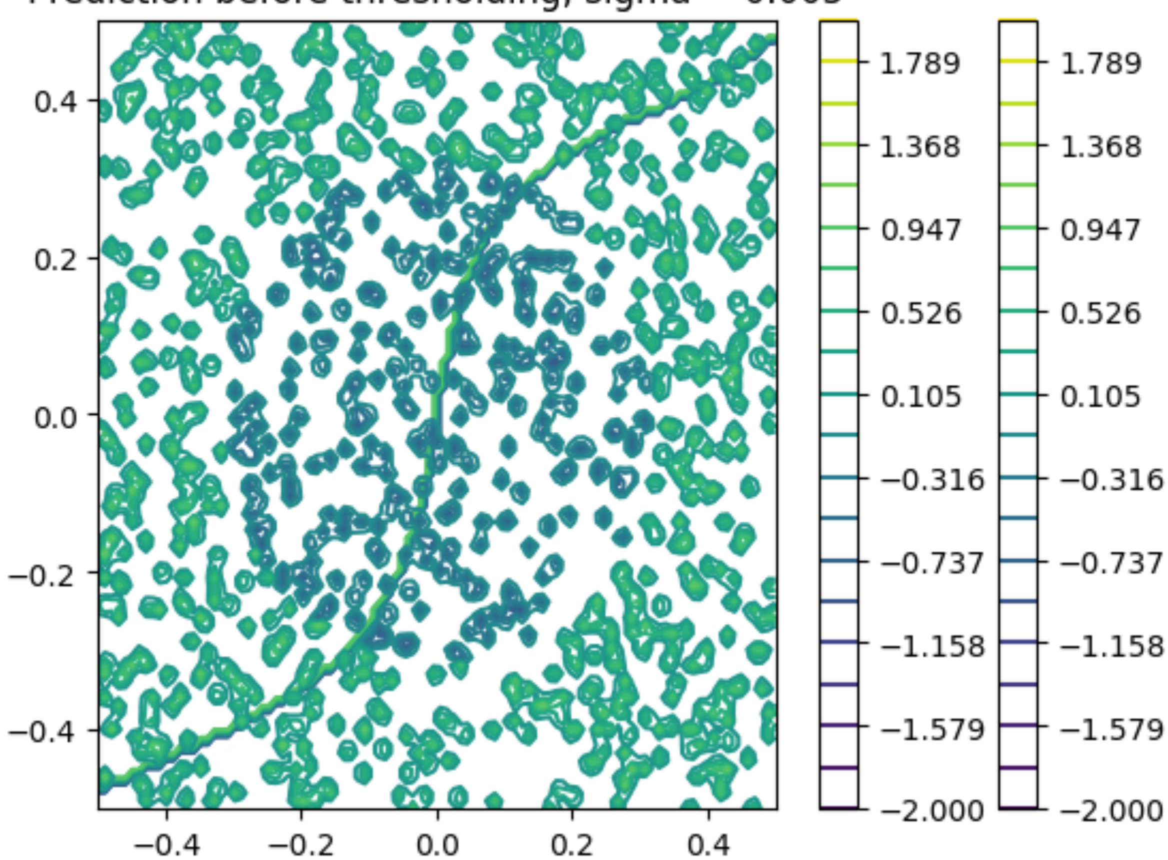
Prediction before thresholding, sigma = 0.5

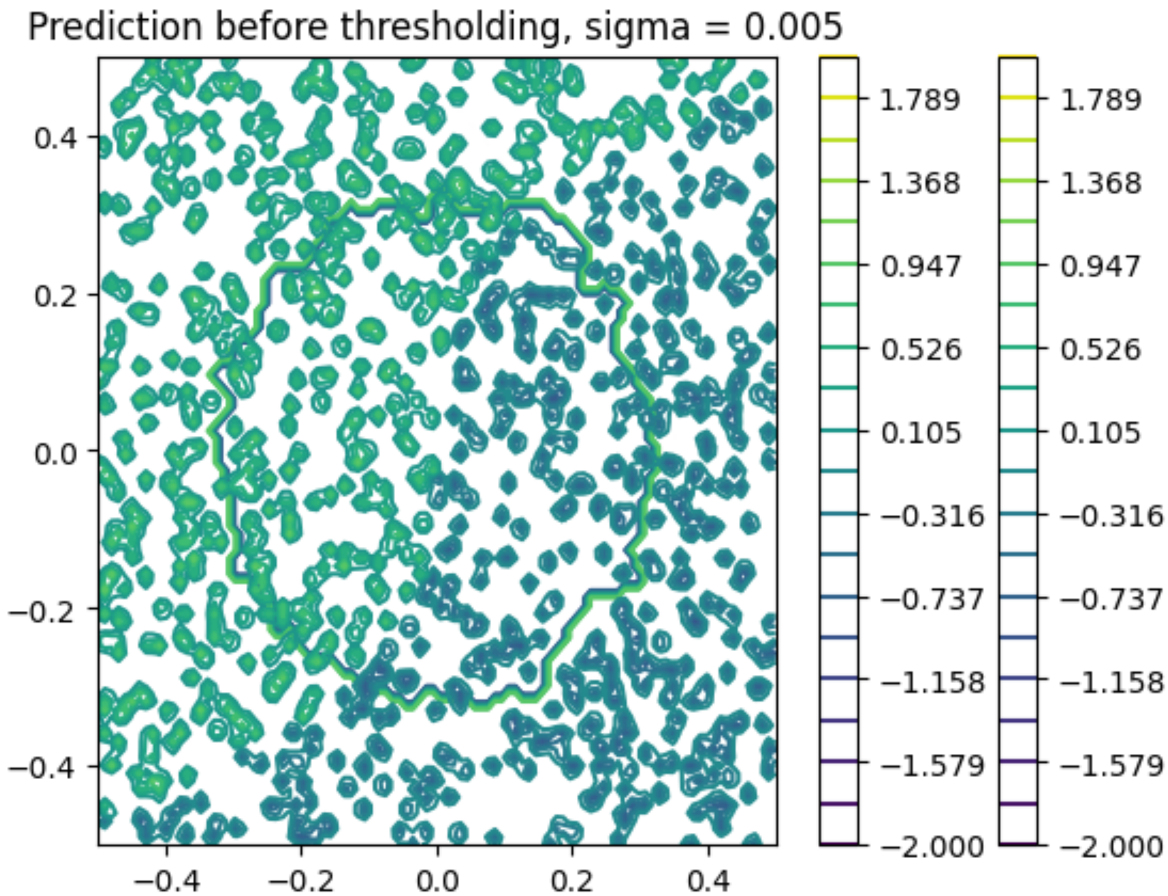


Prediction before thresholding, sigma = 0.5

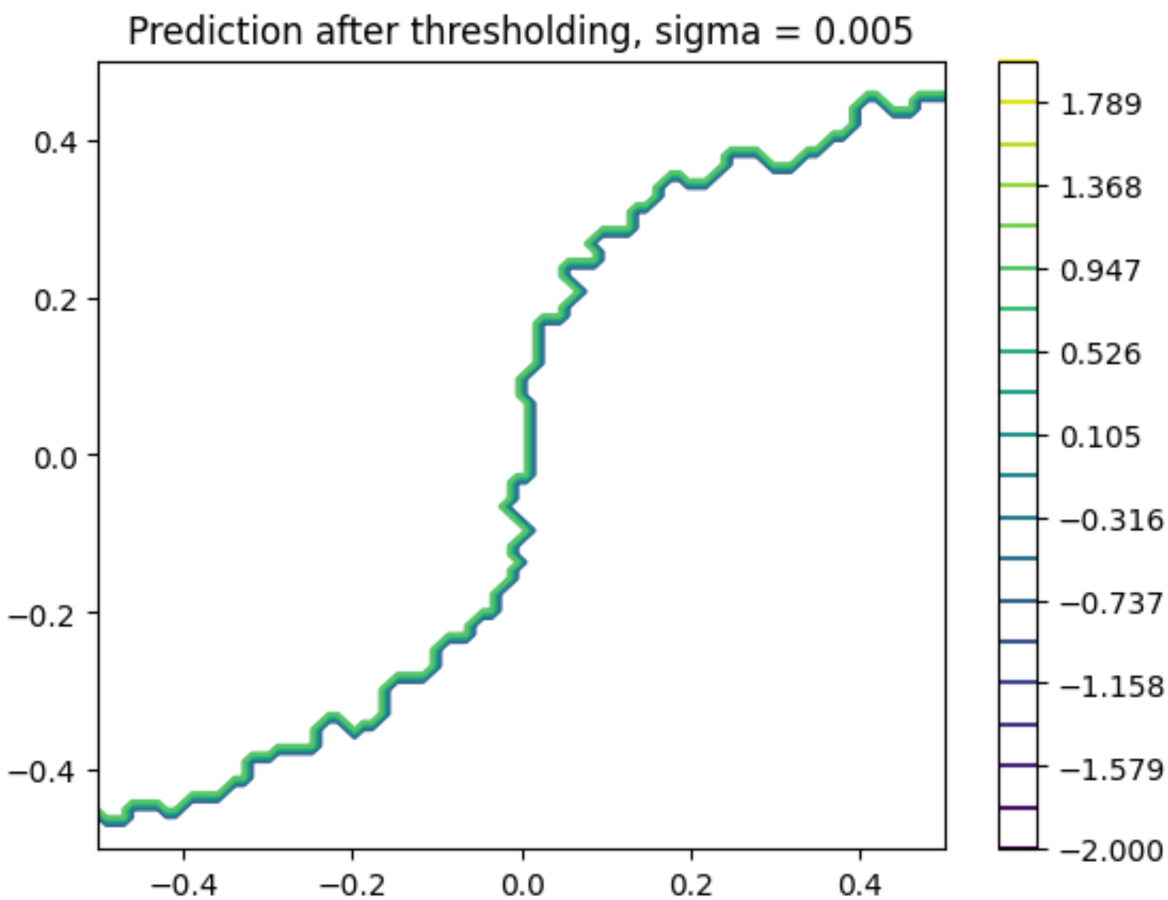


Prediction before thresholding, sigma = 0.005





Prediction before thresholding, sigma = 0.005
0.4
0.2
0.0
−0.2
−0.4
−0.4
−0.2
0.0
0.2
0.4
1.789
1.368
0.947
0.526
0.105
−0.316
−0.737
−1.158
−1.579
−2.000
1.789
1.368
0.947
0.526
0.105
−0.316
−0.737
−1.158
−1.579
−2.000



Prediction after thresholding, sigma = 0.005
0.4
0.2
0.0
−0.2
−0.4
−0.4
−0.2
0.0
0.2
0.4
1.789
1.368
0.947
0.526
0.105
−0.316
−0.737
−1.158
−1.579
−2.000