In [12]:
```
import csv
import numpy as np
import matplotlib.pyplot as plt
from cvxopt import matrix, solvers
```

Part (a) [3.5 points]:

Find the fattest margin line that separates the points in linsep.txt. Please solve the problem using a Quadratic Programming solver. Report the equation of the line as well as the support vectors.

In [115]:
```
linesp_data = []
with open('linsep.txt') as f:
    file = csv.reader(f)
    for line in file:
        linesp_data.append(line)
linesp_data = np.array(linesp_data).astype('float64')
```

In [116]:
```
X = linesp_data[:,:2]
y = linesp_data[:,2]
n_samples, n_features = X.shape
```

In [117]:
```
# Q = y y X^T X
K = np.zeros((n_samples, n_samples))
for i in range(n_samples):
    for j in range(n_samples):
        K[i,j] = np.dot(X[i], X[j])
Q = matrix(np.outer(y, y) * K)
```

In [118]:
```
q = matrix(np.ones((n_samples, 1)) * -1)
A = matrix(y.reshape(1, -1))
b = matrix(np.zeros(1))
G = matrix(np.eye(n_samples) * -1)
h = matrix(np.zeros(n_samples))
```

In [119]:
```
solution = solvers.qp(Q, q, G, h, A, b)
# find the support vectors
alphas = np.array(solution['x'])
idx = (alphas > 1e-4).flatten()
sv = X[idx]
sv_y = y[idx]
alphas = alphas[idx]
```

```
     pcost       dcost       gap    pres   dres
 0: -2.0636e+01 -4.3905e+01  3e+02  2e+01  2e+00
 1: -2.2372e+01 -3.7202e+01  9e+01  5e+00  5e-01
 2: -2.3112e+01 -3.8857e+01  5e+01  2e+00  2e-01
 3: -2.8318e+01 -3.3963e+01  1e+01  4e-01  4e-02
 4: -3.2264e+01 -3.3927e+01  2e+00  1e-02  1e-03
 5: -3.3568e+01 -3.3764e+01  2e-01  1e-03  1e-04
 6: -3.3737e+01 -3.3739e+01  2e-03  1e-05  1e-06
 7: -3.3739e+01 -3.3739e+01  2e-05  1e-07  1e-08
 8: -3.3739e+01 -3.3739e+01  2e-07  1e-09  1e-10
Optimal solution found.
```

In [120]:
```python
# print the support vector
print('---alphas---')
print(alphas)
print('-----sv-----')
print(sv)
print('----sv_y----')
print(sv_y)
```

```
---alphas---
[[33.73875192]
 [ 1.29468506]
 [32.4440672 ]]
-----sv-----
[[0.24979414 0.18230306]
 [0.3917889  0.96675591]
 [0.02066458 0.27003158]]
----sv_y----
[ 1. -1. -1.]
```

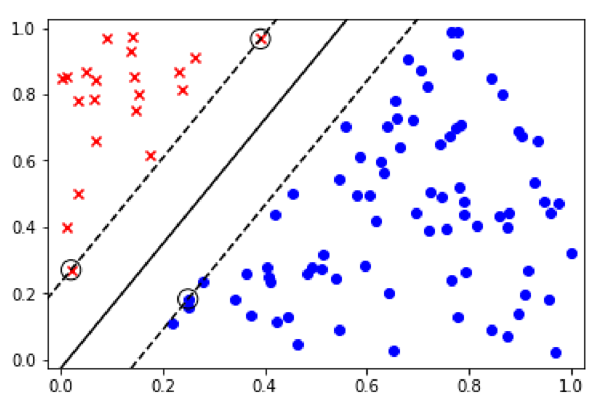
In [121]:
```python
# Calculate w (exclude alpha = 0)
w = np.zeros(n_features)
for n in range(len(alphas)):
    w += alphas[n]*sv_y[n]*sv[n]
w
```

Out[121]:
```
array([ 7.25005616, -3.86188932])
```

In [102]:
```python
# Calculate b
b = sv_y[0] - np.dot(sv[0], w)
b
```

Out[102]:
```
-0.1069872903246214
```

In [103]:
```python
# plot all data points and mark them respectively
pos_idx = np.where(linesp_data[:,2]==1)
neg_idx = np.where(linesp_data[:,2]==-1)
X_pos = linesp_data[pos_idx]
X_neg = linesp_data[neg_idx]
plt.scatter(X_pos[:,0], X_pos[:,1], marker = 'o', color = 'b', label = 'Positive +1')
plt.scatter(X_neg[:,0], X_neg[:,1], marker = 'x', color = 'r', label = 'Negative -1')
# plot maximum margin separating hyperplane
x_plane = np.linspace(-1,1,100)
y_plane = (-b-w[0]*x_plane)/w[1]
plt.plot(x_plane, y_plane, color='black')
plt.plot(x_plane, (1/w[1])+y_plane, '--', color='black')
plt.plot(x_plane, (-1/w[1])+y_plane, '--', color='black')
# circle the support vecotor(sv)
plt.scatter(sv[:,0], sv[:,1], s=150, facecolors='none', edgecolors='k')
plt.xlim(-0.025, 1.025)
plt.ylim(-0.025, 1.025)
plt.show()
```



Part (b) [3.5 points]:

Using a kernel function of your choice along with the same Quadratic Programming solver, find the equation of a curve that separates the points in nonlinsep.txt. Report the kernel function you use as well as the support vectors.

In [161]:
```
nonlinesp_data = []
with open('nonlinsep.txt') as f:
    file = csv.reader(f)
    for line in file:
        nonlinesp_data.append(line)
nonlinesp_data = np.array(nonlinesp_data).astype('float64')
```

In [162]:
```
X = nonlinesp_data[:,:2]
y = nonlinesp_data[:,2]
n_samples, n_features = X.shape
```

In [163]:
```
def RBF_kernel_function(x1, x2, gamma=0.01):
    return np.exp(-gamma * np.linalg.norm(x1 - x2) ** 2)
```

In [164]:
```
# new Q from kernel function(K)
K = np.zeros((n_samples, n_samples))
for i in range(n_samples):
    for j in range(n_samples):
        K[i,j] = RBF_kernel_function(X[i], X[j])
Q = matrix(np.outer(y, y) * K)
```

In [165]:
```
q = matrix(np.ones((n_samples, 1)) * -1)
A = matrix(y.reshape(1, -1))
b = matrix(np.zeros(1))
G = matrix(np.eye(n_samples) * -1)
h = matrix(np.zeros(n_samples))
```

In [166]:
```
solution = solvers.qp(Q, q, G, h, A, b)
alphas = np.array(solution['x'])
# find the support vectors
ind = (alphas > 1e-4).flatten()
sv = X[ind]
sv_y = y[ind]
alphas = alphas[ind]
```

```
     pcost       dcost       gap    pres   dres
 0: -1.6406e+01 -4.8454e+01  3e+02  1e+01  2e+00
 1: -3.2833e+01 -6.7668e+01  1e+02  4e+00  8e-01
 2: -9.1774e+01 -1.2550e+02  1e+02  4e+00  8e-01
 3: -1.8108e+02 -2.2278e+02  1e+02  3e+00  5e-01
 4: -1.8669e+02 -2.0570e+02  4e+01  6e-01  1e-01
 5: -1.8547e+02 -1.8731e+02  3e+00  3e-02  6e-03
 6: -1.8616e+02 -1.8620e+02  6e-02  4e-04  6e-05
 7: -1.8619e+02 -1.8619e+02  6e-04  4e-06  7e-07
 8: -1.8619e+02 -1.8619e+02  6e-06  4e-08  7e-09
Optimal solution found.
```

In [167]:

```
# print the support vector
print('---alphas---')
print(alphas)
print('-----sv-----')
print(sv)
print('----sv_y----')
print(sv_y)
```

```
---alphas---
[[  4.90610978]
 [154.79716375]
 [  6.06570051]
 [ 11.36810383]
 [  9.05592846]
 [ 16.36480164]
 [141.11823703]
 [  9.22492878]
 [ 12.26624767]
 [  7.21879402]]
-----sv-----
[[ 12.74780931   0.19913032]
 [-10.260969     2.07391791]
 [  1.66404809  12.68562818]
 [  1.3393313  -10.29098822]
 [  9.67917724   4.3759541 ]
 [ -9.53754332  -0.51895777]
 [ -9.46760885   2.36139525]
 [  0.20162846  -8.81260121]
 [  9.90143538  -0.31483149]
 [  4.27289989   8.67079427]]
----sv_y----
[ 1.  1.  1.  1.  1. -1. -1. -1. -1. -1.]
```

In [154]:

```
# plot all data points and mark them respectively
pos_idx = np.where(nonlinesp_data[:,2]==1)
neg_idx = np.where(nonlinesp_data[:,2]==-1)
X_pos = nonlinesp_data[pos_idx]
X_neg = nonlinesp_data[neg_idx]
plt.scatter(X_pos[:,0], X_pos[:,1], marker = 'o', color = 'b', label = 'Positive +1')
plt.scatter(X_neg[:,0], X_neg[:,1], marker = 'x', color = 'r', label = 'Negative -1')
# circle the support vecotor(sv)
plt.scatter(sv[:,0], sv[:,1], s=150, facecolors='none', edgecolors='k')
plt.show()
```