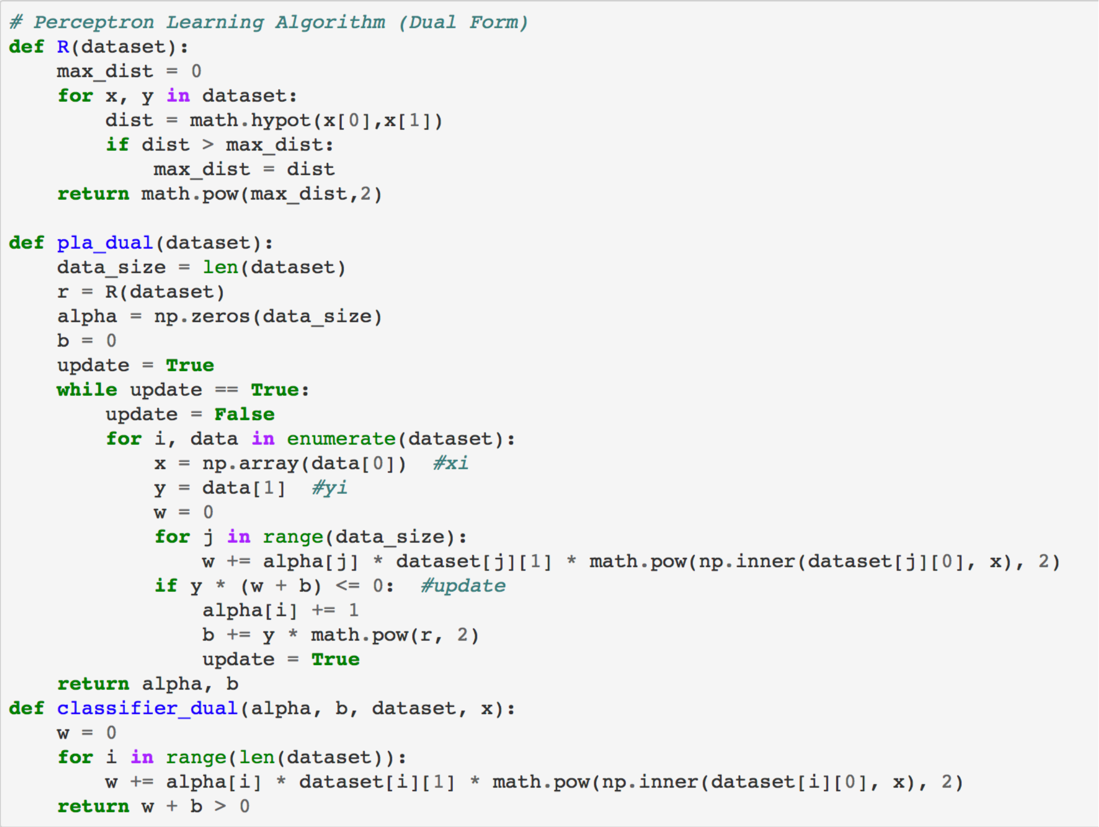
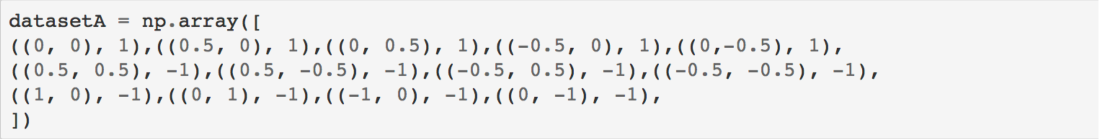
4(a)

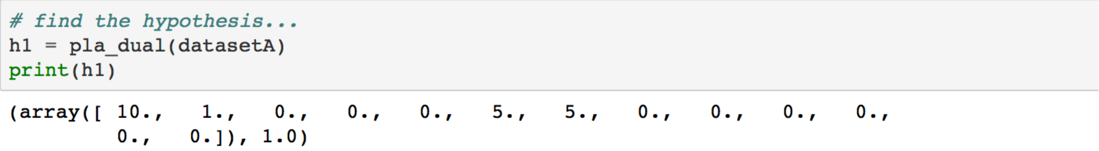


```
# Perceptron Learning Algorithm (Dual Form)
def R(dataset):
    max_dist = 0
    for x, y in dataset:
        dist = math.hypot(x[0],x[1])
        if dist > max_dist:
            max_dist = dist
    return math.pow(max_dist,2)

def pla_dual(dataset):
    data_size = len(dataset)
    r = R(dataset)
    alpha = np.zeros(data_size)
    b = 0
    update = True
    while update == True:
        update = False
        for i, data in enumerate(dataset):
            x = np.array(data[0])  #xi
            y = data[1]  #yi
            w = 0
            for j in range(data_size):
                w += alpha[j] * dataset[j][1] * math.pow(np.inner(dataset[j][0], x), 2)
            if y * (w + b) <= 0:  #update
                alpha[i] += 1
                b += y * math.pow(r, 2)
                update = True
    return alpha, b
def classifier_dual(alpha, b, dataset, x):
    w = 0
    for i in range(len(dataset)):
        w += alpha[i] * dataset[i][1] * math.pow(np.inner(dataset[i][0], x), 2)
    return w + b > 0
```



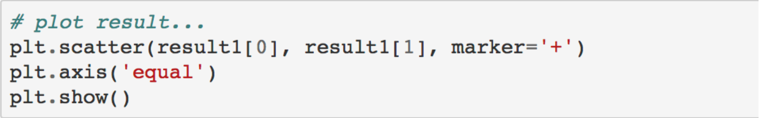
```
datasetA = np.array([
((0, 0), 1),((0.5, 0), 1),((0, 0.5), 1),((-0.5, 0), 1),((0,-0.5), 1),
((0.5, 0.5), -1),((0.5, -0.5), -1),((-0.5, 0.5), -1),((-0.5, -0.5), -1),
((1, 0), -1),((0, 1), -1),((-1, 0), -1),((0, -1), -1),
])
```



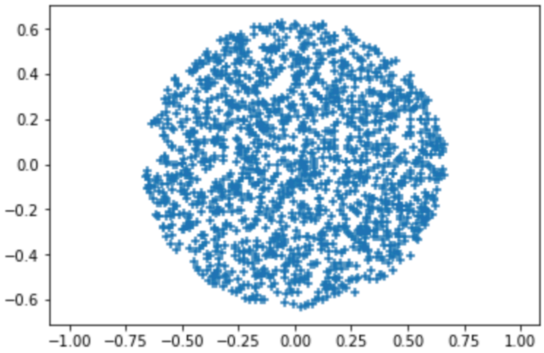
```
# find the hypothesis...
h1 = pla_dual(datasetA)
print(h1)
```

```
(array([ 10.,   1.,   0.,   0.,   0.,   5.,   5.,   0.,   0.,   0.,   0.,
          0.,   0.]), 1.0)
```

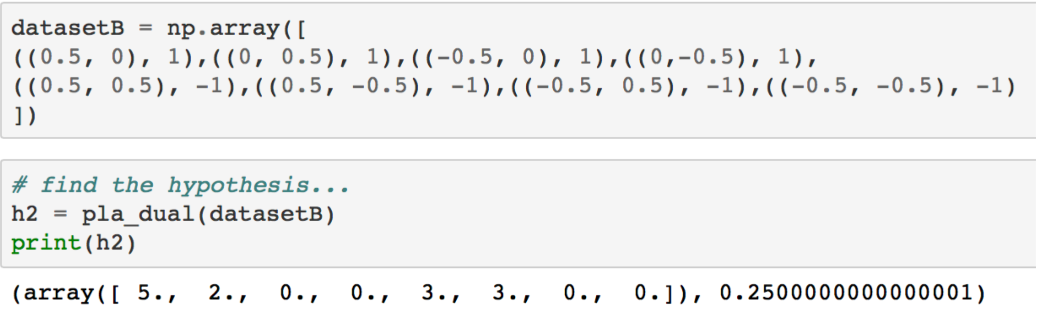
4(b)



```
# plot result...
plt.scatter(result1[0], result1[1], marker='+')
plt.axis('equal')
plt.show()
```



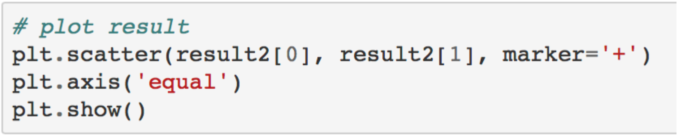
4(c)



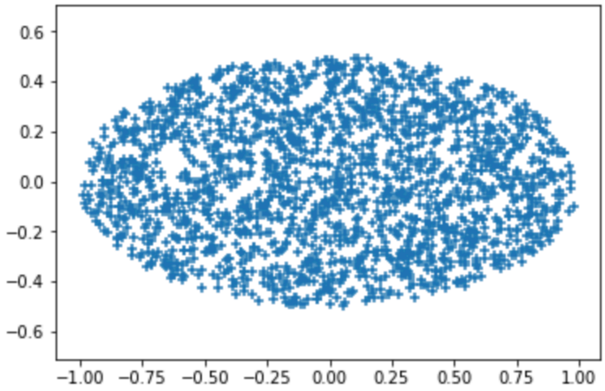
```
datasetB = np.array([
((0.5, 0), 1),((0, 0.5), 1),((-0.5, 0), 1),((0,-0.5), 1),
((0.5, 0.5), -1),((0.5, -0.5), -1),((-0.5, 0.5), -1),((-0.5, -0.5), -1)
])
```

```
# find the hypothesis...
h2 = pla_dual(datasetB)
print(h2)
```

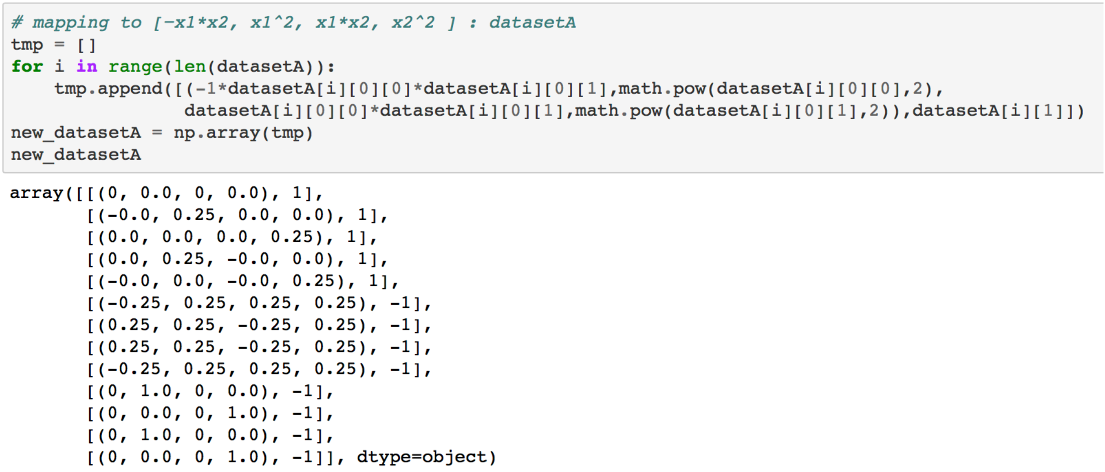
```
(array([ 5.,  2.,  0.,  0.,  3.,  3.,  0.,  0.]), 0.2500000000000001)
```



```
# plot result
plt.scatter(result2[0], result2[1], marker='+')
plt.axis('equal')
plt.show()
```

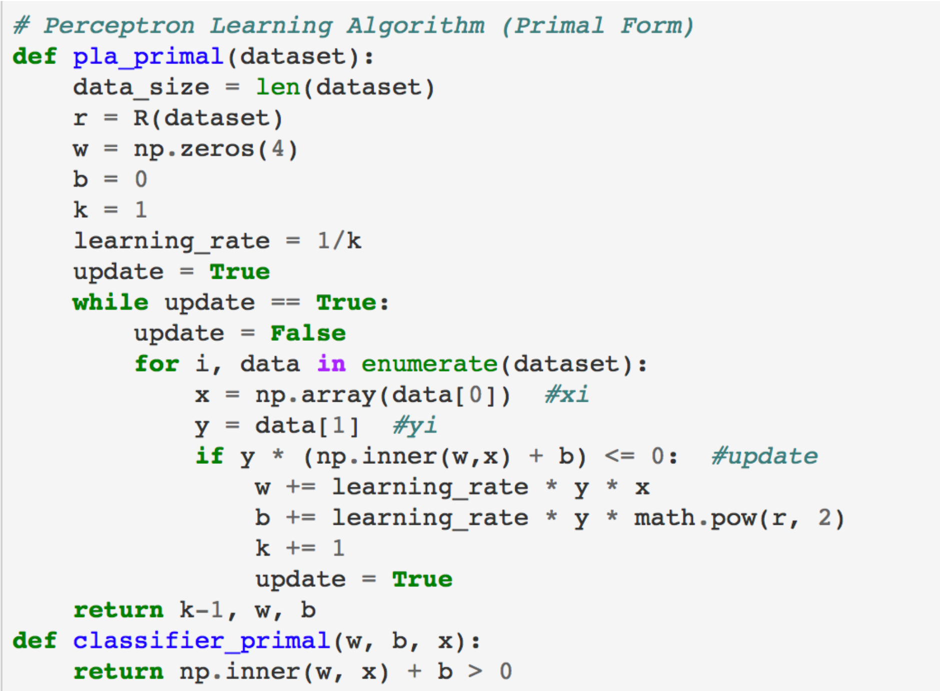


4(d)

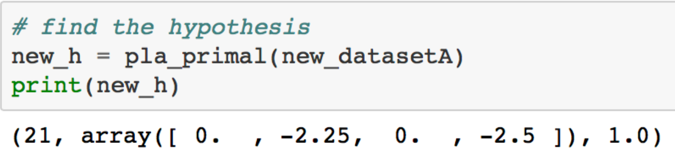


```
# mapping to [-x1*x2, x1^2, x1*x2, x2^2 ] : datasetA
tmp = []
for i in range(len(datasetA)):
    tmp.append([(-1*datasetA[i][0][0]*datasetA[i][0][1],math.pow(datasetA[i][0][0],2),
                datasetA[i][0][0]*datasetA[i][0][1],math.pow(datasetA[i][0][1],2)),datasetA[i][1]])
new_datasetA = np.array(tmp)
new_datasetA
```

```
array([[(0, 0.0, 0, 0.0), 1],
       [(-0.0, 0.25, 0.0, 0.0), 1],
       [(0.0, 0.0, 0.0, 0.25), 1],
       [(0.0, 0.25, -0.0, 0.0), 1],
       [(-0.0, 0.0, -0.0, 0.25), 1],
       [(-0.25, 0.25, 0.25, 0.25), -1],
       [(0.25, 0.25, -0.25, 0.25), -1],
       [(0.25, 0.25, -0.25, 0.25), -1],
       [(-0.25, 0.25, 0.25, 0.25), -1],
       [(0, 1.0, 0, 0.0), -1],
       [(0, 0.0, 0, 1.0), -1],
       [(0, 1.0, 0, 0.0), -1],
       [(0, 0.0, 0, 1.0), -1]], dtype=object)
```



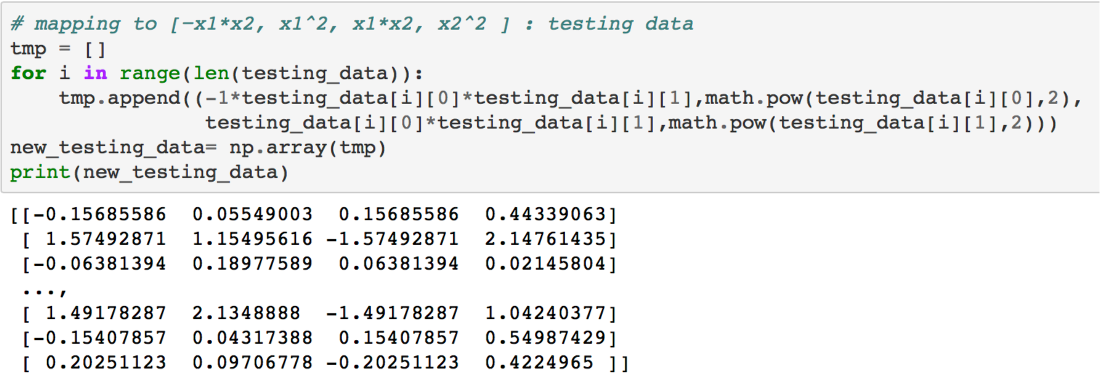
```
# Perceptron Learning Algorithm (Primal Form)
def pla_primal(dataset):
    data_size = len(dataset)
    r = R(dataset)
    w = np.zeros(4)
    b = 0
    k = 1
    learning_rate = 1/k
    update = True
    while update == True:
        update = False
        for i, data in enumerate(dataset):
            x = np.array(data[0])  #xi
            y = data[1]  #yi
            if y * (np.inner(w,x) + b) <= 0:  #update
                w += learning_rate * y * x
                b += learning_rate * y * math.pow(r, 2)
                k += 1
                update = True
    return k-1, w, b
def classifier_primal(w, b, x):
    return np.inner(w, x) + b > 0
```



```
# find the hypothesis
new_h = pla_primal(new_datasetA)
print(new_h)
```

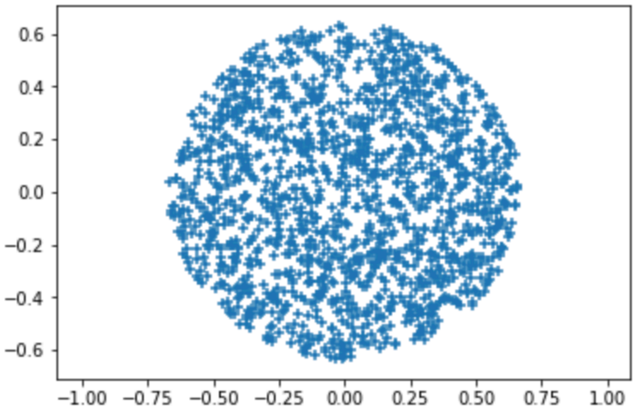
```
(21, array([ 0.  , -2.25,  0.  , -2.5 ]), 1.0)
```

4(e)



```
# mapping to [-x1*x2, x1^2, x1*x2, x2^2 ] : testing data
tmp = []
for i in range(len(testing_data)):
    tmp.append((-1*testing_data[i][0]*testing_data[i][1],math.pow(testing_data[i][0],2),
                testing_data[i][0]*testing_data[i][1],math.pow(testing_data[i][1],2)))
new_testing_data= np.array(tmp)
print(new_testing_data)
```

```
[[-0.15685586  0.05549003  0.15685586  0.44339063]
 [ 1.57492871  1.15495616 -1.57492871  2.14761435]
 [-0.06381394  0.18977589  0.06381394  0.02145804]
 ...,
 [ 1.49178287  2.1348888  -1.49178287  1.04240377]
 [-0.15407857  0.04317388  0.15407857  0.54987429]
 [ 0.20251123  0.09706778 -0.20251123  0.4224965 ]]
```


0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00