Homework 3

1. Generate 1000 realizations of a random variable uniformly distributed in [0,1]. Approximate the corresponding pdf.

Notice that $f(x) \approx \int f(y)K_h(y-x)dy = Ey(K_h(y-x)) \approx \frac{\sum_{i=1}^{N} K_h(y_i-x)}{N}$.
We can use $\frac{\sum_{i=1}^{N} K_h(y_i-x)}{N}$ to approxmate $f(x)$

a) Using the Gaussian kernel

Here we have $K_h(x) = \frac{1}{\sqrt{2\pi h}} e^{\frac{1}{2h}x^2}$
Therefore, we can use `Q11.m` to approximate.

Fig.1.1 is the distribution of 1000 realizations, and Fig.1.2 is the resulting approximations with 3 different value of h.

```
variance =

    0.0002    0.0093    0.0283


support =

    3.0000    1.6240    1.0600
```

Note: when $h = 1$, here we said $support = 3.0000$, but actually it is much larger than 3 because we just focus on $x \in [-1, 2]$.

Notice that as h gets smaller and smaller, though the support gets closer and closer to the interval $[0, 1]$, the pdf in $[0, 1]$ becomes less and less "smooth", which seems like Gibbs phenomenon or overfitting.

b) Using the Laplacian kernel

Now $K_h(x) = \frac{1}{2h} e^{\frac{1}{h}|x|}$, and we can use `Q12.m`.

Fig.1.3 is the distribution of 1000 realizations, and Fig.1.4 is the resulting approximations with 3 different value of h.

```
variance =

    0.0005    0.0130    0.1119


support =

    3.0000    1.6220    1.0300
```

Note: when $h = 1$, here we said $support = 3.0000$, but actually it is much larger than 3 because we just focus on $x \in [-1, 2]$.

Notice that Laplacian kernel is more sensitive to h than Gaussian kernel.

2. Develop a classifier that distinguishes between the two sets

Explain what this minimization tries to achieve.

This minimization can be divided into 2 parts:
$\sum_{i=1}^{N} l(y_i, \phi(X_i))$: the distance between $\phi(X_i)$ and y_i, which is the first 2 term in this minimization.
$\lambda||\phi(X)||^2$: a factor (that smooth the boundary), which is the last term in this minimization.

Here we have $\phi(X) = \sum_{i=1}^{M} a_i K(X, Z_i)$
Therefore, inorder to find the optimum $\phi(x)$, we have to find the optimizer a_i, Z_i, where $i = 1, ..., M$, and M.

a) find the optimum $\phi(x)$

By theorem, the optimum M is N, which is the number of X, and Z_i is X_i, where $i = 1, ..., N$.
Therefore, all we have to find is the optimum a_i.

Let $C = \sum_{i=1}^{N} l(y_i, \phi(X_i)) + \lambda||\phi(X)||^2$.
$C = ||y - Ka||^2 + \lambda a^T K a$, where:
$y = [y_1, y_2, ..., y_N]^T$
$a = [a_1, a_2, ..., a_N]^T$
K is a matrix whose element $k_{ij} = K(X_i, X_j)$.

Notice that K is symmetirc because $K(X_i, X_j) = K(X_j, X_i)$
Since $K(x, y)$ is a positive definite function, we can prove K is a positive definite matrix by Mercer's theorem.

Therefore, $\nabla C = -2K^T(y - Ka) + 2\lambda K a$
Let $\nabla C = 0$. Since K is symmetric and positive definite:
$(K + \lambda I)a = y$
$a = (\lambda I + K)^{-1} y$

Therefore, $\phi(X) = \sum_{i=1}^{N} a_i K(X, X_i)$

We can use `Q21.m` to compute a and $\phi(x)$. The result of a is attached in `Q21-a.txt`.

b) how you are going to use it to classify a new point Xnew

Compute $\phi(X_{new})$, and compare it with 0:
If $\phi(X_{new}) > 0$, classify it as `stars`;
If $\phi(X_{new}) < 0$, classify it as `circles`;
If $\phi(X_{new}) = 0$, classify it as any class we like.

c) the separating boundary

We can use `Q21.m` to plot the boundary.
Notice there is a balance between overfitting and mis-classification. But at least, we can separate stars and circles.

d) Repeat the same process for different values of h and λ.

Fig.2.1, Fig.2.2, and Fig.2.3 are 9 different boundaries with 3 different h and 3 different λ.

e) the far simpler kernel function

We can use `Q22.m` to compute a and $\phi(x)$. The result of a is attached in `Q22-a.txt`.

The mechanism and the boundary of classification are still the same.

Fig.2.4 is 3 different boundaries with 3 different λ
Notice we cannot exactly separate stars and circles in this case. Because the space of $\phi(X)$ is not that "rich" any more.

Here $\forall f(X) \in \phi(X)$ can be written as $f(X) = b_0 + b_1 x_1 + b_2 x_2 + b_3 x_1^2 + b_4 x_2^2 + b_5 x_1 x_2$, where $b_i \in R, i = 1, 2, 3, 4, 5$.
Namely, the Hilbert space generated by this kernel $\{\phi(X)\} = \{b_0 + b_1 x_1 + b_2 x_2 + b_3 x_1^2 + b_4 x_2^2 + b_5 x_1 x_2 | b_i \in R, i = 1, 2, 3, 4, 5\}$

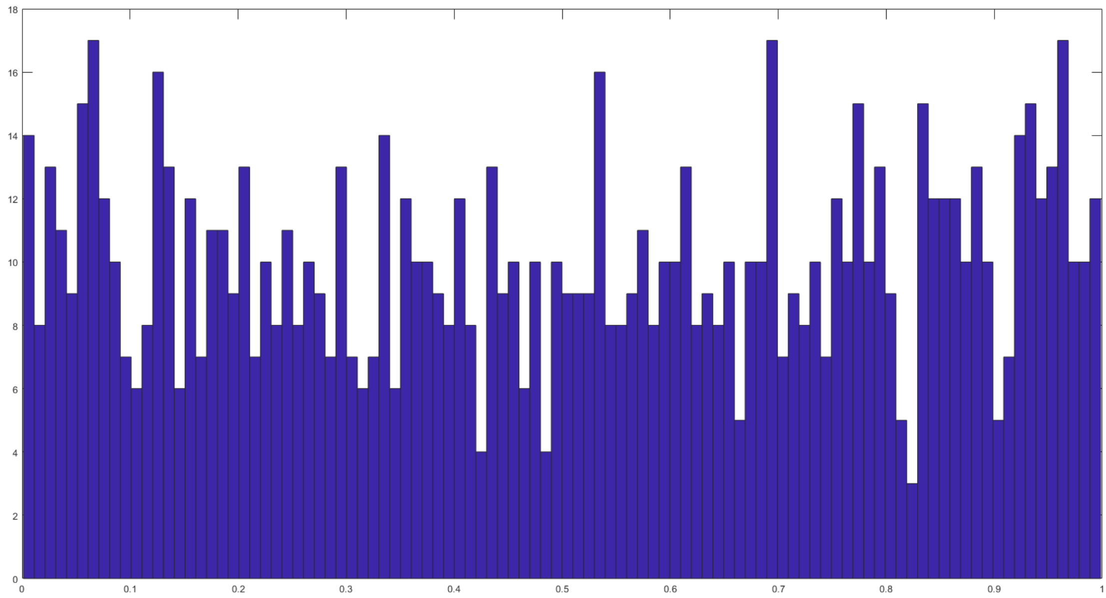

Fig.1.1 the distribution of 1000 random variables in Question 1

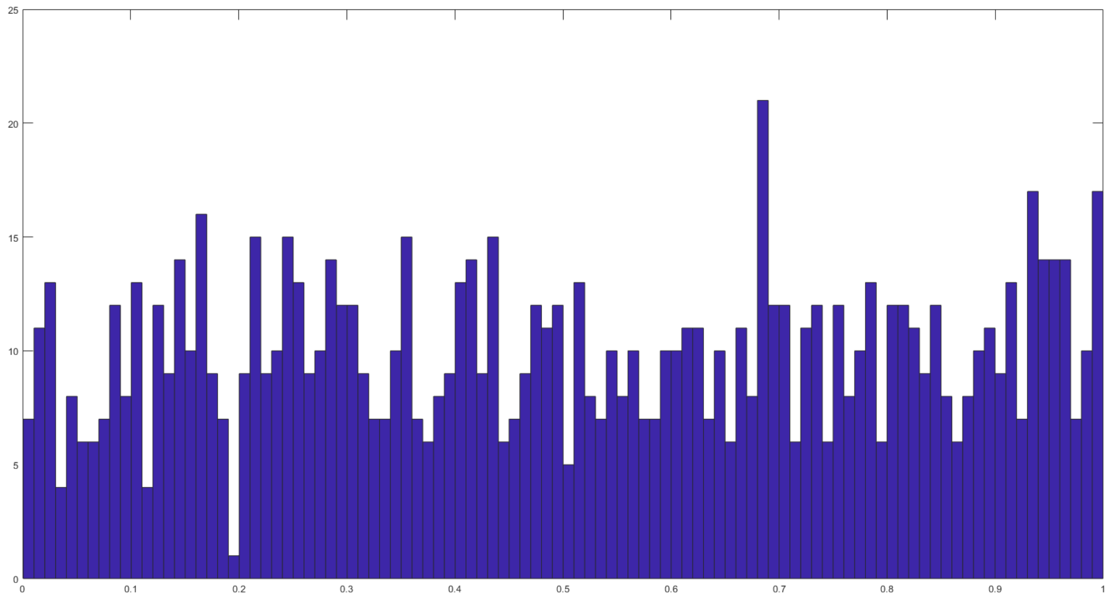

Fig.1.3 the distribution of 1000 random variables in Question 2

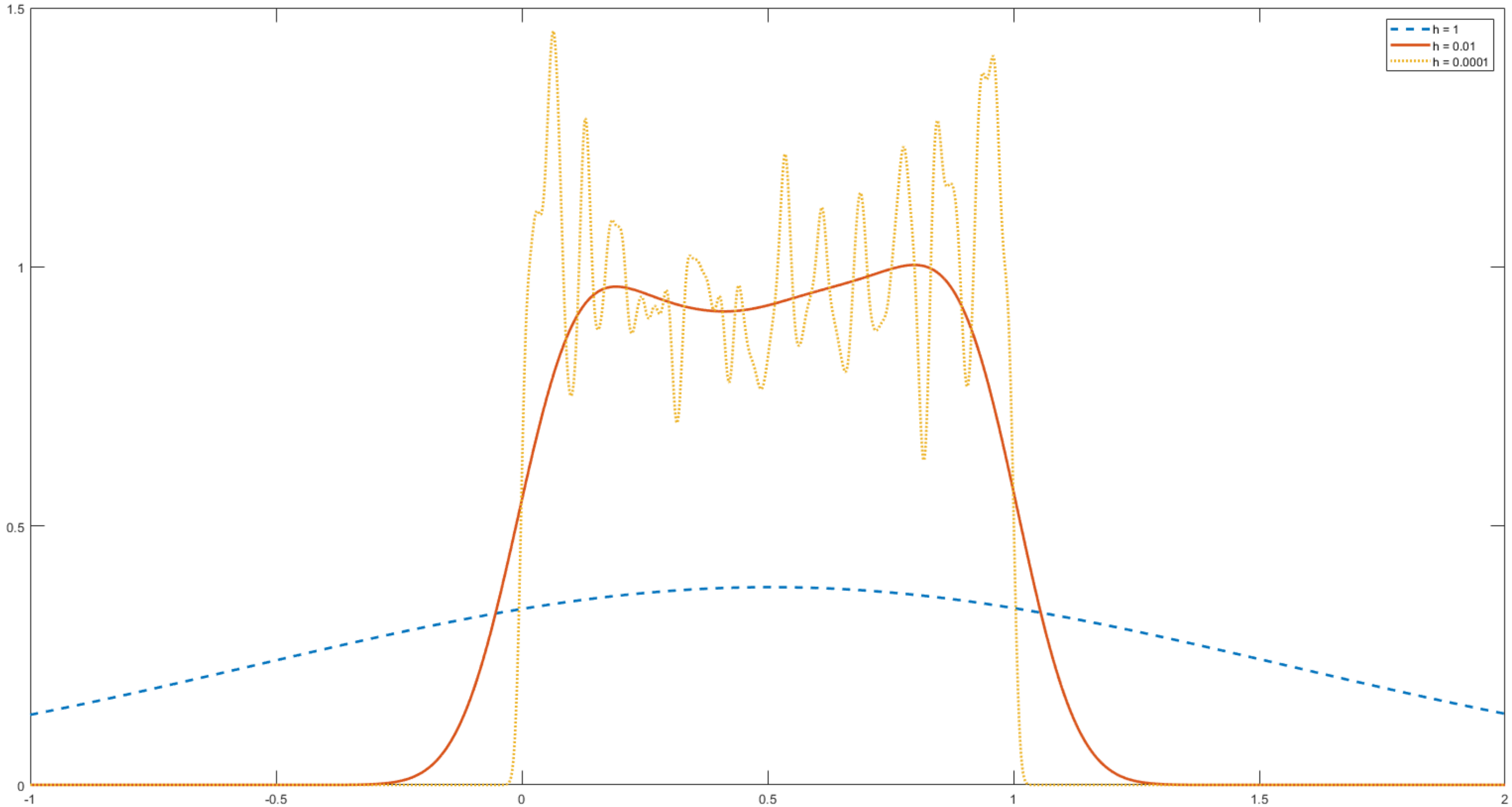

Fig.1.2 the resulting approximations by using Gaussian kernel.

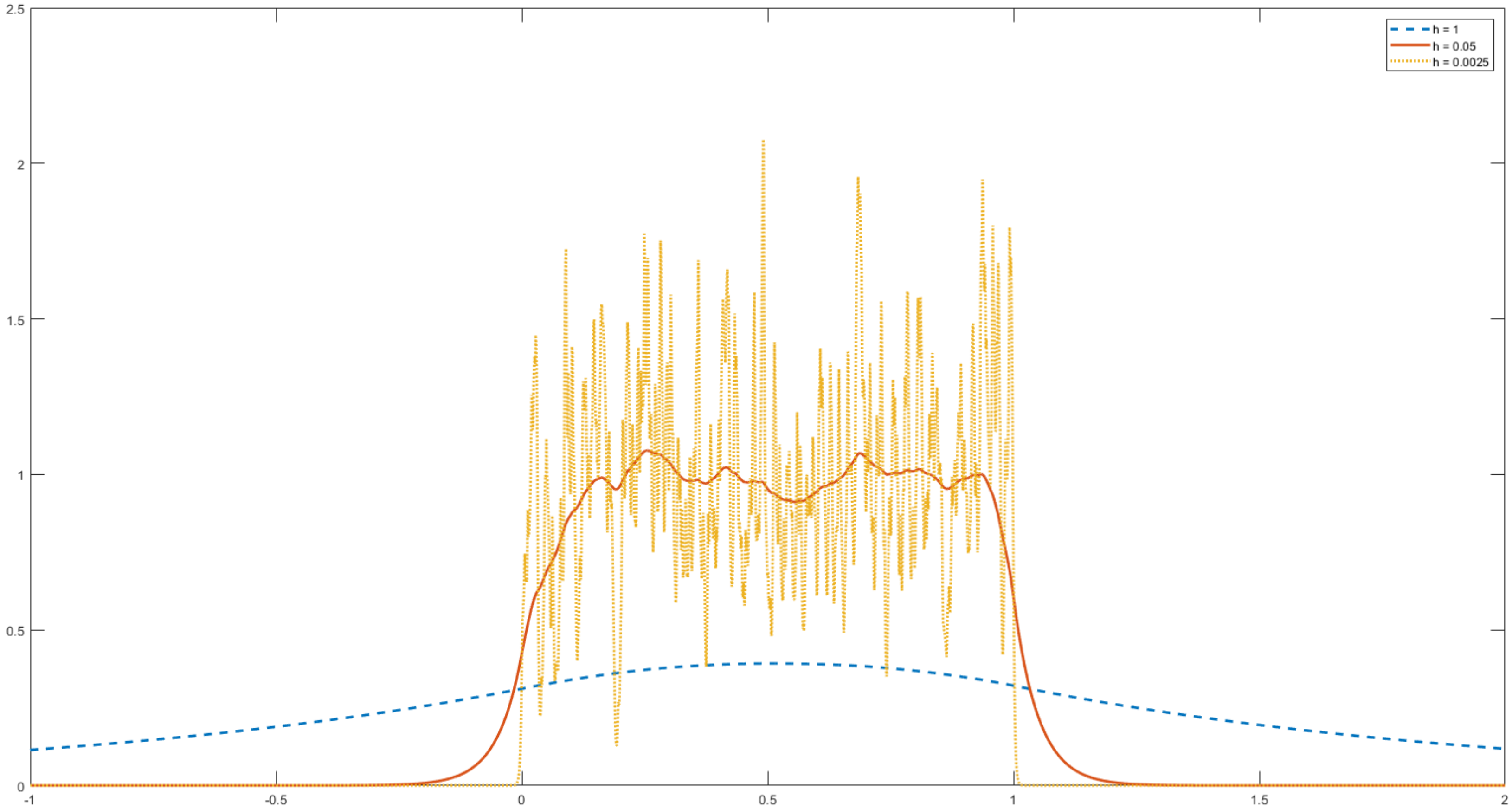

Fig.1.4 the resulting approximations by using Laplacian kernel.

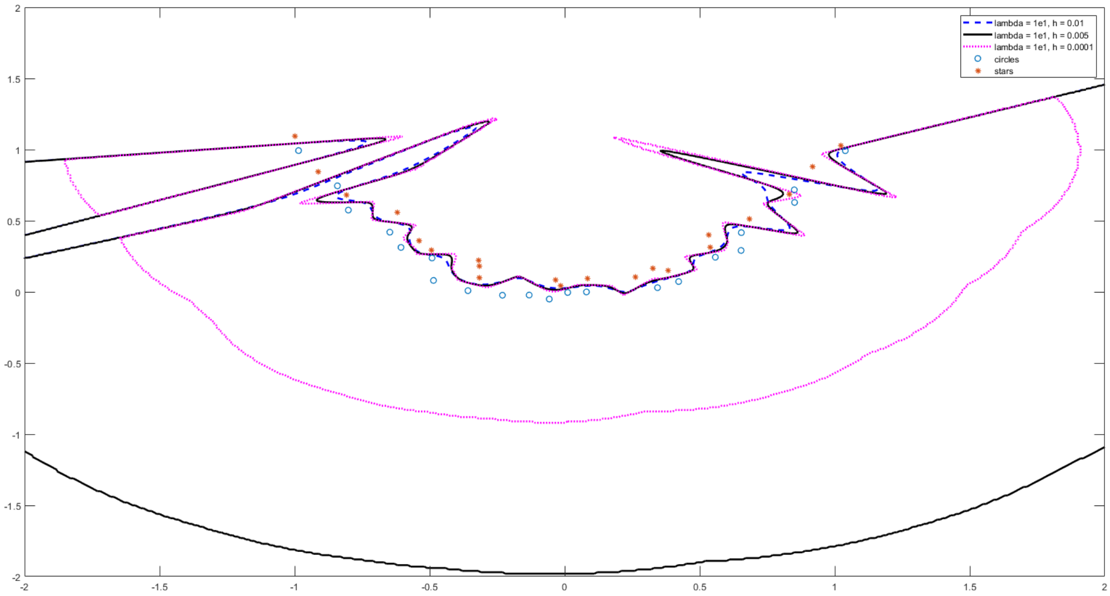

Fig.2.1 the boundary between stars and circles using the Gaussian kernel with $\lambda = 10$

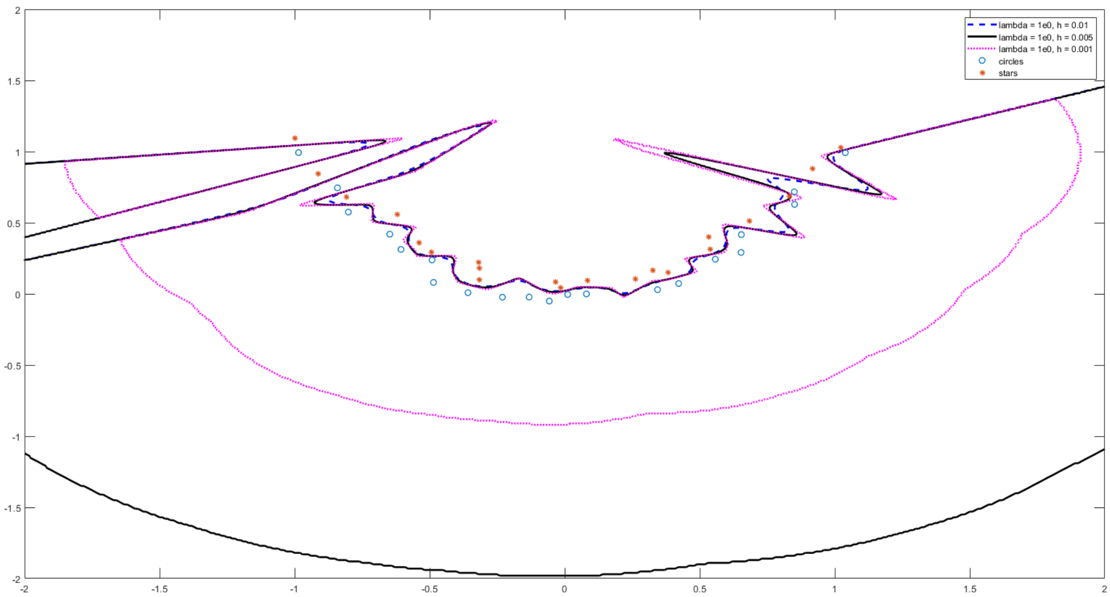

Fig.2.2 the boundary between stars and circles using the Gaussian kernel with $\lambda = 1$

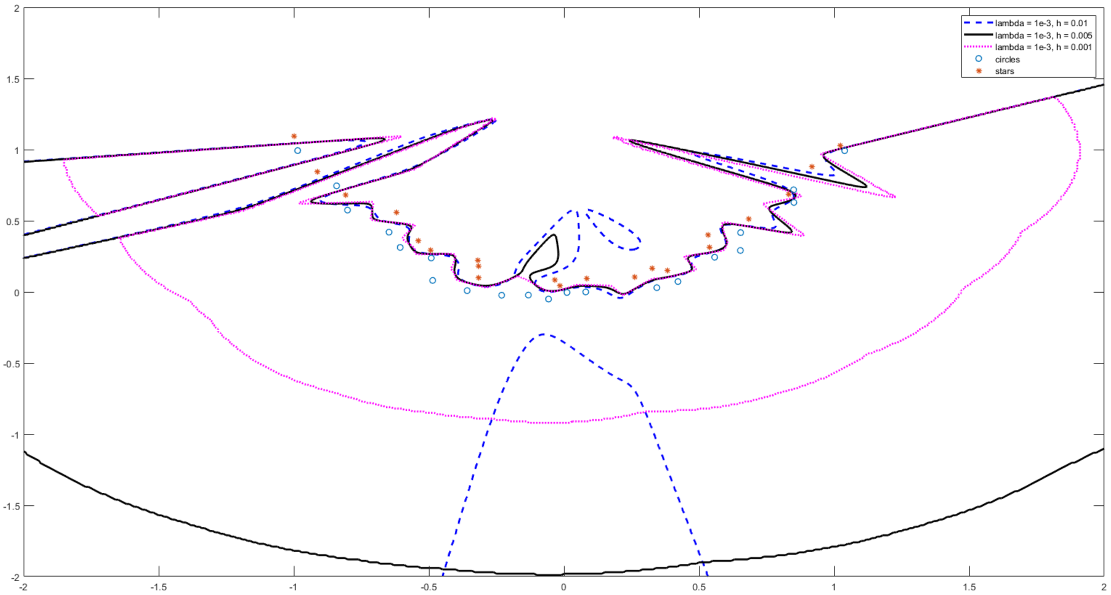

Fig.2.3 the boundary between stars and circles using the Gaussian kernel with λ = 0.001

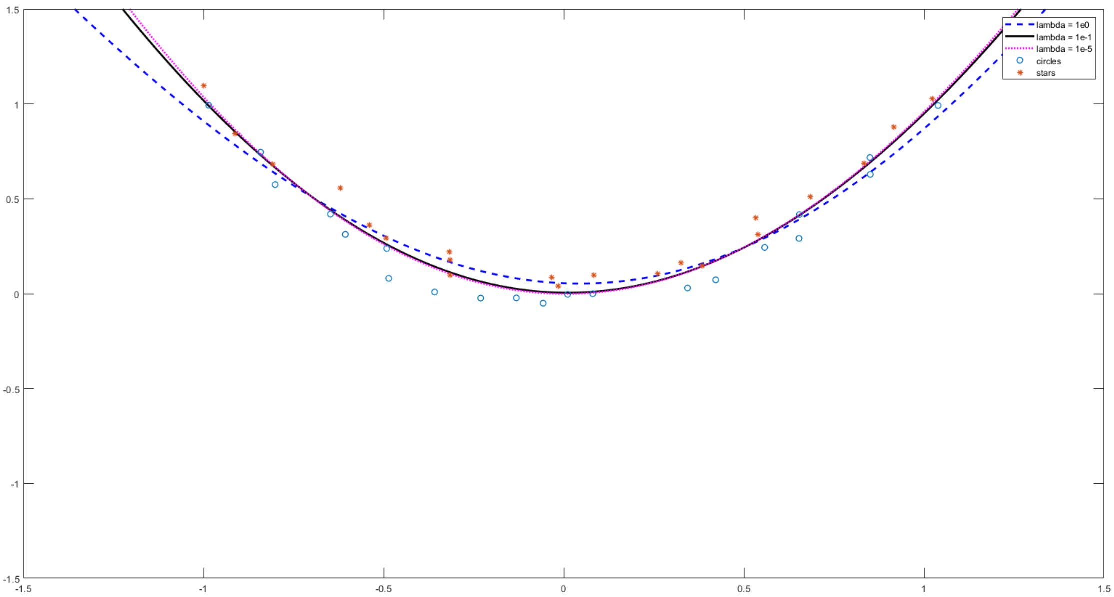

Fig.2.4 the boundary between stars and circles using the "2-order-polynomial" kernel

Question 1

`kG.m`

```
function y = kG(x, h)
%x: vector
%h: scalar, parameter, positive
%y: vector
y = 1 / sqrt(2 * pi * h) * exp(-1 / (2 * h) * x.^2);
```

`kL.m`

```
function y = kL(x, h)
%x: vector
%h: scalar, parameter, positive
%y: vector
y = 1 / (2 * h) * exp(-1 / h * abs(x));
```

`Q11.m`

```
%get 1000 realizations
X = rand(1000, 1);
figure(1)
hist(X, 100)

% approximate
h = [1, 0.01, 0.0001];
x = -1 : 0.001 : 2;
f = zeros(size(x, 2), size(h, 2));
for i = 1 : size(h, 2)
    f(:,i) = sum(kG((X-x), h(i))) / 1000;
end

%plot approximations
figure(2)
p = plot(x, f(:,1), '--', x, f(:,2), '-', x, f(:,3), ':');
for i = 1 : size(p, 1)
    p(i).LineWidth = 2;
end
legend('h = 1', 'h = 0.01', 'h = 0.0001')

%the constant value of the pdf
index = (x >= 0) & (x < 1);
```

```
data = f(index,:);
variance = var(data)

%capturing the support
valid = f > 0.001;
support = zeros(1,3);
for i = 1 : 3
    support(i) = max(x(valid(:,i))) - min(x(valid(:,i)));
end
support
```

`Q12.m`

```
%get 1000 realizations
X = rand(1000, 1);
figure(1)
hist(X, 100)

% approximate
h = [1, 0.05, 0.0025];
x = -1 : 0.001 : 2;
f = zeros(size(x, 2), size(h, 2));
for i = 1 : size(h, 2)
    f(:,i) = sum(kL((X-x), h(i))) / 1000;
end

%plot approximations
figure(2)
p = plot(x, f(:,1), '--', x, f(:,2), '-', x, f(:,3), ':');
for i = 1 : size(p, 1)
    p(i).LineWidth = 2;
end
legend('h = 1', 'h = 0.05', 'h = 0.0025')

%the constant value of the pdf
index = (x >= 0) & (x < 1);
data = f(index,:);
variance = var(data)

%capturing the support
valid = f > 0.001;
support = zeros(1,3);
for i = 1 : 3
    support(i) = max(x(valid(:,i))) - min(x(valid(:,i)));
end
support
```

Question 2

`kG.m`

```
function k = kG(x, y, h)
%x: vector
%y: scalar, I think there should be a way to use matrix multiplication
%k: vector
k = exp(-1 / h * ((x(:,1) - y(1)).^2 +(x(:,2) - y(2)).^2));
```

`kS.m`

```
function k = kS(x, y)
%x: vector
%y: scalar, I think there should be a way to use matrix multiplication
%k: vector
k = (1 + x(:,1) * y(1) + x(:,2) * y(2)).^2;
```

`Q21.m`

```
%load data
load('D:\Users\endlesstory\Desktop\data3-2.mat')
X = [stars; circles];
Y = [ones(size(stars, 1), 1); -ones(size(circles, 1), 1)];

%calculate K
h = [0.01, 0.005, 0.001];
K = zeros(size(X, 1));
for j = 1 : size(X, 1) % I think there should be a way to use matrix multiplication
    for k = 1 : size(h, 2)
        K(:,j,k) = kG(X, X(j,:), h(k));
    end
end

%calculate A
lambda = [1e1, 1e0, 1e-3];
I = eye(size(X, 1));
A = zeros([size(X, 1), size(lambda, 2), size(h, 2)]);
for k = 1 : size(lambda, 2)
    for l = 1 : size(h, 2)
        A(:,k, l) = (lambda(k) * I + K(:,:,l)) \ Y;
    end
end
```

```
%find g(x) = 0 boundary
[x, y] = meshgrid(-2: 0.01: 2);
z = zeros([size(x), size(lambda, 2), size(h, 2)]);
for i = 1 : size(x, 1)
    for j = 1 : size(y, 1)
        for k = 1 : size(lambda, 2)
            for l = 1 : size(h, 2)
                z(i,j,k,l) = sum((A(:,k,l))' * kG(X, [x(i,j), y(i,j)], h(l)));
            end
        end
    end
end

%plot boundary
figure(1)
[~, c] = contour(x, y, z(:,:,1,1), [0,0], 'b--', 'DisplayName', 'lambda = 1e1, h = 0.01')
c.LineWidth = 2;
a11 = A(:,1,1)'
hold on
[~, c] = contour(x, y, z(:,:,1,2), [0,0], 'k-', 'DisplayName', 'lambda = 1e1, h = 0.005')
c.LineWidth = 2;
a12 = A(:,1,2)'
[~, c] = contour(x, y, z(:,:,1,3), [0,0], 'm:', 'DisplayName', 'lambda = 1e1, h = 0.0001'
c.LineWidth = 2;
a13 = A(:,1,3)'
s = scatter(circles(:,1), circles(:,2), 'o', 'DisplayName', 'circles');
s.LineWidth = 1;
s = scatter(stars(:, 1), stars(:, 2), '*', 'DisplayName', 'stars');
s.LineWidth = 1;

figure(2)
[~, c] = contour(x, y, z(:,:,2,1), [0,0], 'b--', 'DisplayName', 'lambda = 1e0, h = 0.01')
c.LineWidth = 2;
hold on
a21 = A(:,2,1)'
[~, c] = contour(x, y, z(:,:,2,2), [0,0], 'k-', 'DisplayName', 'lambda = 1e0, h = 0.005')
c.LineWidth = 2;
a22 = A(:,2,2)'
[~, c] = contour(x, y, z(:,:,2,3), [0,0], 'm:', 'DisplayName', 'lambda = 1e0, h = 0.001')
c.LineWidth = 2;
a23 = A(:,2,3)'
s = scatter(circles(:,1), circles(:,2), 'o', 'DisplayName', 'circles');
s.LineWidth = 1;
s = scatter(stars(:, 1), stars(:, 2), '*', 'DisplayName', 'stars');
s.LineWidth = 1;
```

```
figure(3)
[~, c] = contour(x, y, z(:,:,3,1), [0,0], 'b--', 'DisplayName', 'lambda = 1e-3, h = 0.01'
c.LineWidth = 2;
hold on
a31 = A(:,3,1)'
[~, c] = contour(x, y, z(:,:,3,2), [0,0], 'k-', 'DisplayName', 'lambda = 1e-3, h = 0.005'
c.LineWidth = 2;
a32 = A(:,3,2)'
[~, c] = contour(x, y, z(:,:,3,3), [0,0], 'm:', 'DisplayName', 'lambda = 1e-3, h = 0.001'
c.LineWidth = 2;
a33 = A(:,3,3)'
s = scatter(circles(:,1), circles(:,2), 'o', 'DisplayName', 'circles');
s.LineWidth = 1;
s = scatter(stars(:, 1), stars(:, 2), '*', 'DisplayName', 'stars');
s.LineWidth = 1;
```

`Q22.m`

```
%load data
load('D:\Users\endlesstory\Desktop\data3-2.mat')
X = [stars; circles];
Y = [ones(size(stars, 1), 1); -ones(size(circles, 1), 1)];

%calculate K
K = zeros(size(X, 1));
for j = 1 : size(X, 1) % I think there should be a way to use matrix multiplication
    K(:,j) = kS(X, X(j,:));
end

%calculate A
lambda = [1e0, 1e-1, 1e-5];
I = eye(size(X, 1));
A = zeros(size(X, 1), size(lambda, 2));
for i = 1 : size(lambda, 2)
    A(:,i) = (lambda(i) * I + K) \ Y;
end

%find g(x) = 0 boundary
[x, y] = meshgrid(-1.5: 0.01: 1.5);
z = zeros([size(x), size(lambda, 2)]);
for i = 1 : size(x, 1)
    for j = 1 : size(y, 1)
        for k = 1 : size(lambda, 2)
            z(i,j,k) = sum((A(:,k))' * kS(X, [x(i,j), y(i,j)]));
        end
    end
end
```

```
%plot boundary
[~, c] = contour(x, y, z(:,:,1), [0,0], 'b--', 'DisplayName', 'lambda = 1e0');
c.LineWidth = 2;
hold on
a1 = A(:,1)'
[~, c] = contour(x, y, z(:,:,2), [0,0], 'k-', 'DisplayName', 'lambda = 1e-1');
c.LineWidth = 2;
a2 = A(:,2)'
[~, c] = contour(x, y, z(:,:,3), [0,0], 'm:', 'DisplayName', 'lambda = 1e-5');
c.LineWidth = 2;
a3 = A(:,3)'
s = scatter(circles(:,1), circles(:,2), 'o', 'DisplayName', 'circles');
s.LineWidth = 1;
s = scatter(stars(:, 1), stars(:, 2), '*', 'DisplayName', 'stars');
s.LineWidth = 1;
```

Q21-a.txt

```
val(:,:,1) =
    0.0939    0.6082    1.5723
    0.0933    0.6038    1.8400
    0.0986    0.8264    3.6591
    0.0922    0.5420    1.1752
    0.0942    0.5987    0.7636
    0.0953    0.7108    3.6418
    0.0832    0.3523    1.8671
    0.0811    0.2384   -1.4739
    0.0898    0.5346    2.0293
    0.0883    0.3526   -3.6749
    0.0957    0.7120    8.9337
    0.0913    0.5206    0.7137
    0.0886    0.4682    1.3025
    0.0902    0.5531    2.3644
    0.0843    0.2897   -0.7974
    0.0951    0.6734    2.5018
    0.0899    0.4506    0.4280
    0.0940    0.6169    1.6813
    0.1044    1.1097   18.3732
    0.0915    0.5257    1.6398
    0.0984    0.8763    7.3416
   -0.0944   -0.6238   -1.6547
   -0.0978   -0.8012   -3.4977
   -0.0936   -0.6215   -2.0431
   -0.0917   -0.5229   -0.9509
   -0.0940   -0.6187   -1.5506
   -0.0974   -0.7615   -3.5762
   -0.0899   -0.4582   -0.6526
   -0.0919   -0.5413   -1.5202
   -0.0872   -0.4077   -0.4988
   -0.0851   -0.3677   -1.2800
   -0.0857   -0.3658    0.4737
   -0.0922   -0.6117   -6.9025
   -0.0921   -0.5374    0.3547
   -0.0924   -0.5227   -0.9985
   -0.0930   -0.5572   -1.6456
   -0.0941   -0.6476   -2.3453
   -0.0893   -0.4360   -0.5744
   -0.0949   -0.6450   -1.7807
   -0.0937   -0.6988   -7.4778
   -0.0958   -0.8430  -13.8625
   -0.0991   -0.8942   -7.4273
```

```
val(:,:,2) =

    0.0919    0.5322    1.1367
    0.0914    0.5186    1.0957
    0.0947    0.6357    1.6895
    0.0911    0.5050    1.0186
    0.0916    0.5134    0.9291
    0.0941    0.6362    1.9662
    0.0852    0.3829    0.9004
    0.0832    0.2983    0.0648
    0.0897    0.4876    1.0976
    0.0867    0.3463   -0.3342
    0.0924    0.5961    2.5132
    0.0914    0.5145    1.0185
    0.0897    0.4751    0.9746
    0.0893    0.4819    1.1285
    0.0857    0.3467    0.3026
    0.0928    0.5738    1.4152
    0.0896    0.4551    0.7678
    0.0921    0.5371    1.1601
    0.1018    0.9725    7.0791
    0.0910    0.5019    1.0211
    0.0975    0.7964    3.8984
   -0.0919   -0.5331   -1.1402
   -0.0944   -0.6247   -1.6454
   -0.0918   -0.5323   -1.1700
   -0.0909   -0.4971   -0.9734
   -0.0929   -0.5633   -1.2598
   -0.0956   -0.6732   -2.0617
   -0.0908   -0.4949   -0.9730
   -0.0917   -0.5226   -1.1070
   -0.0896   -0.4651   -0.8711
   -0.0877   -0.4162   -0.7508
   -0.0877   -0.4079   -0.5680
   -0.0920   -0.5663   -2.0650
   -0.0900   -0.4666   -0.6496
   -0.0905   -0.4823   -0.9133
   -0.0915   -0.5160   -1.1016
   -0.0932   -0.5801   -1.4202
   -0.0903   -0.4793   -0.9031
   -0.0922   -0.5409   -1.1712
   -0.0936   -0.6508   -3.2061
   -0.0964   -0.8112   -5.8362
   -0.0975   -0.7973   -3.9014
```

```
val(:,:,3) =

    0.0909    0.5000    0.9990
    0.0909    0.5000    0.9990
    0.0910    0.5013    1.0041
    0.0909    0.5000    0.9990
    0.0909    0.5000    0.9990
    0.0913    0.5119    1.0476
    0.0895    0.4599    0.8511
    0.0895    0.4595    0.8491
    0.0909    0.4996    0.9976
    0.0901    0.4746    0.8981
    0.0906    0.4933    0.9792
    0.0909    0.5000    0.9991
    0.0909    0.4999    0.9985
    0.0907    0.4938    0.9747
    0.0907    0.4935    0.9733
    0.0910    0.5014    1.0046
    0.0909    0.4999    0.9985
    0.0909    0.5000    0.9991
    0.0935    0.5896    1.4246
    0.0909    0.5000    0.9990
    0.0928    0.5644    1.2942
   -0.0909   -0.5000   -0.9990
   -0.0910   -0.5013   -1.0041
   -0.0909   -0.5000   -0.9990
   -0.0909   -0.5000   -0.9990
   -0.0909   -0.5003   -1.0002
   -0.0913   -0.5122   -1.0487
   -0.0909   -0.5000   -0.9990
   -0.0909   -0.5000   -0.9990
   -0.0909   -0.5000   -0.9989
   -0.0909   -0.4995   -0.9972
   -0.0909   -0.4993   -0.9961
   -0.0914   -0.5148   -1.0576
   -0.0908   -0.4980   -0.9908
   -0.0909   -0.4999   -0.9987
   -0.0909   -0.5001   -0.9995
   -0.0910   -0.5015   -1.0051
   -0.0909   -0.5000   -0.9990
   -0.0909   -0.5000   -0.9991
   -0.0911   -0.5076   -1.0359
   -0.0933   -0.5837   -1.4031
   -0.0928   -0.5644   -1.2942
```

Q22-a.txt

0.677010591480610	4.83670013028005	35117.3715207469
0.898433940657055	10.5892788029693	121831.904195552
0.935038257277409	9.49588803182519	95915.4350471848
0.787869236592803	0.372094256039488	-66423.0905464887
0.971163778561711	6.84662649338281	40274.4533636250
1.00779689328008	7.94002502970238	57846.6233813054
0.922868979464173	3.41574482583754	-20797.1138610899
0.981675749250188	5.90870737763430	21673.3738211251
1.09650311616345	10.8205773791824	105500.573383011
0.962691470828535	5.16712452652371	10273.1086474569
1.02098651371386	7.80276360746004	55734.0993489603
0.941043463749687	4.64582685471793	2603.67877725284
0.987145584961127	7.67731910760774	57085.2003081051
1.01418631471257	9.74843005292099	94519.3401405615
0.946424078341004	6.27311467356254	33802.4816966473
0.949421959361785	8.05139427913807	67584.8090426054
0.818467906452973	2.13620282685712	-34185.5337686174
0.877931739375110	6.18384577363642	37987.0455527043
0.895891037117531	8.56620679417943	81938.4687493043
0.788379245140612	4.86126001608317	20791.5491581809
0.817471163160772	7.59258588998862	70732.8140882615
-1.17893672141198	-10.1956653390783	-84227.2542987110
-1.09646277851545	-10.9600646304949	-109380.290016651
-0.905034534364139	-4.34110293269696	-1306.04261323136
-0.950135078679291	-8.68804672960756	-81250.9825124529
-0.854601964920965	-5.29772291801578	-25232.6051167579
-0.911369167722161	-8.70309642279741	-85221.7763600647
-0.686994161929482	0.711343048654931	74881.5660578529
-0.733452632626840	-1.75754318791999	32371.3974978154
-0.797525643411403	-4.63902476353514	-16439.8993974583
-0.855699153981920	-7.06451904415201	-57257.6750736972
-0.843384488459103	-6.30841991713214	-43445.3002884894
-0.919675333249063	-9.46081811672213	-96901.0038952744
-0.924797762883801	-9.37991382170870	-94630.6943455140
-0.856214260142418	-4.64556517695940	-9268.13398233904
-0.848557877144186	-3.71323892670324	7927.31429800677
-0.930228592116772	-6.34494875803499	-35599.9174286679
-0.860363062257172	-2.32169443800487	35196.7917614579
-1.03573970897941	-10.2502002506971	-101319.342016399
-0.982685185631401	-5.83129311707828	-21644.6720675374
-1.11501711925272	-11.7181737700741	-122525.290538022
-1.08776298608111	-8.05329356505157	-54532.7724125734