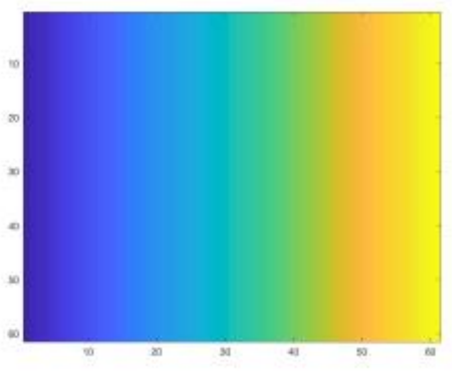
CSCE 636 Neuron Network HW3

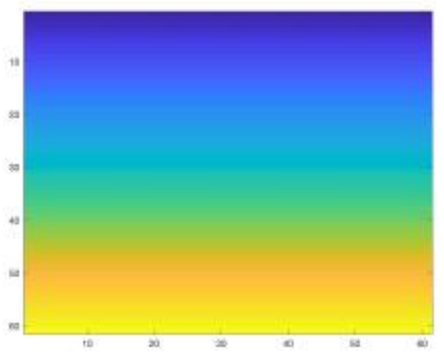
Problem 1.

1. Visualize each RBF in 3D

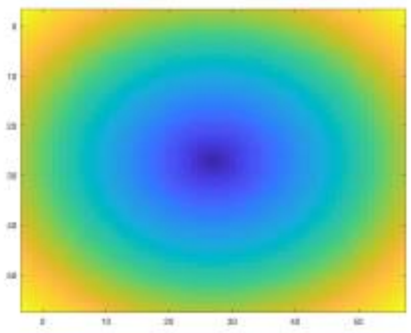
(1) show x coordination matrix



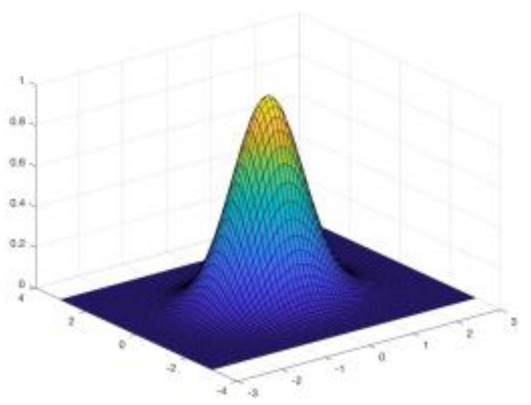
(2) show y coordination matrix

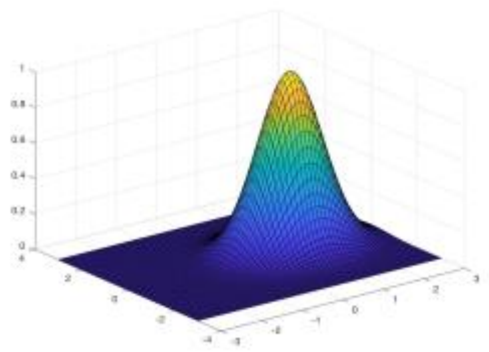


(3) distance from center matrix

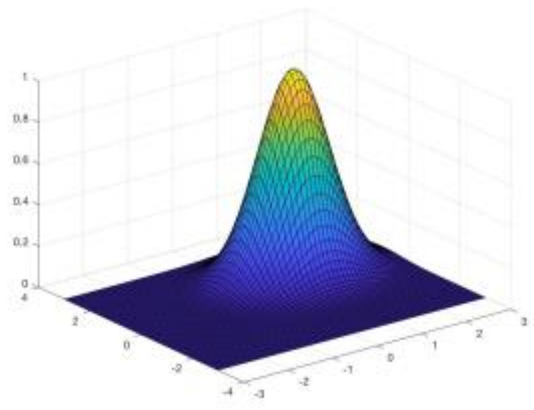


(4) RBF

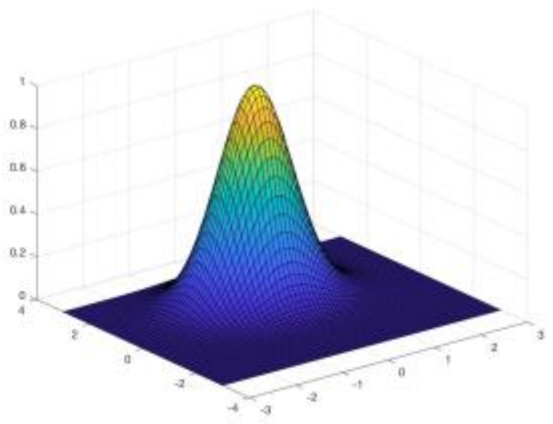




center (0,0)
center (1,0)



center(1,1)



center(0,1)

2. Construct the matrix φ for the RBF network and compute the inverse φ−1

- Gaussian functions (local):

$$\phi(r) = \exp\left(-\frac{r^2}{2\sigma^2}\right)$$

Assume sigma = 0.707

Φ:[1.0000 0.3678 0.3678 0.1353;

0.3678 1.0000 0.1353 0.3678;

0.3678 0.1353 1.0000 0.3678;

0.1353 0.3678 0.3678 1.0000;]

inv(Φ) : [1.3373 -0.4918 -0.4918 0.1809;

-0.4918 1.3373 0.1809 -0.4918;

-0.4918 0.1809 1.3373 -0.4918;

0.1809 -0.4918 -0.4918 1.3373]

3. Calculate the linear weights (w) of the output layer of the network

W = inv(Φ)D =[-0.9837; 1.5182; 1.5182; -0.9837]

Problem 2. Repeat problem 1 with only two hidden units

Two hidden layer, center at t1[1,0], t2[0,1]; Still 4 kinds of input;

- Gaussian functions (local):

$$\phi(r) = \exp\left(-\frac{r^2}{2\sigma^2}\right)$$

Φ is 4*2;

Φ:[0.3678   0.3678;

1.0000   0.1353;

0.1353   1.0000;

0.3678   0.3678]

sudo inv(Φ) : [0.2010, 0.8884, -0.2680, 0.2010;

0.2010, -0.2680, 0.8884, 0.2010]

$$\mathbf{w} = \underbrace{\left(\phi^T \phi\right)^{-1} \phi^T}_{\text{pseudo inverse}} \mathbf{d}.$$

w=[0.6204;

0.6204]

Problem 3. RBF and GRBF programs.

1. **RBF**

Since both RBF and GRBF learning take a 2D input and generates a 1D output ($\lambda = 0$), I could use XOR problem as the model for this question.

Input :[0 0;0 1;1 0;1 1];

Center :[0 0;0 1;1 0;1 1];

Target:[0 ;1 ;1 ;0];

Matlab code:

```
x =[0 0; 0 1; 1 0; 1 1];
d =[0 1 1 0]';
x1 = x(:,1);
x2 = x(:,2);
t=[0 0;0 1;1 0;1 1];
t1 =t(:,1);
t2 =t(:,2);
[m,junk] =size(t);
[N,junk] =size(x);
T1=ones(N,1)*t1';
T2=ones(N,1)*t2';
X1= x1*ones(1,m);
X2=x2*ones(1,m);
sigma = 0.707;
G = exp(-((X1-T1).^2 + (X2-T2).^2)/(2*sigma.^2))
w =inv(G'*G)*G'*d
G*w
```

Result:

```
G =  1.0000   0.3678   0.3678   0.1353

     0.3678   1.0000   0.1353   0.3678

     0.3678   0.1353   1.0000   0.3678

     0.1353   0.3678   0.3678   1.0000
```

w =[-0.9836; 1.5182; 1.5182; -0.9836]

verified result: [-0.0000; 1.0000; 1.0000; -0.0000]

sigma = 0.707; When sigma = 0.2 / 10 result is still perfect. Sigma do not have significant influence.

***2.* GRBF**

2.1 using fixed centers selected at random

where $\mathbf{t}_i \, (i = 1, 2, \ldots, m_1)$ are picked by random from the available inputs $\mathbf{x}_j \, (j = 1, 2, \ldots, N)$.

$\sigma = \frac{d_{\max}}{\sqrt{2m_1}}$, where $d_{\max}$ is the max distance between the chosen centers $\mathbf{t}_i$.

code:

```
x =[0 0; 0 1; 1 0; 1 1];
d =[0 1 1 0]';
x1 = x(:,1);
x2 = x(:,2);
t=[0 0;0 1;1 0;1 1];
[N,junk] = size(x);
r =randperm(N); %chose one row as the center
nCenter = 2; %suppose we have 2 hidden center
r=r(1:nCenter); %represent which rows are chosen to become center
t_select=[t(r,:)];     %build center
[m,junk] = size(t_select);
t1 =t_select(:,1);
t2 =t_select(:,2);
T1=ones(N,1)*t1';
T2=ones(N,1)*t2';
X1= x1*ones(1,m);
X2=x2*ones(1,m);
%find out dmax
dmax = 0;
for i = 1:1:nCenter
  for j = 1:1:nCenter
  temp = sqrt((t_select(i,1)-t_select(j,1)).^2+(t_select(i,2)-  t_select(j,2)).^2);
   if temp > dmax
     dmax =temp;
    end
   end
end
sigma=dmax/sqrt(2*nCenter)
G = exp(-sqrt((X1-T1).^2 + (X2-T2).^2)*(nCenter/dmax.^2))
w =inv(G'*G)*G'*d
G*w
```

Result: if we use less centers then we need large sigma
since   sigma = dmax/sqrt(2*nCenter). I just changed the center number

```
2 centers;
   sigma =0.5000


   G =   0.1353    0.0591
 1.0000    0.1353
 0.0591    0.1353
 0.1353    1.0000
```

```
w =
   1.0243
  -0.0221
output =
   0.1373
   1.0213
   0.0576
   0.1165
```

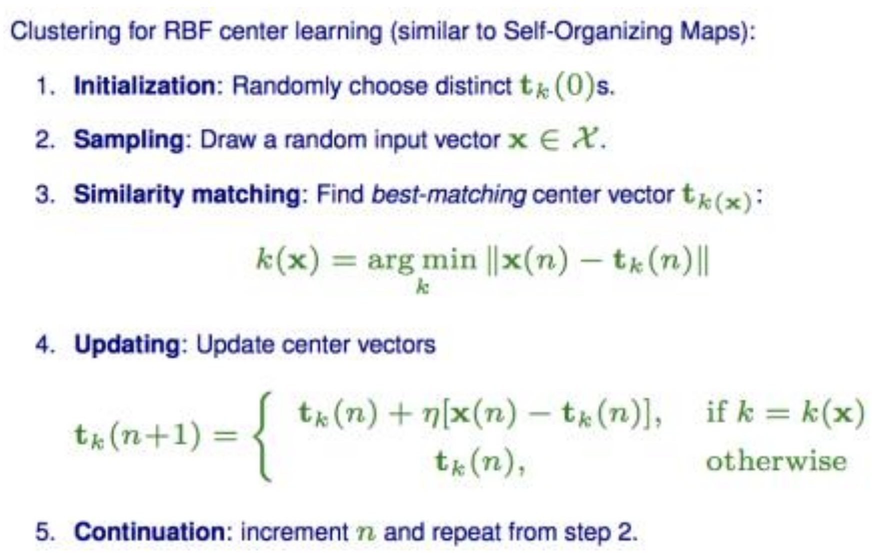
```
3 centers;
sigma = 0.5774

G =

   0.2231    0.2231    0.0498
   0.0498    1.0000    0.2231
   1.0000    0.0498    0.2231
   0.2231    0.2231    1.0000
```

```
w =
   0.9526
   0.9526
  -0.4049
output =

   0.4049
   0.9096
   0.9096
   0.0202
```

```
 4 centers;
sigma =0.5000




G =

   0.1353    0.0183    1.0000    0.1353
   0.0183    0.1353    0.1353    1.0000
   1.0000    0.1353    0.1353    0.0183
   0.1353    1.0000    0.0183    0.1353
```

```
w =
   1.0567
  -0.2809
  -0.2809
   1.0567

output =
   0.0000
   1.0000
   1.0000
   0.0000
```

2.2 self organized selection of centers



Clustering for RBF center learning (similar to Self-Organizing Maps):

1. **Initialization**: Randomly choose distinct $\mathbf{t}_k(0)$s.
2. **Sampling**: Draw a random input vector $\mathbf{x} \in \mathcal{X}$.
3. **Similarity matching**: Find *best-matching* center vector $\mathbf{t}_{k(\mathbf{x})}$:

$$k(\mathbf{x}) = \arg\min_k \|\mathbf{x}(n) - \mathbf{t}_k(n)\|$$

4. **Updating**: Update center vectors

$$\mathbf{t}_k(n+1) = \begin{cases} \mathbf{t}_k(n) + \eta[\mathbf{x}(n) - \mathbf{t}_k(n)], & \text{if } k = k(\mathbf{x}) \\ \mathbf{t}_k(n), & \text{otherwise} \end{cases}$$

5. **Continuation**: increment n and repeat from step 2.

code:

```
clc;
close all;
clear;
%  use the XOR example.
t_raw=[0,0;0,1;1,0;1,1];
x=[0,0,1,1]; y=[0,1,0,1]; d=[0,1,1,0];
[m,junk]=size(t_raw);
[n,junk]=size(x);
sigma=0.707;
deviation= 2* sigma.^2;
%randomly choose distinct tk and input vector
k=randi([1,4],1,2);   %k(1,1) means how many centers k(1,2) means which input is selected
tk_number=randperm(4);  %permutation of 1--4, decide which rows of t_raw are chosen to be center
at first
index=sort(tk_number(1:k(1,1)));
[junk,index_size]=size(index); t=[];
for in=1:1:index_size
   t(in,:)=t_raw(index(in),:);
end
eta=0.15;  %learning rate = 0.15
%similarity matching: find the best-matching center vector tk(x);
kx=[];
for n=1:1:300   %set loop 300 times
   for q=1:1:k(1,1)  %(x(k(1,2),y(k(1,2)) is the chosen input
       kx(q)=sqrt((x(k(1,2))-t(q,1)).^2 + (y(k(1,2))-t(q,2)).^2);
    end
[junk,o]=min(kx);
t(o,1)=t(o,1)+ eta * (x(k(1,2))-t(o,1));  %update centers
t(o,2)=t(o,2)+ eta * (y(k(1,2))-t(o,2));  %update centers
k(1,2)=randi([1,4],1,1);   %choose another input
n=n+1;
end
%construct the matrix of G
G=[];
```

```
for b=1:1:4
   for a=1:1:index_size
       G(b,a) = exp(-deviation*(sqrt((x(b)-t(a,1)).^2+(y(b)-t(a,2)).^2)).^2);
   end
end
w=inv(G'*G)*G'*d;
output= G*w
t
G
```

Result: (if we use 4 centers & sigma=0.2) if we use less centers then we need large sigma

```
output = 0.0000
         1.0000
         1.0000
          0.0000
t =
   0    0
   0    1
   1    0
   1    1
```

```
G =

1.0000    0.9231    0.9231    0.8521
0.9231    1.0000    0.8521    0.9231
 0.9231    0.8521    1.0000    0.9231
0.8521    0.9231    0.9231    1.0000
```

(if we use 2 centers & sigma=0.707)

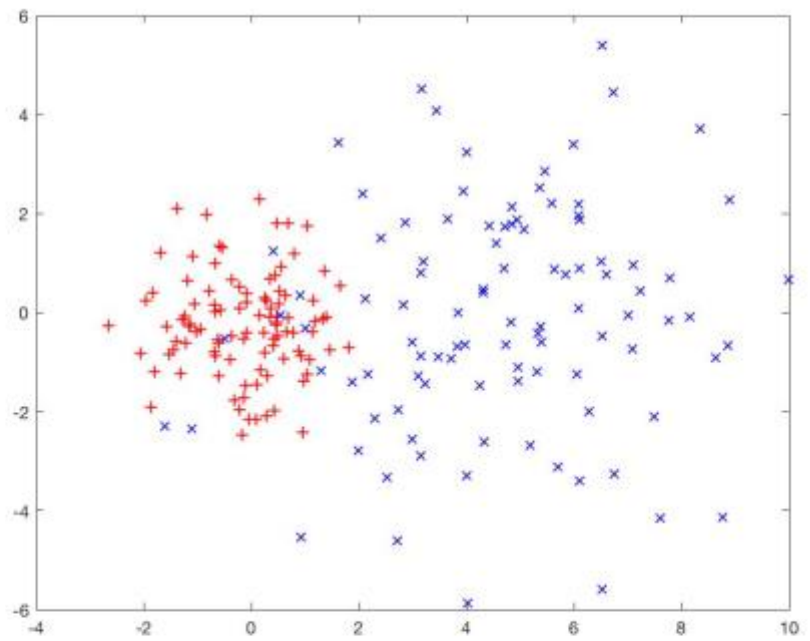
```
output =

   0.4989
   0.4700
   0.4731
   0.4994
t =
        0    0.4367
             1.0000    0.5601

G =

   0.8264    0.2689
   0.7282    0.3033
   0.3041    0.7308
   0.2680    0.8241
```

(if we use 2 centers & sigma=10)

```
output =

   0.0007
   1.0000
   1.0000
   0.0000
```

```
t =    0.5181         0
       0.1428    1.0000
G =

       0.0000    0.0000
       0.0000    0.0169
       0.0000    0.0000
       0.0000    0.0000
```

Problem 4: Test RBF and GRBF ($\lambda = 0$) on a classification task



4.1 test of RBF

code:

```
clear
pos=[randn(100,1),randn(100,1)];
neg=[randn(100,1)*sqrt(5)+5,randn(100,1)*sqrt(5)];
plot(pos(:,1),pos(:,2),'r+',neg(:,1),neg(:,2),'bx');
x=[pos;neg];
t=x;
d=[ones(100,1);zeros(100,1)];
test_pos=[randn(100,1),randn(100,1)];
test_neg=[randn(100,1)*sqrt(5)+5,randn(100,1)*sqrt(5)];
test=[test_pos;test_neg];
sigma=0.5;
deviation= 2*sigma.^2;
phi=[];
for i=1:1:200
  for j=1:1:200
      phi(i,j)=exp(- (1/(2*sigma.^2)) * (sqrt(((x(i,1)-t(j,1)).^2)+((x(i,2)-t(j,2)).^2))).^2);
      phi_test(i,j)=exp(- (1/(2*sigma.^2)) * (sqrt(((test(i,1)-test(j,1)).^2)+((test(i,2)-
test(j,2)).^2))).^2);
  end
end
% weight is based on phi(which is the train set)
w=inv(phi)*d;
output_test=(w'*phi_test)';
output_train=(w' * phi)';
a1=round(output_train);
a2=round(output_test);
c_correctness_train=0;  c_correctness_test=0;
```

```
% caculate accuracy for positive
for i=1:1:100
    if a1(i,1) == 1
        c_correctness_train=c_correctness_train+1;
    end
    if a2(i,1) == 1
        c_correctness_test=c_correctness_test+1;
    end
end
% caculate accuracy for negative
for j=101:1:200
    if a1(j,1) == 0
        c_correctness_train=c_correctness_train+1;
    end
    if a2(j,1) == 0
        c_correctness_test=c_correctness_test+1;
    end
end
accuracy_train = c_correctness_train/200
accuracy_test  = c_correctness_test/200
```

Result:

when sigma = 0.02 :

accuracy_train =1 accuracy_test = 0.9650

when sigma = 0.04 :

accuracy_train =1 accuracy_test = 0.9700

when sigma = 0.5 :

accuracy_train =1 accuracy_test = 0.0750

when sigma = 0.707:

accuracy_train = 1 accuracy_test = 0

4.2 test of GRBF

(1) two random centers picked from the training input set

code:

```
clear
pos=[randn(100,1),randn(100,1)];
neg=[randn(100,1)*sqrt(5)+5,randn(100,1)*sqrt(5)];
plot(pos(:,1),pos(:,2),'r+',neg(:,1),neg(:,2),'bx');
x=[pos;neg];
t=x;
d=[ones(100,1);zeros(100,1)];
test_pos=[randn(100,1),randn(100,1)];
test_neg=[randn(100,1)*sqrt(5)+5,randn(100,1)*sqrt(5)];
test=[test_pos;test_neg];
x1 = x(:,1);
test1 = test(:,1);
x2 = x(:,2);
test2 = test(:,2);
[N junk] = size(x);
r = randperm(N);  %chose one row as the center
nCenter = 2;  %suppose we have 2 hidden center
```

```
r=r(1:nCenter);  %represent  which 2 rows i  choose to become  center
t_select=[t(r,:)]; %build center
t_select_test=[t(r,:)];
[m,junk] = size(t_select);
[m_test,junk] = size(t_select_test);
t1 = t_select(:,1);
t2 = t_select(:,2);
t1_test = t_select_test(:,1);
t2_test = t_select_test(:,2);
T1=ones(N,1)*t1';
T2=ones(N,1)*t2';
T1_test=ones(N,1)*t1_test';
T2_test=ones(N,1)*t2_test';
X1= x1*ones(1,m);
X2=x2*ones(1,m);
%find out  dmax
dmax = 0;
for i = 1:1:nCenter
    for j = 1:1:nCenter
        temp = sqrt((t_select(i,1)-t_select(j,1)).^2+(t_select(i,2)-t_select(j,2)).^2);
        if temp > dmax
            dmax =temp;
        end
    end
end
sigma=dmax/sqrt(2*nCenter);
G = exp(-sqrt((X1-T1).^2 + (X2-T2).^2)*(nCenter/dmax.^2));
G_test = exp(-sqrt((TEST1-T1).^2 + (TEST2-T2).^2)*(nCenter/dmax.^2));
w =inv(G'*G)*G'*d;
output_train = G*w;
output_test = G_test*w;
a1=round(output_train);
a2=round(output_test);
c_correctness_train=0;  c_correctness_test=0;
% caculate accuracy for positive
for i=1:1:100
    if a1(i,1) == 1
        c_correctness_train=c_correctness_train+1;
    end
    if a2(i,1) == 1
        c_correctness_test=c_correctness_test+1;
    end
end
% caculate accuracy for negative
for j=101:1:200
    if a1(j,1) == 0
        c_correctness_train=c_correctness_train+1;
    end
    if a2(j,1) == 0
        c_correctness_test=c_correctness_test+1;
    end
end
accuracy_train = c_correctness_train/200
```

```
        accuracy_test = c_correctness_test/200
```

```
result:
sigma = 2.4211
  accuracy_train = 0.7050
  accuracy_test = 0.7400

sigma = 2.7970
  accuracy_train = 0.8150
  accuracy_test = 0.8150

sigma = 3.4069
  accuracy_train = 0.9400
  accuracy_test = 0.9050
```

(2) two self-organized centers

code:

```
clc;
close all;
clear;
pos=[randn(100,1),randn(100,1)];
neg=[randn(100,1)*sqrt(5)+5,randn(100,1)*sqrt(5)];
x_ori=[pos;neg];
x = x_ori(:,1);
y = x_ori(:,2);
t_raw=x_ori;
d=[ones(100,1);zeros(100,1)];
test_pos=[randn(100,1),randn(100,1)];
test_neg=[randn(100,1)*sqrt(5)+5,randn(100,1)*sqrt(5)];
test=[test_pos;test_neg];
test_x = test(:,1);
test_y = test(:,2);
%randomly choose distinct tk and input vector
k=randi([1,200],1,2);  %k(1,1) means how many centers k(1,2) means which input is
selected, since we have 2 center, k(1,1) is not useful for this question
tk_number=randperm(200); %permutation of 1--200, decide which rows of t_raw are chosen
to be center at first
index=sort(tk_number(1:2));  %choose k(1,1)( which in this problem is 2) rows to be center at
first.
sigma=0.5;
deviation= 2* sigma.^2;
[junk,index_size]=size(index); t=[];
for in=1:1:index_size
    t(in,:)=t_raw(index(in),:);
end
eta=0.15;  %learning rate = 0.15
%similarity matching:find the best-matching center vector tk(x);
kx=[];
```

```
for n=1:1:2000   %set loop 2000 times
    for q=1:1:2  %(x(k(1,2),y(k(1,2)) is the chosen input
        kx(q)=sqrt((x(k(1,2))-t(q,1)).^2 +(y(k(1,2))-t(q,2)).^2);
    end
[junk,o]=min(kx);
%update center t.
%for i=1:1:k(1,1);
t(o,:)=t(o,:)+eta*(x_ori(k(1,2),:)-t(o,:));
%end
k(1,2)=randi([1,4],1,1);   %choose another input
n=n+1;
end
%construct the matrix of G
G=[];
for a=1:1:2   %has 2 centers
for b=1:1:200   %has 200 rows
     G(b,a) = exp(- (1/deviation) *(sqrt((x(b)-t(a,1)).^2+(y(b)-t(a,2)).^2)).^2);
     G_test(b,a)= exp(- (1/deviation) *(sqrt((test_x(b)-t(a,1)).^2+(test_y(b)-t(a,2)).^2)).^2);
end
end
w=inv(G'*G)*G'*d;
output_train = G*w;
output_test = G_test*w;
a1=round(output_train); a2=round(output_test);
c_accuracy_train=0; c_accuracy_test=0;
for i=1:1:100
    if a1(i,1) == 1
        c_accuracy_train=c_accuracy_train+1;
    end
    if a2(i,1) == 1
        c_accuracy_test=c_accuracy_test+1;
    end
end
j=101;
for j=101:1:200
    if a1(j,1) == 0
        c_accuracy_train=c_accuracy_train+1;
    end
    if a2(j,1) == 0
        c_accuracy_test=c_accuracy_test+1;
    end
end
accuracy_train = c_accuracy_train/200
accuracy_test = c_accuracy_test/200
t
```

result:

```
sigma=0.5 ;
  accuracy_train =  0.6450
  accuracy_test = 0.6500
  t =
        0.0450   -0.6299
        0.7595    1.3814

sigma=0.707 ;
  accuracy_train = 0.6700
  accuracy_test = 0.6900
  t =
    6.8546   -0.2590
   -0.3006    0.1257

sigma= 1 ;
  accuracy_train = 0.8250
  accuracy_test = 0.7900
  t =
    0.2485   -0.5605
    4.4031   -2.6041

sigma= 2 ;
  accuracy_train = 0.8800
  accuracy_test =   0.8850
  t =
    1.6763    0.1291
   -0.0042   -1.0777

sigma= 3 ;
  accuracy_train = 0.9450
  accuracy_test =   0.9450
  t =
  -0.0011   -0.9986
    3.0197    2.8717
```

(3) centers [0, 0]T and [5, 0]T

code：

```
clc;
close all;
clear;
pos=[randn(100,1),randn(100,1)];
neg=[randn(100,1)*sqrt(5)+5,randn(100,1)*sqrt(5)];
x=[pos;neg];
%construct center t
t_index=sort(randi([1,200],1,2));
t=[0,5;0,0];
%target value
d=[ones(100,1);zeros(100,1)];
%test matrix
test1=randn(100,2); test2=[randn(100,1)*sqrt(5)+5,randn(100,1)*sqrt(5)]; test=[test1;test2];
sigma = 2
deviation_1= 2* sigma.^2 ;
for i=1:1:2
    for j=1:1:200
         G(j,i) = exp(- (1/deviation_1) * ((x(j,1)-t(i,1)).^2+(x(j,2)-t(i,2)).^2));
         G_test(j,i) = exp(- (1/deviation_1) * ((test(j,1)-t(i,1)).^2+(test(j,2)-t(i,2)).^2));
    end
end
w=inv(G'*G)*G'* d;
output_train= G*w;
output_test=G_test * w;
a1=round(output_train);
a2=round(output_test);
c_accuracy_train=0;
c_accuracy_test=0;
for i=1:1:100
    if a1(i,1) == 1
        c_accuracy_train=c_accuracy_train+1;
    end
    if a2(i,1) == 1
        c_accuracy_test=c_accuracy_test+1;
    end
end
for j=101:1:200
    if a1(j,1) == 0
        c_accuracy_train=c_accuracy_train+1;
    end
    if a2(j,1) == 0
        c_accuracy_test=c_accuracy_test+1;
    end
end
accuracy_train = c_accuracy_train/200
accuracy_test = c_accuracy_test/200
```

result:

sigma= 2 ;

accuracy_train = 0.9550

accuracy_test = 0.9550

```
t =
        0    5
        0    0

sigma= 3 ;
accuracy_train = 0.9200
accuracy_test = 0.9500
t =
        0       5
        1       0
```

(4) conclusion

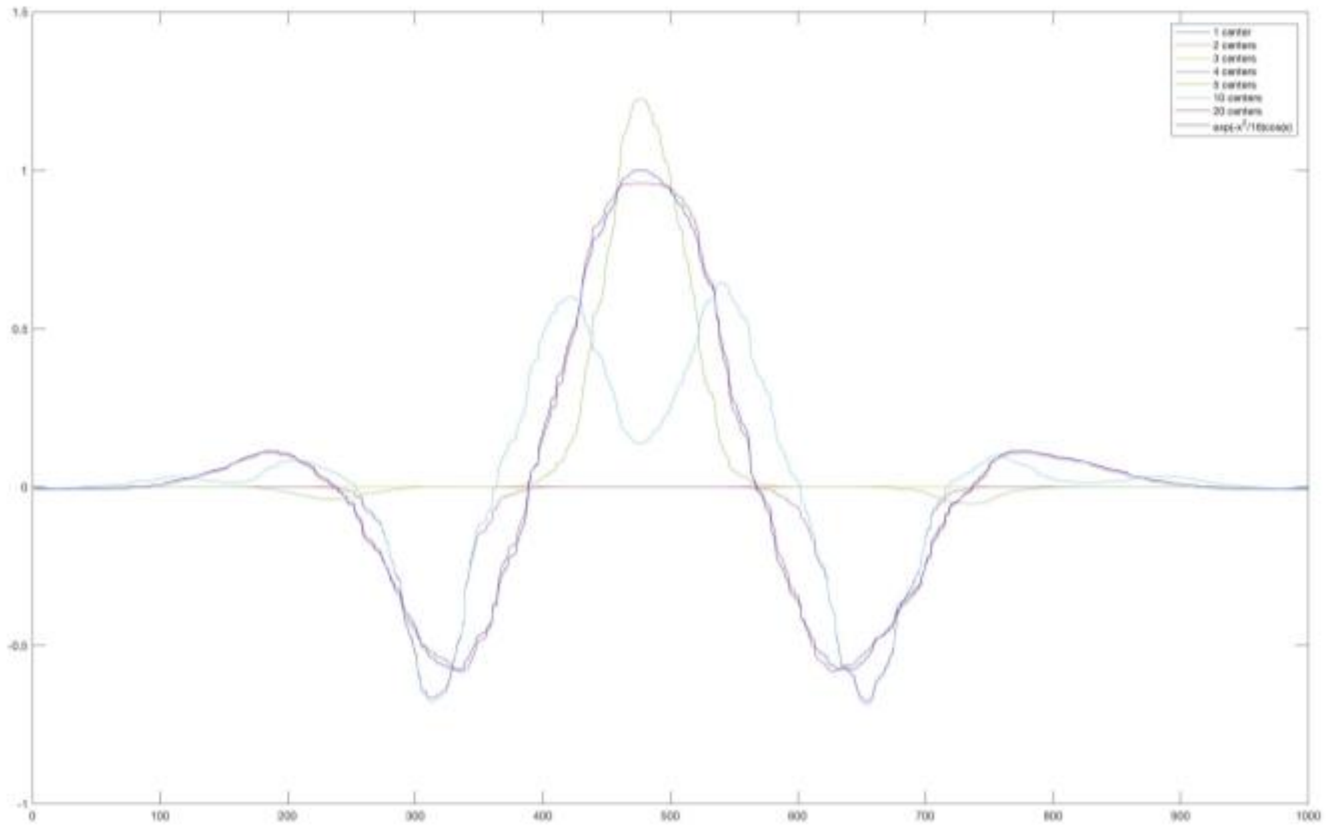
Generally, RBF has the highest accuracy. If we use less centers then we need large sigma. However, it depends on large input sets since it has as many center as input number. Two random centers RBF perform the worst. Nevertheless, The computation is acceptable. The self-organized centers method perform much better than random centers. The last method performs the best.

Problem 5. Experiment with a GRBF network

1. **5.1 fixed standard deviation of 0.5 gradually increase the number of hidden units (2, 3, 4, 5, 10, 20, ...).**

code:

```
clear all;
N=1000;  %number of input
input=rand(N,1)*6*pi -3*pi;  % X[-3pi   3pi]
x=input;
m=[1,2,3,4,5,10,20]; %number of hidden layer
a = cos(x);
b = exp(-x.^2/16);
% set target output
for i=1 : 1: N
     d(i) = a(i)*b(i);
end
target_d = d;
test = sort(x);
a_test = cos(test);
b_test = exp(-test.^2/16);
for i=1 : 1: N
    test_d(i) = a_test(i)*b_test(i);
end
test_target_d = test_d;


for q=1:1:7
     n=m(q);  % n is the number of centers
      %set centers to be at an equal interval
      %across the full range of x (?3? ? x ? 3?)
    interval=(n-1) ;
    t=[];
    for i=1:1:n
        t(i)= 6*pi*(i-1)/interval-3*pi;
     end
     sigma = 0.5 ;
     deviation= 2* sigma.^2 ;
    for i=1:1:n
        for j=1:1:1000
         G(j,i) = exp(- (1/deviation) * ((x(j)-t(i)).^2));
         G_test(j,i) = exp(- (1/deviation) * ((test(j)-t(i)).^2));
         end
     end
     w=inv(G'*G)*G'*target_d;
     output_test= G_test*w;
     plot(output_test)
     hold on
end
hold on
plot(test_target_d,'b')
legend('1 center','2 centers','3 centers','4 centers','5 centers','10 centers','20 centers','exp(-
x^2/16)cos(x)');
```



<u>conclusion:</u>

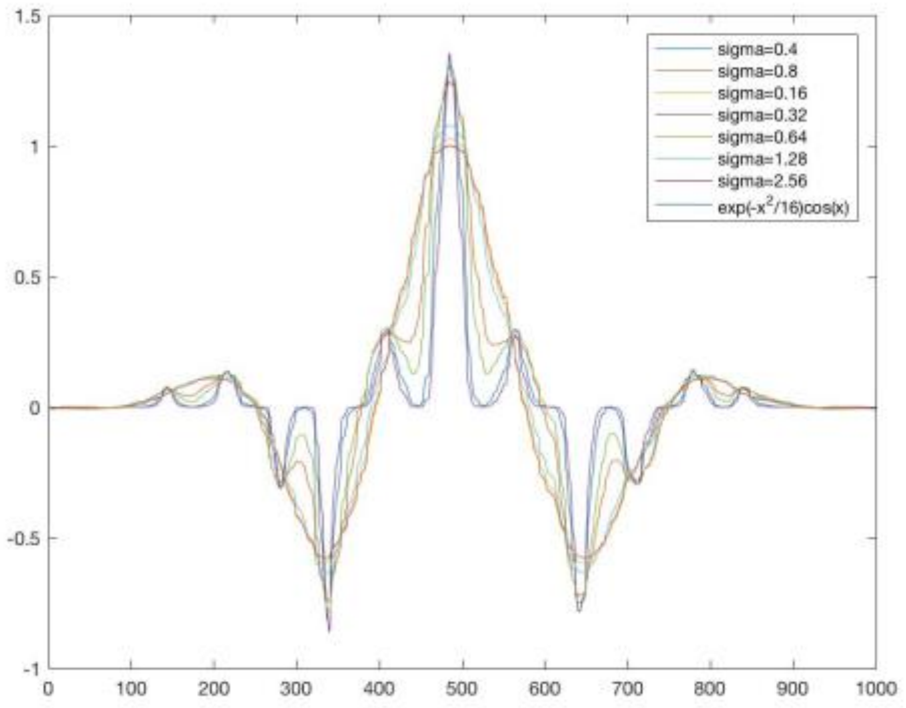
when we have 20 centers we will get best result. Since if we have more hidden unites(centers) we can represent more information for the model.

5.2

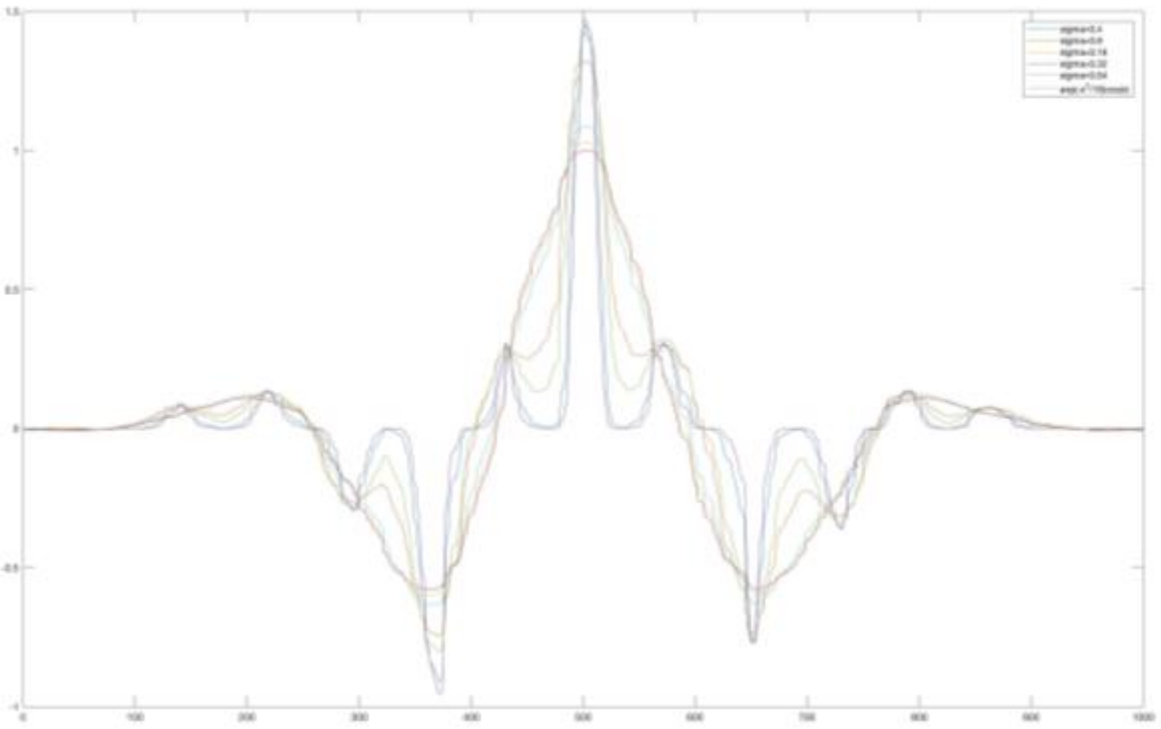
code:

```
clc;
close all;
clear;
N=1000;  %number of input
input=rand(N,1)*6*pi-3*pi;  %X[-3pi  3pi]
x=input;
a = cos(x);
b = exp(-x.^2/16);
for i=1:1:N
    d(i) = a(i)*b(i);
end
target_d = d;
test=sort(rand(N,1)*6*pi-3*pi);
G_test=[];
a_test = cos(test);
b_test = exp(-test.^2/16);
for i=1:1:N
   test_d(i) = a_test(i)*b_test(i);
```

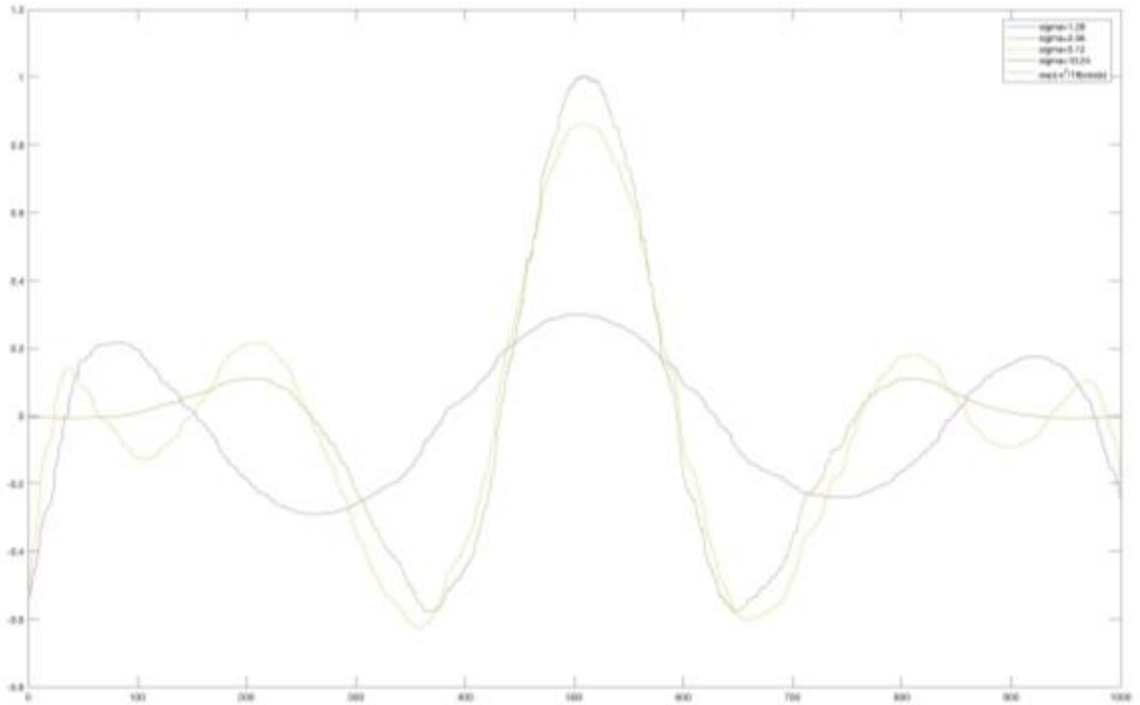
```
end
test_target_d = test_d;
n=15;
interval=(n-1) ; t=[];
for i=1:1:n
t(i)= 6*pi*(i-1)/interval-3*pi ;
end
m=[0.2,0.4,0.8,0.16,0.32,0.64,1.28,2.56];
for k=1:1:8
sigma = m(k);
deviation= 2* sigma.^2 ;
for i=1:1:n
for j=1:1:1000
G(j,i) = exp(- (1/deviation) * ((x(j)-t(i)).^2));
G_test(j,i) = exp(- (1/deviation) * ((test(j)-t(i)).^2));
end
end
w=inv( G'* G)* G'* target_d;
output_test = G_test* w;
plot(output_test)
hold on
end
hold on
plot(test_target_d)
legend('sigma=0.4','sigma=0.8','sigma=0.16','sigma=0.32','sigma=0.64','sigma=1.28','sigma=
2.56','exp(-x^2/16)cos(x)')
```



Part 1 result: sigma start from 0.2 end at 0.64 (0.2,0.4,0.8,0.16,0.32,0.64)



part 2 result: sigma start from 1.28 end at 10.24 (1.28,2.56,5.12,10.24)



<u>Conclusion:</u>

From above we can see that when we fixed number of centers, sigma could not be too small or too large. When sigma is too small, i.e. Sigma = 0.4, we get very bad result. When sigma is too small, i.e. Sigma = 10.24, we also get very bad result. That is because when sigma is too small, the RBF's width will be very small, that may cause under fit. On contrary, if sigma is too large, RBF will get a big width, that may cause over fit.