(4) Plot program: chap6_4plot.m

```
close all;

figure(1);
subplot(211);
plot(t,yd1(:,1),'r',t,y(:,1),'b');
xlabel('time(s)');ylabel('Angle tracking of first link');
subplot(212);
plot(t,yd2(:,1),'r',t,y(:,3),'b');
xlabel('time(s)');ylabel('Angle tracking of second link');

figure(2);
subplot(211);
plot(t,yd1(:,2),'r',t,y(:,2),'b');
xlabel('time(s)');ylabel('Angle speed tracking of first link');
subplot(212);
plot(t,yd2(:,2),'r',t,y(:,4),'b');
xlabel('time(s)');ylabel('Angle speed tracking of second link');

figure(3);
subplot(211);
plot(t,y(:,5),'r',t,u(:,3),'b');
xlabel('time(s)');ylabel('F and Fc');
subplot(212);
plot(t,y(:,6),'r',t,u(:,4),'b');
xlabel('time(s)');ylabel('F and Fc');

figure(4);
subplot(211);
plot(t,u(:,1),'r');
xlabel('time(s)');ylabel('Control input of Link1');
subplot(212);
plot(t,u(:,2),'r');
xlabel('time(s)');ylabel('Control input of Link2');
```

References

1. L.X.Wang, *A Course in Fuzzy Systems and Control*, (Prentice-Hall International, Inc., 1996)
2. L.X. Wang, Stable adaptive fuzzy control of nonlinear systems. IEEE Trans. Fuzzy Syst. **1**(2), 146–155 (1993)
3. P.A. Ioannou, J. Sun, *Robust Adaptive Control*, (PTR Prentice-Hall, 1996), pp. 75–76
4. B.K. Yoo, W.C. Ham, Adaptive control of robot manipulator using fuzzy compensator. IEEE Trans. Fuzzy Syst. **8**(2), 186–199 (2000)

Chapter 7
Neural Networks

7.1 Introduction

Neural networks are networks of nerve cells (neurons) in the brain. The human brain has billions of individual neurons and trillions of interconnections. Neurons are continuously processing and transmitting information to one another.

In 1909, Cajal found that the brain consists of a large number of highly connected neurons which apparently can send very simple excitatory and inhibitory messages to each other and can update their excitations on the basis of these simple messages [1]. A neuron has three major regions: the cell body, the axon (send out messages), and the dendrites (receive messages). The cell body provides the support functions, the structure of the cell. The axon is a branching fiber which carries signals away from the neurons. The dendrites consist of more branching fibers which receive signals from other nerve cells.

The historical reviews of neural networks are as follows:

(1) In 1943, McCulloch and Pitts proposed first mathematical model of the neurons and showed how neuron-like networks could be computed.
(2) The first set of ideas of learning in neural networks was contained in Hebb's book entitled The Organization of Behaviour in 1949.
(3) In 1951, Edmonds and Minsky built their learning machine using Hebb's idea.
(4) The real beginning of a meaningful neuron-like network learning can be traced to the work of Rosenblatt in 1962. Rosenblatt invented a class of simple neuron-like learning networks which is called perceptron neural network.
(5) In a breakthrough paper published in 1982, Hopfield introduced a neural network architecture which is called Hopfield network. This NN can be used to solve optimization problems such as the traveling salesman problem.
(6) An important NN which has been widely used in NN is the back-error propagation or backpropagation (BP). BP NN was first presented in 1974 by Werbos and then was independently reinvented in 1986 by Rumelhart et al. [2].

J. Liu, *Intelligent Control Design and MATLAB Simulation*,
https://doi.org/10.1007/978-981-10-5263-7_7

Their book, Parallel Distributed Processing, introduced a broad perspective of the neural network approaches.

(7) RBF neural networks were addressed in 1988 [3], which have recently drawn much attention due to their good generalization ability and a simple network structure that avoids unnecessary and lengthy calculation as compared to the multilayer feed-forward network (MFN). Past research of universal approximation theorems on RBF have shown that any nonlinear function over a compact set with arbitrary accuracy can be approximated by RBF neural network [4]. There have been significant research efforts on RBF neural control for nonlinear systems.

RBF neural network has three layers: the input layer, the hidden layer, and the output layer. Neurons at the hidden layer are activated by a radial basis function. The hidden layer consists of an array of computing units called hidden nodes. Each hidden node contains a center $\boldsymbol{c}$ vector that is a parameter vector of the same dimension as the input vector $\boldsymbol{x}$, the Euclidean distance between the center and the network input vector $\boldsymbol{x}$ is defined by $\|x(t) - c_j(t)\|$.

7.2 Single Neural Network

From Fig. 7.1, the algorithm of single neural network can be described as

$$Net_i = \sum_j w_{ij}x_j + s_i - \theta_i \tag{7.1}$$

$$u_i = f(Net_i) \tag{7.2}$$

$$y_i = g(u_i) = h(Net_i) \tag{7.3}$$

where $g(u_i) = u_i$, $y_i = f(Net_i)$.

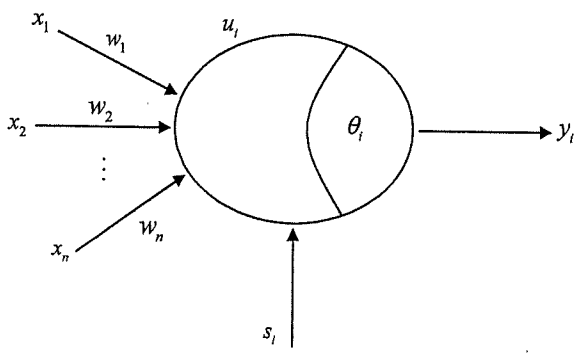


Fig. 7.1 Single NN model

Nonlinearity characteristic function $f(Net_i)$ can be divided as three kinds as follows:

(1) Threshold value

$$f(Net_i) = \begin{cases} 1 & Net_i > 0 \\ 0 & Net_i \le 0 \end{cases} \tag{7.4}$$

The threshold function is shown in Fig. 7.2.

(2) Linear function

$$f(Net_i) = \begin{cases} 0 & Net_i \le Net_{i0} \\ kNet_i & Net_{i0} < Net_i < Net_{i1} \\ f_{\max} & Net_i \ge Net_{i1} \end{cases} \tag{7.5}$$

Choose $Net_{i0} = 30$, $Net_{i1} = 70$, $f_{\max} = 5.0$, the linearity function is shown in Fig. 7.3.

(3) Nonlinear function

Sigmoid function and Gaussian function are often used in neural network. Sigmoid type is expressed as

$$f(Net_i) = \frac{1}{1 + e^{-\frac{Net_i}{T}}} \tag{7.6}$$

Choose $T = 1.0$, the sigmoid function is shown in Fig. 7.4.

Fig. 7.2 Threshold function

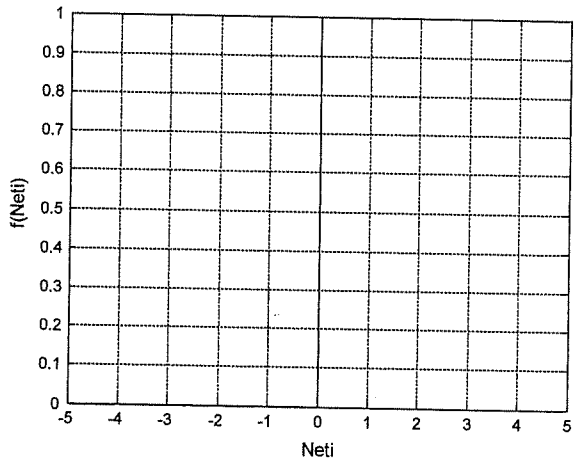


Fig. 7.3 Linearity function

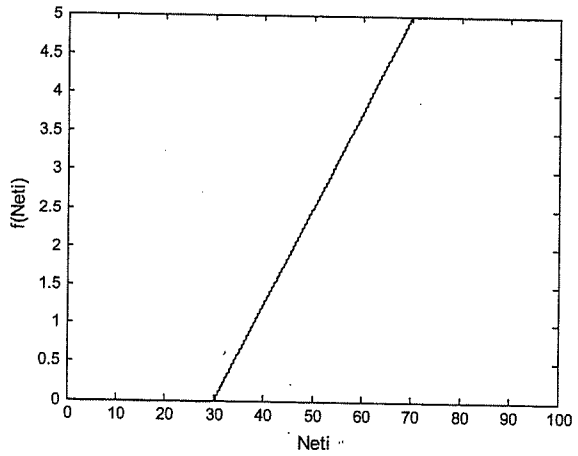
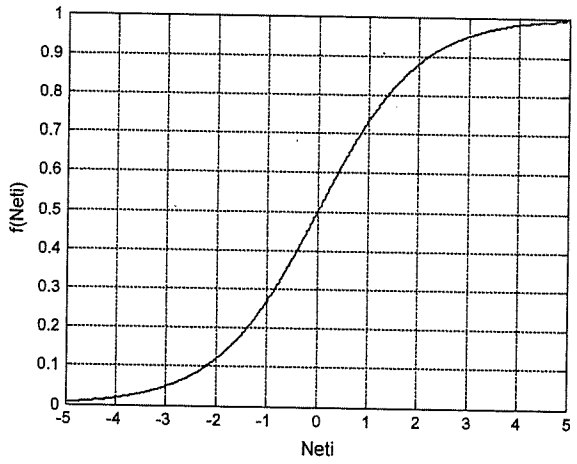


Fig. 7.4 Sigmoid function



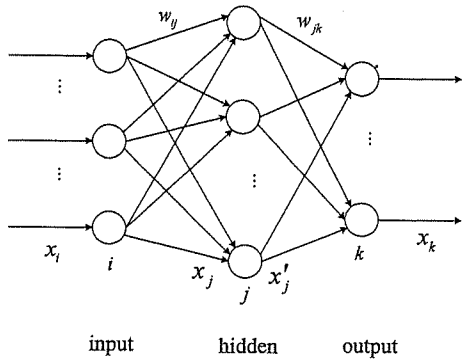
7.3 BP Neural Network Design and Simulation

The backpropagation (BP) neural network is a multilayered neural network. Thus, the BP algorithm employs three or more layers of processing unit (neurons).

7.3.1 *BP Network Structure*

Figure 7.5 shows a structure of a typical three-layered network for the BP algorithm. The leftmost layer of units is the input layer to which the input data is supplied. The layer after it is the hidden layer where the processing units are

Fig. 7.5 BP NN structure



interconnected to the layers before and after it. The rightmost layer is the output layer. The layers shown in Fig. 7.5 are fully interconnected, which means that each processing unit is connected to every unit in the previous layer and in the succeeding layer. However, units are not connected to other units in the same layer.

7.3.2 *Approximation of BP Neural Network*

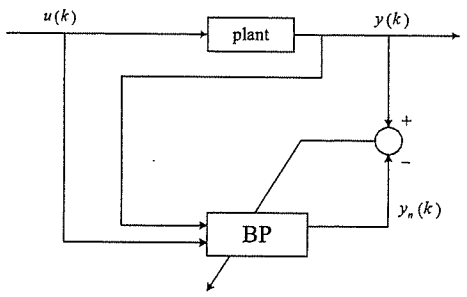
BP neural network scheme for approximation is shown in Fig. 7.6.
BP neural network structure for approximation is shown in Fig. 7.7.
Classical BP neural network algorithm is described as follows:

(1) Feed-forward calculation

Input of hidden layer is

$$x_j = \sum_i w_{ij} x_i \tag{7.7}$$

Fig. 7.6 BP approximation scheme



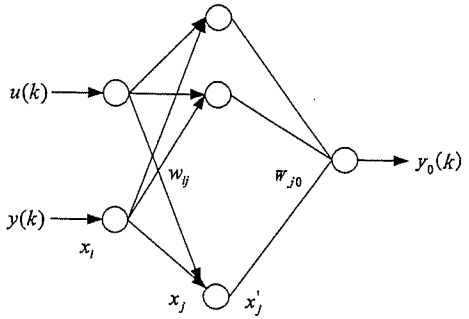


Fig. 7.7 BP neural network structure for approximation

Output of hidden layer is

$$x_j' = f(x_j) = \frac{1}{1+e^{-x_j}} \tag{7.8}$$

then

$$\frac{\partial x_j'}{\partial x_j} = x_j'(1-x_j')$$

Output of output layer is

$$y_o(k) = \sum_j w_{jo} x_j' \tag{7.9}$$

Then, the approximation error is

$$e(k) = y(k) - y_n(k)$$

Error index function is designed as

$$E = \frac{1}{2} e(k)^2 \tag{7.10}$$

(2) Learning algorithm of BP

According to the steepest descent (gradient) method, the learning of weight value w_{jo} is

$$\Delta w_{jo} = -\eta \frac{\partial E}{\partial w_{jo}} = \eta \cdot e(k) \cdot \frac{\partial y_o}{\partial w_{jo}} = \eta \cdot e(k) \cdot x_j'$$

The weight value at time $k+1$ is

$$w_{jo}(k+1) = w_{jo}(k) + \Delta w_{jo}$$

The learning of weight value w_{ij} is

$$\Delta w_{ij} = -\eta \frac{\partial E}{\partial w_{ij}} = \eta \cdot e(k) \cdot \frac{\partial y_o}{\partial w_{ij}}$$

where the chain rule is used, $\frac{\partial y_o}{\partial w_{ij}} = \frac{\partial y_o}{\partial x_j'} \cdot \frac{\partial x_j'}{\partial x_j} \cdot \frac{\partial x_j}{\partial w_{ij}} = w_{jo} \cdot \frac{\partial x_j'}{\partial x_j} \cdot x_i = w_{jo} \cdot x_j'(1-x_j') \cdot x_i$.

The weight value at time $k+1$ is

$$w_{ij}(k+1) = w_{ij}(k) + \Delta w_{ij}$$

Considering the effect of previous weight value change, the algorithm of weight value is

$$w_{jo}(k+1) = w_{jo}(k) + \Delta w_{jo} + \alpha\big(w_{jo}(k) - w_{jo}(k-1)\big) \tag{7.11}$$

$$w_{ij}(t+1) = w_{ij}(t) + \Delta w_{ij} + \alpha(w_{ij}(t) - w_{ij}(t-1)) \tag{7.12}$$

where η is learning rate, α is momentum factor, $\eta \in [0,1]$, $\alpha \in [0,1]$.

By using BP neural network approximation, Jacobian value can be calculated as follows:

$$\frac{\partial y(k)}{\partial u(k)} \approx \frac{\partial y_o(k)}{\partial u(k)} = \frac{\partial y_o(k)}{\partial x_j'} \times \frac{\partial x_j'}{\partial x_j} \times \frac{\partial x_j}{\partial x(1)} = \sum_j w_{jo} x_j' \left(1 - x_j'\right) w_{1j} \tag{7.13}$$

7.3.3 *Simulation Example*

The plant is as follows

$$y(k) = u(k)^3 + \frac{y(k-1)}{1+y(k-1)^2}$$

Input signal is chosen as $u(k) = 0.5\sin(6\pi t)$, let RBF neural network input vector as $\boldsymbol{x} = [u(k)\ y(k)]$, NN structure is chosen as 2-6-1, the initial value of W_{jo}, W_{ij} is chosen as random value in $[-1 \quad +1]$, $\eta = 0.50$, $\alpha = 0.05$.

The program is chap7_1.m, and the results are shown from Figs. 7.8, 7.9, and 7.10.

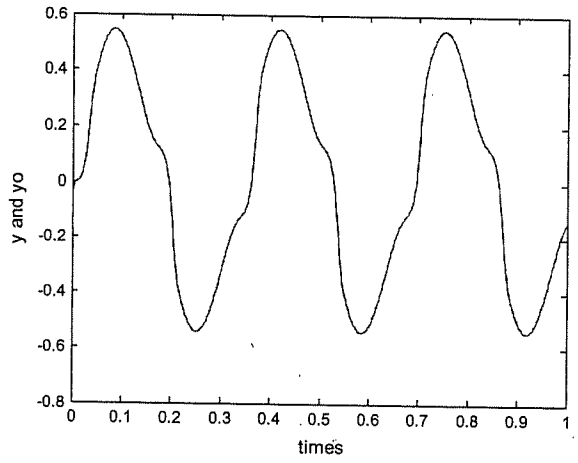


Fig. 7.8 BP approximation

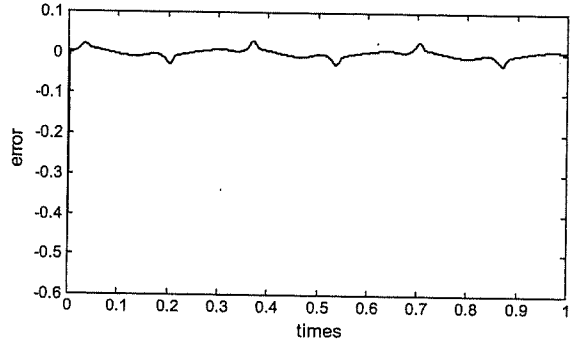


Fig. 7.9 BP approximation error

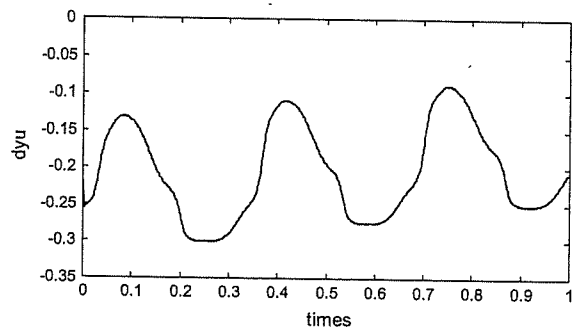


Fig. 7.10 Jacobian value identification

Simulation program: chap7_1.m

```
%BP approximation
clear all;
close all;

xite=0.50;
alfa=0.05;

wjo=rands(6,1);
wjo_1=wjo;wjo_2=wjo_1;

wij=rands(2,6);
wij_1=wij;wij_2=wij;

dwij=0*wij;

x=[0,0]';

u_1=0;
y_1=0;

I=[0,0,0,0,0,0]';
Iout=[0,0,0,0,0,0]';
FI=[0,0,0,0,0,0]';

ts=0.001;
for k=1:1:1000

time(k)=k*ts;
u(k)=0.50*sin(3*2*pi*k*ts);
y(k)=u_1^3+y_1/(1+y_1^2);

x(1)=u(k);
x(2)=y(k);

for j=1:1:6
   I(j)=x'*wij(:,j);
   Iout(j)=1/(1+exp(-I(j)));
end

yo(k)=wjo'*Iout;       % Output of NNI networks

e(k)=y(k)-yo(k);   % Error calculation

wjo=wjo_1+(xite*e(k))*Iout+alfa*(wjo_1-wjo_2);
```

```
for j=1:1:6
  FI(j)=exp(-I(j))/(1+exp(-I(j)))^2;
end

for i=1:1:2
  for j=1:1:6
    dwij(i,j)=e(k)*xite*FI(j)*wjo(j)*x(i);
  end
End
wij=wij_1+dwij+alfa*(wij_1-wij_2);

%%%%%%%%%%%%%%Jacobian%%%%%%%%%%%%%%%%%
yu=0;
for j=1:1:6
  yu=yu+wjo(j)*wij(1,j)*FI(j);
end
dyu(k)=yu;

wij_2=wij_1;wij_1=wij;
wjo_2=wjo_1;wjo_1=wjo;
u_1=u(k);
y_1=y(k);
end
figure(1);
plot(time,y,'r',time,yo,'b');
xlabel('times');ylabel('y and yo');
figure(2);
plot(time,y-yo,'r');
xlabel('times');ylabel('error');
figure(3);
plot(time,dyu);
xlabel('times');ylabel('dyu');
```

7.4 RBF Neural Network Design and Simulation

The radial basis function (RBF) neural network is a multilayered neural network. Like BP neural network structure, RBF algorithm also employs three layers of processing unit (neurons).

The difference between BP and RBF is that RBF have only output layer, and activation function is Gaussian function instead of S function in hidden layer, which will simplify the algorithm and decrease computational burden.

7.4.1 RBF Algorithm

The structure of a typical three-layer RBF neural network is shown as Fig. 7.11.

In RBF neural network, $\boldsymbol{x} = [x_i]^{\mathrm{T}}$ is input vector. Assuming there are mth neural nets, and radial basis function vector in hidden layer of RBF is $\boldsymbol{h} = [h_j]^{\mathrm{T}}$, h_j is Gaussian function value for neural net j in hidden layer, and

$$h_{\mathrm{j}} = \exp\left(-\frac{\|\boldsymbol{x} - c_j\|^2}{2b_j^2}\right) \tag{7.14}$$

where $\boldsymbol{c} = [c_{ij}] = \begin{bmatrix} c_{11} & \cdots & c_{1m} \\ \vdots & \ldots & \vdots \\ c_{n1} & \cdots & c_{nm} \end{bmatrix}$ represents the coordinate value of center point of the Gaussian function of neural net j for the ith input, $i = 1, 2, \ldots, n$, $j = 1, 2, \ldots, m$. For the vector $\boldsymbol{b} = [b_1, \ldots, b_m]^{\mathrm{T}}$, b_j represents the width value of Gaussian function for neural net j.

The weight value of RBF is

$$\boldsymbol{w} = [w_1, \ldots, w_m]^{\mathrm{T}} \tag{7.15}$$

The output of RBF neural network is

$$y(t) = \boldsymbol{w}^{\mathrm{T}}\boldsymbol{h} = w_1h_1 + w_2h_2 + \cdots + w_mh_m \tag{7.16}$$

7.4.2 RBF Design Example with MATLAB Simulation

7.4.2.1 For Structure 1-5-1 RBF Neural Network

Consider a structure 1-5-1 RBF neural network, we have one input as $x = x_1$, and $\boldsymbol{b} = [b_1 \;\; b_2 \;\; b_3 \;\; b_4 \;\; b_5]^{\mathrm{T}}$, $\boldsymbol{c} = [c_{11} \;\; c_{12} \;\; c_{13} \;\; c_{14} \;\; c_{15}]$, $\boldsymbol{h} = [h_1 \;\; h_2 \;\; h_3 \;\; h_4 \;\; h_5]^{\mathrm{T}}$, $\boldsymbol{w} = [w_1 \;\; w_2 \;\; w_3 \;\; w_4 \;\; w_5]$, and $y(t) = \boldsymbol{w}^{\mathrm{T}}\boldsymbol{h} = w_1h_1 + w_2h_2 + w_3h_3 + w_4h_4 + w_5h_5$.

Choose the input as $\sin t$, the output of RBF is shown in Fig. 7.12, the output of hidden neural net is shown in Fig. 7.13.

The Simulink program of this example is chap7_2sim.mdl, and MATLAB programs of the example are given in the Appendix.

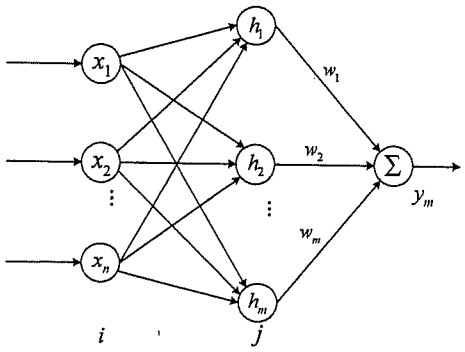


Fig. 7.11 RBF neural network structure

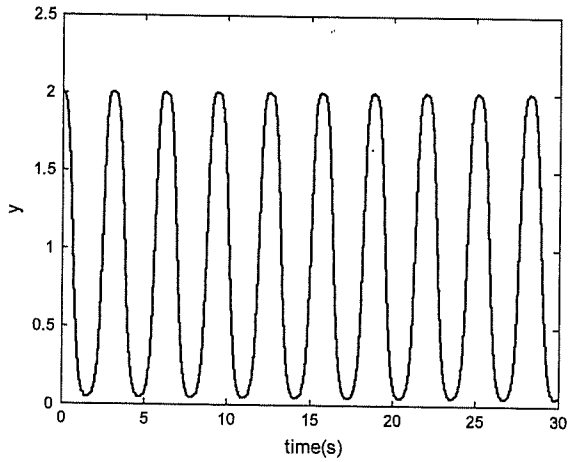


Fig. 7.12 Output of RBF

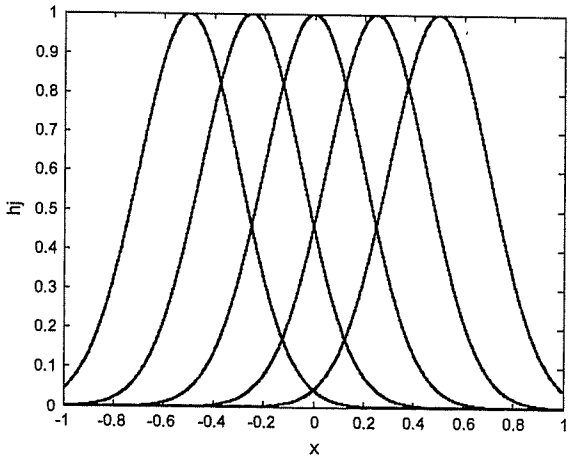
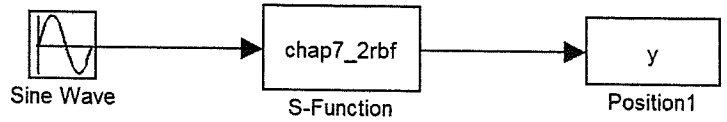
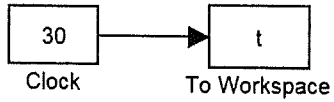


Fig. 7.13 Output of hidden neural net

Simulation programs:

(1) **Simulink main program: chap7_2sim.mdl**





(2) **S function of RBF: chap7_2rbf.m**

```
function [sys,x0,str,ts] = spacemodel(t,x,u,flag)
switch flag,
case 0,
  [sys,x0,str,ts]=mdlInitializeSizes;
case 3,
  sys=mdlOutputs(t,x,u);
case {2,4,9}
  sys=[];
otherwise
  error(['Unhandled flag = ',num2str(flag)]);
end
function [sys,x0,str,ts]=mdlInitializeSizes
sizes = simsizes;
sizes.NumContStates  = 0;
sizes.NumDiscStates  = 0;
sizes.NumOutputs     = 7;
sizes.NumInputs      = 1;
sizes.DirFeedthrough = 1;
sizes.NumSampleTimes = 0;
sys = simsizes(sizes);
x0  = [];
str = [];
ts  = [];
function sys=mdlOutputs(t,x,u)
x=u(1);  %Input Layer
```

```
%i=1
%j=1,2,3,4,5
%k=1
c=[-0.5 -0.25 0 0.25 0.5]; %cij
b=[0.2 0.2 0.2 0.2 0.2]'; %bj

W=ones(5,1); %Wj
h=zeros(5,1); %hj
for j=1:1:5
  h(j)=exp(-norm(x-c(:,j))^2/(2*b(j)*b(j))); %Hidden Layer
end
y=W'*h;  %Output Layer

sys(1)=y;
sys(2)=x;
sys(3)=h(1);
sys(4)=h(2);
sys(5)=h(3);
sys(6)=h(4);
sys(7)=h(5);
```

(3) **Plot program: chap7_2plot.m**

```
close all;

% y=y(:,1);
% x=y(:,2);
% h1=y(:,3);
% h2=y(:,4);
% h3=y(:,5);
% h4=y(:,6);
% h5=y(:,7);

figure(1);
plot(t,y(:,1),'k','linewidth',2);
xlabel('time(s)');ylabel('y');

figure(2);
plot(y(:,2),y(:,3),'k','linewidth',2);
xlabel('x');ylabel('hj');
hold on;
plot(y(:,2),y(:,4),'k','linewidth',2);
hold on;
plot(y(:,2),y(:,5),'k','linewidth',2);
hold on;
plot(y(:,2),y(:,6),'k','linewidth',2);
hold on;
plot(y(:,2),y(:,7),'k','linewidth',2);
```

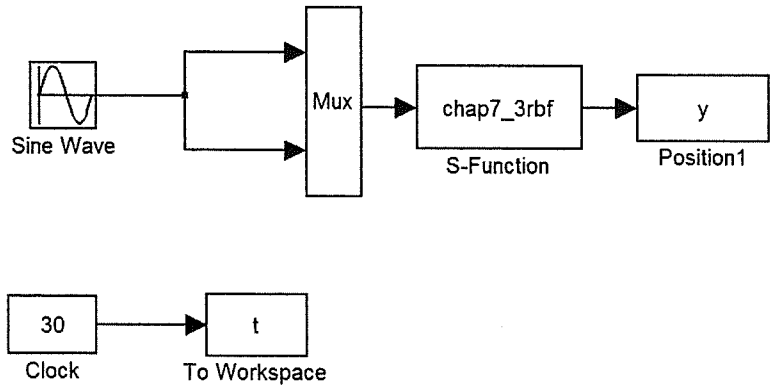

7.4.2.2 For Structure 2-5-1 RBF Neural Network

Consider a structure 2-5-1 RBF neural network, we have $\mathbf{x} = [x_1, x_2]^{\mathrm{T}}$, $\mathbf{b} = [b_1 \quad b_2 \quad b_3 \quad b_4 \quad b_5]^{\mathrm{T}}$, $\mathbf{c} = \begin{bmatrix} c_{11} & c_{12} & c_{13} & c_{14} & c_{15} \\ c_{21} & c_{22} & c_{23} & c_{24} & c_{25} \end{bmatrix}$, $\mathbf{h} = [h_1 \quad h_2 \quad h_3 \quad h_4 \quad h_5]^{\mathrm{T}}$, $\boldsymbol{w} = [w_1 \quad w_2 \quad w_3 \quad w_4 \quad w_5]^{\mathrm{T}}$, and $y(t) = \boldsymbol{w}^{\mathrm{T}}\boldsymbol{h} = w_1h_1 + w_2h_2 + w_3h_3 + w_4h_4 + w_5h_5$.

Two inputs are chosen as $\sin t$, the output of RBF is shown in Fig. 7.14, and the output of hidden neural net is shown in Figs. 7.15 and 7.16.

Simulation programs:

(1) **Simulink main program: chap7_3sim.mdl**



(2) **S function of RBF: chap7_3rbf.m**

```
function [sys,x0,str,ts] = spacemodel(t,x,u,flag)
switch flag,
case 0,
  [sys,x0,str,ts]=mdlInitializeSizes;
case 3,
  sys=mdlOutputs(t,x,u);
case {2,4,9}
  sys=[];
```

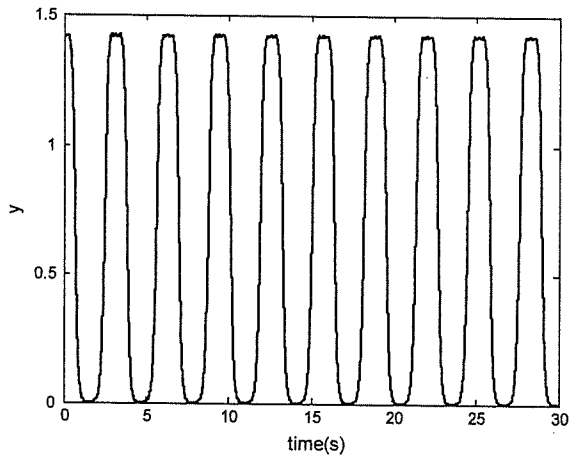


Fig. 7.14 Output of RBF

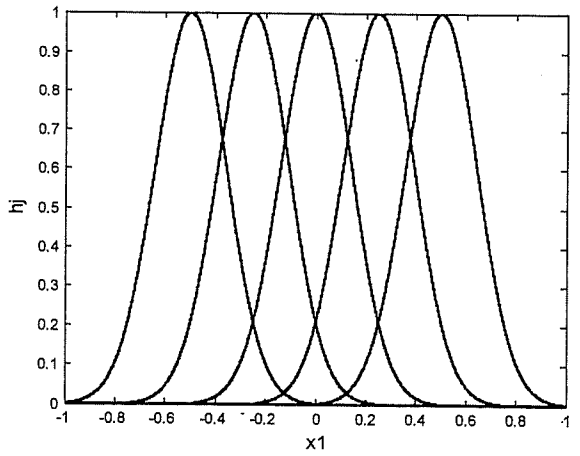


Fig. 7.15 Output of hidden neural net for first input

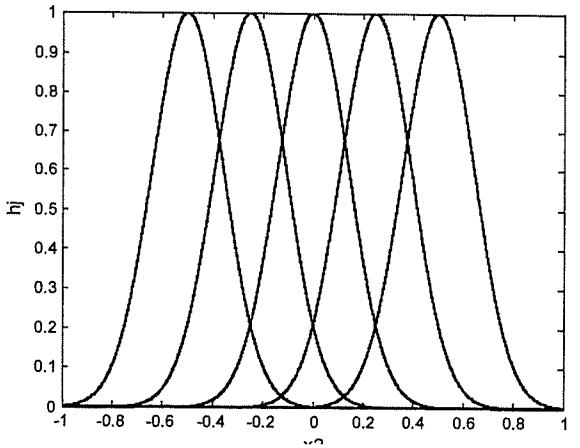


Fig. 7.16 Output of hidden neural net for second input


```
otherwise
  error(['Unhandled flag = ',num2str(flag)]);
end
function [sys,x0,str,ts]=mdlInitializeSizes
sizes = simsizes;
sizes.NumContStates  = 0;
sizes.NumDiscStates  = 0;
sizes.NumOutputs     = 8;
sizes.NumInputs      = 2;
sizes.DirFeedthrough = 1;
sizes.NumSampleTimes = 0;
sys = simsizes(sizes);
x0  = [];
str = [];
ts  = [];
function sys=mdlOutputs(t,x,u)
x1=u(1);  %Input Layer
x2=u(2);
x=[x1 x2]';

%i=2
%j=1,2,3,4,5
%k=1
c=[-0.5 -0.25 0 0.25 0.5;
  -0.5 -0.25 0 0.25 0.5];  %cij
b=[0.2 0.2 0.2 0.2 0.2]';   %bj

W=ones(5,1);  %Wj
h=zeros(5,1); %hj
for j=1:1:5
  h(j)=exp(-norm(x-c(:,j))^2/(2*b(j)*b(j)));  %Hidden Layer
end
yout=W'*h;   %Output Layer

sys(1)=yout;
sys(2)=x1;
sys(3)=x2;
sys(4)=h(1);
sys(5)=h(2);
sys(6)=h(3);
sys(7)=h(4);
sys(8)=h(5);
```

(3) **Plot program: chap7_3plot.m**

```
close all;
% y=y(:,1);
% x1=y(:,2);
% x2=y(:,3);
% h1=y(:,4);
% h2=y(:,5);
% h3=y(:,6);
% h4=y(:,7);
% h5=y(:,8);

figure(1);
plot(t,y(:,1),'k','linewidth',2);
xlabel('time(s)');ylabel('y');

figure(2);
plot(y(:,2),y(:,4),'k','linewidth',2);
xlabel('x1');ylabel('hj');
hold on;
plot(y(:,2),y(:,5),'k','linewidth',2);
hold on;
plot(y(:,2),y(:,6),'k','linewidth',2);
hold on;
plot(y(:,2),y(:,7),'k','linewidth',2);
hold on;
plot(y(:,2),y(:,8),'k','linewidth',2);

figure(3);
plot(y(:,3),y(:,4),'k','linewidth',2);
xlabel('x2');ylabel('hj');
hold on;
plot(y(:,3),y(:,5),'k','linewidth',2);
hold on;
plot(y(:,3),y(:,6),'k','linewidth',2);
hold on;
plot(y(:,3),y(:,7),'k','linewidth',2);
hold on;
plot(y(:,3),y(:,8),'k','linewidth',2);
```


7.5 RBF Neural Network Approximation Based on Gradient Descent Method

7.5.1 RBF Neural Network Approximation

We use RBF neural network to approximate a plant, the structure is shown in Fig. 7.17.

In RBF neural network, $\boldsymbol{x} = [x_1 \quad x_2 \quad \ldots \quad x_n]^T$ is the input vector, and h_j is Gaussian function for neural net j, then

$$h_j = \exp\left(-\frac{\|\boldsymbol{x} - \boldsymbol{c}_j\|^2}{2b_j^2}\right), j = 1, 2, \ldots, m \tag{7.17}$$

where $\boldsymbol{c}_j = [c_{j1}, \ldots, c_{jn}]$ is the center vector of neural net j.

The width vector of Gaussian function is

$$\boldsymbol{b} = [b_1, \ldots, b_m]^T$$

where $b_j > 0$ represents the width value of Gaussian function for neural net j.

The weight value is

$$\boldsymbol{w} = [w_1, \ldots, w_m]^T \tag{7.18}$$

The output of RBF is

$$y_m(t) = w_1 h_1 + w_2 h_2 + \cdots + w_m h_m \tag{7.19}$$

The performance index function of RBF is

$$E(t) = \frac{1}{2}(y(t) - y_m(t))^2 \tag{7.20}$$

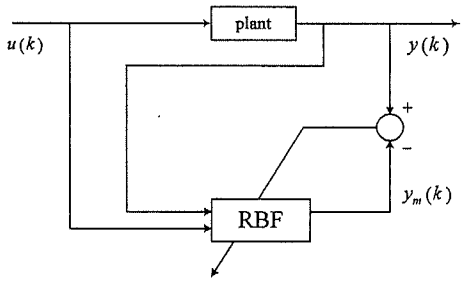


Fig. 7.17 RBF neural network approximation

According to gradient descent method, the parameters can be updated as follows:

$$\Delta w_j(t) = -\eta \frac{\partial E}{\partial w_j} = \eta(y(t) - y_m(t))h_j$$

$$w_j(t) = w_j(t-1) + \Delta w_j(t) + \alpha\big(w_j(t-1) - w_j(t-2)\big) \tag{7.21}$$

$$\Delta b_j = -\eta \frac{\partial E}{\partial b_j} = \eta(y(t) - y_m(t))w_j h_j \frac{\|\boldsymbol{x} - \boldsymbol{c}_j\|^2}{b_j^3} \tag{7.22}$$

$$b_j(t) = b_j(t-1) + \Delta b_j + \alpha\big(b_j(t-1) - b_j(t-2)\big) \tag{7.23}$$

$$\Delta c_{ji} = -\eta \frac{\partial E}{\partial c_{ji}} = \eta(y(t) - y_m(t))w_j \frac{x_j - c_{ji}}{b_j^2} \tag{7.24}$$

$$c_{ji}(t) = c_{ji}(t-1) + \Delta c_{ji} + \alpha\big(c_{ji}(t-1) - c_{ji}(t-2)\big) \tag{7.25}$$

where $\eta \in (0,1)$ is the learning rate, $\alpha \in (0,1)$ is momentum factor.

In RBF neural network approximation, the parameters of $\boldsymbol{c}_i$ and b_i must be chosen according to the scope of the input value. If the parameters $\boldsymbol{c}_i$ and b_i are chosen inappropriately, Gaussian function will not be effectively mapped, and RBF network will be invalid. The gradient descent method is an effective method to adjust $\boldsymbol{c}_i$ and b_i in RBF neural network approximation.

If the initial $\boldsymbol{c}_j$ and $\boldsymbol{b}$ are set in the effective range of inputs of RBF, we can only update weight value with fixed $\boldsymbol{c}_j$ and $\boldsymbol{b}$.

7.5.2 *Simulation Example*

First example: only update w

Using RBF neural network to approximate the following discrete plant

$$G(s) = \frac{133}{s^2 + 25s}$$

Consider a structure 2-5-1 RBF neural network, we choose inputs as $x(1) = u(t)$, $x(2) = y(t)$, and set $\alpha = 0.05$, $\eta = 0.5$. The initial weight value is chosen as random value between 0 and 1.

Choose the input as $u(t) = \sin t$, consider the range of the first input $x(1)$ is $[0,1]$, the range of the second input $x(2)$ is about $[0,10]$, we choose the initial parameters of Gaussian function as $\boldsymbol{c}_j = \begin{bmatrix} -1 & -0.5 & 0 & 0.5 & 1 \\ -10 & -5 & 0 & 5 & 10 \end{bmatrix}^{\mathrm{T}}$, $\boldsymbol{b}_j = 1.5, j = 1,2,3,4,5$.

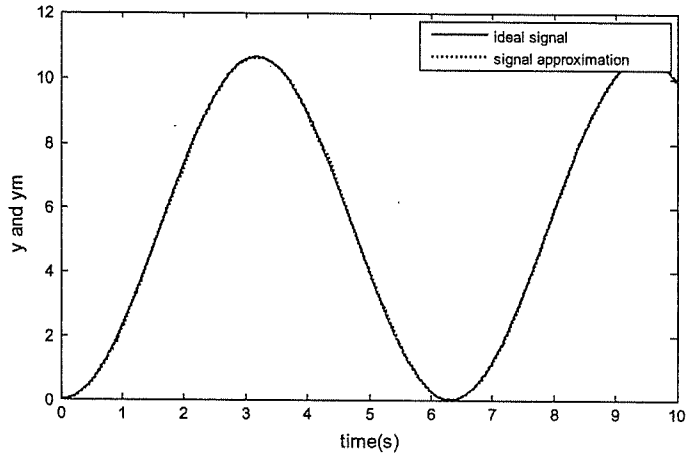
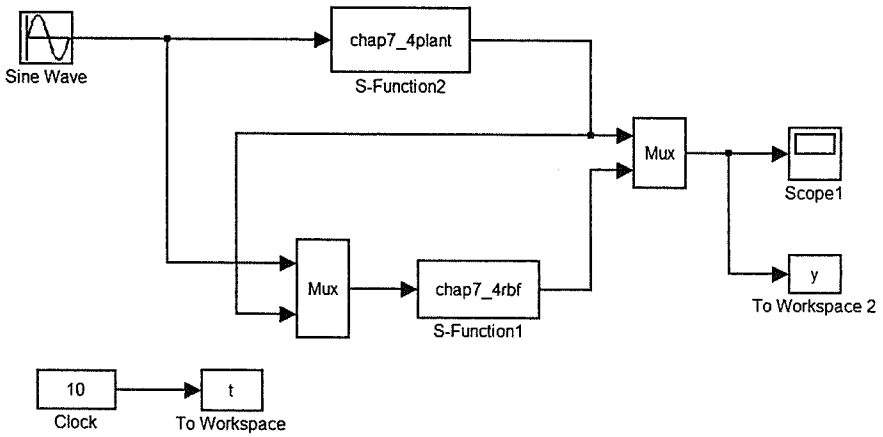


Fig. 7.18 RBF neural network approximation

In the simulation, we only update w with fixed $\boldsymbol{c}_j$ and $\boldsymbol{b}$ in RBF neural network approximation, the results are shown in Fig. 7.18.

Simulation programs:

(1) **Simulink main program: chap7_4sim.mdl**



(2) **S function of RBF: chap7_4rbf.m**

```
function [sys,x0,str,ts]=s_function(t,x,u,flag)
switch flag,
case 0,
```

```
  [sys,x0,str,ts]=mdlInitializeSizes;
case 3,
  sys=mdlOutputs(t,x,u);
case {2, 4, 9 }
  sys = [];
otherwise
  error(['Unhandled flag = ',num2str(flag)]);
end

function [sys,x0,str,ts]=mdlInitializeSizes
sizes = simsizes;
sizes.NumContStates  = 0;
sizes.NumDiscStates  = 0;
sizes.NumOutputs     = 1;
sizes.NumInputs      = 2;
sizes.DirFeedthrough = 1;
sizes.NumSampleTimes = 0;
sys=simsizes(sizes);
x0=[];
str=[];
ts=[];
function sys=mdlOutputs(t,x,u)
persistent w w_1 w_2 b ci
alfa=0.05;
xite=0.5;
if t==0
  b=1.5;
  ci=[-1 -0.5 0 0.5 1;
    -10 -5 0 5 10];
  w=rands(5,1);
  w_1=w;w_2=w_1;
end
ut=u(1);
yout=u(2);
xi=[ut yout]';
for j=1:1:5
  h(j)=exp(-norm(xi-ci(:,j))^2/(2*b^2));
end
ymout=w'*h';

d_w=0*w;
for j=1:1:5 %Only weight value update
 d_w(j)=xite*(yout-ymout)*h(j);
end
```



```
w=w_1+d_w+alfa*(w_1-w_2);

w_2=w_1;w_1=w;
sys(1)=ymout;
```

(3) Plot program: chap7_4plot.m

```
close all;

close all;
figure(1);
plot(t,y(:,1),'r',t,y(:,2),'k:','linewidth',2);
xlabel('time(s)');ylabel('y and ym');
legend('ideal signal','signal approximation');
```

Second example: update w, c_j, b by gradient descent method

Using RBF neural network to approximate the following discrete plant

$$y(k) = u(k)^3 + \frac{y(k-1)}{1+y(k-1)^2}$$

Consider a structure 2-5-1 RBF neural network, and we choose $x(1) = u(k)$, $x(2) = y(k)$, and $\alpha = 0.05$, $\eta = 0.15$. The initial weight value is chosen as random value between 0 and 1. Choose the input as $u(k) = \sin t$, $t = k \times T$, $T = 0.001$, we set the initial parameters of Gaussian function as $\boldsymbol{c}_j = \begin{bmatrix} -1 & -0.5 & 0 & 0.5 & 1 \\ -1 & -0.5 & 0 & 0.5 & 1 \end{bmatrix}^{\mathrm{T}}$, $\boldsymbol{b}_j = 3.0$, $j = 1, 2, 3, 4, 5$.

In the simulation, $M = 1$ indicates only updating $\boldsymbol{w}$ with fixed $\boldsymbol{c}_j$ and $\boldsymbol{b}$ and $M = 2$ indicates updating $\boldsymbol{w}$, $\boldsymbol{c}_j$, $\boldsymbol{b}$, the results are shown from Figs. 7.19 and 7.20.

From the simulation test, we can see that better results can be gotten than only adjusting $\boldsymbol{w}$ by the gradient descent method, especially the initial parameters of Gaussian function $\boldsymbol{c}_j$ and $\boldsymbol{b}$ are chosen not suitably.

Simulation program: chap7_5.m

```
%RBF approximation
clear all;
close all;

alfa=0.05;
xite=0.15;
x=[0,1]';
```

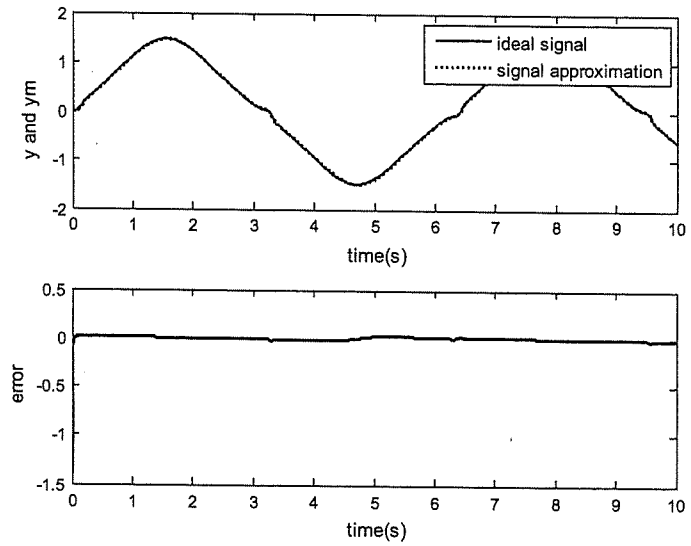


Fig. 7.19 RBF neural network approximation by only updating w (M = 1)

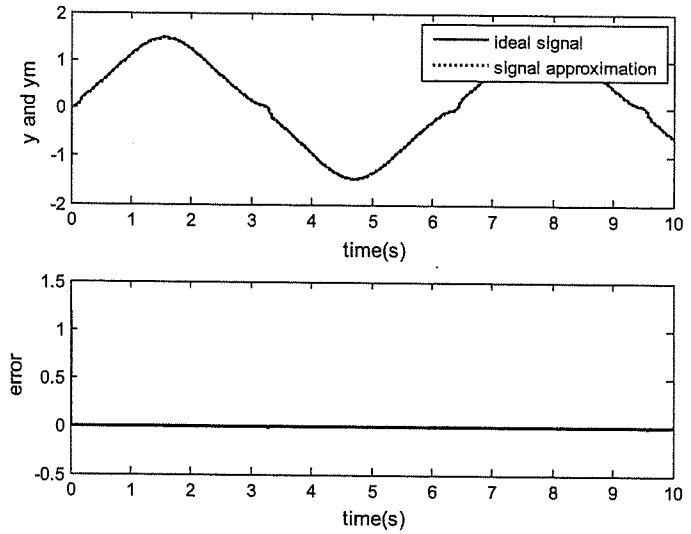


Fig. 7.20 RBF neural network approximation by updating w, b, c (M = 2)


```
b=3*ones(5,1);
c=[-1 -0.5 0 0.5 1;
  -1 -0.5 0 0.5 1];
w=rands(5,1);

w_1=w;w_2=w_1;
c_1=c;c_2=c_1;
b_1=b;b_2=b_1;
d_w=0*w;
d_b=0*b;
y_1=0;

ts=0.001;
for k=1:1:10000

time(k)=k*ts;
u(k)=sin(k*ts);

y(k)=u(k)^3+y_1/(1+y_1^2);

x(1)=u(k);
x(2)=y_1;

for j=1:1:5
  h(j)=exp(-norm(x-c(:,j))^2/(2*b(j)*b(j)));
end
ym(k)=w'*h';
em(k)=y(k)-ym(k);

M=1;
if M==1  %Only weight value update
  d_w(j)=xite*em(k)*h(j);
elseif M==2 %Update w,b,c
  for j=1:1:5
    d_w(j)=xite*em(k)*h(j);
    d_b(j)=xite*em(k)*w(j)*h(j)*(b(j)^-3)*norm(x-c(:,j))^2;
   for i=1:1:2
    d_c(i,j)=xite*em(k)*w(j)*h(j)*(x(i)-c(i,j))*(b(j)^-2);
   end
  end
  b=b_1+d_b+alfa*(b_1-b_2);
  c=c_1+d_c+alfa*(c_1-c_2);
end
w=w_1+d_w+alfa*(w_1-w_2);

y_1=y(k);
```

```
w_2=w_1;
w_1=w;

c_2=c_1;
c_1=c;

b_2=b_1;
b_1=b;
end
figure(1);
subplot(211);
plot(time,y,'r',time,ym,'k:','linewidth',2);
xlabel('time(s)');ylabel('y and ym');
legend('ideal signal','signal approximation');
subplot(212);
plot(time,y-ym,'k','linewidth',2);
xlabel('time(s)');ylabel('error');
```

7.6 Effects of Analysis on RBF Approximation

We consider approximation of the following discrete plant $y(k) = u(k)^3 + \frac{y(k-1)}{1+y(k-1)^2}$

7.6.1 *Effects of Gaussian Function Parameters on RBF Approximation*

In the simulation, we choose $\alpha = 0.05$, $\eta = 0.3$. The initial weight value is chosen as zeros.

From Gaussian function expression, we know that the effect of Gaussian function is related to the design of center vector c_j, width value b_j, and the number of hidden nets. The principle of c_j and b_j design should be as follows:

(1) Width value b_j represents the width of Gaussian function. The bigger value b_j is, the wider Gaussian function is. The width of Gaussian function represents the covering scope for the network input. The wider the Gaussian function is, the greater the covering scope of the network for the input is, otherwise worse covering scope is. Width value b_j should be designed moderate.

(2) Center vector c_j represents the center coordination of Gaussian function for neural net j. The nearer c_j is to the input value, the better sensitivity of Gaussian function is to the input value, otherwise the worse sensitivity is. Center vector c_j should be designed moderate.

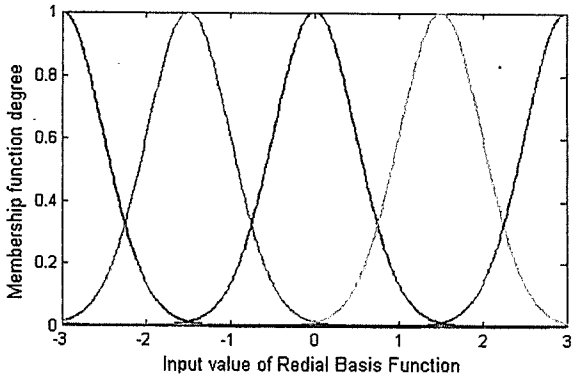


Fig. 7.21 Five Gaussian membership function

(3) The center vector c_j should be designed within the effective mapping of Gaussian membership function. For example, the scope of RBF input value is $[-3, +3]$, and then, the center vector c_j should be set in $[-3, +3]$.

In simulation, we should design the center vector c_j and the width value b_j according to the scope of practical network input value, in other words, the input value must be within the effective mapping of Gaussian membership function. Five Gaussian membership functions are shown in Fig. 7.21.

Simulation program:

Five Gaussian membership function design: chap7_6.m

```
%RBF function
clear all;
close all;

c=[-3 -1.5 0 1.5 3];

M=1;
if M==1
  b=0.50*ones(5,1);
elseif M==2
  b=1.50*ones(5,1);
end

h=[0,0,0,0,0]';

ts=0.001;
for k=1:1:2000

time(k)=k*ts;
```