Chapter 2
RBF Neural Network Design and Simulation

Abstract This chapter introduces RBF neural network design method, gives RBF neural network approximation algorithm based on gradient descent, analyzes the effects of Gaussian function parameters on RBF approximation, and introduces RBF neural network modeling method based on off-line training. Several simulation examples are given.

Keywords Neural network control • Gradient descent rule • Gaussian function • RBF training

2.1 RBF Neural Network Design and Simulation

2.1.1 RBF Algorithm

The structure of a typical three-layer RBF neural network is shown as Fig. 2.1.

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

$$h_j = \exp \left( -\frac{\|x - c_j\|^2}{2b_j^2} \right)$$

(2.1)

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

\[ \mathbf{w} = [w_1, \ldots, w_m]^T. \]  

(2.2)

The output of RBF neural network is

\[ y(t) = \mathbf{w}^T \mathbf{h} = w_1 h_1 + w_2 h_2 + \cdots + w_m h_m. \]  

(2.3)

### 2.1.2 RBF Design Example with Matlab Simulation

#### 2.1.2.1 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

\[ \mathbf{b} = [b_1 \ b_2 \ b_3 \ b_4 \ b_5]^T, \mathbf{c} = [c_{11} \ c_{12} \ c_{13} \ c_{14} \ c_{15}], \mathbf{h} = [h_1 \ h_2 \ h_3 \ h_4 \ h_5]^T, \]

\[ \mathbf{w} = [w_1 \ w_2 \ w_3 \ w_4 \ w_5]^T, \text{ and } y(t) = \mathbf{w}^T \mathbf{h} = w_1 h_1 + w_2 h_2 + w_3 h_3 + w_4 h_4 + w_5 h_5. \]

Choose the input as \( \sin t \); the output of RBF is shown in Fig. 2.2, and the output of hidden neural net is shown in Fig. 2.3.
2.1 RBF Neural Network Design and Simulation

2.1.2.2 For Structure 2-5-1 RBF Neural Network

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

Two inputs are chosen as $\sin t$. The output of RBF is shown in Fig. 2.4, and the output of hidden neural net is shown in Figs. 2.5 and 2.6.

The Simulink program of this example is chap2_1sim.mdl, and the Matlab programs of the example are given in the Appendix.
The Simulink program of this example is chap2_2sim.mdl, and the Matlab programs of the example are given in the Appendix.

2.2 RBF Neural Network Approximation Based on Gradient Descent Method

2.2.1 RBF Neural Network Approximation

We use RBF neural network to approximate a plant; the structure is shown in Fig. 2.7.
In RBF neural network, \( x = [x_1 \ x_2 \ \cdots \ x_n]^T \) is the input vector, and \( h_j \) is Gaussian function for neural net \( j \), then

\[
h_j = \exp\left(-\frac{\|x - c_j\|^2}{2b_j^2}\right), \quad j = 1, 2, \ldots, m. \tag{2.4}
\]

where \( c_j = [c_{j1}, \ldots, c_{jn}] \) is the center vector of neural net \( j \).

The width vector of Gaussian function is

\[
b = [b_1, \ldots, b_m]^T
\]

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

The weight value is

\[
w = [w_1, \ldots, w_m]^T \tag{2.5}
\]

The output of RBF is

\[
y_m(t) = w_1h_1 + w_2h_2 + \cdots + w_mh_m. \tag{2.6}
\]

The performance index function of RBF is

\[
E(t) = \frac{1}{2} (y(t) - y_m(t))^2. \tag{2.7}
\]

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

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

\[
w_j(t) = w_j(t-1) + \Delta w_j(t) + \alpha (w_j(t-1) - w_j(t-2)) \tag{2.8}
\]

\[
\Delta b_j = -\eta \frac{\partial E}{\partial b_j} = \eta (y(t) - y_m(t))w_jh_j \frac{\|x - c_j\|^2}{b_j^3} \tag{2.9}
\]
\[ b_j(t) = b_j(t - 1) + \Delta b_j + \alpha(b_j(t - 1) - b_j(t - 2)) \quad (2.10) \]

\[ \Delta c_{ji} = -\eta \frac{\partial E}{\partial c_{ji}} = \eta(y(t) - y_m(t))w_j \frac{x_j - c_{ji}}{b^2_j} \quad (2.11) \]

\[ c_{ji}(t) = c_{ji}(t - 1) + \Delta c_{ji} + \alpha(c_{ji}(t - 1) - c_{ji}(t - 2)) \quad (2.12) \]

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

In RBF neural network approximation, the parameters of \( c_i \) and \( b_i \) must be chosen according to the scope of the input value. If the parameter values 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 \( c_i \) and \( b_i \) in RBF neural network approximation.

If the initial \( c_j \) and \( b \) are set in the effective range of input of RBF, we can only update weight value with fixed \( c_j \) and \( b \).

### 2.2.2 Simulation Example

#### 2.2.2.1 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, and we choose \( x(1) = u(t) \), \( x(2) = y(t) \), and \( \alpha = 0.05 \), \( \eta = 0.5 \). The initial weight value is chosen as random value between 0 and 1. Consider the range of the first input is [0,1] and the range of the second input is about [0,10]; we choose the initial parameters of Gaussian function as \( c_j = \begin{bmatrix} -1 & -0.5 & 0 & 0.5 & 1 \\ -10 & -5 & 0 & 5 & 10 \end{bmatrix}^T \), \( b_j = 1.5, \ j = 1, 2, 3, 4, 5 \).

Choose the input as \( u(t) = \sin t \) : in the simulation, we only update \( w \) with fixed \( c_j \) and \( b \) in RBF neural network approximation. The results are shown in Fig. 2.8.

The Simulink program of this example is chap2_3sim.mdl, and the Matlab programs of the example are given in the Appendix.

#### 2.2.2.2 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, and the initial parameters of Gaussian function are chosen as

$$c_j = \begin{bmatrix} -1 & -0.5 & 0 & 0.5 & 1 \end{bmatrix}^T, \quad b_j = 3.0, \quad j = 1, 2, 3, 4, 5.$$ 

Choose the input as $u(k) = \sin t$, $t = k \times T$, $T = 0.001$: in simulation, $M = 1$ indicates only update $w$ with fixed $c_j$ and $b$ and $M = 2$ indicates update $w$, $c_j$, $b$ in RBF neural network approximation; the initial value of the input is set as $[0,1]$, and the results are shown from Figs. 2.9 and 2.10.

From the simulation test, we can see that better results can be gotten by the gradient descent method, especially the initial parameters of Gaussian function $c_j$ and $b$ are chosen not suitably.

The program of this example is chap2_4.m, which is given in the Appendix.

2.3 Effect of Gaussian Function Parameters on RBF Approximation

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. 2.9 RBF neural network approximation by only updating $w(M = 1)$

Fig. 2.10 RBF neural network approximation by updating $w,b,c(M = 2)$
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 $[-3, +3]$, 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; thus, input value the can be within the effective mapping of Gaussian membership function. Five Gaussian membership functions are shown in Fig. 2.11.

In the simulation, we choose the input of RBF as $0.5 \sin(2\pi t)$ and set the structure as 2-5-1. By changing $c_j$ and $b_j$ value, the effects of $c_j$ and $b_j$ on RBF approximation are given.

Now we analyze the effect of different $c_j$ and $b_j$ on RBF approximation as follows:

1. RBF approximation with moderate $b_j$ and $c_j$ ($M_b = 1, M_c = 1$)
2. RBF approximation with improper $b_j$ and moderate $c_j$ ($M_b = 2, M_c = 1$)
3. RBF approximation with moderate $b_j$ and improper $c_j$ ($M_b = 1, M_c = 2$)
4. RBF approximation with improper $b_j$ and $c_j$ ($M_b = 2, M_c = 2$)

The results are shown from Figs. 2.12, 2.13, 2.14, and 2.15. From the results, we can see if we design improper $c_j$ and $b_j$, the RBF approximation performance will not be ensured.

The program of this example is chap2_5.m and chap2_6.m, which are given in the Appendix.
2.4 Effect of Hidden Nets Number on RBF Approximation

From Gaussian function expression, besides the moderate center vector $c_j$ and width value $b_j$, the approximation error is also related to the number of hidden nets.

In the simulation, we choose $\alpha = 0.05, \eta = 0.3$. The initial weight value is chosen as zeros, and the parameter of Gaussian function is chosen as $b_j = 1.5$. The inputs of RBF are $u(k) = \sin t$ and $y(k)$. Set the structure as 2-m-1: $m$ represents the
number of hidden nets. We analyze the effect of different number of hidden nets on RBF approximation as \( m = 1 \), \( m = 3 \), and \( m = 7 \). According to the practical scope of the two inputs \( u(k) \) and \( y(k) \), for different \( m \), the parameter \( c_j \) is chosen \( c_j = 0 \), \( c_j = \frac{1}{3} [-1 \quad 0 \quad 1]^T \) and \( c_j = \frac{1}{9} [-3 \quad -2 \quad -1 \quad 0 \quad 1 \quad 2 \quad 3]^T \), respectively.

Fig. 2.14  RBF approximation with moderate \( b_j \) and improper \( c_j \) (Mb = 1, Mc = 2)

Fig. 2.15  RBF approximation with improper \( b_j \) and \( c_j \) (Mb = 2, Mc = 2)
The results are shown from Figs. 2.16, 2.17, 2.18, 2.19, 2.20, and 2.21. From the results, we can see that the more number the hidden nets is chosen, the smaller the approximation error can be received.

It should be noted that the more number the hidden nets is chosen, to prevent from divergence, the smaller value of $\eta$ should be designed.

The program of this example is chap2_7.m, which is given in the Appendix.
Fig. 2.18  Three Gaussian functions with three hidden nets \((m = 3)\)

Fig. 2.19  Approximation with three hidden nets \((m = 3)\)
Fig. 2.20  Seven Gaussian functions with seven hidden nets \( (m = 7) \)

Fig. 2.21  Approximation with seven hidden nets \( (m = 7) \)
2.5 RBF Neural Network Training for System Modeling

2.5.1 RBF Neural Network Training

We use RBF neural network to train a data vector with multi-input and multi-output or to model a system off-line.

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

$$h_j = \exp\left(-\frac{\|x - c_j\|^2}{2b_j^2}\right), \quad j = 1, 2, \ldots, m \quad (2.13)$$

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

The width vector of Gaussian function is

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

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

The weight value is

$$w = [w_1, \ldots, w_m]^T \quad (2.14)$$

The output of RBF is

$$y_l = w_1h_1 + w_2h_2 + \cdots + w_mh_m \quad (2.15)$$

where $y_l^d$ denotes the ideal output, $l = 1, 2, \ldots, N$.

The error of the $l$th output is

$$e_l = y_l^d - y_l.$$ 

The performance index function of the training is

$$E(t) = \sum_{l=1}^{N} e_l^2. \quad (2.16)$$

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

$$\Delta w_j(t) = -\eta \frac{\partial E}{\partial w_j} = \eta \sum_{l=1}^{N} e_lh_j$$
\[ w_j(t) = w_j(t-1) + \Delta w_j(t) + \alpha(w_j(t-1) - w_j(t-2)) \]  

(2.17)

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

### 2.5.2 Simulation Example

#### 2.5.2.1 First Example: A MIMO Data Sample Training

Choosing three inputs and two outputs data as a training sample, which are shown in Table 2.1.

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

RBF network structure is chosen as 3-5-1. The choice of Gaussian function parameter values \( c_{ij} \) and \( b_j \) must be chosen according to the scope of practical input value. According to the practical scope of \( x_1 \) and \( x_2 \), the parameters of \( c_i \) and \( b_i \) are designed as:

\[
\begin{bmatrix}
-1 & -0.5 & 0 & 0.5 & 1 \\
-1 & -0.5 & 0 & 0.5 & 1 \\
-1 & -0.5 & 0 & 0.5 & 1 \\
\end{bmatrix}
\]

and 10, the initial weight value is chosen as random value in the interval of \([-1, 1]\), and \( \eta = 0.10 \) and \( \alpha = 0.05 \) are chosen.

Firstly, we run chap2_8a.m, set the error index as \( E = 10^{-20} \). Error index change is shown as Fig. 2.22, and the trained weight values are saved as wfile.dat.
Then we run chap2_8b.m, use wfile.dat, the test results with two samples are shown in Table 2.2. From the results, we can see that good modeling performance can be received.

The programs of this example are chap2_8a.m and chap2_8b.m, which are given in the Appendix.

2.5.2.2 Second Example: System Modeling

Consider a nonlinear discrete-time system as

\[
y(k) = \frac{0.5y(k-1)(1 - y(k-1))}{1 + \exp(-0.25y(k-1))} + u(k-1).
\]

To model the system above, we choose RBF neural network. The network structure is chosen as 2-5-1, according to the practical scope of two inputs; the parameters of \( c_i \) and \( b_i \) are designed as \( \begin{bmatrix} -3 & -2 & -1 & 0 & 1 & 2 & 3 \\ -3 & -2 & -1 & 0 & 1 & 2 & 3 \end{bmatrix} \) and 1.5. Each element of the initial weight vector is chosen as 0.10; \( \eta = 0.50 \) and \( \alpha = 0.05 \) are chosen.

Firstly, we run chap2_9a.m, the input is chosen as \( x = [u(k) \quad y(k)] \), \( u(k) = \sin t \), and \( t = k \times ts \), \( ts = 0.001 \) represents sampling time. The number of samples is chosen as \( NS = 3,000 \). After 500 steps training off-line, we get the error index change as Fig. 2.23. The trained weight values and Gaussian function parameters are saved as wfile.dat.

### Table 2.2 Test samples and results

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.970</td>
<td>0.001</td>
</tr>
<tr>
<td>1.000</td>
<td>0.000</td>
</tr>
</tbody>
</table>

Fig. 2.23  Error index change
Then we run chap2_9b.m, use wfile.dat, the test results with input \( \sin t \) are shown in Fig. 2.24. From the results, we can see that good modeling performance can be received.

The programs of this example are chap2_9a.m and chap2_9b.m, which are given in the Appendix.

2.6 RBF Neural Network Approximation

Since any nonlinear function over a compact set with arbitrary accuracy can be approximated by RBF neural network \([1, 2]\), RBF neural network can be used to approximate uncertainties in the control systems.

For example, to approximate the function \( f(x) \), the algorithm of RBF is expressed as

\[
h_j = g \left( \frac{||x - c_{ij}||^2}{b_j^2} \right)
\]

\[
f = W^* h(x) + \varepsilon
\]

where \( x \) is the input vector, \( i \) denotes input neural net number in the input layer, \( j \) denotes hidden neural net number in the hidden layer, \( h = [h_1, h_2, \ldots, h_n]^T \) denotes the output of hidden layer, \( W^* \) is ideal weight vector, and \( \varepsilon \) is approximation error, \( \varepsilon \leq \varepsilon_N \).

In the control system, if we use RBF to approximate \( f \), we often choose the system states as the input of RBF neural network. For example, we can choose

![Fig. 2.24  Modeling test](image)
the tracking error and its derivative value as the input vector, that is, \( x = [e \ \dot{e}]^T \), then the output of RBF is

\[
\hat{f}(x) = \hat{W}^T h(x)
\]  

(2.18)

where \( \hat{W} \) is the estimated weight vector, which can be tuned by the adaptive algorithm in the Lyapunov stability analysis.

Appendix

Programs for Sect. 2.1.2.1

Simulink main program: chap2_1sim.mdl

RBF function: chap2_1rbf.m

```matlab
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 = [];
```
\begin{verbatim}
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);

    Plot program: chap2_1plot.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;
\end{verbatim}
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);

Programs for Sect. 2.1.2.2

Simulink main program: chap2_2sim.mdl

RBF function: chap2_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 = 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; %cij
     -0.5 -0.25 0 0.25 0.5];
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);

Plot program: chap2_2plot.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);

**Programs for Sect. 2.2.2.1**

Simulink main program: chap2_3sim.mdl

![Diagram of Simulink model](image)

S function for plant: chap2_3rbf.m

```matlab
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 = [];
else
    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;

S function for plant: chap2_3plant.m

function [sys,x0,str,ts]=s_function(t,x,u,flag)
switch flag,
function [sys,x0,str,ts] = mdlInitializeSizes
sizes = simsizes;
sizes.NumContStates = 2;
sizes.NumDiscStates = 0;
sizes.NumOutputs = 1;
sizes.NumInputs = 1;
sizes.DirFeedthrough = 0;
sizes.NumSampleTimes = 0;
sys = simsizes(sizes);
x0 = [0, 0];
str = [];
ts = [];
function sys = mdlDerivatives(t,x,u)
sys(1) = x(2);
sys(2) = -25*x(2) + 133*u;
function sys = mdlOutputs(t,x,u)
sys(1) = x(1);

Plot program: chap2_3plot.m

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');

Programs for Sect. 2.2.2.2

Matlab program: chap2_4.m

% RBF identification
clear all;
close all;
alfa=0.05;
xite=0.15;
x=[0,1]’;

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=2;
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;
    end
    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
\[
\begin{align*}
  w &= w_1 + d_w + \alpha (w_1 - w_2); \\
  y_1 &= y(k); \\
  w_2 &= w_1; \\
  w_1 &= w; \\
  c_2 &= c_1; \\
  c_1 &= c; \\
  b_2 &= b_1; \\
  b_1 &= b;
\end{align*}
\]

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’);

Programs for Sect. 2.3

Program of Gaussian membership function design: chap2_5.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;
% RBF function
x(1)=3*sin(2*pi*k*ts);
for j=1:1:5
    h(j)=exp(-norm(x-c(:,j))^2/(2*b(j)*b(j)));
end

x1(k)=x(1);
%First Redial Basis Function
h1(k)=h(1);
%Second Redial Basis Function
h2(k)=h(2);
%Third Redial Basis Function
h3(k)=h(3);
%Fourth Redial Basis Function
h4(k)=h(4);
%Fifth Redial Basis Function
h5(k)=h(5);
end
figure(1);
plot(x1,h1,'b');
figure(2);
plot(x1,h2,'g');
figure(3);
plot(x1,h3,'r');
figure(4);
plot(x1,h4,'c');
figure(5);
plot(x1,h5,'m');
figure(6);
hold on;plot(x1,h1,'b');
hold on;plot(x1,h2,'g');
hold on;plot(x1,h3,'r');
hold on;plot(x1,h4,'c');
hold on;plot(x1,h5,'m');
xlabel('Input value of Redial Basis Function');ylabel('Membership function degree');

Program of RBF approximation to test the effect of b and c: chap2_6.m

%RBF approximation test
clear all;
close all;
alfa=0.05;
xite=0.5;
x=[0,0]';

%The parameters design of Guassian Function
The input of RBF \((u(k), y(k))\) must be in the effect range of Guassian function overlay

The value of \(b\) represents the widenth of Guassian function overlay

\[
Mb = 1;
\]

if \(Mb = 1\) %The width of Guassian function is moderate

\[
b = 1.5^*\text{ones}(5, 1);
\]

elseif \(Mb = 2\) %The width of Guassian function is too narrow, most overlap of the function is near to zero

\[
b = 0.0005^*\text{ones}(5, 1);
\]

end

The value of \(c\) represents the center position of Guassian function overlay

the NN structure is 2-5-1: \(i = 2; j = 1, 2, 3, 4, 5; k = 1\)

\[
Mc = 1;
\]

if \(Mc = 1\) %The center position of Guassian function is moderate

\[
c = [-1.5 -0.5 0 0.5 1.5;
   -1.5 -0.5 0 0.5 1.5]; %cij
\]

elseif \(Mc = 2\) %The center position of Guassian function is improper

\[
c = 0.1^*[-1.5 -0.5 0 0.5 1.5;
   -1.5 -0.5 0 0.5 1.5]; %cij
\]

end

\[w = \text{rands}(5, 1);\]

\[w_1 = w; w_2 = w_1;\]

\[y_1 = 0;\]

\[ts = 0.001;\]

for \(k = 1:1:2000\)

\[\text{time}(k) = k^*ts;\]

\[u(k) = 0.50^*\sin(1^*2^*\pi^*k^*ts);\]

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

\[x(1) = u(k);\]

\[x(2) = y(k);\]

for \(j = 1:1:5\)

\[h(j) = \exp(-\text{norm}(x-c(:, j))^2/(2*b(j)^*b(j)));\]

end

\[ym(k) = w^'*h';\]

\[em(k) = y(k) - ym(k);\]

\[d_w = \text{xite}^*em(k)^*h';\]

\[w = w_1 + d_w + \alpha^*(w_1 - w_2);\]
\[ y_1 = y(k); \]
\[ w_2 = w_1; w_1 = w; \]
\[
\begin{align*}
\text{end} \\
\text{figure(1);} \\
\text{plot(time,y,'r',time,ym,'b:','linewidth',2);} \\
\text{xlabel('time(s)');ylabel('y and ym');} \\
\text{legend('Ideal value','Approximation value');}
\end{align*}
\]

**Programs for Sect. 2.4**

Program of RBF approximation to test the effect of hidden nets number: chap2_7.m

```
% RBF approximation test
clear all;
close all;
alfa = 0.05;
xite = 0.3;
x = [0,0]';

% The parameters design of Guassian Function
% The input of RBF \((u(k),y(k))\) must be in the effect range of
% Guassian function overlay
% The value of \( b \) represents the wideth of Guassian function overlay
bj = 1.5;  % The width of Guassian function
% The value of \( c \) represents the center position of Guassian function overlay
% the NN structure is 2-m-1: i=2; j=1,2,...,m; k=1
M = 3;    % Different hidden nets number
if M == 1  % only one hidden net
    m = 1;
c = 0;
elseif M == 2
    m = 3;
c = 1/3*[-1 0 1;
              -1 0 1];
elseif M == 3
    m = 7;
c = 1/9*[-3 -2 -1 0 1 2 3;
              -3 -2 -1 0 1 2 3];
end
w = zeros(m,1);
```
w_1=w;w_2=w_1;
y_1=0;
ts=0.001;
for k=1:1:5000
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(k);
for j=1:1:m
    h(j)=exp(-norm(x-c(:,j))^2/(2*bj^2));
end
ym(k)=w'*h';
em(k)=y(k)-ym(k);
d_w=xite*em(k)*h';
w=w_1+d_w+alfa*(w_1-w_2);
y_1=y(k);
w_2=w_1;w_1=w;
x1(k)=x(1);
for j=1:1:m
    H(j,k)=h(j);
end
if k==5000
    figure(1);
    for j=1:1:m
        plot(x1,H(j,:),’linewidth’,2);
        hold on;
    end
    xlabel(’Input value of Radial Basis Function’);ylabel(’Membership function degree’);
end
end
figure(2);
subplot(211);
plot(time,y,’r’,time,ym,’b:’,’linewidth’,2);
xlabel(’time(s)’);ylabel(’y and ym’);
legend(’Ideal value’,’Approximation value’);
subplot(212);
plot(time,y-ym,’r’,’linewidth’,2);
xlabel(’time(s)’);ylabel(’Approximation error’);
Programs for Sect. 2.5.2.1

Program of RBF training: chap2_8a.m

% RBF Training for MIMO
clear all;
close all;
xite=0.10;
alfa=0.05;
W=rands(5,2);
W_1=W;
W_2=W_1;
h=[0,0,0,0,0]’;
c=2*[-0.5 -0.25 0 0.25 0.5;
    -0.5 -0.25 0 0.25 0.5;
    -0.5 -0.25 0 0.25 0.5]; % cij
b=10; % b_j
xs=[1,0,0]; % Ideal Input
ys=[1,0]; % Ideal Output
OUT=2;
NS=1;
k=0;
E=1.0;
while E>=1e-020
    % for k=1:1:1000
    k=k+1;
times(k)=k;
    for s=1:1:NS % MIMO Samples
        x=xs(s,:);
        for j=1:1:5
            h(j)=exp(-norm(x’-c(:,j))^2/(2*b^2)); % Hidden Layer
        end
        yl=W’*h; % Output Layer
        el=0;
y=ys(s,:);
        for l=1:1:OUT
            el=el+0.5*(y(l)-yl(l))^2; % Output error
        end
        es(s)=el;
    E=0;
    if s==NS
        for s=1:1:NS

\[ E = E + \text{es}(s); \]
end
error = y - yl';
dW = xite * h * error;

\[ W = W_1 + dW + \text{alfa} \times (W_1 - W_2); \]

\[ W_2 = W_1; W_1 = W; \]
end % End of for
Ek(k) = E;
end % End of while
figure(1);
plot(times, Ek, 'r', 'linewidth', 2);
xlabel('k'); ylabel('Error index change');
save wfile b c W;

Program of RBF test: chap2_8b.m

\% Test RBF
clear all;
load while b c W;

\% N Samples
x = [0.970, 0.001, 0.001;
     1.000, 0.000, 0.000];

NS = 2;
h = zeros(5, 1); \% h_j
for i = 1:1:NS
    for j = 1:1:5
        h(j) = exp(-norm(x(i,:) - c(:, j))^2 / (2 * b^2)); \% Hidden Layer
    end
    yl(i,:) = W'*h; \% Output Layer
end
yl

**Programs for Sect. 2.5.2.2**

Program of RBF training: chap2_9a.m

\% RBF Training for a Plant
clear all;
close all;

\% t = 0.001;
xite = 0.50;
alfa=0.05;
u_1=0;y_1=0;
fx_1=0;
W=0.1*ones(1,7);
W_1=W;
W_2=W_1;
h=zeros(7,1);
c1=[-3 -2 -1 0 1 2 3];
c2=[-3 -2 -1 0 1 2 3];
c=[c1;c2];
b=1.5; %bj
NS=3000;
for s=1:1:NS %Samples
u(s)=sin(s*ts);
fx(s)=0.5*y_1*(1-y_1)/(1+exp(-0.25*y_1));
y(s)=fx_1+u_1;
end
k=0;
for k=1:1:500
k=k+1;
times(k)=k;
for s=1:1:NS %Samples
x=[u(s),y(s)];
for j=1:1:7
h(j)=exp(-norm(x’-c(:,j))^2/(2*b^2)); %Hidden Layer
end
yl(s)=W*h; %Output Layer
es(s)=el;
E=0;
end
end
error=y(s)-yl(s);
dW=xite*h’*error;
\[ W = W_1 + dW + \alpha (W_1 - W_2); \]

\[ W_2 = W_1; W_1 = W; \]

end % End of for

Ek(k) = E;
end % End of while

figure(1);
plot(times, Ek, 'r', 'linewidth', 2);
xlabel('k'); ylabel('Error index change');
save wfile b c W NS;

Program of RBF test: chap2_9b.m

% Online RBF Estimation for Plant

clear all;
load wfile b c W NS;

ts = 0.001;
 u_1 = 0; y_1 = 0;
 fx_1 = 0;
 h = zeros(7, 1);
 for k = 1:1:NS
    times(k) = k;
    u(k) = sin(k*ts);
    fx(k) = 0.5*y_1*(1-y_1)/(1+exp(-0.25*y_1));
    y(k) = fx_1 + u_1;
    x = [u(k), y(k)];
    for j = 1:1:7
        h(j) = exp(-norm(x' - c(:, j))^2 / (2*b^2)); % Hidden Layer
    end
    yp(k) = W*h; % Output Layer
    u_1 = u(k); y_1 = y(k);
    fx_1 = fx(k);
 end
figure(1);
plot(times, y, 'r', times, yp, 'b-.', 'linewidth', 2);
xlabel('times'); ylabel('y and yp');

References