学院：数据科学与计算机学院　　专业：计算机科学与技术
学号：郑 康 泽　　学号：17341213

智能控制与计算智能

第九章作业

9-1　参照RBF网络直接模型参考自适应控制算法，试推导BP网络直接模型参考自适应控制算法。

设参考模型输出为$y_m(k)$，控制系统要求对象的输出$y(k)$能够跟踪参考模型的输出$y_m(k)$。则跟踪误差为$ec(k)=y_m(k)-y(k)$，控制目标函数为$E(k)=\frac{1}{2}ec^2(k)$。
控制器为网络的输出：

$$u(k)=\sum_{j=1}^{m}w_j'(k)f(h_j(k))$$

其中w_j'为BP网络第j个隐层神经元与输出层之间的连接权，h_j为第j个隐层神经元的输入，f为隐层神经元的激活函数，m为隐层神经元的个数。
在BP网络结构中，$\mathbf{x}=\{x_1,x_2,\dots,x_n\}^T$为网络的输入，第$i$个输入层神经元$x_i$与第$j$个隐层神经元之间的连接权为$w_{ij}$，则第$j$隐层神经元的输入为

$$h_j=\sum_{i=1}^{n}w_{ij}x_i$$

按照梯度下降法及链式法则，可得隐层神经元与输出层之间连接权的学习算法如下：

$$\Delta w_j'(k)=-\eta\frac{\partial E(k)}{\partial w_j'(k)}=\eta ec(k)\frac{\partial y_(k)}{\partial u(k)}f(h_j(k))$$
$$w_j'(k+1)=w_j'(k)+\Delta w_j'(k)+\alpha(w_j'(k)-w_j'(k-1))$$

其中η为学习速率，α为动量因子。
同理，输入层神经元与隐层神经元之间连接权的学习算法如下：

$$\Delta w_{ij}(k+1) = -\eta \frac{\partial E(k)}{\partial w_{ij}(k)} = \eta ec(k) \frac{\partial y(k)}{\partial u(k)} w'_j \frac{\mathrm{d}f}{\mathrm{d}h_j(k)} x_i$$
$$w_{ij}(k+1) = w_{ij}(k) + \eta \Delta w_{ij}(k) + \alpha(w_{ij}(k) - w_{ij}(k-1))$$

9-2　参照RBF网络的自校正控制方法，设计基于RBF网络的模型参考自校正控制器，并进行Matlab仿真。被控对象为 $y(k) = 0.8\sin(y(k-1)) + 15u(k-1)$，采样周期为$T = 0.001$，参考模型为 $y_m(k) = 0.6y_m(k-1) + r(k)$，$r(k)$为正弦信号，$r(k) = 0.5\sin(2\pi kT)$。

根据被控对象，则自校正控制器的控制算法为：

$$u(k) = -\frac{Ng[\bullet]}{Nf[\bullet]} + \frac{y_m(k+1)}{Nf[\bullet]}$$

其中$Ng[\bullet]$是利用RBF网络估计出来的参数，目的是逼近 $g[\bullet] = 0.8\sin(y(k-1))$，$Nf[\bullet]$是利用另一个RBF网络估计出来的参数，目的是逼近$f[\bullet] = 15$。可以看出，如果能够逼近成功的话，那么：

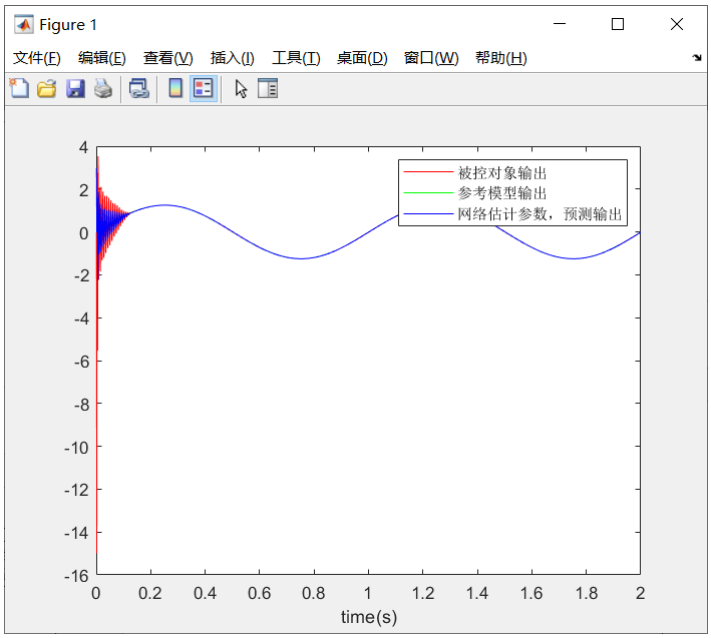
$$y(k) = g[\bullet] + f[\bullet]u(k-1) = g[\bullet] + f[\bullet](-\frac{Ng[\bullet]}{Nf[\bullet]} + \frac{y_m(k)}{Nf[\bullet]}) = y_m(k)$$

从而实现了跟踪控制。
代码如下：

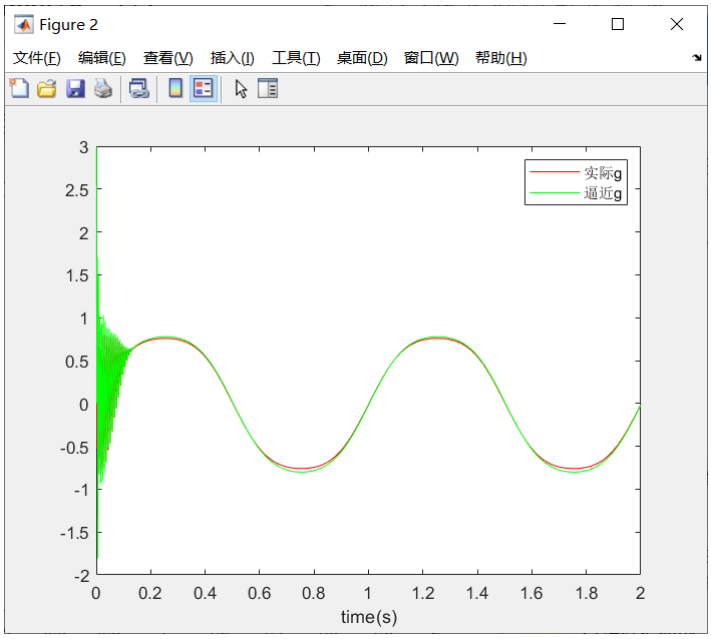
```
% 基于RBF网络的模型参考自校正控制器
clear;
close;

xite1 = 0.15;                    % 网络1的学习率
xite2 = 0.50;                    % 网络2的学习率
alfa = 0.05;

w = 0.5 * ones(6, 1);            % 网络1隐层到输出层的权重
v = 0.5 * ones(6, 1);            % 网络2隐层到输出层权重

c = 0.5 * ones(1, 6);            % 中心
b = 5 * ones(6, 1);              % 宽度
h = zeros(6, 1);                 % 隐层输出

w_1 = w; w_2 = w_1;              % 前两步的w
v_1 = v; v_2 = v_1;              % 前两步的v
u_1 = 0; y_1 = 0; ym_1 = 0;      % 前一步的u、y、ym

ts = 0.001;                      % 采样时间
for k = 1:2000
    time(k) = k * ts;            % x轴

    r(k) = 0.5 * sin(2 * pi * k * ts);       % 参考模型信号
```

```
    ym(k) = 0.6 * ym_1 + r(k);                  % 参考模型输出

    g(k) = 0.8 * sin(y_1);
    f(k) = 15;
    y(k) = g(k) + f(k) * u_1;                   % 被控对象的输出

    % 网络1和网络2的隐层输出
    for j = 1:6
        h(j) = exp(-norm(y(k) - c(:, j))^2 / (2 * b(j) * b(j)));
    end

    Ng(k) = w' * h;                             % 网络1的输出（逼近g）
    Nf(k) = v' * h;                             % 网络2的输出（逼近f）
    yn(k) = Ng(k) + Nf(k) * u_1;                % 根据网络估计的参数预测被控对象的输出

    e(k) = y(k) - yn(k);                        % 网络的误差

    % 更新w
    d_w = 0 * w;
    for j = 1:6
        d_w(j) = xite1 * e(k) * h(j);
    end
    w = w_1 + d_w + alfa * (w_1 - w_2);

    % 更新v
    d_v = 0 * v;
    for j = 1:6
        d_v(j) = xite2 * e(k) * h(j) * u_1;
    end
    v = v_1 + d_v + alfa * (v_1 - v_2);

    % 自校正控制器的输出（被控对象的控制输入）
    u(k) = (ym(k) - Ng(k)) / Nf(k);

    % 更新记录
    u_1 = u(k); y_1 = y(k); ym_1 = ym(k);
    w_2 = w_1; w_1 = w;
    v_2 = v_1; v_1 = v;
end

% 画图
figure(1);
plot(time, y, 'r', time, ym, 'g', time, yn, 'b');
xlabel('time(s)');
legend('被控对象输出', '参考模型输出', '网络估计参数，预测输出');
figure(2);
plot(time, g, 'r', time, Ng, 'g');
xlabel('time(s)');
legend('实际g', '逼近g');
figure(3);
plot(time, f, 'r', time, Nf, 'g');
xlabel('time(s)');
legend('实际f', '逼近f');
figure(4);
plot(time, e);
xlabel('time(s)');
legend('网络误差');
```

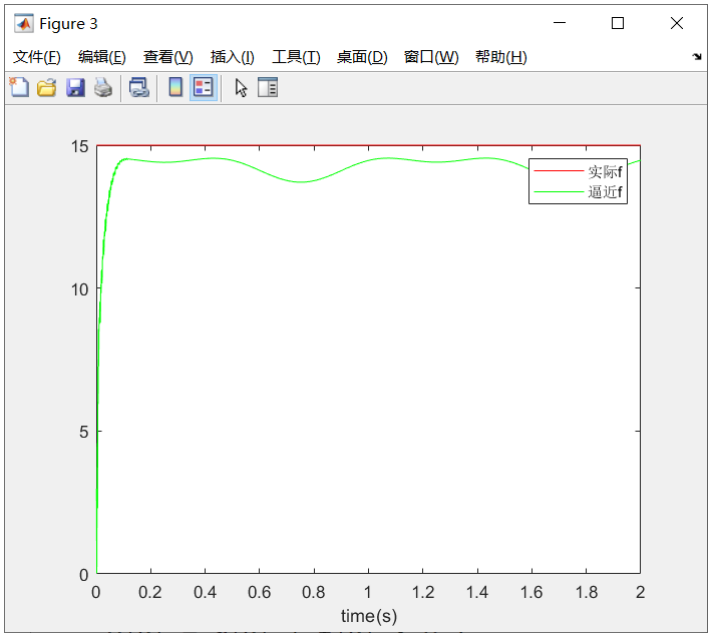
跟踪输出：



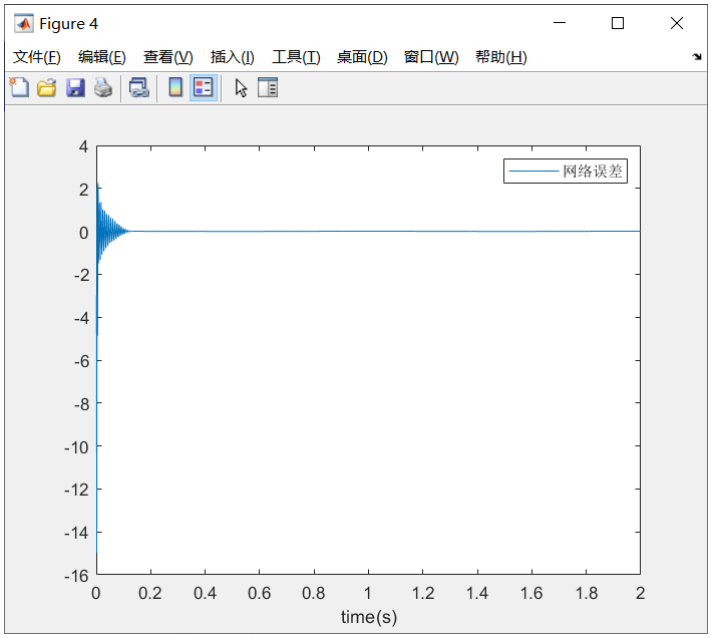
$Ng[\bullet]$和$g[\bullet]$：



$Nf[\bullet]$和$f[\bullet]$：



跟踪误差：



9-3　已知一非线性系统

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

给定的期望轨迹为

$$y_d(k)=\sin\frac{2\pi k}{25}+\sin\frac{2\pi k}{10}$$

试采用RBF神经网络进行自适应控制，其中Jacobian信息由RBF网络辨识，并进行Matlab仿真。

题目要求用RBF网络辨识Jacobian，但课本并没有这个例子，所以我搜索了一下，找到了这篇论文，其中就有利用RBF网络辨识Jacobian。其中，网络的输入为上一步的输入u_1、上一步的输出y_1和当前的输出$y(k)$，网络的输出为$y_p(k)$，目标函数为$E(k)=\frac{1}{2}(y(k)-y_p(k))^2$，可以看出网络的目标就是使得$y_p(k)$逼近$y(k)$。那么这个网络为什么能辨识Jacobian呢？根据论文，它是这么定义的：

$$\text{Jacobian}=\frac{\partial y(k)}{\partial u(k)}=\frac{\partial y_p(k)}{\partial u(k)}=\sum_{j=1}^{m}w_j h_j\frac{\|\mathbf{x}-\mathbf{c}_j\|}{b_j^2}$$

其中，m为RBF网络的隐层神经个数，w_j为隐层第j个神经元连接输出层的权值，h_j为隐层第j个神经元的输出，$\mathbf{c}_j$为隐层第j个神经元的中心矢量，b_j为隐层第j个神经元的基宽参数。

将这个辨识Jacobian的RBF网络加入到“RBF网络直接模型参考自适应控制”的示例程序中，调了一下午参数，似乎没什么用，也不知道这样的网络是否正确，反正最后的结果是失败的。以下是程序：

```
% RBF网络自适应控制
clear;
close;

xite = 0.35;
alfa = 0.05;

u_1 = 0;
y_1 = 0;

% NNC
x = [0, 0]';
c = 0.5 * ones(2, 6);
b = 5 * ones(6, 1);
w = 0.5 * ones(6, 1);
h = [0, 0, 0, 0, 0, 0]';

c_1 = c; c_2 = c;
b_1 = b; b_2 = b;
w_1 = w; w_2 = w;

% NNI
xx = [0, 0, 0]';
cc = [-3 -2 -1 1 2 3;
      -3 -2 -1 1 2 3;
      -3 -2 -1 1 2 3];
bb = 1 * ones(6, 1);
ww = 1 * rand(6, 1);
hh = [0, 0, 0, 0, 0, 0]';
cc_1 = cc; cc_2 = cc;
bb_1 = bb; bb_2 = bb;
ww_1 = ww; ww_2 = w;

ts = 0.01;
```

```
for k = 1:100
    time(k) = k * ts;
    ym(k) = sin(2 * pi * k / 25) + sin(2 * pi * k / 10);
    y(k) = y_1 / (1 + y_1^2) + u_1^3;

    for j = 1:6
        h(j) = exp(-norm(x - c(:, j))^2 / (2 * b(j) * b(j)));
    end
    u(k) = w' * h;
    ec(k) = ym(k) - y(k);

    % RBF辨识Jacobian
    xx(1) = u_1;
    xx(2) = y_1;
    xx(3) = y(k);
    for j = 1:6
        hh(j) = exp(-norm(xx - cc(:, j))^2 / (2 * bb(j) * bb(j)));
    end
    yy(k) = ww' * hh;
    e(k) = y(k) - yy(k);
    dyu(k) = 0;
    for j = 1:6
        dyu(k) = dyu(k) + ww(j) * hh(j) * norm(xx - cc(:, j)) * bb(j)^-2;
    end

    % 更新NNC参数
    d_w = 0 * w;
    for j=1:6
        d_w(j) = xite * ec(k) * h(j) * dyu(k);
    end
    w = w_1 + d_w + alfa * (w_1 - w_2);

    M = 1;
    if M == 1
        d_b = 0 * b;
        for j = 1:6
            d_b(j) = xite * ec(k) * w(j) * h(j) * (b(j)^-3) * norm(x - c(:,
j))^2 * dyu(k);
        end
        b = b_1 + d_b + alfa * (b_1 - b_2);

        d_c = 0 * c;
        for j=1:6
            for i = 1:2
                d_c(i, j) = xite * ec(k) * w(j) * h(j) * (x(i) - c(i,j)) *
(b(j)^-2) * dyu(k);
            end
        end
        c = c_1 + d_c + alfa * (c_1 - c_2);
    elseif M == 2
       b = b_1;
       c = c_1;
    end

    % 更新NNI参数
    d_ww = 0 * ww;
    for j = 1:6
        d_ww(j) = xite * e(k) * hh(j);
```

```
    end
    ww = ww_1 + d_ww + alfa * (ww_1 - ww_2);

%     d_bb = 0 * bb;
%         for j = 1:6
%             d_bb(j) = xite * e(k) * ww(j) * hh(j) * (bb(j)^-3) * norm(xx -
cc(:, j))^2;
%         end
%         bb = bb_1 + d_bb + alfa * (bb_1 - bb_2);
%
%         d_cc = 0 * cc;
%         for j=1:6
%             for i = 1:3
%                 d_cc(i, j) = xite * e(k) * ww(j) * hh(j) * (xx(i) - cc(i,j)) *
(bb(j)^-2);
%             end
%         end
%         cc = cc_1 + d_cc + alfa * (cc_1 - cc_2);

    % 更新记录
    u_1 = u(k);
    y_1 = y(k);

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

    w_2 = w_1; w_1 = w;
    c_2 = c_1; c_1 = c;
    b_2 = b_1; b_1 = b;


    ww_2 = ww_1; ww_1 = ww;
    cc_2 = cc_1; cc_1 = cc;
    bb_2 = bb_1; bb_2 = bb;
end
% 画图
figure(1);
plot(time, ym, 'r', time, y, 'b');
xlabel('time(s)');
legend('期望轨迹', '实际轨迹')
% figure(2);
% plot(time,ym-y,'r');
% xlabel('time(s)');ylabel('tracking error');
```

因为基本上没跟上期望的轨迹，所以这里就不展示结果了，网络结构应该是有问题，但又不知道怎么改。