APPENDIX B

MATLAB Programs to Implement the SA-DT Method

```
% pid_train3.m

   global nsys_good nsys_bad

   nsys_good = 0;
   nsys_bad = 0;

   max_sastep = 100;     % used for tree.6683
   saalpha = 0.2;
   num_train = 200;

   num_rules = 25;

   num_samples = num_train*max_sastep;
   num_attrib = 7;

   ! rm -f train_good.sam train_bad.sam

   % write titles:
   write_titles

   pid_initial3

   ncount = num_train;
   while (ncount >1)
      ncount = ncount - 1;
      pid_sa3_train;
   end

%  pid_plots3;

   save pid_train3.mat max_sastep saalpha num_train nsys_good nsys_bad


% pid_initial3.m

   % plant:
   % jdb_plant:
%  num = [1 5 14.09];
%  den = [1 10 45.25 122 199.5 145];
```

```
    %gz_plant:
    num = [1 30 300];
    den = [1 10 45 120 200 300];

    % simulation parameters:
    dt = 0.05;
    Tf = 120;
    t = [0:dt:Tf]';

    % set point change:
    set_pt = 1.0;
    sp_change = 0.2;

    % input singal:
    u0 = set_pt*ones(t);
    usp_ch = sp_change*ones(t);

    % choices of noise signals
    rand('normal');
    un1 = 0.1*set_pt*rand(t);                % random noise
    un2 = 0.1*set_pt*sin(10*t);              % sinusoidal random noise
    un3 = 0.1*set_pt*sign(sin(10*t));        % rectangular noise
    un4 = telegram(length(t),0.1*set_pt,0.5); % pseudo telegram noise

    % where is the noise added and which one will be added?
    lent = length(u0);               % length of the simulation
    be_sp_ch = fix(lent*1/3);        % begining point to change set point
    be_ns = fix(lent*2/3);           % beging point to add the noise
    um1 = zeros(u0);
    um2 = um1;
    um1(be_sp_ch:lent) = usp_ch(be_sp_ch:lent);
    um2(be_ns:lent) = un4(be_ns:lent);
    u = u0 + um1 + um2;

    % to get steady state performance:
    us = set_pt*ones(1:be_sp_ch-1);    % input that makes sys steady
    ts = t(1:be_sp_ch-1);              % time when sys becomes steady

    % desired output:
    des_outw = u0 + um1;     % whole length of des_out

    % the part used for objective calculation:
    u_obj = u(be_sp_ch:lent);
    t_obj = t(be_sp_ch:lent);


% File: rules.m
% Example ruleset
    % example ruleset:
```

```
pn1 = -0.15;     % makes 4 rules
pn2 = -0.10;     % makes 4 rules
pn3 = -0.05;     % makes 4 rules
pp1 = 0.05;     % makes 4 rules
pp2 = 0.10;     % makes 4 rules
pp3 = 0.15;     % makes 4 rules
ruleset = [    pn1     0     0     0
        0     pn1     0     0
        0     0     pn1     0
        0     0     0     pn1
        pn2     0     0     0
        0     pn2     0     0
        0     0     pn2     0
        0     0     0     pn2
        pn3     0     0     0
        0     pn3     0     0
        0     0     pn3     0
        0     0     0     pn3
        pp1     0     0     0
        0     pp1     0     0
        0     0     pp1     0
        0     0     0     pp1
        pp2     0     0     0
        0     pp2     0     0
        0     0     pp2     0
        0     0     0     pp2
        pp3     0     0     0
        0     pp3     0     0
        0     0     pp3     0
        0     0     0     pp3
        0     0     0     0];

% File: pid_safun3.m

function [obest,xbest,o_v,rule_v,prob] = ...
    pid_safun(s,x,ruleset,desobj,max_step,alpha,num,den,t,u,...
    des_outw,lent,be_sp_ch,be_ns);

    len = length(x);
    [num_rule,num_par] = size(ruleset);

    prob = (1.0/num_rule)*ones(num_rule,1);   % initial probability
    prob = prob/sum(prob);    % normalize
    xinterval = 1:num_rule;

    eval(s);
    obj = obj_v(2);
    objv_old = obj_v;
    len_obj_v = length(obj_v);
```

```
 minobj = obj;
 xbest = x;
 num_step=1;
 text = sprintf('Err(%.0f) = %12.5f',num_step, obj);
 disp(text);
 rule_v(1) = 0;  % initialize
 o_v(1) = obj;
 while (minobj>desobj & num_step <= max_step)
     num_step = num_step+1;
             sample = arbprob_simp3(1,prob,xinterval);
             rule = sample;
    rule_v(num_step) = rule;
    xold = x;
             x = ([ruleset(rule,:)]').* x + x;     % update x

    eval(s);
    obj = obj_v(2);
    objv_old = obj_v;
    o_v(num_step) = obj;
    %
    if (obj > o_v(num_step-1))
        accept_prob = exp(-alpha*num_step);
        rand_p = rand;
        alarm = obj_v(1);
        if (accept_prob<rand_p | alarm==1)  % back to original
            x = xold;
            obj = o_v(num_step-1);
            rule_v(num_step)=num_rule; %last rule:no change
            o_v(num_step) = obj;
            disp('      ---- back to previous system ----')
            if (alarm==1)
                write_bad;
            end
        end
        if (rule ~= num_rule)
            prob(rule) = prob(rule) - 0.005;
        end
        if (prob(rule)<0)
            prob(rule) = 0;
        end
    else   % increase the probability of the good rule:
        if (rule ~= num_rule)
            prob(rule) = prob(rule)+ 0.005;
        end
        % write down the good results
        write_good;
    end
    prob = prob/sum(prob);     % normalize
    text = sprintf('Err(%.0f) = %12.5f',num_step, o_v(num_step));
    disp(text)
    if (obj < minobj)
```

```
            minobj = obj;
            xbest = x;
        end
    end

    obest = minobj;             % best objective obtained


% pid_apply3s.m

    % load the tree:
    load tree.10173
    trainsam = 10173;

    num_rules = 25;
    num_apply = 100;

    pid_initial3

    desobj = 3.0;

    max_apstep = 50;

    step = 0;
    num_suc = 0;
    num_imp = 0;
    num_wor = 0;
    num_nch = 0;
    while (step < num_apply)
        step = step + 1;
        pid_apply_simu3s;
    end

    num_v = [num_suc num_imp num_nch num_wor];
    num_v = num_v/num_apply;


% pid_sa3_apply.m

    % keep trying until an initial controller is good enough.
    test_alarm = 1;
    rand('uniform');
    while (test_alarm == 1 )  % indicating an unstable system
        pidi = 0.2*[rand rand rand rand]';
        [Aci,Bci,Cci,Dci]=consys(num,den,pidi);
        tobjv = pid_obj(pidi,num,den,t,u,des_outw,lent,be_sp_ch,be_ns);
        test_alarm = tobjv(1);
    end

    alarm = test_alarm;
    % simulate the initial system
        [Aci,Bc,Cci,Dci]=consys(num,den,pidi);
        [yti,xti] = lsim(Aci,Bci,Cci,Dci,u,t);

    intobj = tobjv(2);

%     Calling procedure:
    s = 'obj_v=pid_obj(x,num,den,t,u,des_outw,lent,be_sp_ch,be_ns);';
    x = pidi;
                          % example rule set, to give 'ruleset'
    rules;
    [num_rule,num_par] = size(ruleset);

    eval(s);
    obj = obj_v(2);

    minobj = obj;
    xbest = x;
    num_step=0;

    rule_v(1) = 0;  % initialize
    o_v(1) = obj;
    alarm = obj_v(1);
    qua_v = [obj_v(2) obj_v(3) obj_v(4)];
    xold  = x;
    sample = [xold(1) xold(2) xold(3) xold(4) qua_v]';

%    text = sprintf('\nErr(0) = %12.5f', intobj);
%    disp(text);
    while (minobj>desobj & num_step <= max_apstep & alarm==0)
       num_step = num_step+1;
       alarm = obj_v(1);
       qua_v = [obj_v(2) obj_v(3) obj_v(4)];
       sample = [xold(1) xold(2) xold(3) xold(4) qua_v]';
       class = dtree(tree,sample);
              rule = class;
      rule_v(num_step) = rule;
              x = ([ruleset(rule,:)]').* x + x;     % update x
      eval(s);    % update obj_v;
      obj = obj_v(2);
      o_v(num_step) = obj;
%      text = sprintf('Err(%.0f) = %12.5f',num_step, o_v(num_step));
%      disp(text)
      if (obj < minobj)
         minobj = obj;
         xbest = x;
      end
      xold = x;
end
```

```
if (alarm == 1)  % the system become unstable
    pidf = xold;
    minobj = intobj;
end

pidf = x;

if (minobj <= desobj)
    num_suc = num_suc + 1;
elseif (minobj < intobj)
    num_imp = num_imp + 1;
elseif (minobj == intobj)
    num_nch = num_nch + 1;
else
    num_wor = num_wor + 1;
end
```