FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Sistemas de Decisão

3º Trabalho

Sintonização de um controlador PID através de técnicas de programação evolutiva

João Miguel Carvalho 49341

Índice

Introdução

Pretende-se desenvolver um controlador PID projectado através de técnicas de computação evolutiva, para o processo térmico Feedback PCT 37-100 (Figura 1), tendo em vista o controlo da temperatura de saída do escoamento de ar.

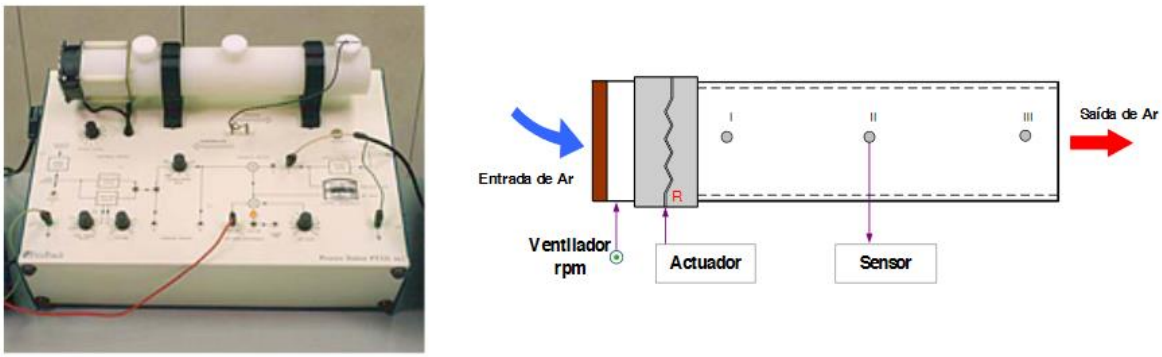

Figura 1 – processo térmico Feedback PCT 37-100

A computação evolutiva pode ser encontrada em diversas áreas, sendo neste caso em especial as áreas da identificação de sistemas (modelação) e optimização. A optimização é caracterizada pela procura da solução, tendo em conta um modelo conhecido e descrição da saída desejada, enquanto que a identificação envolve a determinação da relação entre as entradas e as saídas.

Para este trabalho, na identificação do sistema, utilizamos um modelo ARX (3,2,1), cujo vector de parametrização é encontrado com o algoritmo PSO (enxame de partículas), enquanto que o controlador PID é sintonizado com o algoritmo genético.

1. Enxame de partículas (PSO)

A optimização por enxame de partículas é um método estocástico que explora a analogia com o comportamento social de conjuntos de animais organizados em enxames, cardumes ou bandos, com objectivo de perseguir um ponto óptimo no espaço. Neste método, cada indivíduo ou partícula contém

posição e velocidade que são inicializados aleatoriamente. Após essa inicialização, os indivíduos são avaliados através da função de aptidão(*fitness)*, que avalia o quão bom é o resultado.

O algoritmo é iterativo e actualiza os parâmetros velocidade e posição de cada um dos seus indivíduos no final de cada ciclo, seguindo o melhor resultado de cada indivíduo (pi) em termos de fitness e o melhor resultado global (g). Estes parâmetros vão sendo actualizados a partir da equação :

$$v_i^{k+1} = v_i^k + c_1 \cdot rand \cdot \left(p_i^{best} - p_i\right) + c_2 \cdot rand \cdot \left(g_{best} - p_i\right)$$

$$p_i^{k+1} := p_i^k + v_i^{k+1}$$

equação 1.1

Sendo c1 e c2 os factores de aprendizagem e *rand* um número aleatório.

O PSO necessita de alguns parâmetros para a sua sintonização, tais como o número de partículas, a dimensão das partículas, a velocidade máxima e o factor de aprendizagem. Todos estes parâmetros influenciam directamente o funcionamento do algoritmo.

Por exemplo, pretende-se optimizar a função $f(x) = -\sqrt{x}\,.\sin(x)\;.\sqrt{y}.\sin(y)$. Escolheu-se como factores de aprendizagem pessoal e social c1 e c2 = 1.49, número de partículas = 800, e critério de paragem como 1600 iterações.

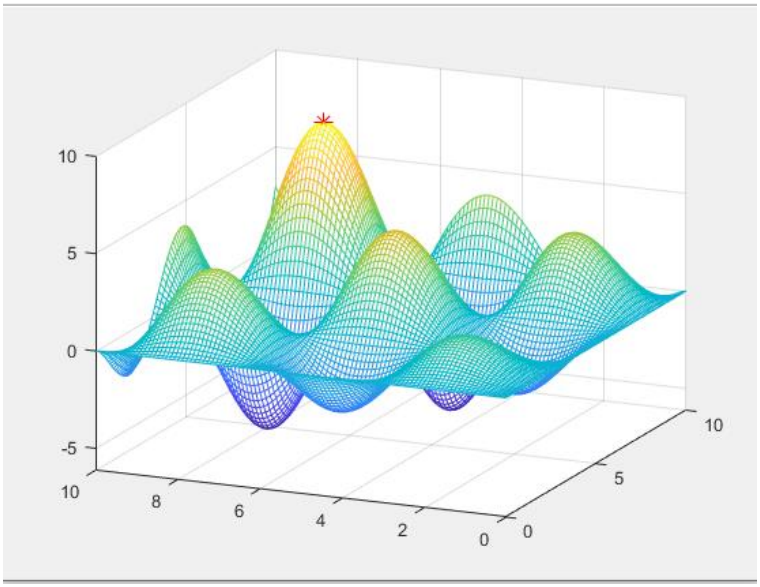

Figura 2- representação de f(x) e do máximo global (ponto a vermelho)

2. Algoritmo genético

O algoritmo genético é também um algoritmo de busca pelo óptimo global, considerando um conjunto de valores (população). Neste algoritmo, à semelhança do PSO, os indivíduos são inicializados aleatoriamente e avaliados por uma função de aptidão(fitness) , mas desta vez, os indivíduos sofrem operações de cruzamento. Os indivíduos mais aptos, têm uma maior probabilidade de serem seleccionados para reprodução, prevalecendo assim o "gene" mais forte. A seleção acontece através duma experiência aleatória com uma determinada distribuição de probabilidades, como é no caso do "método da roleta". Após o cruzamento, dá-se uma nova geração, que também vai ser avaliada com a função fitness e posteriormente será feita uma nova seleção com base neste valor, sendo este ciclo repetido N vezes. É comum, após a recombinação, aplicar-se uma operação de mutação, com objectivo de impedir que o algoritmo fique preso em óptimos locais, trazendo um pouco o factor aleatório.

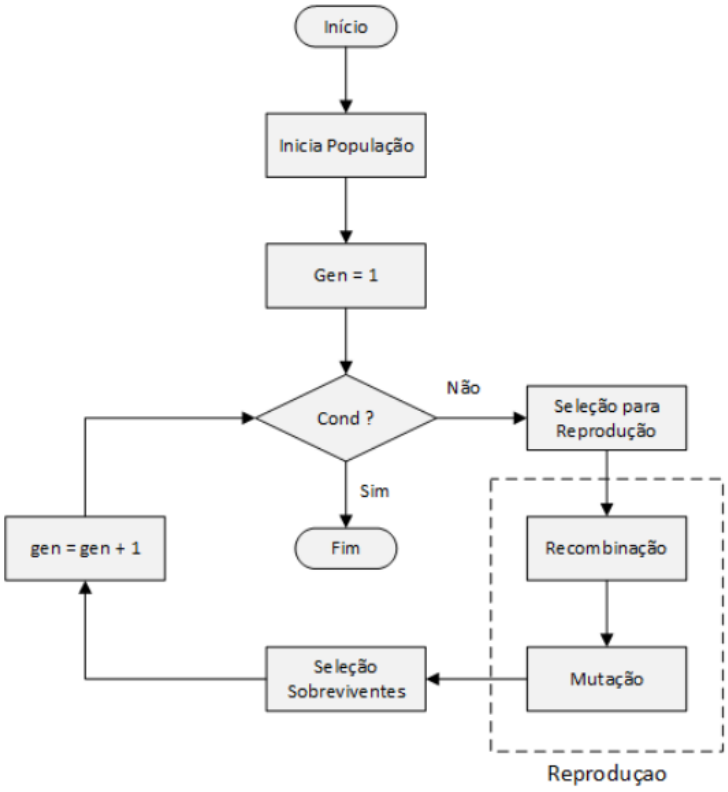

Figura 3- Fluxograma do algoritmo genético

Tendo em conta o problema de optimização, são necessários adaptar alguns parâmetros, tais como o tamanho da população, o número de gerações, a taxa de cruzamento e a taxa de mutação.

Projecto

1. Modelação do sistema

Primeiramente, o sistema foi excitado com dois conjuntos de sinais pseudoaleatórios com 750 amostras cada, extraindo daí os dados de estimação e os de validação.

Utilizando os dados de estimação, aproximou-se a dinâmica sistema com um modelo ARX(3,2,1), cujos parâmetros foram obtidos pelo algoritmo PSO, limitados entre [-5;5]. Os factores de aprendizagem, o número máximo de iterações e o número de partículas foram, respectivamente, os seguintes:

- C1(individual), C2(social) = 1.49;
- MaxIter = 300;
- SwarmSize = 500.

A função de aptidão (*fitness)* usada foi baseada no critério do erro quadrático entre a saída real do processo e o modelo.

O algoritmo foi corrido 5x, e a cada uma foi calculado o erro, para avaliação da solução. Os erros obtidos foram: **26.5533, 26.5367, 26.5286, 26.57341, 26.54639**. Como esperado, escolheu-se a solução com menor erro, cujos parâmetros do modelo ARX são:

- A=[0.0441 -0.6548 -0-0550]
- B=[0.2357 0.2057]

Podemos ver na figura 4 o resultado da simulação do modelo, comparada com a saída real.

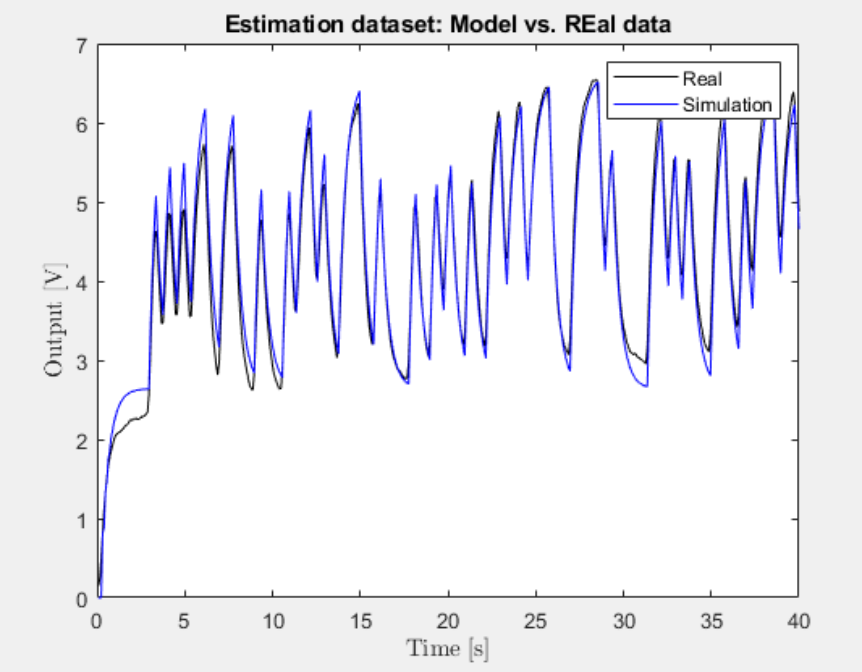

Figura 4 – Comparação do modelo ARX calculado com a saída real

2. Controlo do sistema

Para o controlo do sistema utilizamos um controlador PID, cujos ganhos são determinados através do algoritmo genético, limitados entre [5; 0.01]. Recorreu-se à representação da população do tipo double com 200 indivíduos e correu-se o algoritmo durante 50 gerações. A seleção dos indivíduos para reprodução foi feita através do método da roleta e com uma taxa de recombinação de 0.85.

O critério de desempenho utilizado foi uma ponderação entre o erro quadrático e a variação da ação de controlo, designadamente:

$$J(e,u) = \sum p \cdot \big(r(k) - y(k)\big)^2 + \sum q \cdot (u(k))^2 + \sum w \cdot (u(k) - u(k-1))^2$$

Foram feitas duas experiências com diferentes pesos p, q e w, cada uma considerando uma referência composta por uma sucessão de ondas quadradas de amplitudes {2,0; 4,0; 3,0, 4,5; 3,0} V, cada uma com contendo 150 amostras.

Simulação 1:

- p = 10;
- q = 0.1;
- w = 0.1.

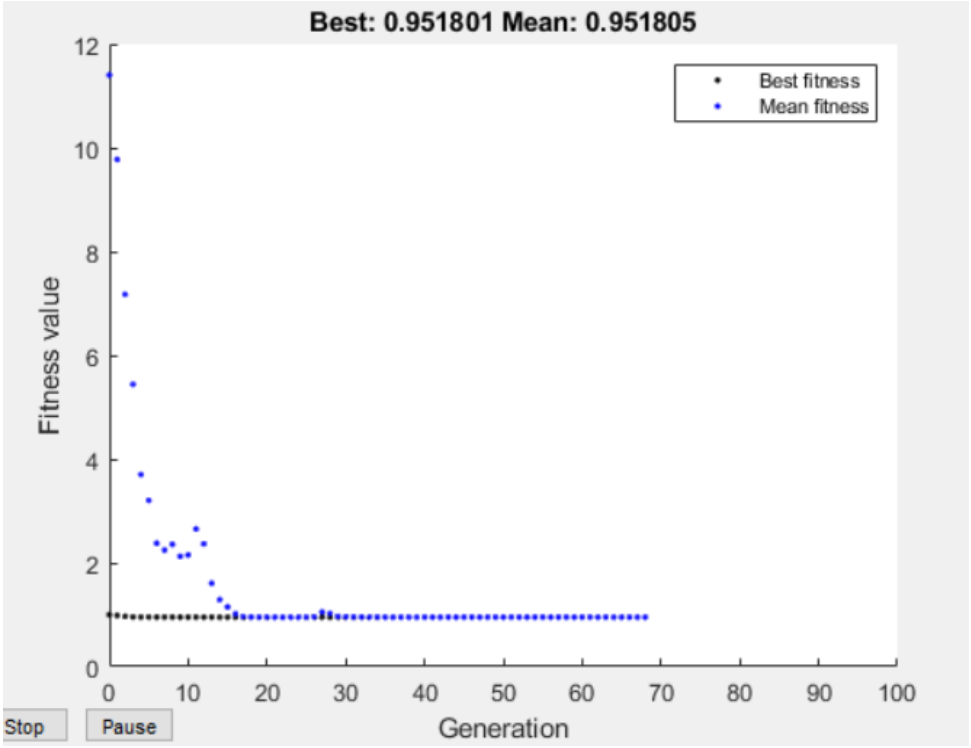

Figura 5 – Evolução geracional do erro da simulação 1

Resultando os ganhos:

- KP = 1.8
- KI = 5
- KD = 0.01

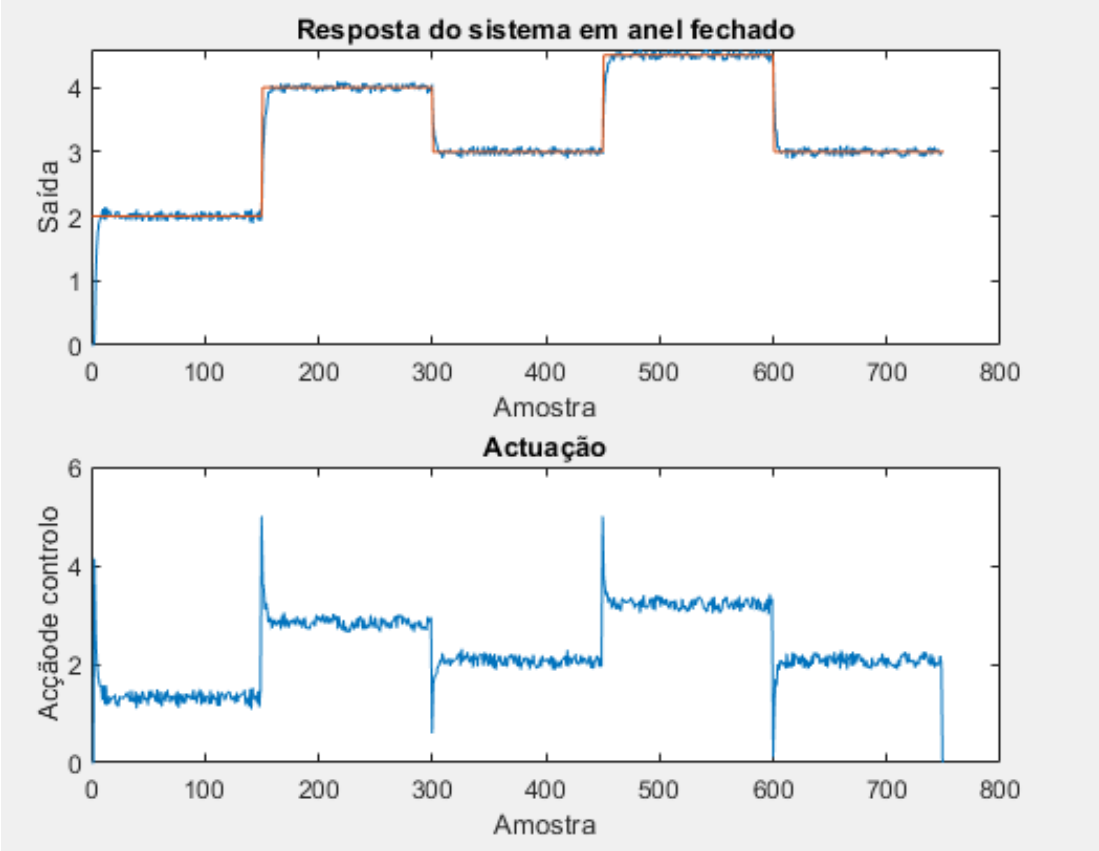

Figura 6 – Resposta do sistema em anel fechado da simulação 1

Simulação 2:

Nesta simulação pretendeu-se que a resposta do sistema fosse mais lenta para evitar sobrelevações. Penalizou-se, portanto, a ação de controlo.

- p = 15;
- q = 2;
- w = 0.1.

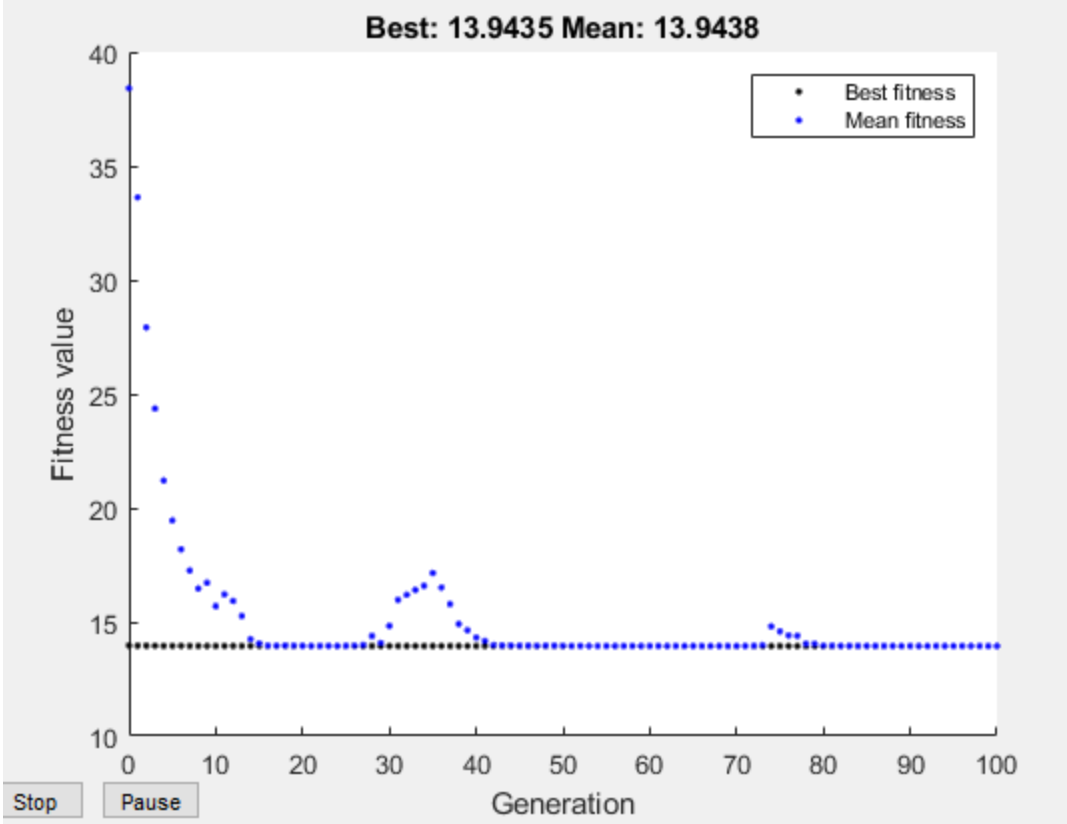

Figura 7 – Evolução geracional do erro da simulação 2

Resultando os ganhos:

- KP = 2.0565;
- KI = 2.2282;
- KD = 0.0101.

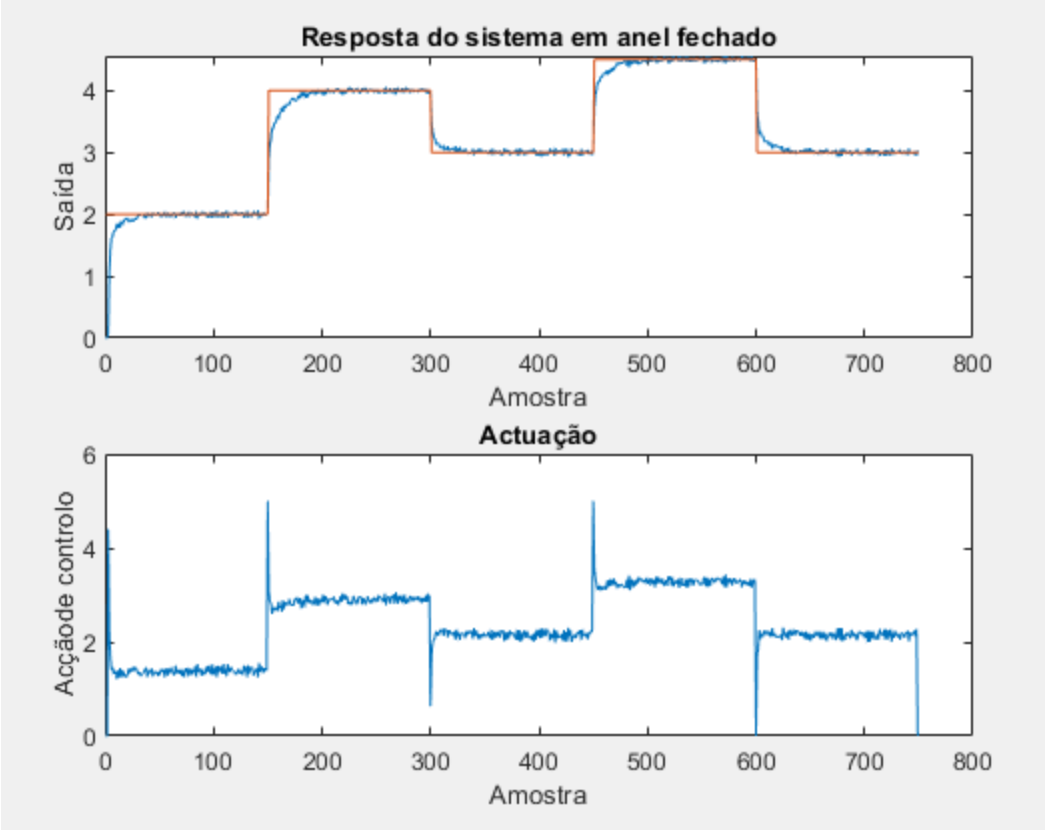

Figura 8 – Resposta do sistema em anel fechado da simulação 2

Teste no processo

Após uma análise detalhada e devida validação do comportamento do controlador em modo simulação em MatLab, o mesmo foi testado para as mesmas duas configurações de pesos do critério de desempenho, considerando também a mesma referência.

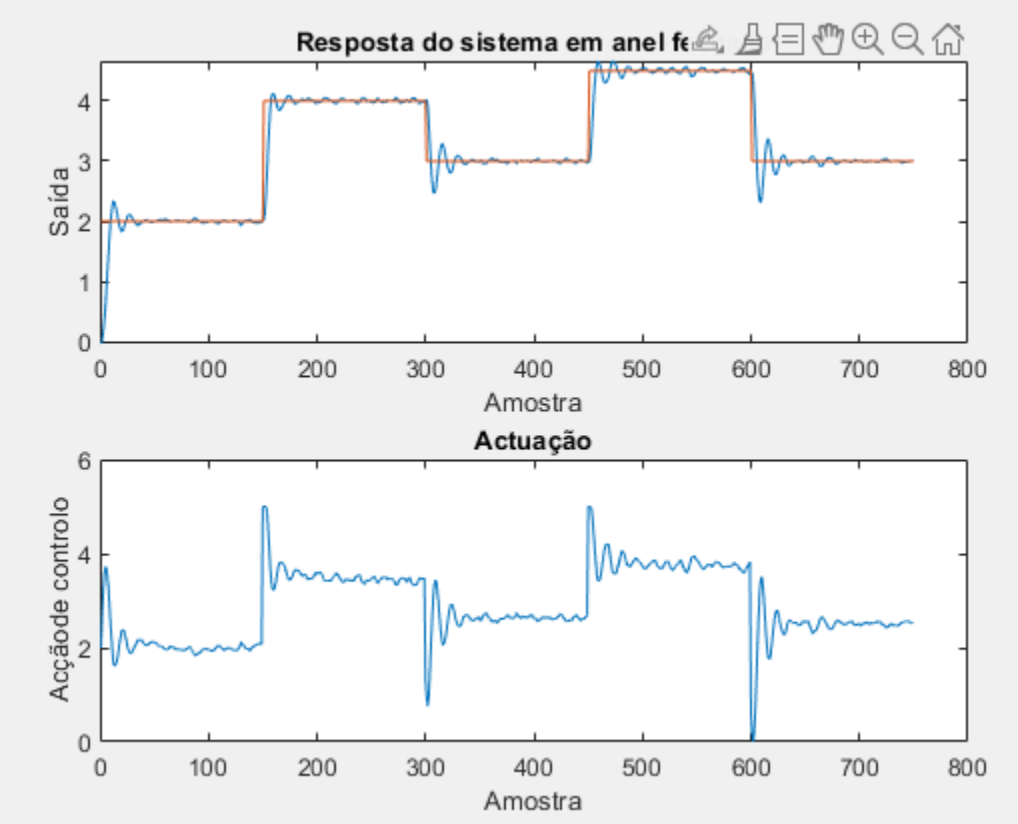

Figura 9 -Teste no processo da primeira configuração de pesos do critério de desempenho

Ao contrário do que aconteceu na simulação, este controlador apresenta uma sobrelevação, com um pequeno comportamento oscilatório ao longo da referência.

Por o sistema apresentar uma sobrelevação, aumentou-se a penalização da ação de controlo, tendo como resultado o que está apresentado na figura 10.

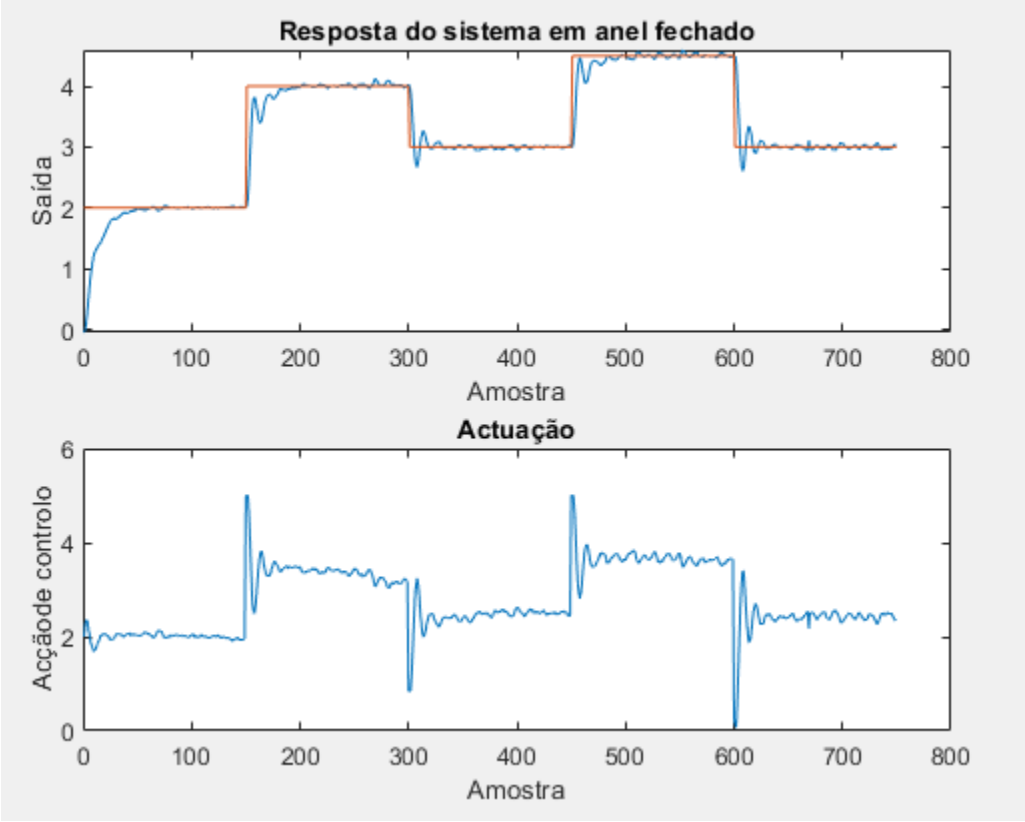

Figura 10- Teste no processo da segunda configuração de pesos do critério de desempenho

A primeira configuração apresenta um somatório de erro quadrático de 63.6223, enquanto que a segunda, apresenta um erro quadrático de 72.2727.

Conclusão

Os algoritmos de computação evolutiva mostraram-se ser bastante úteis para a optimização de valores, no que toca à sua simplicidade na implementação.

No entanto, pela análise dos resultados obtidos experimentalmente, quando comparados com as simulações com o modelo obtido através do algoritmo PSO, vemos que este não captou na perfeição as dinâmicas do processo. Pelos resultados do teste no processo, vemos que o sistema tem um comportamento oscilatório que não foi captado na simulação.

Todas as tarefas foram concluídas com sucesso e sem grandes problemas no dia do teste no processo no laboratório, devido à simplicidade de implementação do algoritmo.

Anexo

ModelPSO.m

```
close all; clear all; clc;

% dctr;
% yreal;

% Load modelData; % File with data collected from the plant

load dataset.mat

% yreal = Ye; % dados reais do processo
dctr = Ue;
yreal = Ye;

%limites para os parâmetros ai, bi
% ARX(3,2,1) -> no. de parâmetros do modelo
UB = [5 5 5 5 5];
LB = [-5 -5 -5 -5 -5];

% Adjustment parameters PSO
c1 = 1.49; % Default SelfAdjustment
c2 = 1.49; % Default SocialAdjustment

options = optimoptions(@particleswarm,'MaxIter',300,'SelfAdjustment',c1,...
    'SocialAdjustment',c2,'SwarmSize',500,'Display','iter');

fun = @(theta)modelFitness(theta,dctr,yreal);

[theta,fval] = particleswarm(fun,5,LB,UB,options)
theta
clc

save modelpar.mat theta -mat

modelsimul

%theta
```

ModelFitness.m

```
function f = modelFitness(theta,dctr,yreal)

% dctr;    % Control action
% yreal;   % Plant output

% ARX(3,2,1)
f = 0;
N = length(yreal);
y = zeros(N);

for  index = 4 : N

    y(index) = -theta(1)*y(index-1) - theta(2)*y(index-2) - theta(3)*y(index-
3)...
        + theta(4)*dctr(index-2) + theta(5)*dctr(index-3);

    f = f + ((y(index)-yreal(index))^2)/(N-4); % MSE - Mean Squared error

end

end
```

ModelSimul.m

```
load dataset.mat
load modelpar
N = length(Ye);
y = zeros(N);

Ne = length(Ue);
yme = zeros(Ne,1);
kmax = Ne;
Ts = 0.08;

for  index = 4 : Ne

    yme(index) = -theta(1)*yme(index-1) - theta(2)*yme(index-2) -
theta(3)*yme(index-3)...
        + theta(4)*dctr(index-2) + theta(5)*dctr(index-3);

end

figure
plot((1:kmax)*Ts, Ye,'k',(1:kmax)*Ts,yme,'b')
legend('Real','Simulation')
title('Estimation dataset: Model vs. REal data')
xlabel({'Time [s]'}, 'Interpreter', 'latex')
ylabel({'Output [V]'}, 'Interpreter', 'latex')
```

ctrlAG.m

```
close all; clear all; clc;

% dctr;
% yreal;

% Load modelData; % File with data collected from the plant

load dataset.mat
load modelpar.mat
load weightsctrl.mat

% yreal = Ye; % dados reais do processo
%dctr = Ue;
r = [2*ones(150,1); 4*ones(150,1); 3*ones(150,1); 4.5*ones(150,1);
3*ones(150,1)];

%limites para os parâmetros ai, bi
% ARX(3,2,1) -> no. de parâmetros do modelo
UB = [5 5 5];
LB = [0.1 0.1 0.01];
nv = 3;
%niter = 100;
% p = weights(1);
% q = weights(2);
% w = weights(3);
p = 10;
q = 0.1;
w = 0.1;

% Adjustment parameters PSO

c1 = 0.85; % Default SelfAdjustment
%c2 = 1.49; % Default SocialAdjustment

%options = optimoptions(@particleswarm,'MaxIter',300,'SelfAdjustment',c1,...
    %'SocialAdjustment',c2,'SwarmSize',500,'Display','iter');



%options = optimoptions('ga','ConstraintTolerance',1e-6,'PlotFcn',
@gaplotbestf, 'CrossoverFraction', c1, 'SelectionFcn','selectionroulette',
'MaxGenerations', niter,  'PopulationType', 'doubleVector');
%options = optimoptions('ga','ConstraintTolerance',1e-6,'PlotFcn',
@gaplotbestf, 'CrossoverFraction', c1, 'SelectionFcn','selectionroulette',
'MaxGenerations', niter, 'InitialPopulationMatrix',[100 3],
'PopulationType', 'doubleVector');


fun = @(Ks)ctrlFitness(Ks, theta,r, p,q,w);

options = gaoptimset(@ga);
```

```
options = gaoptimset('PopulationType', 'doubleVector', 'PopulationSize', 300,
'CrossoverFraction', c1, 'Generations', 100, 'SelectionFcn',
@selectionroulette, 'PlotFcns', @gaplotbestf);

[Ks,fval] = ga(fun,3,[],[],[],[],LB,UB,[],[],options);


clc

save ctrlpar.mat Ks -mat

%modelsimul

Ks
```

ctrlFitness.m

```
function f = ctrlFitness(Ks, theta,r, p,q,w)

% dctr;    % Control action
% yreal;   % Plant output

% ARX(3,2,1)
f = 0;
N = length(r);
y = zeros(N,1);
ek = zeros(N,1);
u = zeros(N,1);
%u(1:4) = r(1:4);

for  index = 3 : N-1
    u(index) = uctrl(u(index-1), ek(index), ek(index-1), ek(index-2), Ks(1),
Ks(2), Ks(3), 0.08);
    y(index+1) = -theta(1)*y(index) - theta(2)*y(index-1) -
theta(3)*y(index)...
            + theta(4)*u(index) + theta(5)*u(index-1);
    ek(index+1) = r(index+1) - y(index+1);

        f = f + (p * (r(index +1) - y(index + 1))^2 + q * (u(index ))^2 +...
        w * (u(index) - u(index -1))^2)/(N -4);
end

end
```

PID_SIM.m

```
clear all, clc, close all;

load ctrlpar.mat
load modelpar.mat

r = [2*ones(150,1); 4*ones(150,1); 3*ones(150,1); 4.5*ones(150,1);
3*ones(150,1)];
N = length(r);
y = zeros(N,1);
ek = zeros(N,1);
u = zeros(N,1);
f = 0;


for  index = 3 : N-1
    u(index,1) = uctrl(u(index-1,1), ek(index,1), ek(index-1,1), ek(index-
2,1), Ks(1), Ks(2), Ks(3), 0.08);
    y(index+1,1) = -theta(1)*y(index,1) - theta(2)*y(index-1,1) -
theta(3)*y(index-2)...
            + theta(4)*u(index,1) + theta(5)*u(index-1,1) + rand*0.15;
    ek(index+1,1) = r(index+1,1) - y(index+1,1);


end
erro = sumsqr(ek);
subplot(2,1,1),plot(y(2:end)), hold on, plot(r(1:end)),hold off;
title('Resposta do sistema em anel fechado')
ylabel('Saída'), xlabel('Amostra')
subplot(2,1,2), plot(u(2:end))
title('Actuação')
ylabel('Acçãode controlo'), xlabel('Amostra')
```

Processo.m

```
%%% Script para testar o controlador%%%
clear all, close,clc
load ctrlpar.mat

r = [2*ones(150,1);4*ones(150,1);3*ones(150,1);
4.5*ones(150,1);3*ones(150,1)];
%r = [3*ones(450,1)];
Ts =  0.08; % Definir intervalo de amostragem
usbinit% Inicialização da placa de aquisição

% ************************************************
N = length(r);
u = zeros(N,1);
ek = r;


for index= 1:N
    y(index,1)= usbread(0);
           ek(index,1) = r(index,1) - y(index,1);
    tic% Inicia cronómetro
           ek(index,1) = r(index,1) - y(index,1);
     if index<= 2
         u(index,1) = r(index);
     else
        u(index,1) = uctrl(u(index-1,1), ek(index,1), ek(index-1,1),
ek(index-2,1), Ks(1), Ks(2), Ks(3), 0.08);

     end
        u(index,1) = max(min(u(index,1),5),0); % Saturação da excitação
        usbwrite(u(index),0)
        Dt = toc; % Stop cronómetro
        pause(Ts-Dt)
        index
end
usbwrite(0,0)



subplot(2,1,1), plot(y(1:end),'r'),hold on, plot(r(1:end),'g'),hold off,
title('Resposta do sistema em anel fechado')
ylabel('Saida'), xlabel('Amostra')
subplot(2,1,2), plot(u(1:end))
title('Actuacao')
ylabel('Accao de controlo'), xlabel('Amostra')

save expdata.mat r u y -mat
```