Aprendizagem Computacional

Trabalho Prático 4
Controlo difuso e sistemas neuro-difusos

João Silva nº2012131780
Noé Godinho nº 2011159459

15 de Dezembro de 2015

Índice

1 Introdução

Este trabalho tem como objetivo a implementação de 2 tipos de sistemas difusos:

- **Controladores difusos**: Estes controladores fazem parte de um sistema, que representam um hipotético processo real. Foram implementados dois tipos, *Mamdani* e *Sugeno*.
- **Sistemas neuro-difusos**: Estes sistemas são utilizados na modelação de processos e sistemas. Num sistema qualquer, aplica-se-lhe uma entrada e obtém-se uma saída.

A lógica difusa estende a lógica booleana admitindo valores lógicos intermédios entre o falso e o verdadeiro.

2 Controladores difusos

Os controladores difusos possuem um conjunto de regras lógicas, constituídas por três fases:

- Fase de entrada.
- Fase de processamento.
- Fase de saída.

Na fase de entrada mapeia a entrada recebida pelo controlador nas funções de pertença e valores de verdade apropriados. Depois, na fase de processamento, as regras recebidas à entrada são invocadas e combinam o resultado, sendo ele devolvido na fase de saída.

Neste trabalho, foram implementados controladores do tipo *Mamdani* e *Sugeno*, cada um com 9 e 25 regras, com apoio da *Fuzzy Logic Toolbox* do *Matlab*.

2.1 Implementação

Para a implementação desta parte A do trabalho descrito, foi criado um *script* com nome run.m que dá ao utilizador a escolher entre as seguintes opções na linha de comandos:

- Tipo de função de referência no modelo: *Square* ou *Sin*.
- Tipo de controlador a usar: *Mamdani* ou *Sugeno*.

- Número de regras: 9 ou 25.
- Método de desfuzificação: *Centroid* ou *Mom* no caso do *Mamdani* e *Wtaver* ou *Wtsum* no caso do *Sugeno*.
- Tipo de perturbação no modelo: sem perturbação, com perturbação no atuador ou com perturbação na carga.

Uma vez feitas as escolhas, o *script* irá carregar o controlador e modelo correspondente.

Também foram criados os modelos, com o *Simulink* e os controladores, com o *fuzzy*. Existem seis modelos diferentes, cada um com a função de referência *Square* ou *Sin* e com os tipos de perturbação acima falados.

Por último, foram criados oito controladores, com os tipos disponíveis, o número de regras e o método de desfuzificação acima falados e função de membro, *gaussmf*.

2.2 Função de transferência

Cada grupo tem uma função de transferência a aplicar no modelo, resultando em processos diferentes a simular.

No caso do nosso grupo, a função de transferência é a seguinte:

$$\frac{4}{s^3 + 3s^2 + 4s + 2}$$

2.3 Regras

Tal como foi explicado acima, foram aplicados nos controladores 9 e 25 regras.

Ao aplicar mais regras, ou seja, regras diferentes, é alterada a ação tomada pelo controlador. Se forem incrementadas o número de regras, isto é, deixar as regras anteriores intactas e criar novas, é esperada uma melhoria da performance do controlador.

A seguir são apresentadas as tabelas com o número de regras e as suas combinações, em que:

- N - Negativo
- ZE - Zero
- P - Positivo
- NB - Negativo Grande

- NS - Negativo Pequeno
- PB - Positivo Grande
- PS - Positivo Pequeno

e_k \ Δe_k	N	ZE	P
N	N	N	Z
ZE	N	Z	P
P	Z	P	P

Figura 1: Tabela das 9 regras

e_k \ Δe_k	NB	NS	ZE	PS	PB
NB	NB	NB	NB	NS	ZE
NS	NB	NB	NS	ZE	PS
ZE	NB	NS	ZE	PS	PB
PS	NS	ZE	PS	PB	PB
PB	ZE	PS	PB	PB	PB

Figura 2: Tabela das 25 regras

2.4 Modelo

O modelo usado para a implementação dos controladores foi o seguinte:

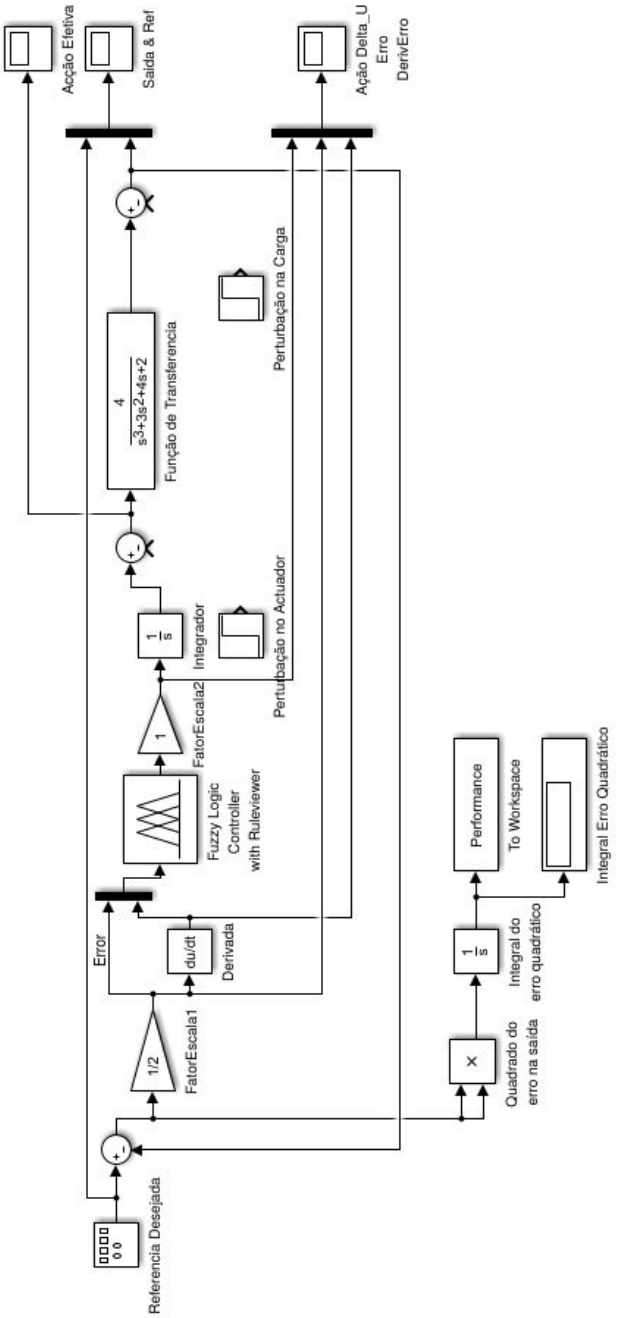

Figura 3: Modelo implementado

2.5 Execução

Uma vez executados todos os testes, foram obtidos os seguintes resultados com T = 500.0:

Referência	Controlador	Regras	Desfuzificação	Perturbação	Performance
Square	Mamdani	9	Centroid	Sem	15.51
Square	Mamdani	9	Centroid	Carga	20.36
Square	Mamdani	9	Centroid	Atuador	24.84
Square	Mamdani	9	Mom	Sem	363.3
Square	Mamdani	9	Mom	Carga	258.7
Square	Mamdani	9	Mom	Atuador	310.6
Square	Mamdani	25	Centroid	Sem	10.64
Square	Mamdani	25	Centroid	Carga	12.57
Square	Mamdani	25	Centroid	Atuador	15.07
Square	Mamdani	25	Mom	Sem	54.83
Square	Mamdani	25	Mom	Carga	28.66
Square	Mamdani	25	Mom	Atuador	33.4
Square	Sugeno	9	Wtaver	Sem	9.851
Square	Sugeno	9	Wtaver	Carga	11.72
Square	Sugeno	9	Wtaver	Atuador	14.46
Square	Sugeno	9	Wtsum	Sem	14.63
Square	Sugeno	9	Wtsum	Carga	16.54
Square	Sugeno	9	Wtsum	Atuador	22.23
Square	Sugeno	25	Wtaver	Sem	10.88
Square	Sugeno	25	Wtaver	Carga	12.68
Square	Sugeno	25	Wtaver	Atuador	15.93
Square	Sugeno	25	Wtsum	Sem	419.8
Square	Sugeno	25	Wtsum	Carga	3762
Square	Sugeno	25	Wtsum	Atuador	3801

Referência	Controlador	Regras	Desfuzificação	Perturbação	Performance
Sin	Mamdani	9	Centroid	Sem	20.85
Sin	Mamdani	9	Centroid	Carga	22.87
Sin	Mamdani	9	Centroid	Atuador	26.05
Sin	Mamdani	9	Mom	Sem	225.3
Sin	Mamdani	9	Mom	Carga	105.6
Sin	Mamdani	9	Mom	Atuador	145.2
Sin	Mamdani	25	Centroid	Sem	0.08708
Sin	Mamdani	25	Centroid	Carga	1.998
Sin	Mamdani	25	Centroid	Atuador	4.472
Sin	Mamdani	25	Mom	Sem	43.07
Sin	Mamdani	25	Mom	Carga	24.5
Sin	Mamdani	25	Mom	Atuador	39.58
Sin	Sugeno	9	Wtaver	Sem	0.04935
Sin	Sugeno	9	Wtaver	Carga	1.891
Sin	Sugeno	9	Wtaver	Atuador	4.625
Sin	Sugeno	9	Wtsum	Sem	0.02217
Sin	Sugeno	9	Wtsum	Carga	1.911
Sin	Sugeno	9	Wtsum	Atuador	7.606
Sin	Sugeno	25	Wtaver	Sem	0.028
Sin	Sugeno	25	Wtaver	Carga	1.803
Sin	Sugeno	25	Wtaver	Atuador	5.046
Sin	Sugeno	25	Wtsum	Sem	332.8
Sin	Sugeno	25	Wtsum	Carga	323.9
Sin	Sugeno	25	Wtsum	Atuador	340.4

2.6 Conclusão

Como esperado, a função de referência *Sin* tem valores melhores de desempenho, no geral, do que a função *Square*.

A função *Sin* é uma função mais suave do que a função *Square*. As abruptas variações da função *Square* tem um efeito negativo no controlador, o que faz com que resulte em erros mais pronunciados.

Também, observa-se maior desempenho quando o número de regras é 25 e que, no nosso modelo, o controlador *Mamdani* é o que apresenta melhores resultados.

Por último, no geral, quando não existe uma carga a perturbar o sistema, obtém-se melhores resultados, isto é porque o controlador necessita adaptar-se à perturbação, causando uma perda do desempenho.

3 Sistemas neuro-difusos

Estes sistemas são muito utilizados na modelação de processos e sistemas. A partir de um sistema definido aplica-se-lhe uma entrada e obtém-se uma saída.

No caso de sistemas dinâmicos, com memória, descritos por equações de diferenças, temos uma saída y num instante k, que depende de algumas saídas e entradas, u, anteriores. Para sistemas com inércia, a entrada só irá afetar a saída no instante seguinte.

Assim, temos a seguinte equação:

$$y(k) = f(y(k-1), y(k-2), ..., u(k-1), u(k-2), ...) \tag{1}$$

A partir daí é necessário usar uma técnica de *clustering* para determinação das regras iniciais e otimizar com vista a diminuir o erro obtido, sendo ele a diferença entre a saída pretendida e a saída do sistema difuso.

3.1 Modelo

Para começar, foi criado um modelo no *Simulink* que permitisse gerar um bom conjunto de dados entrada-saída, para assim obter um bom conjunto de regras.

Assim, foi criado o seguinte modelo:

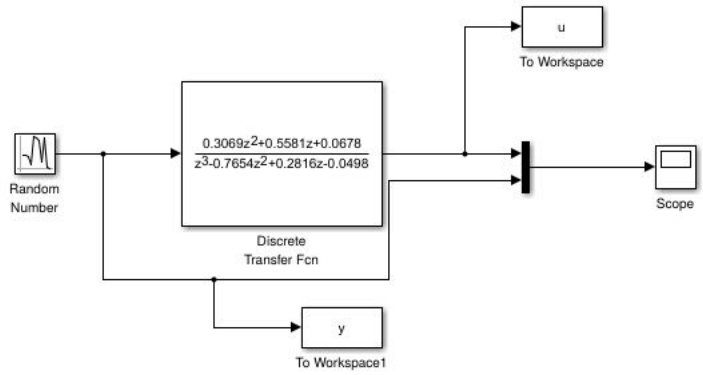

Figura 4: Modelo implementado

3.2 Função de transferência discreta

Para obter a função de transferência discreta, foi necessário usar a função de transferência mostrada na secção 2.2.

Assim, tem-se a seguinte equação:

$$\mathrm{G(s)} = G(s) = \frac{4}{s^3 + 3s^2 + 4s + 2}$$

Logo, foi necessário usar o seguinte código para obter a função de transferência discreta:

```
>>num = 10;
>>den = [1 3 4 2];
>>[numd, dend] = c2dm(num, den, 1, 'zoh');
```

Assim, as variáveis *numd* e *dend* irão conter o numerador e denominador, respetivamente, da função de transferência discreta que é necessária:

$$G(z) = \frac{0.3069z^2 + 0.5581z + 0.0678}{z^3 - 0.7654z^2 + 0.2816z - 0.0498}$$

3.3 Matriz de dados

Depois da execução do modelo com a função de transferência discreta, é necessário a criação da matriz de dados para *clustering*.

Esta matriz é necessária para a geração das regras difusas, explicadas na próxima secção, a partir do *clustering*.

A matriz terá um tamanho de $(N-k)x7$, em que N será o tamanho dos vetores de entrada e saída gerados pelo modelo acima explicado e k é o instante em que se inicializa o *target* (não há valores negativos de k).

Em seguida é explicado o conteúdo de cada coluna, tendo como $k = 3$:

- A primeira coluna contém a série temporal da saída deslocada **duas** linhas para cima ($i = 2..(N-1)$).
- A segunda coluna contém a série temporal da saída deslocada **uma** linha para cima ($i = 1..(N-2)$).
- A terceira coluna contém a série temporal da saída como se obteve na simulação ($i = 0..(N-3)$).
- A quarta coluna contém a série temporal da entrada deslocada **duas** linhas para cima ($i = 2..(N-1)$).

- A quinta coluna contém a série temporal da entrada deslocada **uma** linha para cima ($i = 1..(N-2)$).
- A sexta coluna contém a série temporal da entrada como se obteve na simulação ($i = 0..(N-3)$).
- Por último, a sétima coluna contém a série temporal deslocada **três** linhas para cima, ela é o *target* (($i = 3..N$).

3.4 Regras difusas

Se forem aplicadas regras difusas (com TSK de ordem zero) na equação 1 genérica, obtém-se as seguintes regras genéricas:

IF $y(k-1)$ is A_1 AND $y(k-2)$ is A_2 AND $y(k-i)$ is A_i AND $y(k-n)$ is A_n AND

$u(k-1)$ is A_{n+1} AND $u(k-2)$ is A_{n+2} AND $u(k-j)$ is A_{n+j} AND $u(k-n)$ is A_{2n}

THEN $y(k)$ is α

Neste trabalho, é necessário aplicar regras difusas na seguinte equação:

$$y(k) = f(y(k-1), y(k-2), y(k-3), u(k-1), u(k-2), u(k-3)$$

Assim, obtiveram-se as seguintes regras genéricas para a equação anterior:

IF $y(k-1)$ is A_1 AND $y(k-2)$ is A_2 AND $y(k-3)$ is A_3 AND

$u(k-1)$ is A_4 AND $u(k-2)$ is A_5 AND $u(k-3)$ is A_6 THEN $y(k)$ is α

O nosso *target* está localizado no instante 3 ($k = 3$), tal como explicado na secção anterior, logo obtém-se as seguintes regras:

IF $y(2)$ is A_1 AND $y(1)$ is A_2 AND $y(0)$ is A_3 AND

$u(2)$ is A_4 AND $u(1)$ is A_5 AND $u(0)$ is A_6 THEN $y(3)$ is α

3.5 Modelo do cálculo do desempenho

Também foi criado mais um modelo no *Simulink* que mostrasse o desempenho do controlador criado:

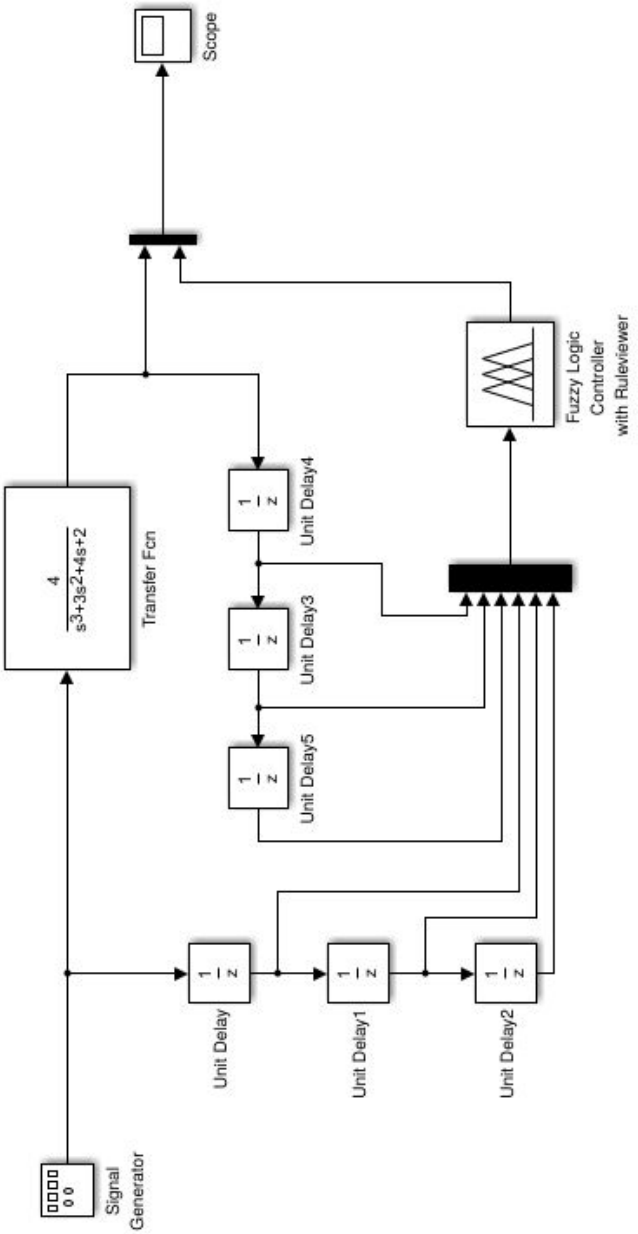

Figura 5: Modelo implementado

3.6 Implementação e resultados

Para iniciar o programa, é necessário correr o ficheiro *runB.m*, que irá verificar se existem as variáveis de entrada e saída, necessárias para a verificação dos *clusters* e criação de regras difusas. Caso não existam em ficheiro ou no *workspace*, irá abrir o modelo do *Simulink* para que o utilizador execute a simulação.

Uma vez executada a simulação, as variáveis de entrada e saída, com os dados gerados, irão estar disponíveis no *workspace*, sendo assim necessário re-executar o *runB.m* para criar a matriz de dados a utilizar. O utilizador também pode guardar essas variáveis de entrada, num ficheiro *uy.mat*, executando o ficheiro *save_script.m*.

Após a criação da matriz de dados, é possível detetar os *clusters* da mesma, usando o método subtrativo. Para isso, executa-se na *shell* do *Matlab* o comando *findcluster* e carrega-se a matriz de dados (*matrix.dat*) e clica-se no botão *start*.

Assim, obtém-se os seguintes *clusters*:

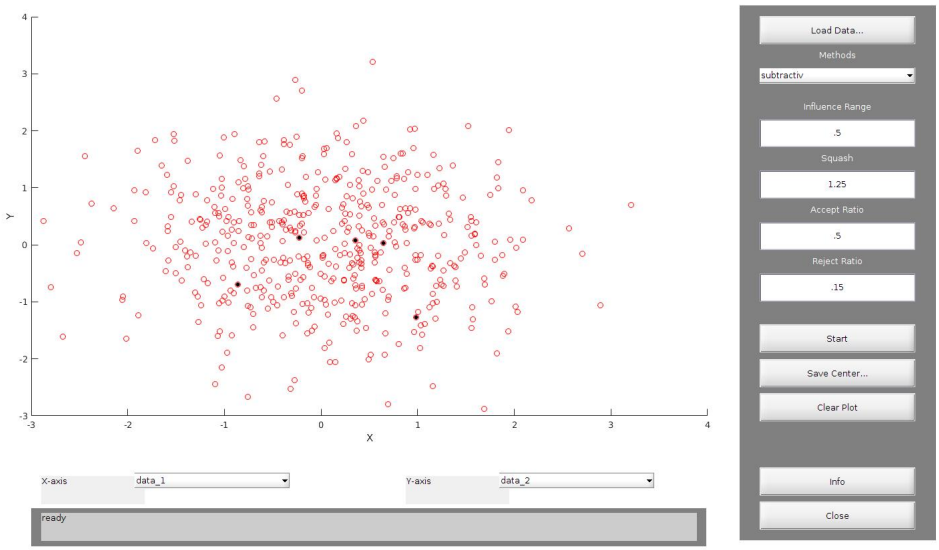

Figura 6: *Clusters* detectados

Como se verifica, existem 5 *clusters* diferentes na matriz de dados criada.

Após a verificação dos *clusters*, executa-se o comando *anfisedit* na *shell* do *Matlab*, de maneira a poder criar o controlador a ser usado no modelo que analisa a *performance* do mesmo, para poder verificar o desempenho das regras difusas criadas a partir do controlador.

Para isso, é necessário carregar a matriz a partir do botão *Load Data*, gerar o controlador com a opção *Sub. Clustering*, para usar o método subtrativo, guardar o mesmo no *workspace* e/ou num ficheiro (*fuzzymodel.fis*), escolher um número de *epochs* (no nosso caso foram 300), clicar em *train now* e por último em *test now*. Desta maneira, obteve-se o seguinte gráfico:

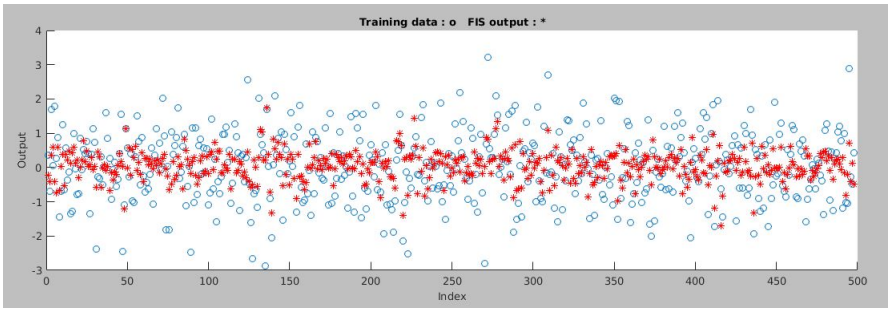

Figura 7: Gráfico da criação do controlador com treino

Por último, foram executados três instâncias do modelo referido na secção 3.5, com as ondas *Square*, *Sine* e *Sawtooth* com T = 500:

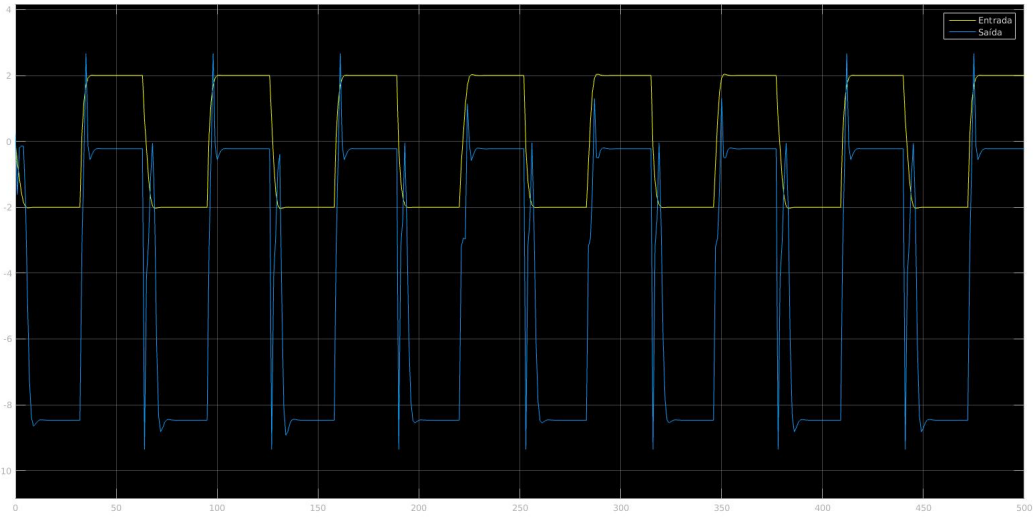

Figura 8: Gráfico do resultado da simulação do modelo com onda *Square*

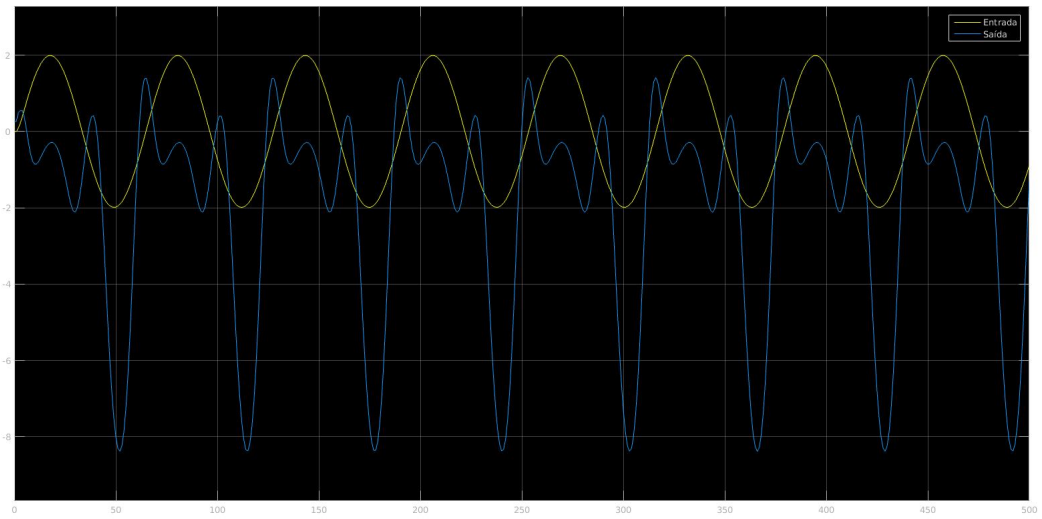

Figura 9: Gráfico do resultado da simulação do modelo com onda *Sine*

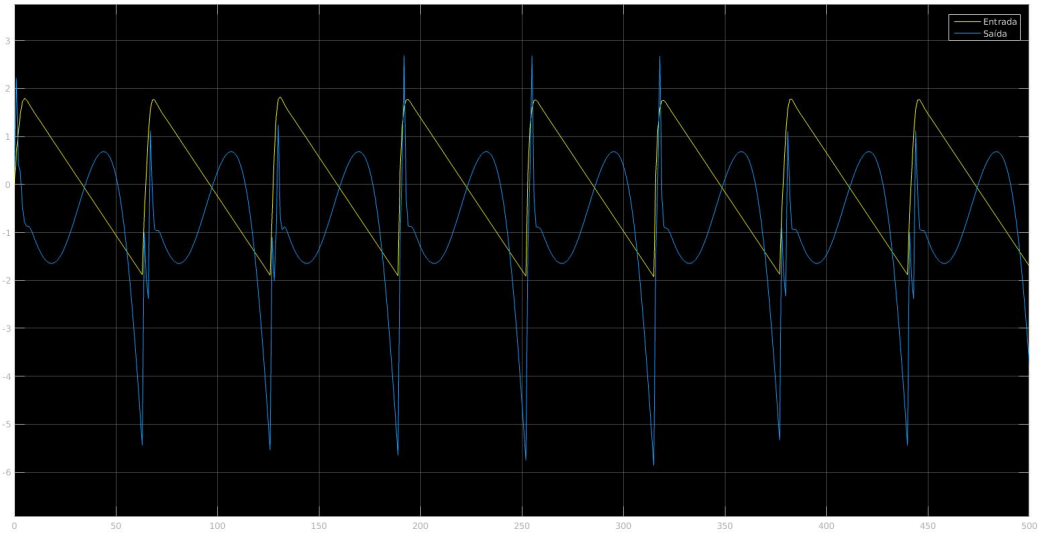

Figura 10: Gráfico do resultado da simulação do modelo com onda *Sawtooth*

3.7 Conclusão

Como é possível observar, as ondas *Sine* e *Sawtooth* mostram valores mais suaves de saída, o que era de esperar. Isso verifica uma melhoria de desempenho em relação à onda *Square*, que tem transições mais bruscas, tal como na parte A deste trabalho.

Em relação à matriz de dados, pode observar-se uma boa dispersão dos dados, o que permitiu uma boa criação do controlador com regras difusas. Também foram detetados vários *clusters*, o que mostra a dispersão dos dados.