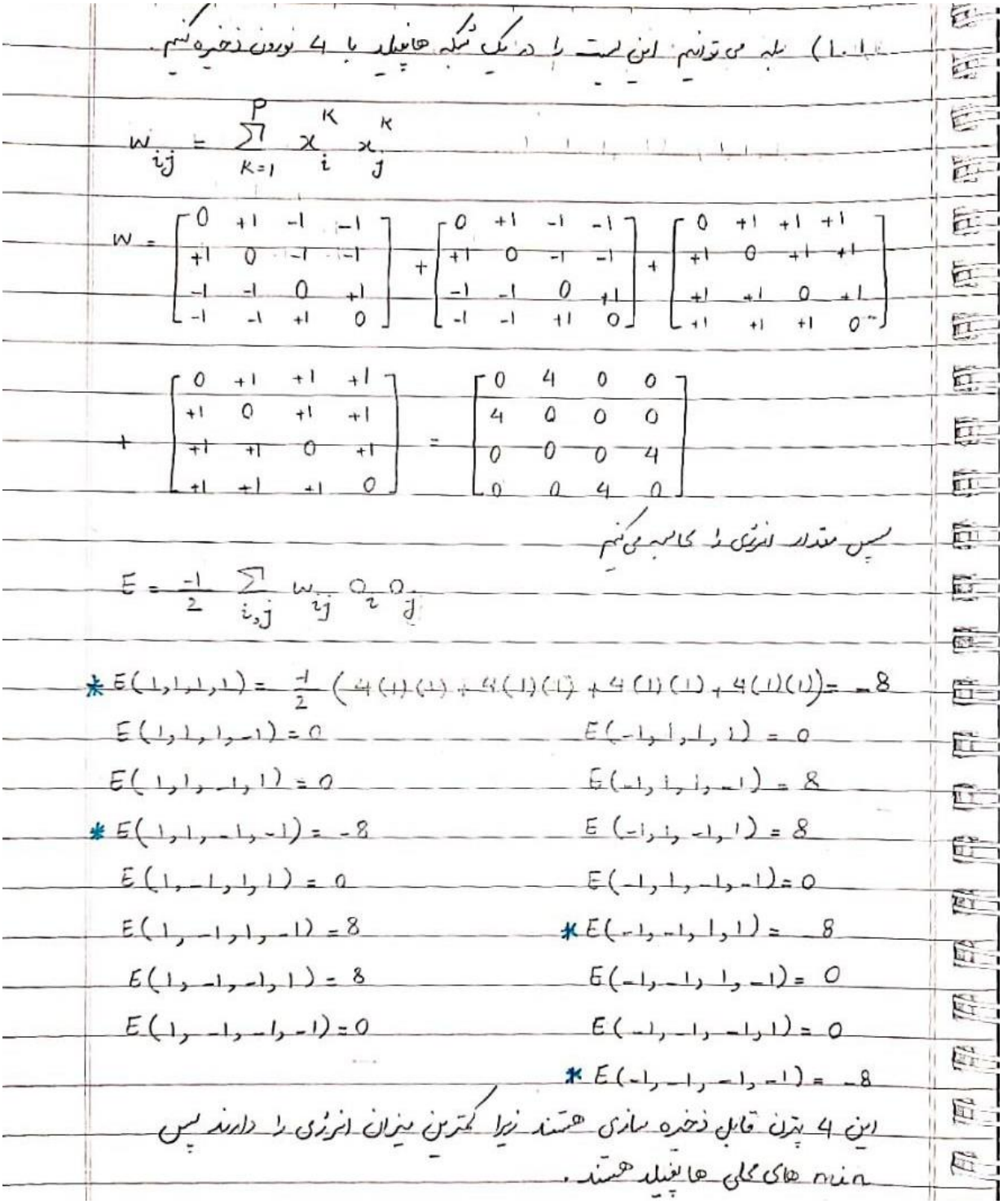
به نام خدا

گزارش تمرین سوم مبانی هوش

غزل زمانی نژاد، 97522166

1. شبکه Hopfield



1.2.در این سوال می خواهیم دو پترن را در شبکه هاپفیلد ذخیره سازی کنیم. برای این کار، کلاسی با نام Hopfield تعریف می کنیم که constructor آن پترن ها را به عنوان ورودی می گیرد. سپس باید ماتریس وزن را به صورت دو بعدی و با ابعاد (طول هر پترن، طول هر پترن) تشکیل دهیم. برای پر کردن ماتریس وزن از متد calculate_weights کمک می گیریم. در این متد به کمک حلقه های for ابتدا تک تک عناصر یک پترن خاص را در هم ضرب می کنیم، سپس با مقدار قبلی وزن ها جمع می کنیم و هم در سطر و هم در ستون ماتریس وزن می نویسیم. (به گونه ای که این ماتریس متقارن شود.)

```
weights:  [[ 0.  0.  2. -2.  0. -2.]
 [ 0.  0.  0.  0. -2.  0.]
 [ 2.  0.  0. -2.  0. -2.]
 [-2.  0. -2.  0.  0.  2.]
 [ 0. -2.  0.  0.  0.  0.]
 [-2.  0. -2.  2.  0.  0.]]
```

مثال از نوع ضرب کردن در یک پترن:

[1, 2, 3] = [0, 2, 3,
2, 0, 6,
3, 6, 0]

پس از آن نوبت به تست کردن شبکه هاپفیلد می رسد.

ابتدا شبکه را با پترن (1, 1, 1, -1, -1, -1) تست می کنیم. از متد sync_update استفاده می کنیم. در این متد ابتدا یک حلقه while داریم که در آن به تعداد cycleهای مورد نظر این کار انجام می شود. در یک حلقه for، پترن تست را در هر سطر از ماتریس وزن ضرب داخلی می کنیم. سپس با توجه به sign آن، مقدار عنصر iام پترن تست را تغییر می دهیم. اگر پترن بدست آمده با پترنی که در حلقه قبل بدست آمده بود برابر باشد، پترن تست stable است. که این پترن بعد از iteration سوم پایدار می شود.

```
iteration: 0
after update:  [1 1 1 1 1 1]
iteration: 1
after update:  [-1 -1 -1 -1 -1 -1]
iteration: 2
after update:  [-1 -1 -1 -1 -1 -1]
test pattern is stable
```

سپس پترن (-1, 1, 1, -1, -1, -1) را وارد شبکه می کنیم. و متد sync_update را بر روی آن صدا می زنیم. و مشاهده می کنیم که این پترن مرتب در حال تغییر است و یک در میان جا به جا می شود. پس این پترن پایدار نیست.

```
iteration: 0
after update:  [-1  1  1  1  1  1]
iteration: 1
after update:  [ 1 -1 -1  1 -1  1]
iteration: 2
after update:  [-1 -1  1  1 -1  1]
iteration: 3
after update:  [ 1 -1 -1  1 -1  1]
iteration: 4
after update:  [-1 -1  1  1 -1  1]
iteration: 5
after update:  [ 1 -1 -1  1 -1  1]
iteration: 6
after update:  [-1 -1  1  1 -1  1]
iteration: 7
after update:  [ 1 -1 -1  1 -1  1]
iteration: 8
after update:  [-1 -1  1  1 -1  1]
iteration: 9
after update:  [ 1 -1 -1  1 -1  1]
iteration: 10
after update:  [-1 -1  1  1 -1  1]
iteration: 11
after update:  [ 1 -1 -1  1 -1  1]
iteration: 12
after update:  [-1 -1  1  1 -1  1]
iteration: 13
after update:  [ 1 -1 -1  1 -1  1]
iteration: 14
after update:  [-1 -1  1  1 -1  1]
iteration: 15
after update:  [ 1 -1 -1  1 -1  1]
iteration: 16
after update:  [-1 -1  1  1 -1  1]
iteration: 17
after update:  [ 1 -1 -1  1 -1  1]
iteration: 18
after update:  [-1 -1  1  1 -1  1]
iteration: 19
after update:  [ 1 -1 -1  1 -1  1]
pattern is repeating with cycle of length 2
```

1.3.این سوال مشابه سوال قبل است. با این تفاوت که برای آپدیت کردن از متد async_update استفاده می کنیم.

باید برای هر یک از اندازه های فونت، ابتدا تصاویر A تا J را در شبکه ذخیره کنیم و سپس شبکه را با تصاویر نویزی حروف امتحان کنیم.

توضیح برای فونت 16: ابتدا متد save_images را صدا می زنیم. در آن حروف الفبا را با فونت 16 تشکیل می دهیم و به عنوان تصاویر را ذخیره می کنیم. آن ها را در یک لیست می ریزیم و return می کنیم. سپس سایز بزرگترین عکس را پیدا می کنیم و به عنوان سایز شبکه در نظر می گیریم. بعد متد data_test را با ورودی لیست تصاویر و سایز شبکه هاپفیلد صدا می زنیم. در آن تمامی عکس ها را با سایز شبکه resize می کنیم تا تمامی تصاویر هم اندازه شوند. سپس آن ها را به numpy array تبدیل می کنیم. بعد به جای عناصری که 0 هستند (پیکسل های سفید تصویر) در آرایه 1- می نویسیم و سایر عناصر را با 1 پر می کنیم. و به لیست پترن ها append می کنیم. اکنون نوبت به ساختن تصاویر نویزی از پترن ها می رسد. این کار را با تابع noisy_data انجام می دهیم. این تابع تصویر و درصد نویز را به عنوان ورودی دریافت می کند. سپس در یک حلقه for، عددی را به صورت

رندوم generate می کند. اگر آن عدد از درصد نویز ورودی کمتر باشد به جای آن صفر یا یک می گذارد. یا اگر آن عدد از مکمل نویز ورودی بیشتر باشد به جای آن صفر یا یک می گذارد. در غیر این صورت آن پیکسل را تغییر نمی دهد. این کار را برای نویز 10، 30 و 60 درصد انجام می دهیم. پس از ساختن پترن ها و تست دیتا، نوبت به train و test شبکه می رسد. ابتدا شبکه را با پترن ها train می کنیم و وزن ها را بدست می آوریم (مشابه سوال قبل). سپس تابع async_update را بر روی تک تک تصاویر نویزی صدا می زنیم. در این متد، در یک حلقه for به تعداد cycle ها تست دیتا را آپدیت می کنیم. نحوه آپدیت به صورت asynchronous است؛ یعنی یکی از ایندکس های پترن تست را به صورت رندوم انتخاب می کنیم. سپس حاصل ضرب داخلی آن پترن در وزن [ایندکس رندوم انتخاب شده] را بدست می آوریم و با توجه به sign آن آن ایندکس از پترن را آپدیت می کنیم.
در هر لوپ، مقدار انرژی را نیز بر اساس فرمول زیر محاسبه می کنیم:

$$E = -\frac{1}{2}\sum_{i}\sum_{j} w_{ij}V_iV_j$$

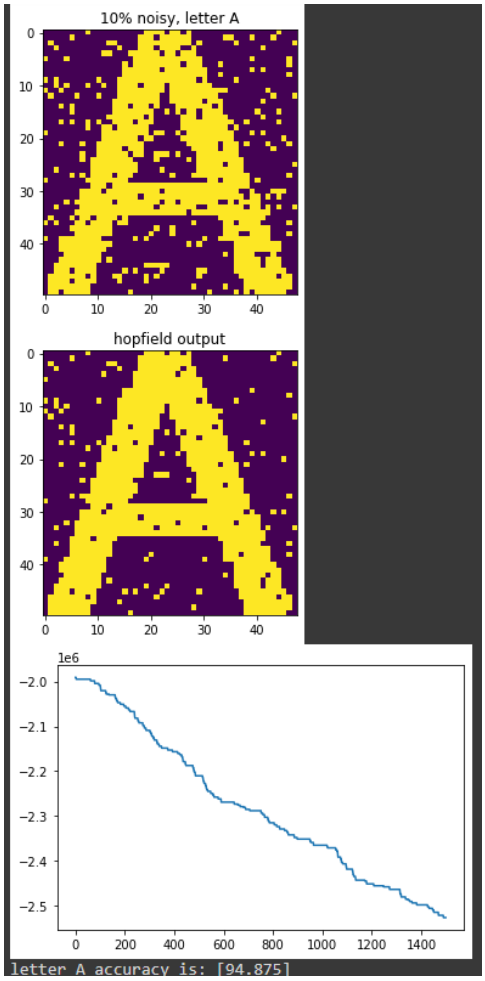
و در هر لوپ مقادیر انرژی را در یک لیست ذخیره می کنیم. در پایان لوپ خروجی و لیست انرژی را برمی گردانیم.
در پایان با تابع print_output تصویر نویزی، خروجی شبکه هاپفیلد و تابع انرژی را چاپ می کنیم. مشاهده می شود که تابع انرژی در گذر زمان به کمترین مقدار خودش رسیده است.
هم چنین برای محاسبه دقت از فرمول زیر استفاده می کنیم:

$$\frac{\text{تعداد عناصر}(pattern == ouput)}{\text{طول پترن}} * 100$$

و برای هر اندازه فونت و درصد نویز، میانگین دقت آن سایز و درصد نویز را محاسبه می کنیم.
نتایج در جدول زیر مشاهده می شود:

	10%	30%	60%
16	76.98	76.85	76.53
32	84.36	82.05	72.92
64	89.68	77.89	61.49

نمونه یکی از خروجی ها در شکل زیر مشاهده می شود:


10% noisy, letter A
hopfield output
1e6
letter A accuracy is: [94.875]

1.2) برای طراحی یک کنترلر فازی در ابتدا متغیرها را مشخص می‌کنیم. در این جا

متغیرهای ورودی میزان کثیفی و نوع کثیفی لباس، خروجی زمان شستشو است.

T D H

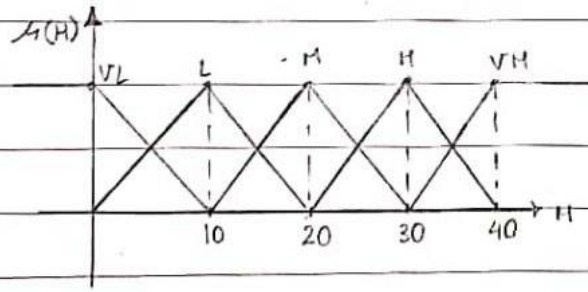
پس برای هر متغیرها، linguistic term تعیین می‌کنیم

linguistic terms (H) = { VL, L, M, H, VH }

linguistic terms(D) = { sweaty, bloody, inky, greasy }

linguistic terms(T) = { VL, L, M, H, VH }

از تابع مثلثی به عنوان fuzzification function استفاده می‌کنیم چون پیاده سازی آن ساده‌تر است. تابع membership هر متغیر را رسم کرده و آنها را کامل می‌کنیم.



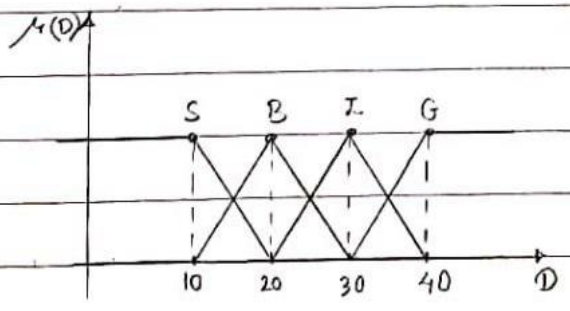
خیلی کم $H \leq 10$

کم $0 \leq H \leq 20$

متوسط $10 \leq H \leq 30$

زیاد $20 \leq H \leq 40$

خیلی زیاد $30 \leq H$



عرق $D \leq 20$

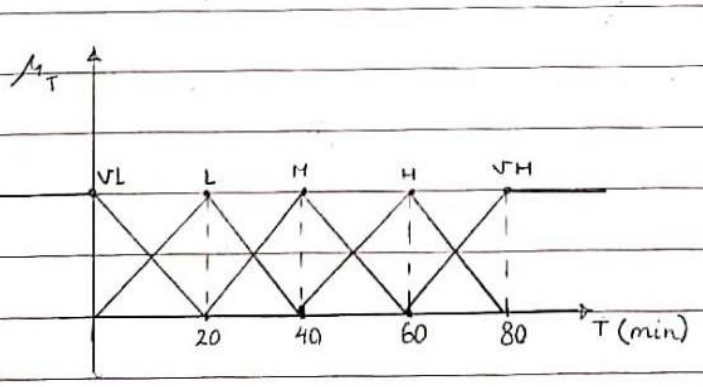
خونی $10 \leq D \leq 30$

جوهری $20 \leq D \leq 40$

روغنی $30 \leq D$

$$\mu_H = \begin{cases} \mu_{VL}(h) = \begin{cases} 1 & h \le 0 \\ -0.1h+1 & 0 \le h \le 10 \end{cases} \\ \mu_L(h) = \begin{cases} 0.1h & 0 \le h \le 10 \\ -0.1h+2 & 10 \le h \le 20 \end{cases} \\ \mu_M(h) = \begin{cases} 0.1h-1 & 10 \le h \le 20 \\ -0.1h+3 & 20 \le h \le 30 \end{cases} \\ \mu_H(h) = \begin{cases} 0.1h-2 & 20 \le h \le 30 \\ -0.1h+4 & 30 \le h \le 40 \end{cases} \\ \mu_{VH}(h) = \begin{cases} 0.1h-3 & 30 \le h \le 40 \\ 1 & 40 \le h \end{cases} \end{cases}$$

$$\mu_D = \begin{cases} \mu_S(d) = \begin{cases} 1 & d \le 10 \\ -0.1d+2 & 10 \le d \le 20 \end{cases} \\ \mu_B(d) = \begin{cases} 0.1d-1 & 10 \le d \le 20 \\ -0.1d+3 & 20 \le d \le 30 \end{cases} \\ \mu_I(d) = \begin{cases} 0.1d-2 & 20 \le d \le 30 \\ -0.1d+4 & 30 \le d \le 40 \end{cases} \\ \mu_C(d) = \begin{cases} 0.1d-3 & 30 \le d \le 40 \\ 1 & 40 \le d \end{cases} \end{cases}$$



خیلی کم $T \le 20$

کم $0 \le T \le 40$

متوسط $20 \le T \le 60$

زیاد $40 \le T \le 80$

خیلی زیاد $60 \le T$

$$
\mu_T = \begin{cases}
\mu_{VL}(t) = \begin{cases} 1 & t \leq 0 \\ \frac{-1}{20} t + 1 & 0 \leq t \leq 20 \end{cases} \\[2ex]
\mu_{L}(t) = \begin{cases} \frac{1}{20} t & 0 \leq t \leq 20 \\ \frac{-1}{20} t + 2 & 20 \leq t \leq 40 \end{cases} \\[2ex]
\mu_{M}(t) = \begin{cases} \frac{1}{20} t - 1 & 20 \leq t \leq 40 \\ \frac{-1}{20} t + 3 & 40 \leq t \leq 60 \end{cases} \\[2ex]
\mu_{H}(t) = \begin{cases} \frac{1}{20} t - 2 & 40 \leq t \leq 60 \\ \frac{-1}{20} t + 4 & 60 \leq t \leq 80 \end{cases} \\[2ex]
\mu_{VH}(t) = \begin{cases} \frac{1}{20} t - 3 & 60 \leq t \leq 80 \\ 1 & 80 \leq t \end{cases}
\end{cases}
$$

سپس به تعیین قواعد می‌پردازیم:

به کمک expert ها

1) if type of dirt is sweaty & dirtiness is H, then time is medium

2) " " " bloody & " " very low, " " low.

3) " " " inky & " " high, " " high.

4) " " " greasy & " " medium, " " high

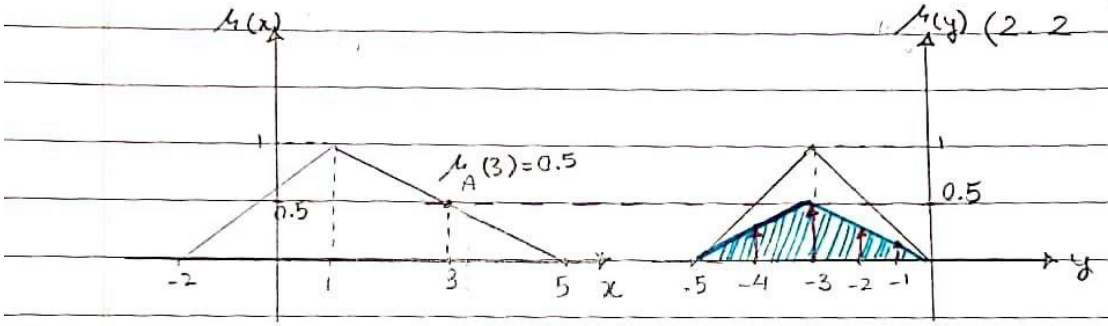
ادامه قواعد را به کمک یک جدول نمایش می‌دهیم.

D \ H	VL	L	M	H	VH
S	VL	VL	M	M	H
B	L	L	M	H	VH
I	L	M	H	H	VH
G	M	H	H	VH	VH

برای agregation از قاعده Mamdani استفاده می‌کنم.

برای defuzzification از تابع mean of maxima استفاده می‌کنم.

2.2



$A \to B$

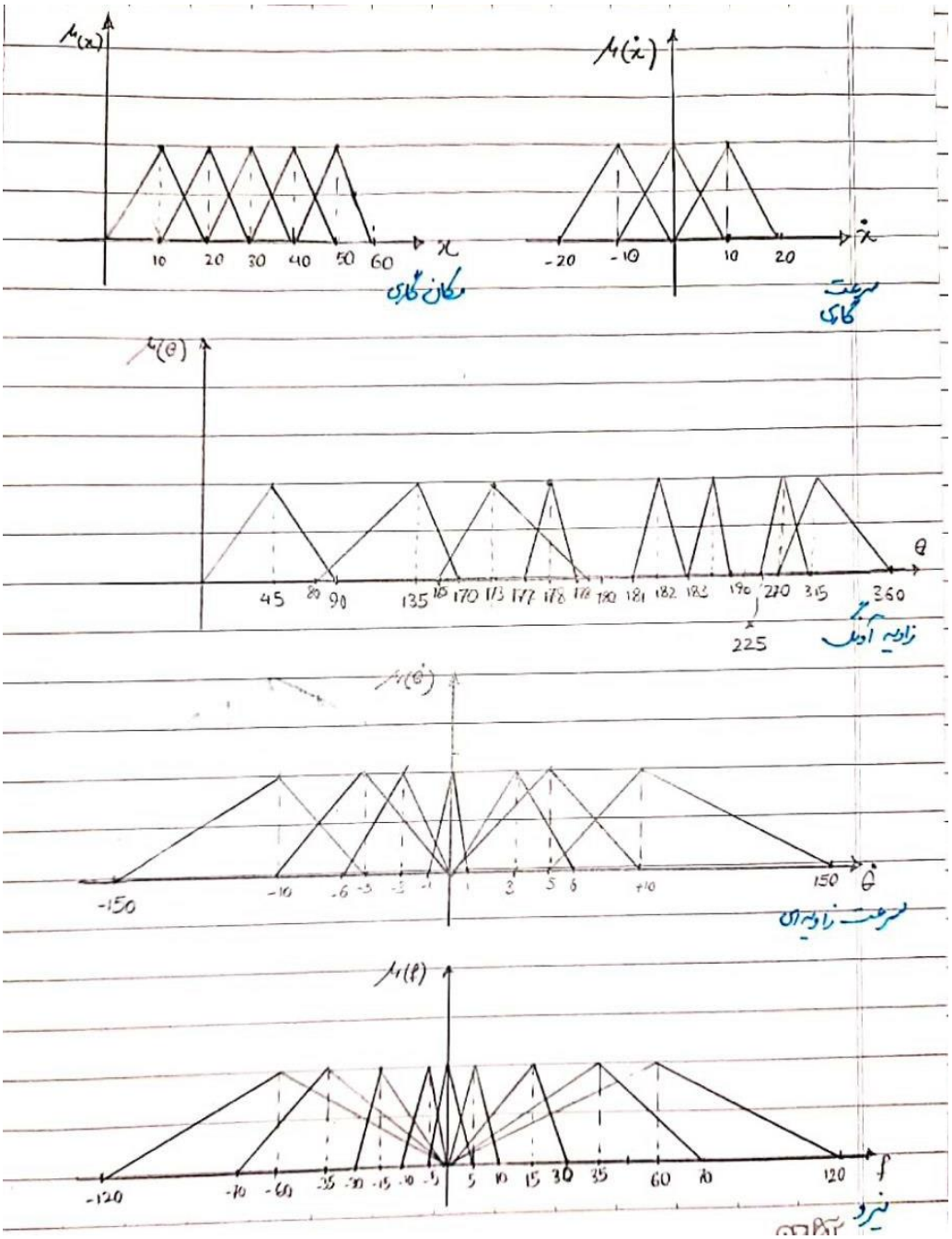
$A' \to ?$ $\quad \mu_{B'} = \mu_A(2) \cdot \mu_B = 0.5\,\mu_B = \frac{\mu_B}{2}$

$$\Rightarrow \mu_{B'} = \left\{ \left(-4, \frac{0.5}{2}\right) \left(-3, \frac{1}{2}\right) \left(-2, \frac{0.67}{2}\right) \left(-1, \frac{0.33}{2}\right) \right\}$$

$$= \left\{ (-4, 0.25)(-3, 0.5)(-2, 0.335)(-1, 0.165) \right\}$$

3.2 در این سوال باید برای همه ورودی ها و خروجی ترم های فازی تعریف کنیم و سپس قوانینی برای حل مسئله تعریف کنیم.

نمودار membership function ورودی ها و خروجی مطابق تصویر زیر است:



این ترم ها را در بلاک های

FUZZIFY

…

END_ FUZZIFY

تعریف می کنیم.

سپس قوانینی که بدست آورده ایم را در بلاک

RULEBLOCK

…

END_ RULEBLOCK

تعریف می کنیم.

(ربع های این دایره یکی از ربع های اصلی عقبتر است.)

قانون 1: اگر زاویه در ربع 4 دایره و زیاد باشد و سرعت زاویه ای در خلاف جهت عقربه های ساعت و کم باشد، نیروی بسیار زیاد به سمت چپ وارد شود.

قانون 2: اگر زاویه در ربع 1 دایره و کم باشد و سرعت زاویه ای در جهت عقربه های ساعت و کم باشد، نیروی بسیار زیاد به سمت راست وارد شود.

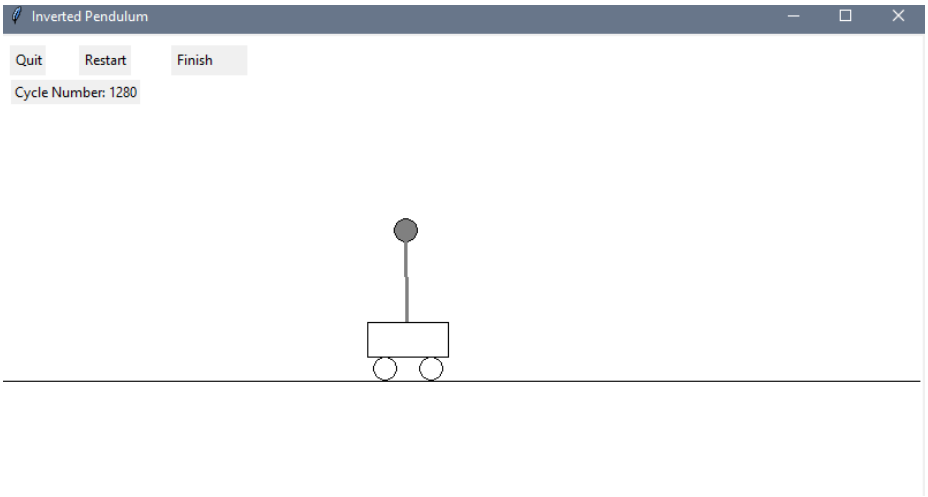
قانون 3: اگر زاویه در ربع 3 دایره و کم باشد و سرعت زاویه ای در خلاف جهت عقربه های ساعت و کم باشد، نیروی بسیار زیاد به سمت چپ وارد شود.

قانون 4: اگر زاویه در ربع 2 دایره و زیاد باشد و سرعت زاویه ای در جهت عقربه های ساعت باشد، نیروی بسیار زیاد به سمت راست وارد شود.

قانون 5: اگر زاویه در ربع 3 دایره و بسیار به 180 درجه نزدیک باشد و سرعت زاویه ای در خلاف جهت عقربه های ساعت باشد، نیروی کم به سمت چپ وارد شود.

قانون 6: اگر زاویه در ربع 2 دایره و بسیار به 180 درجه نزدیک باشد و سرعت زاویه ای در جهت عقربه های ساعت باشد، نیروی کم به سمت راست وارد شود.

نتیجه را در تصویر زیر مشاهده می کنیم:


Inverted Pendulum
Quit
Restart
Finish
Cycle Number: 1280

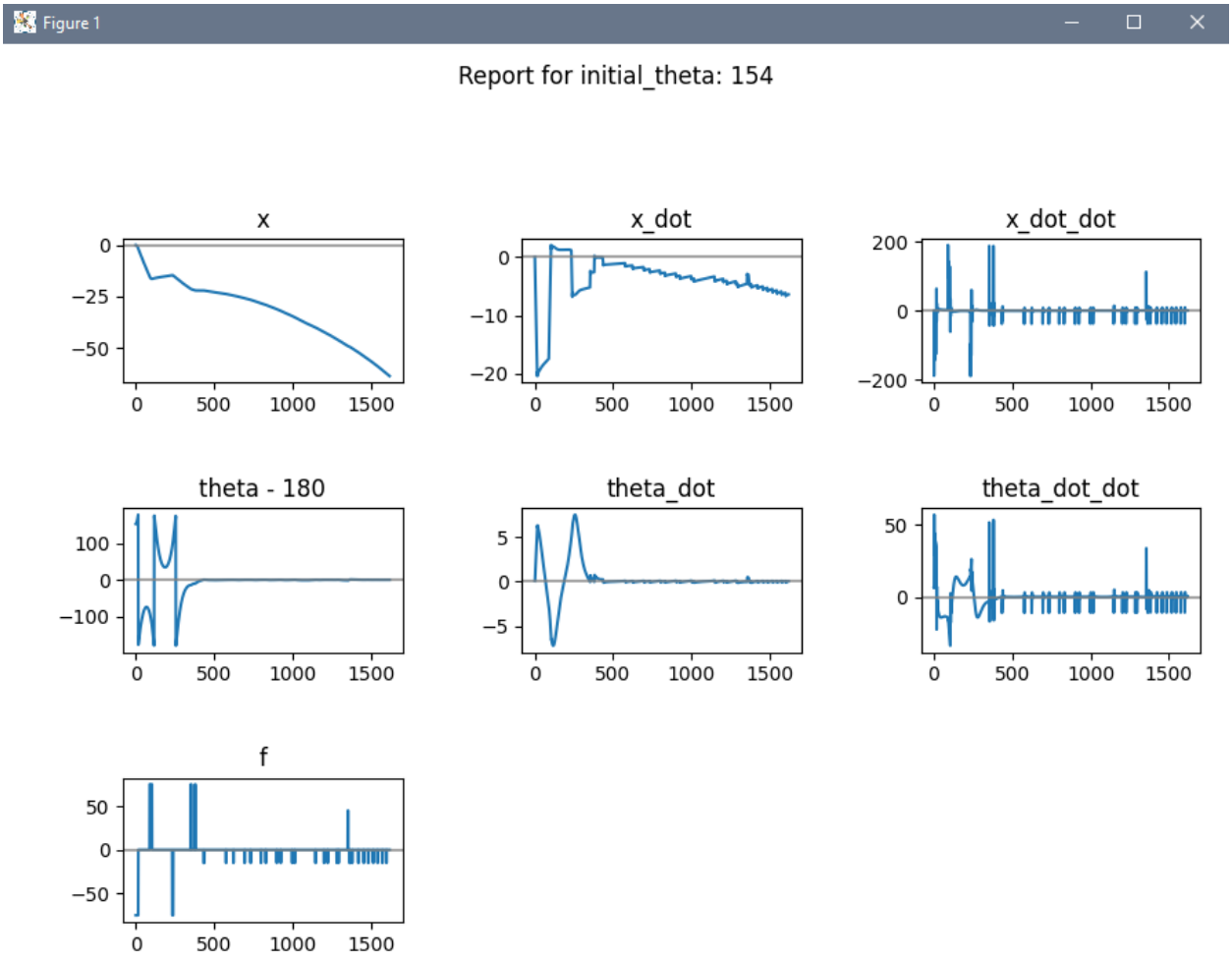


Figure 1
Report for initial_theta: 154
x
x_dot
x_dot_dot
theta - 180
theta_dot
theta_dot_dot
f