**Universidad de Chile**
**Facultad de Ciencias Físicas y Matemáticas**
**Departamento de Ingeniería Eléctrica**
**EL7038 – Introducción a la Teoría de Conjuntos Difusos y Sistemas Inteligentes**

**Proyecto 1**

# Controlador por Lógica Difusa

**Autor:** Mario Vicuña
**Profesor:** Claudio Held
**Profesor Auxiliar:** Gabriel Orellana
**Ayudante:** Diego Jimenez
**Fecha de Entrega:** 08/05/2018

# Abstract

El presente trabajo se centra en la modelación de un sistema de control de presión mediante un CLD y un *setting* dado (entiéndase por esto un set de reglas difusas, diversos métodos de desdifusión, las ecuaciones que definen el modelo y las entradas y salidas al controlador por lógica difusa, además de condiciones iniciales y deseadas).

El set de reglas se resume en un mapa, con el cual se pueden simular los primeros modelos para generar un entendimiento inicial de la planta, y en forma particular haciéndose un análisis de los comportamientos de los controladores sustentados con distintos métodos de desdifusión al aplicarse a un mismo problema.

Tras esto, se haya una nueva interpretación de los valores obtenidos durante el control, obtenible a partir del plano de entradas a través del tiempo. Se encuentran relaciones entre la topología de este con la forma propia de la respuesta del sistema de control previamente discutida y se verifica la consistencia de ambas herramientas.

Se hace un breve análisis de robustez enfocado principalmente en hallar aquellas reglas que permitan inicializar el proceso de control. En base a esto se repiten las pruebas para verificar qué tan fundamentales estas son para la ya mencionada inicialización del sistema de control. Finalmente se verifica que las reglas 1 y 11 juegan un rol fundamental (al menos al inicio del proceso), sin las cuales el CLD es incapaz de realizar control alguno.

# Índice General

# 1. Descripción

El presente trabajo busca ser una introducción más profunda en el área de los controladores por lógica difusa (CLD), tanto en su concepto, modelamiento, comprensión, interpretación y manejo.

Para esto se proponen una serie de estudios. Los primeros de simulación para tener una base de trabajo sobre la cual seguir desarrollando conocimiento; visualicación de información y datos con el fin de definir relaciones entre los distintos métodos y adquirir técnicas que permiten lograr una mejor consistencia y/o detección de fallos a la hora de diseñar un CLD.

Se trata también el tema de robustez frente a cambios en el mapa de reglas difusas, a modo de aterrizar el conocimiento y reforzar la noción de que la lógica difusa permite una abstracción de variables matemáticas a un entorno más lingüístico y de razocinio humano y que, no en desmedro de esto, posee sentido lógico y sustancia tal que se presta como una herramienta útil.

## 2. Mapa de Reglas Difusas

Los conjuntos elementales con los que se trabajará en este experimento están definidos mediante su notación abreviada para su soporte y núcleo. Estos representan los conceptos difusos de Negativo grande, Negativo medio, Negativo pequeño, Negativo ínfimo, Cero, Positivo ínfimo, Positivo pequeño, Positivo medio y Positivo grande, respectivamente. En consecuencia, los conjuntos son, escritos en su forma normalizada en el rango $[-1, 1]$:

$$Ng = (-1.0;\ -1.0;\ -0.7;\ -0.4)$$

$$Nm = (-0.7;\ -0.5;\ -0.4;\ -0.2)$$

$$Np = (-0.4;\ -0.3;\ -0.2;\ -0.1)$$

$$Ni = (-0.2;\ -0.1;\ 0.0;\ 0.0)$$

$$Ce = (-0.1;\ -0.0;\ 0.0;\ 0.1)$$

$$Pi = (0.0;\ 0.0;\ 0.1; 0.2)$$

$$Pp = (0.1;\ 0.2;\ 0.3;\ 0.4)$$

$$Pm = (0.2;\ 0.4;\ 0.5;\ 0.7)$$

$$Pg = (0.4; 0.7; 1.0; 1.0)$$

Sin embargo, el rango de definición para los conjuntos difusos es expandido a $[-15, 15]$. Luego, las funciones de membresía para los 9 conjuntos construidas sobre el intervalo $[-15, 15]$ se muestran en la figura 1:

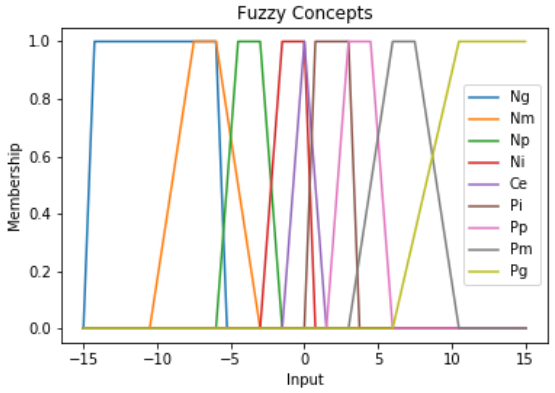



*Figura 1: Funciones de pertenencia para los conceptos difusos Ng, Nm, Np, Ni, CE, Pi, Pp, Pm y PG.*

Dentro del mapa de reglas difusas se incluyen "conceptos" definidos por (de_a c1 c2), lo cual representa la unión de los conjuntos definidos para los conceptos anteriormente descritos, los cuales son:

$$(de\_a\ Ng\ Pp) \equiv Ng \cup Pp$$

$$(de\_a\ Ng\ Nm) \equiv Ng \cup Nm$$

$$(de\_a\ Ng\ Np) \equiv Ng \cup Np$$

$$(de\_a\ Np\ Pi) \equiv Np \cup Pi$$

$$(de\_a\ Ni\ Pi) \equiv Ni \cup Pi$$

$$(de\_a\ Pm\ Pg) \equiv Pm \cup Pg$$

$$(de\_a\ Ni\ Pp) \equiv Ni \cup Pp$$

$$(de\_a\ Pp\ Pg) \equiv Pp \cup Pg$$

$$(de\_a\ Np\ Pg) \equiv Np \cup Pg$$

En la figura 2 se aprecian los resultados de la composición de conjuntos descrita anteriormente para los nueve casos a aplicar en la implementación de las reglas difusas que se definirán más adelante.

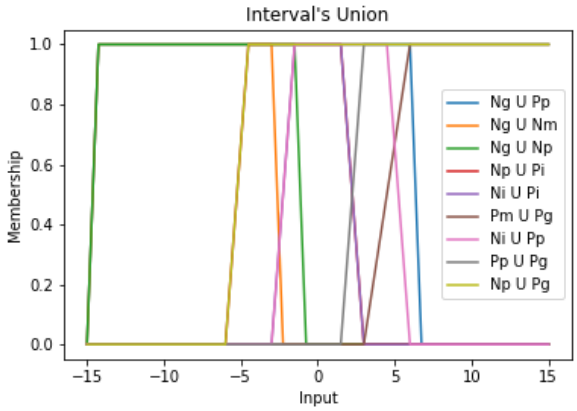



*Figura 2: Funciones de pertenencia para las uniones de conceptos definidas.*

Toda la información anterior se resume en la tabla 1, la cual consiste en el mapa de reglas difusas definido para el controlador a implementar.

*Tabla 1: Mapa de reglas difusas.*

| | | **Entrada 2: TP** | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | Ng U Pp | Ng U Np | Np U Pi | Ng U Nm | Pm U Pg | Ce | Ni U Pp | Pp U Pg | Np U Pg | Pp | Np |
| **Entrada 1: EP** | Ng | Pg | | | | | | | | | | |
| | Ng U Nm | | Pm | | | | | | | | | |
| | Np | | | Pm | | | | | | | | |
| | Ni | | | | Pm | Np | | | | | Ce | Pp |
| | Ni U Pi | | | | | | Ce | | | | | |
| | Pi | | | | Pp | Nm | | | | | Np | Ce |
| | Pp | | | | | | | Nm | | | | |
| | Pm U Pg | | | | | | | | Nm | | | |
| | Pg | | | | | | | | | Ng | | |
| | Ng U Np | | | | | Pg | | | | | | |
| | Pp U Pg) | | | | Ng | | | | | | | |

# 3. Resultados Preliminares

Una vez programadas las distintas funciones a utilizar por el controlador (en particular las referentes a los métodos de desdifusión) se procede a realizar las simulaciones preliminares, con el fin de mostrar el comportamiento del controlador frente a las distintas condiciones iniciales, considerando los siguientes 3 métodos de desdifusión: *Mean of Maximum* (*MoM), Center of Gravity* (*CoG) y Center of Area* (*CoA)* o de alturas.

De esta forma en las figuras 3, 4 y 5, se muestran las respuestas pertenecientes a los 3 controladores implementados (uno por cada método), frente a las condiciones iniciales de presión de 500, 550 y 600, respectivamente.

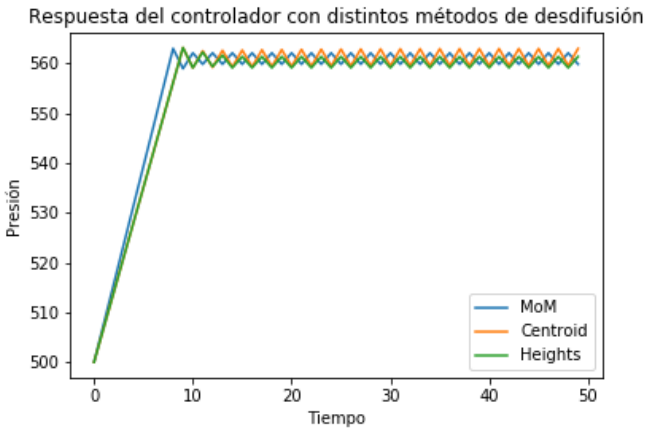



*Figura 3: Respuesta de los controladores para una presión inicial de 500.*

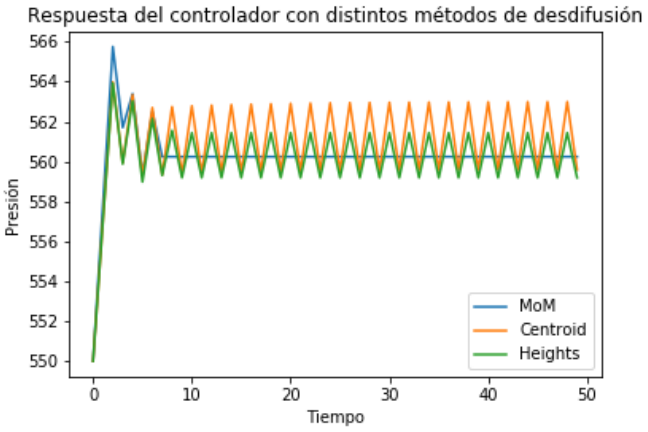



*Figura 4: Respuesta de los controladores para una presión inicial de 550.*

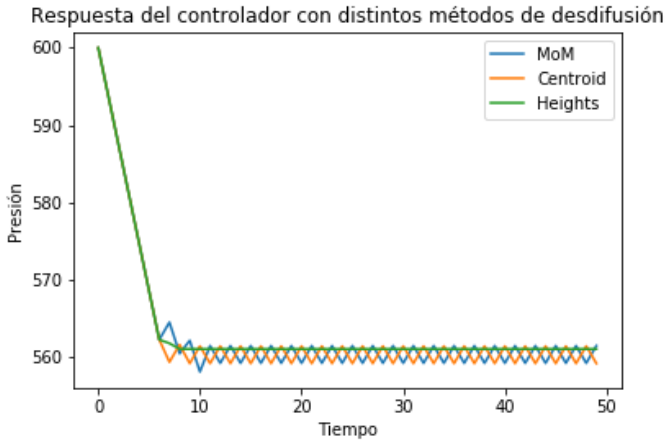



*Figura 5: Respuesta de los controladores para una presión inicial de 600*

De los gráficos anteriores, se aprecia que, frente a distintas condiciones iniciales, cada controlador por sí solo no es consistente, en el sentido de que no exhibe un mismo comportamiento sustancial para los distintos casos, más allá de converger y mantener la respuesta en un entorno del punto de operación deseado. Esto hace que un controlador con cierto método defusificador no sea el más adecuado para todos los puntos iniciales.

Por ejemplo, para el caso de $P_i = 500$ se hace evidente que la mejor opción es el controlador implementado con *MoM*, puesto que el tiempo de subida es menor, así como las oscilaciones en torno al punto de operación. Se puede considerar que el segundo lugar en rendimiento lo ocupa el método de las alturas, pues su comportamiento es similar al anterior, pero con un tiempo de subida levemente mayor.

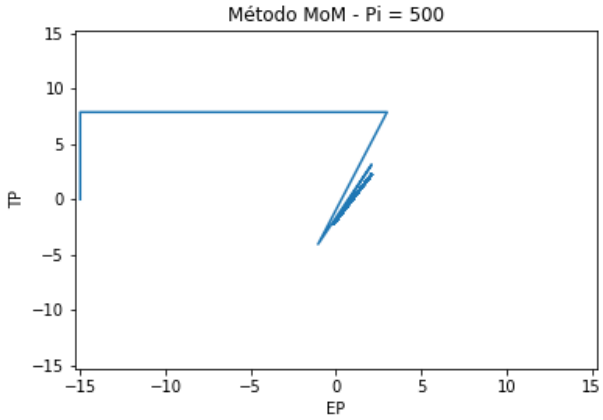
Para el caso en que $P_i = 550$, puede decirse que el orden de desempeño es el mismo, pero por distintas razones. Si bien el método *MoM* presenta el mayor *overshoot* de los tres controladores, este alcanza cero error en estado estacionario, no así los otros dos. Sin embargo, nuevamente se tiene que el controlador con defusificador por centro de área presenta una menor magnitud en su oscilación en torno a la presión objetivo.

Para el tercer caso ($P_i = 600$), el único controlador que logra estabilizarse es el que ocupa el método de las alturas, sin embargo, el error en estado estacionario no es cero. Por otro lado, pese a tener un tiempo de "subida" mayor, el controlador con *MoM* presenta un mayor *overshoot* que aquel implementado con el método del centroide, aunque la magnitud de la oscilación una vez pasado el transciente es la misma.

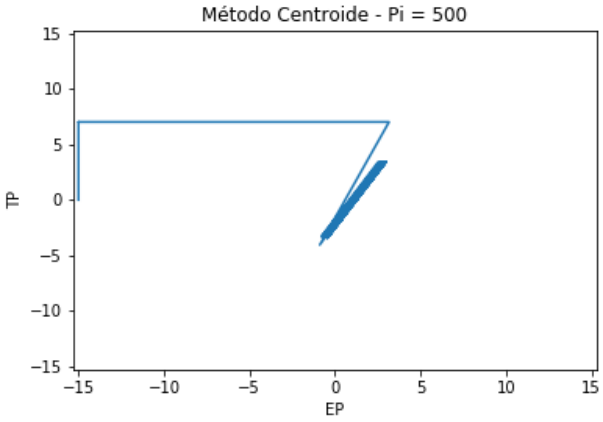
# 4. Plano de Entradas

Es posible monitorear la convergencia de la respuesta del controlador a partir de su plano de entradas. En forma particular, es posible analizar el encaje de cada regla activada y su relación con dicha convergencia para analizar el efecto de la regla sobre el sistema y su importancia respecto al resto de las reglas del mapa; así como también su concordancia con los resultados obtenidos en la sección anterior.

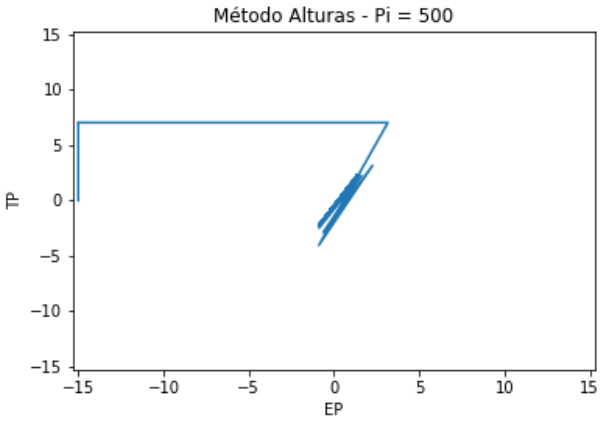
En la figura 6 se muestran los planos de entradas para los tres controladores bajo una condición inicial de 500.




*a)*




*b)*

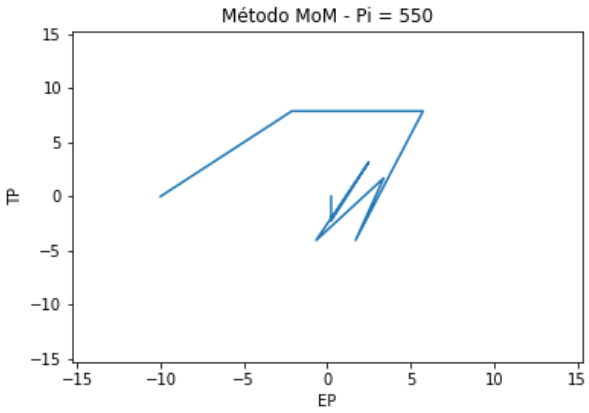



*c)*

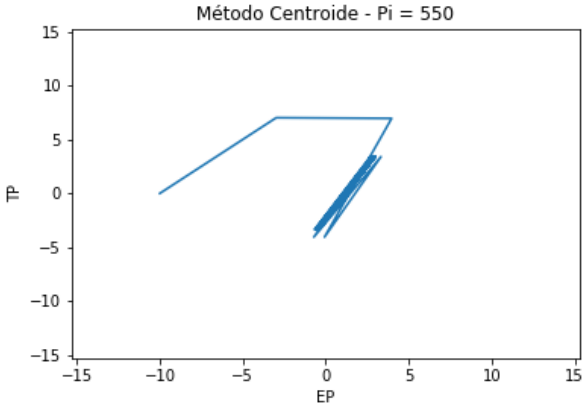
*Figura 6: Planos de entradas para los distintos controladores considerando una presión inicial de 500: a) MoM, b) Centroide y c) Alturas.*

Como se aprecia en la figura 6, el parecido entre los planos de entradas se condice con el gran parecido discutido anteriormente entre las respuestas de los tres controladores para este caso. Además, se distingue la convergencia más rápida para el caso *MoM*, debido a que en su plano de entradas de la figura 6a) posee una menor varianza en torno al cero.

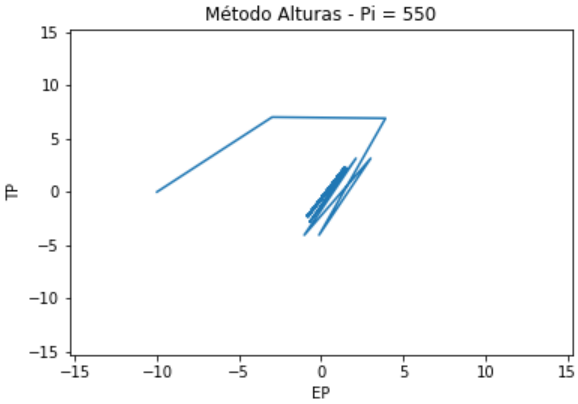
La figura 7 equivale a la 6, con la diferencia de que en esta la condición inicial para la presión de es 550. Para la figura 8 es análogo, tomando como presión inicial un valor de 600.




*d)*




*e)*

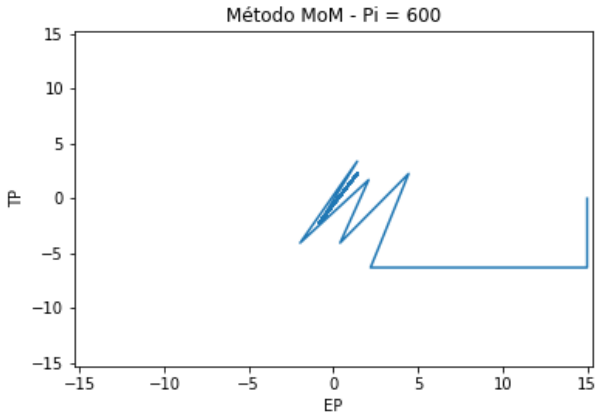



*f)*

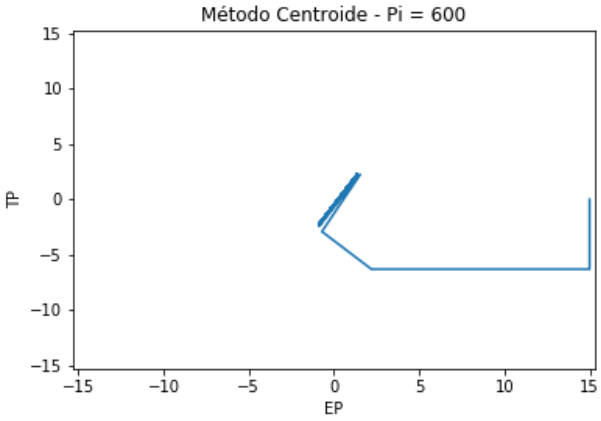
*Figura 7: Planos de entradas para los distintos controladores considerando una presión inicial de 550: a) MoM, b) Centroide y c) Alturas.*

Si bien a grandes rasgos lo mencionado respecto a la figura 6 también se satisface para la figura 7, cabe hacer las aclaraciones pertinentes que vinculan lo observado en el plano de entradas con la salida del sistema. En la figura 7a) se aprecia que, una vez alcanzado cierto grado de convergencia, la varianza de la curva descrita en el plano es mayor que las que se aprecian en los gráficos 7b) y 7c), pero alcanza el origen del plano en un número reducido de iteraciones. La interpretación anterior se condice con las etapas transciente y estacionaria vistas en la figura 4: la mayor varianza en las primeras iteraciones causa el a su vez mayor *overshoot*; y el que se alcance cero error en estado estacionario se corresponde con la baja cantidad de cambios en los valores de las entradas, pues estas convergen rápidamente al origen del plano.

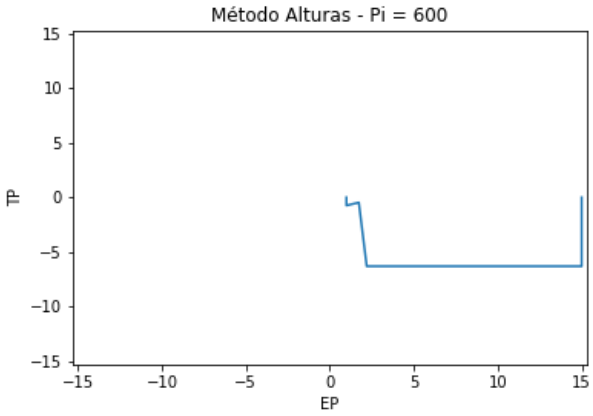
Similar a lo anterior, se aprecia que el plano de entradas para el controlador diseñado con método de las alturas posee una mayor varianza en las cercanías del origen, de ahí que la magnitud de la parte oscilatoria de la respuesta del controlador sea mayor en este diseño que en el implementado con centro de gravedad como método de defusificación.




*g)*




*h)*




*i)*

*Figura 8: Planos de entradas para los distintos controladores considerando una presión inicial de 600: a) MoM, b) Centroide y c) Alturas.*

El aspecto más destacable de la figura 8 está en la figura 8c), pues se describe una trayectoria corta, compuesta por una reducida cantidad de puntos o estados, y dicha curva no se mueve en torno al origen, si no que prácticamente va directo a ubicarse en una vecindad pequeña de dicho punto (pero sin alcanzarlo). Lo anterior va muy de la mano con la suavidad de la respuesta del controlador con defusificación por centro de área, con ausencia de *overshoot*, y que no alcanza cero error en estado estacionario, debido justamente a que no se alcanza el origen del plano de entradas.

Cabe destacar que justamente la curva en el plano de entradas que presenta mayor varianza es justamente la que corresponde con la del controlador con defusificación *MoM*, debido al *overhsoot* y la oscilación remanente del sistema.

# 5. Remoción de Reglas y Robustez

Dentro del *script* utilizado, se hayan dos bloques utilizados para mostrar la variable denotada por *rule_map*, la cual contiene las activaciones de todas las reglas antes de ser procesadas mediante agregación y defusificación.

En la tabla 2 se muestran las activaciones dominantes de las reglas para las 3 primeras iteraciones, según el controlador y la condición inicial:

*Tabla 2: Principales reglas gatilladas en las tres primeras iteraciones, según condición inicial y método de defusificación delcontrolador.*

| | **Controlador** | | | |
|---|---|---|---|---|
| **Condición Inicial** | | ***MoM*** | **Centroide** | **Alturas** |
| | $\boldsymbol{P_i = 500}$ | i) $R_1$<br>ii) $R_{16}$<br>iii) $R_{16}$ | i) $R_1$<br>ii) $R_{16}$<br>iii) $R_{16}$ | i) $R_1$<br>ii) $R_{16}$<br>iii) $R_{16}$ |
| | $\boldsymbol{P_i = 550}$ | i) $R_1$<br>ii) $R_{16}$<br>iii) $R_{10}$ | i) $R_1$<br>ii) $R_{16}$<br>iii) $R_{11}$ | i) $R_1$<br>ii) $R_{16}$<br>iii) $R_{10}$ |
| | $\boldsymbol{P_i = 600}$ | i) $R_{11}$<br>ii) $R_{17}$<br>iii) $R_{17}$ | i) $R_{11}$<br>ii) $R_{17}$<br>iii) $R_{17}$ | i) $R_{11}$<br>ii) $R_{17}$<br>iii) $R_{17}$ |

De la tabla anterior, se observa claramente la tendencia a iniciar el proceso de control con la regla 1 si $P_i < 600$ y la regla 11 en caso contrario. Si bien también se aprecia una marcada tendencia respecto a la segunda y tercera regla que se activan, lo primordial es verificar qué ocurre en caso de eliminarse las reglas que "inicializan" el control del sistema.

Para esto, se repetirán los primeros experimentos, pero, según corresponda, eliminando la regla 1 u 11 del mapa de reglas difusas.

De esta forma se obtienen los gráficos de las figuras 9, 10 y 11, que corresponden a la respuesta del sistema de control para las condiciones iniciales $P_i = 500, P_i = 550$ y $P_i = 600$, respectivamente; considerando la remoción respectiva de la regla inicial.

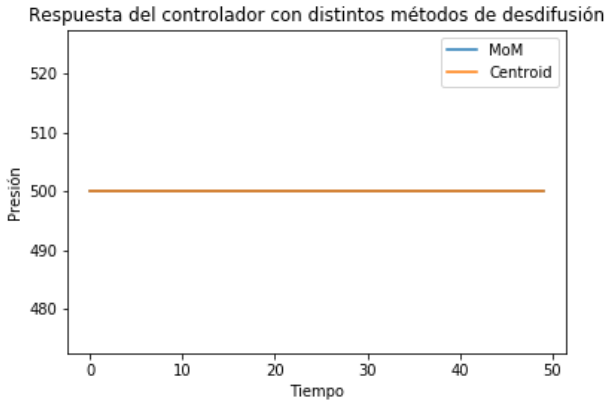


*Figura 9: Respuesta de los controladores para una presión inicial de 500, considerando la remoción de la regla 1.*

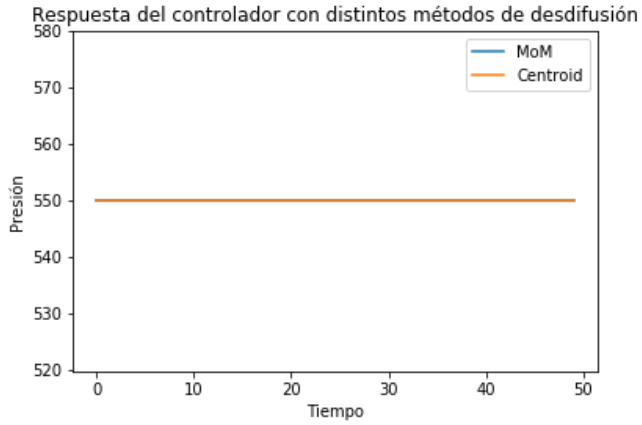



*Figura 10: Respuesta de los controladores para una presión inicial de 550, considerando la remoción de la regla 1.*

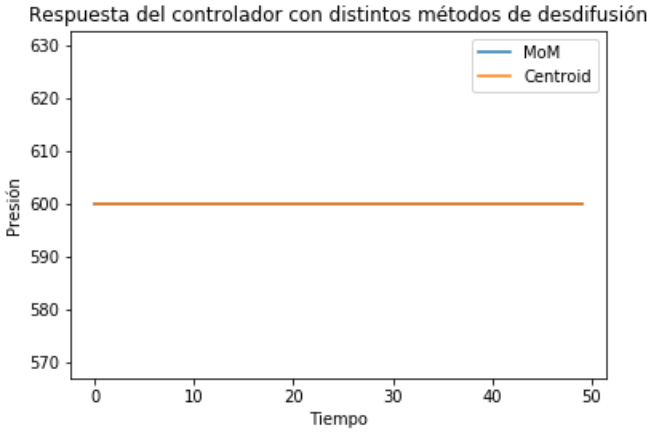



*Figura 11: Respuesta de los controladores para una presión inicial de 600, considerando la remoción de la regla 11.*

Como cabría de esperar, al remover la regla que permite comenzar el proceso de control en los casos estudiados, el sistema se vuelve incapaz de controlar su presión, manteniéndola estática siempre que no exista otras perturbaciones que alteren dicho valor.

Cabe destacar que en los gráficos anteriores no aparecen los controladores con desdifusión por método de las alturas. Esto es así puesto que sin la regla 11, el programa no era capaz de ejecutarse sin aplicar manejo de excepciones (como divisiones por cero) cosa que escapa a lo pertinente al propio control, por lo que se opta por dejar dicho controlador fuera de discusión, al menos más allá de verificar la importancia de la regla de inicialización eliminada en el proceso como tal.

# 6. Conclusiones

Se logra modelar y manejar sistemas de control por lógica difusa, interpretar los resultados obtenidos por estos por una diversa gama de herramientas visuales, como lo son la propia respuesta de la planta como función del tiempo así como los planos de entradas.

Se analiza la relación entre el mapa de reglas difusas, su gatillamiento y su efecto a través del tiempo en el fenómeno del control difuso.

En términos generales se cumplieron los objetivos más allá de los percances computacionales y de librerías enfrentados, pero adquiriendo nuevas herramientas que, al menos parcialmente, ayudaron en el desarrollo del trabajo realizado, ya sea para comprensión tanto de la parte computacional como de los fenómenos a modelar, corroboración e resultados o entendimiento de los contenidos tratados.

# 7. Código Anexo

A continuación, se muestra el código de la implementación final del programa.

Se debe destacar que para el correcto funcionamiento del programa se debe contar con *Python* y las librerías *skfuzzy*, *numpy* y *matplotlib*. Además, se recomienda correr el programa en un *jupyter notebook* por la sencillez de su visualización, en la que además pueden verse todos los gráficos mostrados en el presente informe de manera simultánea.

```
1. # coding: utf-8
2.
3. # In[1]:
4.
5.
6. ## Primera versión con skfuzzy, más adelante se hace todo a mano
   por relativa simplicidad
7.
8. #!/usr/bin/env python2
9. # -*- coding: utf-8 -*-
10.      import numpy as np
11.      import skfuzzy.control as ctrl
12.      import skfuzzy as fuzz
13.      import matplotlib
14.      import matplotlib.pyplot as plt
15.
16.      def de_a(A, B, literal = True):
17.          # Se considera que A y B son conjuntos difusos
   trapezoidales
18.          de_a = []
19.          if literal:
20.              de_a.append(A[0])
21.              de_a.append(A[1])
22.              de_a.append(B[2])
23.              de_a.append(B[3])
24.          else:
25.              for i in range(0,4):
26.                  if i < 2:
27.                      de_a.append(min(A[i],B[i]))
28.                  else:
29.                      de_a.append(max(A[i],B[i]))
30.          return de_a
31.
32.
33.      # In[2]:
34.
35.
36.      EP = ctrl.Antecedent(np.arange(-15, 15.05, 0.05), 'EP')
37.      TP = ctrl.Antecedent(np.arange(-15, 15.05, 0.05), 'TP')
38.      dH = ctrl.Consequent(np.arange(-15, 15.05, 0.05), 'dH')
39.
40.      names = ['Ng', 'Nm', 'Np', 'Ni', 'Ce', 'Pi', 'Pp', 'Pm', 'Pg'
   ]
41.
42.      Ng = (np.asarray([-1.0, -1.0, -0.7, -0.4]) * 15).tolist()
43.      Nm = (np.asarray([-0.7, -0.5, -0.4, -0.2]) * 15).tolist()
```

```
44.      Np = (np.asarray([-0.4, -0.3, -0.2, -0.1]) * 15).tolist()
45.      Ni = (np.asarray([-0.2, -0.1, 0.0, 0.0]) * 15).tolist()
46.      Ce = (np.asarray([-0.1, -0.0, 0.0, 0.1]) * 15).tolist()
47.      Pi = (np.asarray([0.0, 0.0, 0.1, 0.2]) * 15).tolist()
48.      Pp = (np.asarray([0.1, 0.2, 0.3, 0.4]) * 15).tolist()
49.      Pm = (np.asarray([0.2, 0.4, 0.5, 0.7]) * 15).tolist()
50.      Pg = (np.asarray([0.4, 0.7, 1.0, 1.0]) * 15).tolist()
51.
52.      NgUPp = de_a(Ng, Pp); names.append('NgUPp')
53.      NgUNm = de_a(Ng, Nm); names.append('NgUNm')
54.      NgUNp = de_a(Ng, Np); names.append('NgUNp')
55.      NpUPi = de_a(Np, Pi); names.append('NpUPi')
56.      NiUPi = de_a(Ni, Pi); names.append('NiUPi')
57.      PmUPg = de_a(Pm, Pg); names.append('PmUPg')
58.      NiUPp = de_a(Ni, Pp); names.append('NiUPp')
59.      PpUPg = de_a(Pp, Pg); names.append('PpUPg')
60.      NpUPg = de_a(Np, Pg); names.append('NpUPg')
61.
62.      EP['Ng'] = fuzz.trapmf(EP.universe, Ng)
63.      EP['Nm'] = fuzz.trapmf(EP.universe, Nm)
64.      EP['Np'] = fuzz.trapmf(EP.universe, Np)
65.      EP['Ni'] = fuzz.trapmf(EP.universe, Ni)
66.      EP['Ce'] = fuzz.trapmf(EP.universe, Ce)
67.      EP['Pi'] = fuzz.trapmf(EP.universe, Pi)
68.      EP['Pp'] = fuzz.trapmf(EP.universe, Pp)
69.      EP['Pm'] = fuzz.trapmf(EP.universe, Pm)
70.      EP['Pg'] = fuzz.trapmf(EP.universe, Pg)
71.
72.      EP.view()
73.
74.      EP['NgUPp'] = fuzz.trapmf(EP.universe, NgUPp)
75.      EP['NgUNm'] = fuzz.trapmf(EP.universe, NgUNm)
76.      EP['NgUNp'] = fuzz.trapmf(EP.universe, NgUNp)
77.      EP['NpUPi'] = fuzz.trapmf(EP.universe, NpUPi)
78.      EP['NiUPi'] = fuzz.trapmf(EP.universe, NiUPi)
79.      EP['PmUPg'] = fuzz.trapmf(EP.universe, PmUPg)
80.      EP['NiUPp'] = fuzz.trapmf(EP.universe, NiUPp)
81.      EP['PpUPg'] = fuzz.trapmf(EP.universe, PpUPg)
82.      EP['NpUPg'] = fuzz.trapmf(EP.universe, NpUPg)
83.
84.      EP.view()
85.
86.      TP['Ng'] = fuzz.trapmf(TP.universe, Ng)
87.      TP['Nm'] = fuzz.trapmf(TP.universe, Nm)
88.      TP['Np'] = fuzz.trapmf(TP.universe, Np)
89.      TP['Ni'] = fuzz.trapmf(TP.universe, Ni)
90.      TP['Ce'] = fuzz.trapmf(TP.universe, Ce)
91.      TP['Pi'] = fuzz.trapmf(TP.universe, Pi)
92.      TP['Pp'] = fuzz.trapmf(TP.universe, Pp)
93.      TP['Pm'] = fuzz.trapmf(TP.universe, Pm)
94.      TP['Pg'] = fuzz.trapmf(TP.universe, Pg)
95.
96.      TP.view()
97.
98.      TP['NgUPp'] = fuzz.trapmf(TP.universe, NgUPp)
99.      TP['NgUNm'] = fuzz.trapmf(TP.universe, NgUNm)
100.     TP['NgUNp'] = fuzz.trapmf(TP.universe, NgUNp)
```

```
    TP['NpUPi'] = fuzz.trapmf(TP.universe, NpUPi)
    TP['NiUPi'] = fuzz.trapmf(TP.universe, NiUPi)
    TP['PmUPg'] = fuzz.trapmf(TP.universe, PmUPg)
    TP['NiUPp'] = fuzz.trapmf(TP.universe, NiUPp)
    TP['PpUPg'] = fuzz.trapmf(TP.universe, PpUPg)
    TP['NpUPg'] = fuzz.trapmf(TP.universe, NpUPg)

    TP.view()

    dH['Ng'] = fuzz.trapmf(dH.universe, Ng)
    dH['Nm'] = fuzz.trapmf(dH.universe, Nm)
    dH['Np'] = fuzz.trapmf(dH.universe, Np)
    dH['Ni'] = fuzz.trapmf(dH.universe, Ni)
    dH['Ce'] = fuzz.trapmf(dH.universe, Ce)
    dH['Pi'] = fuzz.trapmf(dH.universe, Pi)
    dH['Pp'] = fuzz.trapmf(dH.universe, Pp)
    dH['Pm'] = fuzz.trapmf(dH.universe, Pm)
    dH['Pg'] = fuzz.trapmf(dH.universe, Pg)

    dH.view()

    dH['NgUPp'] = fuzz.trapmf(dH.universe, NgUPp)
    dH['NgUNm'] = fuzz.trapmf(dH.universe, NgUNm)
    dH['NgUNp'] = fuzz.trapmf(dH.universe, NgUNp)
    dH['NpUPi'] = fuzz.trapmf(dH.universe, NpUPi)
    dH['NiUPi'] = fuzz.trapmf(dH.universe, NiUPi)
    dH['PmUPg'] = fuzz.trapmf(dH.universe, PmUPg)
    dH['NiUPp'] = fuzz.trapmf(dH.universe, NiUPp)
    dH['PpUPg'] = fuzz.trapmf(dH.universe, PpUPg)
    dH['NpUPg'] = fuzz.trapmf(dH.universe, NpUPg)

    dH.view()


    rule1 = ctrl.Rule(antecedent = ((EP['Pg'] & TP['NpUPg']) |
                                    (EP['PpUPg'] & TP['NgUNm'])),
                      consequent = dH['Ng'], label = 'rule Ng')

    rule2 = ctrl.Rule(antecedent = ((EP['Pi'] & TP['PmUPg']) |
                                    (EP['Pp'] & TP['NiUPp']) |
                                    (EP['PmUPg'] & TP['PpUPg'])),
                      consequent = dH['Nm'], label = 'rule Nm')

    rule3 = ctrl.Rule(antecedent = ((EP['Ni'] & TP['PmUPg']) |
                                    (EP['Pi'] & TP['Pp'])),
                      consequent = dH['Np'], label = 'rule Np')

    rule5 = ctrl.Rule(antecedent = ((EP['NiUPi'] & TP['Ce']) |
                                    (EP['Ni'] & TP['Pp']) |
                                    (EP['Pi'] & TP['Np'])),
                      consequent = dH['Ce'], label = 'rule Ce')

    rule7 = ctrl.Rule(antecedent = ((EP['Pi'] & TP['NgUNm']) |
                                    (EP['Ni'] & TP['Np'])),
                      consequent = dH['Pp'], label = 'rule Pp')

    rule8 = ctrl.Rule(antecedent = ((EP['NgUNm'] & TP['NgUNp']) |
```

```
                                        (EP['Np'] & TP['NpUPi']) |
                                        (EP['Ni'] & TP['NgUNm'])),
                        consequent = dH['Pm'], label = 'rule Pm')

    rule9 = ctrl.Rule(antecedent = ((EP['Ng'] & TP['NgUPp']) |
                                        (EP['NgUNp'] & TP['PmUPg'])),
                        consequent = dH['Pg'], label = 'rule Pg')

    rules = [rule1, rule2, rule3, rule5, rule7, rule8, rule9]

    PO = 560
    K = 0.6
    initialConditions = [500, 550, 600]
    defuzzMethods = ['mom', 'centroid', 'heights']

    Pmom = []
    Pcentroid = []
    Pheights = []


    # In[3]:


    ## MoM method for Pi = 500

    dH.defuzzify_method = defuzzMethods[0]

    system = ctrl.ControlSystem(rules)
    sim = ctrl.ControlSystemSimulation(system)

    Pi = initialConditions[0]
    P = np.zeros(50)
    P[0] = Pi
    DH = np.zeros(50)


    for i in range(1,50):

        EPt = (P[i - 1] - PO)
        if EPt > 15:
            EPt = 15
        elif EPt < -15:
            EPt = -15

        sim.input['EP'] = EPt

        if i == 1:
            sim.input['TP'] = 0
        else:
            TPt = P[i - 1] - P[i - 2]
            if TPt > 15:
                TPt = 15
            elif TPt < -15:
                TPt = -15
            sim.input['TP'] = TPt

        sim.compute()
```

```
215.
216.         dHt = int(sim.output['dH'])
217.         if dHt > 15:
218.             dHt = 15
219.         elif dHt < -15:
220.             dHt = -15
221.         DH[i] = dHt
222.         dP = dHt * K
223.         P[i] = P[i - 1] + dP
224.
225.     Pmom.append(P)
226.     plt.figure()
227.     plt.plot(P,'-*')
228.     plt.show()
229.
230.
231.     # In[4]:
232.
233.
234.     ## MoM method for Pi = 550
235.
236.     dH.defuzzify_method = defuzzMethods[0]
237.
238.     system = ctrl.ControlSystem(rules)
239.     sim = ctrl.ControlSystemSimulation(system)
240.
241.     Pi = initialConditions[1]
242.     P = np.zeros(50)
243.     P[0] = Pi
244.     DH = np.zeros(50)
245.
246.
247.     for i in range(1,50):
248.
249.         EPt = (P[i - 1] - PO)
250.         if EPt > 15:
251.             EPt = 15
252.         elif EPt < -15:
253.             EPt = -15
254.
255.         sim.input['EP'] = EPt
256.
257.         if i == 1:
258.             sim.input['TP'] = 0
259.         else:
260.             TPt = P[i - 1] - P[i - 2]
261.             if TPt > 15:
262.                 TPt = 15
263.             elif TPt < -15:
264.                 TPt = -15
265.             sim.input['TP'] = TPt
266.
267.         sim.compute()
268.
269.         dHt = int(sim.output['dH'])
270.         if dHt > 15:
271.             dHt = 15
```

```
        elif dHt < -15:
            dHt = -15
        DH[i] = dHt
        dP = dHt * K
        P[i] = P[i - 1] + dP

    Pmom.append(P)
    plt.figure()
    plt.plot(P,'-*')
    plt.show()


    # In[5]:


    ## MoM method for Pi = 600

    dH.defuzzify_method = defuzzMethods[0]

    system = ctrl.ControlSystem(rules)
    sim = ctrl.ControlSystemSimulation(system)

    Pi = initialConditions[2]
    P = np.zeros(50)
    P[0] = Pi
    DH = np.zeros(50)


    for i in range(1,50):

        EPt = (P[i - 1] - PO)
        if EPt > 15:
            EPt = 15
        elif EPt < -15:
            EPt = -15

        sim.input['EP'] = EPt

        if i == 1:
            sim.input['TP'] = 0
        else:
            TPt = P[i - 1] - P[i - 2]
            if TPt > 15:
                TPt = 15
            elif TPt < -15:
                TPt = -15
            sim.input['TP'] = TPt

        sim.compute()

        dHt = int(sim.output['dH'])
        if dHt > 15:
            dHt = 15
        elif dHt < -15:
            dHt = -15
        DH[i] = dHt
        dP = dHt * K
```

```
            P[i] = P[i - 1] + dP

    Pmom.append(P)
    plt.figure()
    plt.plot(P,'-*')
    plt.show()


    # In[6]:


    ## Centroid method for Pi = 500

    dH.defuzzify_method = defuzzMethods[1]

    system = ctrl.ControlSystem(rules)
    sim = ctrl.ControlSystemSimulation(system)

    Pi = initialConditions[0]
    P = np.zeros(50)
    P[0] = Pi
    DH = np.zeros(50)


    for i in range(1,50):

        EPt = (P[i - 1] - PO)
        if EPt > 15:
            EPt = 15
        elif EPt < -15:
            EPt = -15

        sim.input['EP'] = EPt

        if i == 1:
            sim.input['TP'] = 0
        else:
            TPt = P[i - 1] - P[i - 2]
            if TPt > 15:
                TPt = 15
            elif TPt < -15:
                TPt = -15
            sim.input['TP'] = TPt

        sim.compute()

        dHt = int(sim.output['dH'])
        if dHt > 15:
            dHt = 15
        elif dHt < -15:
            dHt = -15
        DH[i] = dHt
        dP = dHt * K
        P[i] = P[i - 1] + dP

    Pcentroid.append(P)
    plt.figure()
```

```
    plt.plot(P,'-*')
    plt.show()


    # In[7]:


    ## Centroid method for Pi = 550

    dH.defuzzify_method = defuzzMethods[1]

    system = ctrl.ControlSystem(rules)
    sim = ctrl.ControlSystemSimulation(system)

    Pi = initialConditions[1]
    P = np.zeros(50)
    P[0] = Pi
    DH = np.zeros(50)


    for i in range(1,50):

        EPt = (P[i - 1] - PO)
        if EPt > 15:
            EPt = 15
        elif EPt < -15:
            EPt = -15

        sim.input['EP'] = EPt

        if i == 1:
            sim.input['TP'] = 0
        else:
            TPt = P[i - 1] - P[i - 2]
            if TPt > 15:
                TPt = 15
            elif TPt < -15:
                TPt = -15
            sim.input['TP'] = TPt

        sim.compute()

        dHt = int(sim.output['dH'])
        if dHt > 15:
            dHt = 15
        elif dHt < -15:
            dHt = -15
        DH[i] = dHt
        dP = dHt * K
        P[i] = P[i - 1] + dP

    Pcentroid.append(P)
    plt.figure()
    plt.plot(P,'-*')
    plt.show()
```

```
# In[8]:


## Centroid method for Pi = 600

dH.defuzzify_method = defuzzMethods[1]

system = ctrl.ControlSystem(rules)
sim = ctrl.ControlSystemSimulation(system)

Pi = initialConditions[2]
P = np.zeros(50)
P[0] = Pi
DH = np.zeros(50)


for i in range(1,50):

    EPt = (P[i - 1] - PO)
    if EPt > 15:
        EPt = 15
    elif EPt < -15:
        EPt = -15

    sim.input['EP'] = EPt

    if i == 1:
        sim.input['TP'] = 0
    else:
        TPt = P[i - 1] - P[i - 2]
        if TPt > 15:
            TPt = 15
        elif TPt < -15:
            TPt = -15
        sim.input['TP'] = TPt
    print(EPt, TPt)
    sim.compute()

    dHt = int(sim.output['dH'])
    if dHt > 15:
        dHt = 15
    elif dHt < -15:
        dHt = -15
    print(dHt)
    DH[i] = dHt
    dP = dHt * K
    P[i] = P[i - 1] + dP

Pcentroid.append(P)
plt.figure()
plt.plot(P,'-*')
plt.show()


# In[9]:


```

```
    plt.figure()
    plt.plot(Pmom[0], label = 'MoM')
    plt.plot(Pcentroid[0], label = 'Centroid')
    plt.legend()
    plt.show()


    # In[10]:


    plt.figure()
    plt.plot(Pmom[1], label = 'MoM')
    plt.plot(Pcentroid[1], label = 'Centroid')
    plt.legend()
    plt.show()


    # In[11]:


    plt.figure()
    plt.plot(Pmom[2], label = 'MoM')
    plt.plot(Pcentroid[2], label = 'Centroid')
    plt.legend()
    plt.show()


    # In[12]:


    ## Segunda versión sin skfuzzy. Se procede a redefinir todo para empezar.
    import numpy
    import matplotlib.pyplot as plt
    import pandas as pd
    import os
    import xlsxwriter
    import math

    def de_a(A, B, literal = True):
        # Se considera que A y B son conjuntos difusos trapezoidales
        de_a = []
        if literal:
            de_a.append(A[0])
            de_a.append(A[1])
            de_a.append(B[2])
            de_a.append(B[3])
        else:
            for i in range(0,4):
                if i < 2:
                    de_a.append(min(A[i],B[i]))
                else:
                    de_a.append(max(A[i],B[i]))
        return de_a

    def trapezoid_membership(x, Set):
```

```
555.         assert len(Set) == 4
556.         if -15.0 <= x <= Set[0]:
557.             if -15.0 == Set[0]:
558.                 return 1
559.             return 0
560.         elif Set[0] <= x <= Set[1]:
561.             if Set[0] == Set[1]:
562.                 m = 0
563.             else:
564.                 m = 1/(Set[1] - Set[0])
565.             return m*x - m*Set[0]
566.         elif Set[1] <= x <= Set[2]:
567.             return 1
568.         elif Set[2] <= x <= Set[3]:
569.             if Set[0] == Set[1]:
570.                 m = 0
571.             else:
572.                 m = -1/(Set[3] - Set[2])
573.             return m*x - m*Set[2] + 1
574.         elif Set[3] <= x <= 15.0:
575.             return 0
576.         else:
577.             print("Out of bounds")
578.             return
579.
580.     def trapezoid_membership_vector(E, Set):
581.         output = []
582.         for i in range(0,len(E)):
583.             output.append(trapezoid_membership(E[i],Set))
584.         return output
585.
586.     def fuzzy_rule(E1, mf1, E2, mf2, domain, out_mf):
587.         c1 = trapezoid_membership_vector (E1, mf1)
588.         c2 = trapezoid_membership_vector(E2, mf2)
589.         c1_and_c2 = []
590.         for i in range(0,len(c1)):
591.             c1_and_c2.append(min(c1[i],c2[i]))
592.         out_function = trapezoid_membership_vector(domain, out_mf
    )
593.         output = []
594.         for i in range(0,len(c1_and_c2)):
595.             output.append(numpy.minimum(c1_and_c2[i],out_function
    ))
596.         return numpy.asarray(output)
597.
598.     def fuzzy_rule_antecedent(E1, mf1, E2, mf2):
599.         c1 = trapezoid_membership_vector (E1, mf1)
600.         c2 = trapezoid_membership_vector(E2, mf2)
601.         c1_and_c2 = []
602.         for i in range(0,len(c1)):
603.             c1_and_c2.append(min(c1[i],c2[i]))
604.         return c1_and_c2
605.
606.     def aggregation(rule_map):
607.         output = []
608.         for row in range(0, len(rule_map[0])):
609.             aggregated_row = []
```

```
610.             for column in range(0, len(rule_map[0][0])):
611.                 current_max = -math.inf
612.                 for rule in range(0, len(rule_map)):
613.                     if rule_map[rule][row][column] >= current_max
    :
614.                         current_max = rule_map[rule][row][column]
615.                 aggregated_row.append(current_max)
616.             output.append(aggregated_row)
617.         return output
618.
619.     def COG(interval, function):
620.         output = []
621.         for row in range(0, len(function)):
622.             num = sum((function[row])*interval)
623.             den = sum(function[row])
624.             if num == 0 and den == 0:
625.                 num = 0
626.             else:
627.                 num = num/den
628.             output.append(num)
629.         return output
630.
631.     def MOM(interval, function):
632.         M = max(function[0])
633.         for i in range(0, len(function[0])):
634.             if function[0][i] != M:
635.                 function[0][i] = 0
636.         return COG(interval, function)
637.
638.     def Heights(interval, function):
639.         return function
640.
641.     def defuzz(interval, function, method):
642.         return method(interval, function)
643.
644.
645.     # In[13]:
646.
647.
648.     n_points = 41
649.     Domain = numpy.linspace(-15.0, 15.0, n_points)
650.     #Domain = np.arange(-15, 15, 0.05)
651.
652.     Ng = (numpy.asarray([-1.0, -1.0, -0.7, -0.4]) * 15).tolist()
653.     Nm = (numpy.asarray([-0.7, -0.5, -0.4, -0.2]) * 15).tolist()
654.     Np = (numpy.asarray([-0.4, -0.3, -0.2, -0.1]) * 15).tolist()
655.     Ni = (numpy.asarray([-0.2, -0.1, 0.0, 0.0]) * 15).tolist()
656.     Ce = (numpy.asarray([-0.1, -0.0, 0.0, 0.1]) * 15).tolist()
657.     Pi = (numpy.asarray([0.0, 0.0, 0.1, 0.2]) * 15).tolist()
658.     Pp = (numpy.asarray([0.1, 0.2, 0.3, 0.4]) * 15).tolist()
659.     Pm = (numpy.asarray([0.2, 0.4, 0.5, 0.7]) * 15).tolist()
660.     Pg = (numpy.asarray([0.4, 0.7, 1.0, 1.0]) * 15).tolist()
661.
662.     fNg = trapezoid_membership_vector(Domain, Ng)
663.     fNm = trapezoid_membership_vector(Domain, Nm)
664.     fNp = trapezoid_membership_vector(Domain, Np)
665.     fNi = trapezoid_membership_vector(Domain, Ni)
```

```
    fCe = trapezoid_membership_vector(Domain, Ce)
    fPi = trapezoid_membership_vector(Domain, Pi)
    fPp = trapezoid_membership_vector(Domain, Pp)
    fPm = trapezoid_membership_vector(Domain, Pm)
    fPg = trapezoid_membership_vector(Domain, Pg)

    fig = plt.figure()
    plt.plot(Domain, fNg, label = 'Ng')
    plt.plot(Domain, fNm, label = 'Nm')
    plt.plot(Domain, fNp, label = 'Np')
    plt.plot(Domain, fNi, label = 'Ni')
    plt.plot(Domain, fCe, label = 'Ce')
    plt.plot(Domain, fPi, label = 'Pi')
    plt.plot(Domain, fPp, label = 'Pp')
    plt.plot(Domain, fPm, label = 'Pm')
    plt.plot(Domain, fPg, label = 'Pg')
    plt.xlabel('Input')
    plt.ylabel('Membership')
    plt.title('Fuzzy Concepts')
    plt.legend()
    fig.savefig('Concepts.png')
    plt.show()

    NgUPp = de_a(Ng, Pp)
    NgUNm = de_a(Ng, Nm)
    NgUNp = de_a(Ng, Np)
    NpUPi = de_a(Np, Pi)
    NiUPi = de_a(Ni, Pi)
    PmUPg = de_a(Pm, Pg)
    NiUPp = de_a(Ni, Pp)
    PpUPg = de_a(Pp, Pg)
    NpUPg = de_a(Np, Pg)

    fNgUPp = trapezoid_membership_vector(Domain, NgUPp)
    fNgUNm = trapezoid_membership_vector(Domain, NgUNm)
    fNgUNp = trapezoid_membership_vector(Domain, NgUNp)
    fNpUPi = trapezoid_membership_vector(Domain, NpUPi)
    fNiUPi = trapezoid_membership_vector(Domain, NiUPi)
    fPmUPg = trapezoid_membership_vector(Domain, PmUPg)
    fNiUPp = trapezoid_membership_vector(Domain, NiUPp)
    fPpUPg = trapezoid_membership_vector(Domain, PpUPg)
    fNpUPg = trapezoid_membership_vector(Domain, NpUPg)

    fig = plt.figure()
    plt.plot(Domain, fNgUPp, label = 'Ng U Pp')
    plt.plot(Domain, fNgUNm, label = 'Ng U Nm')
    plt.plot(Domain, fNgUNp, label = 'Ng U Np')
    plt.plot(Domain, fNpUPi, label = 'Np U Pi')
    plt.plot(Domain, fNiUPi, label = 'Ni U Pi')
    plt.plot(Domain, fPmUPg, label = 'Pm U Pg')
    plt.plot(Domain, fNiUPp, label = 'Ni U Pp')
    plt.plot(Domain, fPpUPg, label = 'Pp U Pg')
    plt.plot(Domain, fNpUPg, label = 'Np U Pg')
    plt.xlabel('Input')
    plt.ylabel('Membership')
    plt.title('Interval\'s Union')
    plt.legend()
```

```
723.     fig.savefig('Unions.png')
724.     plt.show()
725.
726.
727.     # In[14]:
728.
729.
730.     PO = 560
731.     K = 0.6
732.     timeHorizon = 50
733.     initialConditions = [500.0, 550.0, 600.0]
734.     defuzzMethods = [MOM, COG, Heights]
735.     Pmom = []
736.     Pcentroid = []
737.     Pheights = []
738.     EPmom = []
739.     TPmom = []
740.     EPcentroid = []
741.     TPcentroid = []
742.     EPheights = []
743.     TPheights = []
744.
745.
746.     # In[15]:
747.
748.
749.     defuzzify_method = defuzzMethods[0]
750.     Pinitial = initialConditions[0]
751.     P = np.zeros(50)
752.     P[0] = Pinitial
753.     DH = np.zeros(50)
754.     EP = []
755.     TP = []
756.     for t in range(1, timeHorizon):
757.         EPt = (P[t - 1] - PO)
758.         if EPt > 15:
759.             EPt = 15
760.         elif EPt < -15:
761.             EPt = -15
762.         EPt *= 1.0
763.         EP.append(EPt)
764.         if t == 1:
765.             TPt = 0
766.         else:
767.             TPt = P[t - 1] - P[t - 2]
768.             if TPt > 15:
769.                 TPt = 15
770.             elif TPt < -15:
771.                 TPt = -15
772.         TPt *= 1.0
773.         TP.append(TPt)
774.
775.         # Given Rules:
776.         rule_1 = fuzzy_rule([EPt], Ng, [TPt], NgUPp, Domain, Pg)
777.         rule_2 = fuzzy_rule([EPt], NgUNm, [TPt], NgUNp, Domain, P
    m)
778.         rule_3 = fuzzy_rule([EPt], Np, [TPt], NpUPi, Domain, Pm)
```

```
779.          rule_4 = fuzzy_rule([EPt], Ni, [TPt], NgUNm, Domain, Pm)
780.          rule_5 = fuzzy_rule([EPt], Ni, [TPt], PmUPg, Domain, Np)
781.          rule_6 = fuzzy_rule([EPt], NiUPi, [TPt], Ce, Domain, Ce)
782.          rule_7 = fuzzy_rule([EPt], Pi, [TPt], NgUNm, Domain, Pp)
783.          rule_8 = fuzzy_rule([EPt], Pi, [TPt], PmUPg, Domain, Nm)
784.          rule_9 = fuzzy_rule([EPt], Pp, [TPt], NiUPp, Domain, Nm)
785.          rule_10 = fuzzy_rule([EPt], PmUPg, [TPt], PpUPg, Domain,
    Nm)
786.          rule_11 = fuzzy_rule([EPt], Pg, [TPt], NpUPg, Domain, Ng)
787.          rule_12 = fuzzy_rule([EPt], Ni, [TPt], Pp, Domain, Ce)
788.          rule_13 = fuzzy_rule([EPt], Ni, [TPt], Np, Domain, Pp)
789.          rule_14 = fuzzy_rule([EPt], Pi, [TPt], Np, Domain, Ce)
790.          rule_15 = fuzzy_rule([EPt], Pi, [TPt], Pp, Domain, Np)
791.          rule_16 = fuzzy_rule([EPt], NgUNp, [TPt], PmUPg, Domain,
    Pg)
792.          rule_17 = fuzzy_rule([EPt], PpUPg, [TPt], NgUNm, Domain,
    Ng)
793.
794.          rule_map=[rule_1, rule_2, rule_3, rule_4, rule_5, rule_6,
     rule_7, rule_8,
795.                    rule_9, rule_10, rule_11, rule_12, rule_13, rul
    e_14, rule_15, rule_16, rule_17]
796.          aggregated_rules = aggregation(rule_map)
797.          #dHt = COG(Domain, aggregated_rules)
798.          dHt = defuzz(Domain, aggregated_rules, defuzzify_method)
799.          if dHt[0] > 15:
800.              dHt[0] = 15
801.          elif dHt[0] < -15:
802.              dHt[0] = -15
803.          DH[t] = dHt[0]
804.          dP = dHt[0] * K
805.          P[t] = P[t - 1] + dP
806.
807.      Pmom.append(P)
808.      EPmom.append(EP)
809.      TPmom.append(TP)
810.      plt.figure()
811.      plt.plot(P,'-*')
812.      plt.show()
813.
814.
815.      # In[16]:
816.
817.
818.      defuzzify_method = defuzzMethods[1]
819.      Pinitial = initialConditions[0]
820.      P = np.zeros(50)
821.      P[0] = Pinitial
822.      DH = np.zeros(50)
823.      EP = []
824.      TP = []
825.      for t in range(1, timeHorizon):
826.          EPt = (P[t - 1] - PO)
827.          if EPt > 15:
828.              EPt = 15
829.          elif EPt < -15:
830.              EPt = -15
```

```
831.          EPt *= 1.0
832.          EP.append(EPt)
833.          if t == 1:
834.              TPt = 0
835.          else:
836.              TPt = P[t - 1] - P[t - 2]
837.              if TPt > 15:
838.                  TPt = 15
839.              elif TPt < -15:
840.                  TPt = -15
841.          TPt *= 1.0
842.          TP.append(TPt)
843.
844.          # Given Rules:
845.          rule_1 = fuzzy_rule([EPt], Ng, [TPt], NgUPp, Domain, Pg)
846.          rule_2 = fuzzy_rule([EPt], NgUNm, [TPt], NgUNp, Domain, P
   m)
847.          rule_3 = fuzzy_rule([EPt], Np, [TPt], NpUPi, Domain, Pm)
848.          rule_4 = fuzzy_rule([EPt], Ni, [TPt], NgUNm, Domain, Pm)
849.          rule_5 = fuzzy_rule([EPt], Ni, [TPt], PmUPg, Domain, Np)
850.          rule_6 = fuzzy_rule([EPt], NiUPi, [TPt], Ce, Domain, Ce)
851.          rule_7 = fuzzy_rule([EPt], Pi, [TPt], NgUNm, Domain, Pp)
852.          rule_8 = fuzzy_rule([EPt], Pi, [TPt], PmUPg, Domain, Nm)
853.          rule_9 = fuzzy_rule([EPt], Pp, [TPt], NiUPp, Domain, Nm)
854.          rule_10 = fuzzy_rule([EPt], PmUPg, [TPt], PpUPg, Domain,
   Nm)
855.          rule_11 = fuzzy_rule([EPt], Pg, [TPt], NpUPg, Domain, Ng)
856.          rule_12 = fuzzy_rule([EPt], Ni, [TPt], Pp, Domain, Ce)
857.          rule_13 = fuzzy_rule([EPt], Ni, [TPt], Np, Domain, Pp)
858.          rule_14 = fuzzy_rule([EPt], Pi, [TPt], Np, Domain, Ce)
859.          rule_15 = fuzzy_rule([EPt], Pi, [TPt], Pp, Domain, Np)
860.          rule_16 = fuzzy_rule([EPt], NgUNp, [TPt], PmUPg, Domain,
   Pg)
861.          rule_17 = fuzzy_rule([EPt], PpUPg, [TPt], NgUNm, Domain,
   Ng)
862.
863.          rule_map=[rule_1, rule_2, rule_3, rule_4, rule_5, rule_6,
    rule_7, rule_8,
864.                    rule_9, rule_10, rule_11, rule_12, rule_13, rul
   e_14, rule_15, rule_16, rule_17]
865.          aggregated_rules = aggregation(rule_map)
866.          dHt = defuzz(Domain, aggregated_rules, defuzzify_method)
867.          if dHt[0] > 15:
868.              dHt[0] = 15
869.          elif dHt[0] < -15:
870.              dHt[0] = -15
871.          DH[t] = dHt[0]
872.          dP = dHt[0] * K
873.          P[t] = P[t - 1] + dP
874.
875.      Pcentroid.append(P)
876.      EPcentroid.append(EP)
877.      TPcentroid.append(TP)
878.      plt.figure()
879.      plt.plot(P,'-*')
880.      plt.show()
881.
```

```
882.
883.     # In[17]:
884.
885.
886.     defuzzify_method = defuzzMethods[2]
887.     Pinitial = initialConditions[0]
888.     P = np.zeros(50)
889.     P[0] = Pinitial
890.     DH = np.zeros(50)
891.     EP = []
892.     TP = []
893.     for t in range(1, timeHorizon):
894.         EPt = (P[t - 1] - PO)
895.         if EPt > 15:
896.             EPt = 15
897.         elif EPt < -15:
898.             EPt = -15
899.         EPt *= 1.0
900.         EP.append(EPt)
901.         if t == 1:
902.             TPt = 0
903.         else:
904.             TPt = P[t - 1] - P[t - 2]
905.             if TPt > 15:
906.                 TPt = 15
907.             elif TPt < -15:
908.                 TPt = -15
909.         TPt *= 1.0
910.         TP.append(TPt)
911.
912.         # Given Rules:
913.         rule_1 = fuzzy_rule([EPt], Ng, [TPt], NgUPp, Domain, Pg)
914.         rule_2 = fuzzy_rule([EPt], NgUNm, [TPt], NgUNp, Domain, P
    m)
915.         rule_3 = fuzzy_rule([EPt], Np, [TPt], NpUPi, Domain, Pm)
916.         rule_4 = fuzzy_rule([EPt], Ni, [TPt], NgUNm, Domain, Pm)
917.         rule_5 = fuzzy_rule([EPt], Ni, [TPt], PmUPg, Domain, Np)
918.         rule_6 = fuzzy_rule([EPt], NiUPi, [TPt], Ce, Domain, Ce)
919.         rule_7 = fuzzy_rule([EPt], Pi, [TPt], NgUNm, Domain, Pp)
920.         rule_8 = fuzzy_rule([EPt], Pi, [TPt], PmUPg, Domain, Nm)
921.         rule_9 = fuzzy_rule([EPt], Pp, [TPt], NiUPp, Domain, Nm)
922.         rule_10 = fuzzy_rule([EPt], PmUPg, [TPt], PpUPg, Domain,
    Nm)
923.         rule_11 = fuzzy_rule([EPt], Pg, [TPt], NpUPg, Domain, Ng)
924.         rule_12 = fuzzy_rule([EPt], Ni, [TPt], Pp, Domain, Ce)
925.         rule_13 = fuzzy_rule([EPt], Ni, [TPt], Np, Domain, Pp)
926.         rule_14 = fuzzy_rule([EPt], Pi, [TPt], Np, Domain, Ce)
927.         rule_15 = fuzzy_rule([EPt], Pi, [TPt], Pp, Domain, Np)
928.         rule_16 = fuzzy_rule([EPt], NgUNp, [TPt], PmUPg, Domain,
    Pg)
929.         rule_17 = fuzzy_rule([EPt], PpUPg, [TPt], NgUNm, Domain,
    Ng)
930.
931.         rule_map=[rule_1, rule_2, rule_3, rule_4, rule_5, rule_6,
     rule_7, rule_8,
932.                   rule_9, rule_10, rule_11, rule_12, rule_13, rul
    e_14, rule_15, rule_16, rule_17]
```

```
933.          COGs = []
934.          for rule in rule_map:
935.              COGs.append(COG(Domain, rule))
936.          rule_1 = fuzzy_rule_antecedent([EPt], Ng, [TPt], NgUPp)
937.          rule_2 = fuzzy_rule_antecedent([EPt], NgUNm, [TPt], NgUNp
   )
938.          rule_3 = fuzzy_rule_antecedent([EPt], Np, [TPt], NpUPi)
939.          rule_4 = fuzzy_rule_antecedent([EPt], Ni, [TPt], NgUNm)
940.          rule_5 = fuzzy_rule_antecedent([EPt], Ni, [TPt], PmUPg)
941.          rule_6 = fuzzy_rule_antecedent([EPt], NiUPi, [TPt], Ce)
942.          rule_7 = fuzzy_rule_antecedent([EPt], Pi, [TPt], NgUNm)
943.          rule_8 = fuzzy_rule_antecedent([EPt], Pi, [TPt], PmUPg)
944.          rule_9 = fuzzy_rule_antecedent([EPt], Pp, [TPt], NiUPp)
945.          rule_10 = fuzzy_rule_antecedent([EPt], PmUPg, [TPt], PpUP
   g)
946.          rule_11 = fuzzy_rule_antecedent([EPt], Pg, [TPt], NpUPg)
947.          rule_12 = fuzzy_rule_antecedent([EPt], Ni, [TPt], Pp)
948.          rule_13 = fuzzy_rule_antecedent([EPt], Ni, [TPt], Np)
949.          rule_14 = fuzzy_rule_antecedent([EPt], Pi, [TPt], Np)
950.          rule_15 = fuzzy_rule_antecedent([EPt], Pi, [TPt], Pp)
951.          rule_16 = fuzzy_rule_antecedent([EPt], NgUNp, [TPt], PmUP
   g)
952.          rule_17 = fuzzy_rule_antecedent([EPt], PpUPg, [TPt], NgUN
   m)
953.
954.          rule_map=[rule_1, rule_2, rule_3, rule_4, rule_5, rule_6,
    rule_7, rule_8,
955.                    rule_9, rule_10, rule_11, rule_12, rule_13, rul
   e_14, rule_15, rule_16, rule_17]
956.          num = 0
957.          den = 0
958.          for i in range(0, len(rule_map)):
959.              cog = COGs[i][0]
960.              mu = rule_map[i][0]
961.              num += cog*mu
962.              den += mu
963.          aggregated_rules = [num/den]
964.
965.          dHt = defuzz(Domain, aggregated_rules, defuzzify_method)
966.          if dHt[0] > 15:
967.              dHt[0] = 15
968.          elif dHt[0] < -15:
969.              dHt[0] = -15
970.          DH[t] = dHt[0]
971.          dP = dHt[0] * K
972.          P[t] = P[t - 1] + dP
973.
974.      Pheights.append(P)
975.      EPheights.append(EP)
976.      TPheights.append(TP)
977.      plt.figure()
978.      plt.plot(P,'-*')
979.      plt.show()
980.
981.
982.      # In[18]:
983.
```

```
984.
985.     fig = plt.figure()
986.     plt.plot(Pmom[0], label = 'MoM')
987.     plt.plot(Pcentroid[0], label = 'Centroid')
988.     plt.plot(Pheights[0], label = 'Heights')
989.     plt.legend()
990.     plt.xlabel('Tiempo')
991.     plt.ylabel('Presión')
992.     plt.title('Respuesta del controlador con distintos métodos de
    desdifusión')
993.     fig.savefig('from500.png')
994.     plt.show()
995.
996.
997.     # In[19]:
998.
999.
1000.    defuzzify_method = defuzzMethods[0]
1001.    Pinitial = initialConditions[1]
1002.    P = np.zeros(50)
1003.    P[0] = Pinitial
1004.    DH = np.zeros(50)
1005.    EP = []
1006.    TP = []
1007.    for t in range(1, timeHorizon):
1008.        EPt = (P[t - 1] - PO)
1009.        if EPt > 15:
1010.            EPt = 15
1011.        elif EPt < -15:
1012.            EPt = -15
1013.        EPt *= 1.0
1014.        EP.append(EPt)
1015.        if t == 1:
1016.            TPt = 0
1017.        else:
1018.            TPt = P[t - 1] - P[t - 2]
1019.            if TPt > 15:
1020.                TPt = 15
1021.            elif TPt < -15:
1022.                TPt = -15
1023.        TPt *= 1.0
1024.        TP.append(TPt)
1025.
1026.        # Given Rules:
1027.        rule_1 = fuzzy_rule([EPt], Ng, [TPt], NgUPp, Domain, Pg)
1028.        rule_2 = fuzzy_rule([EPt], NgUNm, [TPt], NgUNp, Domain, P
    m)
1029.        rule_3 = fuzzy_rule([EPt], Np, [TPt], NpUPi, Domain, Pm)
1030.        rule_4 = fuzzy_rule([EPt], Ni, [TPt], NgUNm, Domain, Pm)
1031.        rule_5 = fuzzy_rule([EPt], Ni, [TPt], PmUPg, Domain, Np)
1032.        rule_6 = fuzzy_rule([EPt], NiUPi, [TPt], Ce, Domain, Ce)
1033.        rule_7 = fuzzy_rule([EPt], Pi, [TPt], NgUNm, Domain, Pp)
1034.        rule_8 = fuzzy_rule([EPt], Pi, [TPt], PmUPg, Domain, Nm)
1035.        rule_9 = fuzzy_rule([EPt], Pp, [TPt], NiUPp, Domain, Nm)
1036.        rule_10 = fuzzy_rule([EPt], PmUPg, [TPt], PpUPg, Domain,
    Nm)
1037.        rule_11 = fuzzy_rule([EPt], Pg, [TPt], NpUPg, Domain, Ng)
```

```
        rule_12 = fuzzy_rule([EPt], Ni, [TPt], Pp, Domain, Ce)
        rule_13 = fuzzy_rule([EPt], Ni, [TPt], Np, Domain, Pp)
        rule_14 = fuzzy_rule([EPt], Pi, [TPt], Np, Domain, Ce)
        rule_15 = fuzzy_rule([EPt], Pi, [TPt], Pp, Domain, Np)
        rule_16 = fuzzy_rule([EPt], NgUNp, [TPt], PmUPg, Domain,
   Pg)
        rule_17 = fuzzy_rule([EPt], PpUPg, [TPt], NgUNm, Domain,
   Ng)

        rule_map=[rule_1, rule_2, rule_3, rule_4, rule_5, rule_6,
    rule_7, rule_8,
                  rule_9, rule_10, rule_11, rule_12, rule_13, rul
   e_14, rule_15, rule_16, rule_17]
        aggregated_rules = aggregation(rule_map)
        #dHt = COG(Domain, aggregated_rules)
        dHt = defuzz(Domain, aggregated_rules, defuzzify_method)
        if dHt[0] > 15:
            dHt[0] = 15
        elif dHt[0] < -15:
            dHt[0] = -15
        DH[t] = dHt[0]
        dP = dHt[0] * K
        P[t] = P[t - 1] + dP

    Pmom.append(P)
    EPmom.append(EP)
    TPmom.append(TP)
    plt.figure()
    plt.plot(P,'-*')
    plt.show()


    # In[20]:


    defuzzify_method = defuzzMethods[1]
    Pinitial = initialConditions[1]
    P = np.zeros(50)
    P[0] = Pinitial
    DH = np.zeros(50)
    EP = []
    TP = []
    for t in range(1, timeHorizon):
        EPt = (P[t - 1] - PO)
        if EPt > 15:
            EPt = 15
        elif EPt < -15:
            EPt = -15
        EPt *= 1.0
        EP.append(EPt)
        if t == 1:
            TPt = 0
        else:
            TPt = P[t - 1] - P[t - 2]
            if TPt > 15:
                TPt = 15
            elif TPt < -15:
```

```
                TPt = -15
        TPt *= 1.0
        TP.append(TPt)

        # Given Rules:
        rule_1 = fuzzy_rule([EPt], Ng, [TPt], NgUPp, Domain, Pg)
        rule_2 = fuzzy_rule([EPt], NgUNm, [TPt], NgUNp, Domain, P
   m)
        rule_3 = fuzzy_rule([EPt], Np, [TPt], NpUPi, Domain, Pm)
        rule_4 = fuzzy_rule([EPt], Ni, [TPt], NgUNm, Domain, Pm)
        rule_5 = fuzzy_rule([EPt], Ni, [TPt], PmUPg, Domain, Np)
        rule_6 = fuzzy_rule([EPt], NiUPi, [TPt], Ce, Domain, Ce)
        rule_7 = fuzzy_rule([EPt], Pi, [TPt], NgUNm, Domain, Pp)
        rule_8 = fuzzy_rule([EPt], Pi, [TPt], PmUPg, Domain, Nm)
        rule_9 = fuzzy_rule([EPt], Pp, [TPt], NiUPp, Domain, Nm)
        rule_10 = fuzzy_rule([EPt], PmUPg, [TPt], PpUPg, Domain,
   Nm)
        rule_11 = fuzzy_rule([EPt], Pg, [TPt], NpUPg, Domain, Ng)
        rule_12 = fuzzy_rule([EPt], Ni, [TPt], Pp, Domain, Ce)
        rule_13 = fuzzy_rule([EPt], Ni, [TPt], Np, Domain, Pp)
        rule_14 = fuzzy_rule([EPt], Pi, [TPt], Np, Domain, Ce)
        rule_15 = fuzzy_rule([EPt], Pi, [TPt], Pp, Domain, Np)
        rule_16 = fuzzy_rule([EPt], NgUNp, [TPt], PmUPg, Domain,
   Pg)
        rule_17 = fuzzy_rule([EPt], PpUPg, [TPt], NgUNm, Domain,
   Ng)

        rule_map=[rule_1, rule_2, rule_3, rule_4, rule_5, rule_6,
    rule_7, rule_8,
                  rule_9, rule_10, rule_11, rule_12, rule_13, rul
   e_14, rule_15, rule_16, rule_17]
        aggregated_rules = aggregation(rule_map)
        dHt = defuzz(Domain, aggregated_rules, defuzzify_method)
        if dHt[0] > 15:
            dHt[0] = 15
        elif dHt[0] < -15:
            dHt[0] = -15
        DH[t] = dHt[0]
        dP = dHt[0] * K
        P[t] = P[t - 1] + dP

    Pcentroid.append(P)
    EPcentroid.append(EP)
    TPcentroid.append(TP)
    plt.figure()
    plt.plot(P,'-*')
    plt.show()


    # In[21]:


    defuzzify_method = defuzzMethods[2]
    Pinitial = initialConditions[1]
    P = np.zeros(50)
    P[0] = Pinitial
    DH = np.zeros(50)
```

```
    EP = []
    TP = []
    for t in range(1, timeHorizon):
        EPt = (P[t - 1] - PO)
        if EPt > 15:
            EPt = 15
        elif EPt < -15:
            EPt = -15
        EPt *= 1.0
        EP.append(EPt)
        if t == 1:
            TPt = 0
        else:
            TPt = P[t - 1] - P[t - 2]
            if TPt > 15:
                TPt = 15
            elif TPt < -15:
                TPt = -15
        TPt *= 1.0
        TP.append(TPt)

        # Given Rules:
        rule_1 = fuzzy_rule([EPt], Ng, [TPt], NgUPp, Domain, Pg)
        rule_2 = fuzzy_rule([EPt], NgUNm, [TPt], NgUNp, Domain, P
   m)
        rule_3 = fuzzy_rule([EPt], Np, [TPt], NpUPi, Domain, Pm)
        rule_4 = fuzzy_rule([EPt], Ni, [TPt], NgUNm, Domain, Pm)
        rule_5 = fuzzy_rule([EPt], Ni, [TPt], PmUPg, Domain, Np)
        rule_6 = fuzzy_rule([EPt], NiUPi, [TPt], Ce, Domain, Ce)
        rule_7 = fuzzy_rule([EPt], Pi, [TPt], NgUNm, Domain, Pp)
        rule_8 = fuzzy_rule([EPt], Pi, [TPt], PmUPg, Domain, Nm)
        rule_9 = fuzzy_rule([EPt], Pp, [TPt], NiUPp, Domain, Nm)
        rule_10 = fuzzy_rule([EPt], PmUPg, [TPt], PpUPg, Domain,
   Nm)
        rule_11 = fuzzy_rule([EPt], Pg, [TPt], NpUPg, Domain, Ng)
        rule_12 = fuzzy_rule([EPt], Ni, [TPt], Pp, Domain, Ce)
        rule_13 = fuzzy_rule([EPt], Ni, [TPt], Np, Domain, Pp)
        rule_14 = fuzzy_rule([EPt], Pi, [TPt], Np, Domain, Ce)
        rule_15 = fuzzy_rule([EPt], Pi, [TPt], Pp, Domain, Np)
        rule_16 = fuzzy_rule([EPt], NgUNp, [TPt], PmUPg, Domain,
   Pg)
        rule_17 = fuzzy_rule([EPt], PpUPg, [TPt], NgUNm, Domain,
   Ng)

        rule_map=[rule_1, rule_2, rule_3, rule_4, rule_5, rule_6,
    rule_7, rule_8,
                  rule_9, rule_10, rule_11, rule_12, rule_13, rul
   e_14, rule_15, rule_16, rule_17]
        COGs = []
        for rule in rule_map:
            COGs.append(COG(Domain, rule))
        rule_1 = fuzzy_rule_antecedent([EPt], Ng, [TPt], NgUPp)
        rule_2 = fuzzy_rule_antecedent([EPt], NgUNm, [TPt], NgUNp
   )
        rule_3 = fuzzy_rule_antecedent([EPt], Np, [TPt], NpUPi)
        rule_4 = fuzzy_rule_antecedent([EPt], Ni, [TPt], NgUNm)
        rule_5 = fuzzy_rule_antecedent([EPt], Ni, [TPt], PmUPg)
```

```
        rule_6 = fuzzy_rule_antecedent([EPt], NiUPi, [TPt], Ce)
        rule_7 = fuzzy_rule_antecedent([EPt], Pi, [TPt], NgUNm)
        rule_8 = fuzzy_rule_antecedent([EPt], Pi, [TPt], PmUPg)
        rule_9 = fuzzy_rule_antecedent([EPt], Pp, [TPt], NiUPp)
        rule_10 = fuzzy_rule_antecedent([EPt], PmUPg, [TPt], PpUPg)
        rule_11 = fuzzy_rule_antecedent([EPt], Pg, [TPt], NpUPg)
        rule_12 = fuzzy_rule_antecedent([EPt], Ni, [TPt], Pp)
        rule_13 = fuzzy_rule_antecedent([EPt], Ni, [TPt], Np)
        rule_14 = fuzzy_rule_antecedent([EPt], Pi, [TPt], Np)
        rule_15 = fuzzy_rule_antecedent([EPt], Pi, [TPt], Pp)
        rule_16 = fuzzy_rule_antecedent([EPt], NgUNp, [TPt], PmUPg)
        rule_17 = fuzzy_rule_antecedent([EPt], PpUPg, [TPt], NgUNm)

        rule_map=[rule_1, rule_2, rule_3, rule_4, rule_5, rule_6, rule_7, rule_8,
                  rule_9, rule_10, rule_11, rule_12, rule_13, rule_14, rule_15, rule_16, rule_17]
        num = 0
        den = 0
        for i in range(0, len(rule_map)):
            cog = COGs[i][0]
            mu = rule_map[i][0]
            num += cog*mu
            den += mu
        aggregated_rules = [num/den]

        dHt = defuzz(Domain, aggregated_rules, defuzzify_method)
        if dHt[0] > 15:
            dHt[0] = 15
        elif dHt[0] < -15:
            dHt[0] = -15
        DH[t] = dHt[0]
        dP = dHt[0] * K
        P[t] = P[t - 1] + dP

    Pheights.append(P)
    EPheights.append(EP)
    TPheights.append(TP)
    plt.figure()
    plt.plot(P,'-*')
    plt.show()


    # In[22]:


    fig = plt.figure()
    plt.plot(Pmom[1], label = 'MoM')
    plt.plot(Pcentroid[1], label = 'Centroid')
    plt.plot(Pheights[1], label = 'Heights')
    plt.legend()
    plt.xlabel('Tiempo')
    plt.ylabel('Presión')
```

```
    plt.title('Respuesta del controlador con distintos métodos de desdifusión')
    fig.savefig('from550.png')
    plt.show()


    # In[23]:


    defuzzify_method = defuzzMethods[0]
    Pinitial = initialConditions[2]
    P = np.zeros(50)
    P[0] = Pinitial
    DH = np.zeros(50)
    EP = []
    TP = []
    for t in range(1, timeHorizon):
        EPt = (P[t - 1] - PO)
        if EPt > 15:
            EPt = 15
        elif EPt < -15:
            EPt = -15
        EPt *= 1.0
        EP.append(EPt)
        if t == 1:
            TPt = 0
        else:
            TPt = P[t - 1] - P[t - 2]
            if TPt > 15:
                TPt = 15
            elif TPt < -15:
                TPt = -15
        TPt *= 1.0
        TP.append(TPt)

        # Given Rules:
        rule_1 = fuzzy_rule([EPt], Ng, [TPt], NgUPp, Domain, Pg)
        rule_2 = fuzzy_rule([EPt], NgUNm, [TPt], NgUNp, Domain, Pm)
        rule_3 = fuzzy_rule([EPt], Np, [TPt], NpUPi, Domain, Pm)
        rule_4 = fuzzy_rule([EPt], Ni, [TPt], NgUNm, Domain, Pm)
        rule_5 = fuzzy_rule([EPt], Ni, [TPt], PmUPg, Domain, Np)
        rule_6 = fuzzy_rule([EPt], NiUPi, [TPt], Ce, Domain, Ce)
        rule_7 = fuzzy_rule([EPt], Pi, [TPt], NgUNm, Domain, Pp)
        rule_8 = fuzzy_rule([EPt], Pi, [TPt], PmUPg, Domain, Nm)
        rule_9 = fuzzy_rule([EPt], Pp, [TPt], NiUPp, Domain, Nm)
        rule_10 = fuzzy_rule([EPt], PmUPg, [TPt], PpUPg, Domain, Nm)
        rule_11 = fuzzy_rule([EPt], Pg, [TPt], NpUPg, Domain, Ng)
        rule_12 = fuzzy_rule([EPt], Ni, [TPt], Pp, Domain, Ce)
        rule_13 = fuzzy_rule([EPt], Ni, [TPt], Np, Domain, Pp)
        rule_14 = fuzzy_rule([EPt], Pi, [TPt], Np, Domain, Ce)
        rule_15 = fuzzy_rule([EPt], Pi, [TPt], Pp, Domain, Np)
        rule_16 = fuzzy_rule([EPt], NgUNp, [TPt], PmUPg, Domain, Pg)
        rule_17 = fuzzy_rule([EPt], PpUPg, [TPt], NgUNm, Domain, Ng)
```

```
1295.
1296.         rule_map=[rule_1, rule_2, rule_3, rule_4, rule_5, rule_6,
    rule_7, rule_8,
1297.                   rule_9, rule_10, rule_11, rule_12, rule_13, rul
   e_14, rule_15, rule_16, rule_17]
1298.         aggregated_rules = aggregation(rule_map)
1299.         #dHt = COG(Domain, aggregated_rules)
1300.         dHt = defuzz(Domain, aggregated_rules, defuzzify_method)
1301.         if dHt[0] > 15:
1302.             dHt[0] = 15
1303.         elif dHt[0] < -15:
1304.             dHt[0] = -15
1305.         DH[t] = dHt[0]
1306.         dP = dHt[0] * K
1307.         P[t] = P[t - 1] + dP
1308.
1309.     Pmom.append(P)
1310.     EPmom.append(EP)
1311.     TPmom.append(TP)
1312.     plt.figure()
1313.     plt.plot(P,'-*')
1314.     plt.show()
1315.
1316.
1317.     # In[24]:
1318.
1319.
1320.     defuzzify_method = defuzzMethods[1]
1321.     Pinitial = initialConditions[2]
1322.     P = np.zeros(50)
1323.     P[0] = Pinitial
1324.     DH = np.zeros(50)
1325.     EP = []
1326.     TP = []
1327.     for t in range(1, timeHorizon):
1328.         EPt = (P[t - 1] - PO)
1329.         if EPt > 15:
1330.             EPt = 15
1331.         elif EPt < -15:
1332.             EPt = -15
1333.         EPt *= 1.0
1334.         EP.append(EPt)
1335.         if t == 1:
1336.             TPt = 0
1337.         else:
1338.             TPt = P[t - 1] - P[t - 2]
1339.             if TPt > 15:
1340.                 TPt = 15
1341.             elif TPt < -15:
1342.                 TPt = -15
1343.         TPt *= 1.0
1344.         TP.append(TPt)
1345.
1346.         # Given Rules:
1347.         rule_1 = fuzzy_rule([EPt], Ng, [TPt], NgUPp, Domain, Pg)
1348.         rule_2 = fuzzy_rule([EPt], NgUNm, [TPt], NgUNp, Domain, P
   m)
```

```
1349.         rule_3 = fuzzy_rule([EPt], Np, [TPt], NpUPi, Domain, Pm)
1350.         rule_4 = fuzzy_rule([EPt], Ni, [TPt], NgUNm, Domain, Pm)
1351.         rule_5 = fuzzy_rule([EPt], Ni, [TPt], PmUPg, Domain, Np)
1352.         rule_6 = fuzzy_rule([EPt], NiUPi, [TPt], Ce, Domain, Ce)
1353.         rule_7 = fuzzy_rule([EPt], Pi, [TPt], NgUNm, Domain, Pp)
1354.         rule_8 = fuzzy_rule([EPt], Pi, [TPt], PmUPg, Domain, Nm)
1355.         rule_9 = fuzzy_rule([EPt], Pp, [TPt], NiUPp, Domain, Nm)
1356.         rule_10 = fuzzy_rule([EPt], PmUPg, [TPt], PpUPg, Domain,
   Nm)
1357.         rule_11 = fuzzy_rule([EPt], Pg, [TPt], NpUPg, Domain, Ng)
1358.         rule_12 = fuzzy_rule([EPt], Ni, [TPt], Pp, Domain, Ce)
1359.         rule_13 = fuzzy_rule([EPt], Ni, [TPt], Np, Domain, Pp)
1360.         rule_14 = fuzzy_rule([EPt], Pi, [TPt], Np, Domain, Ce)
1361.         rule_15 = fuzzy_rule([EPt], Pi, [TPt], Pp, Domain, Np)
1362.         rule_16 = fuzzy_rule([EPt], NgUNp, [TPt], PmUPg, Domain,
   Pg)
1363.         rule_17 = fuzzy_rule([EPt], PpUPg, [TPt], NgUNm, Domain,
   Ng)
1364.
1365.         rule_map=[rule_1, rule_2, rule_3, rule_4, rule_5, rule_6,
    rule_7, rule_8,
1366.                   rule_9, rule_10, rule_11, rule_12, rule_13, rul
   e_14, rule_15, rule_16, rule_17]
1367.         aggregated_rules = aggregation(rule_map)
1368.         dHt = defuzz(Domain, aggregated_rules, defuzzify_method)
1369.         if dHt[0] > 15:
1370.             dHt[0] = 15
1371.         elif dHt[0] < -15:
1372.             dHt[0] = -15
1373.         DH[t] = dHt[0]
1374.         dP = dHt[0] * K
1375.         P[t] = P[t - 1] + dP
1376.
1377.     Pcentroid.append(P)
1378.     EPcentroid.append(EP)
1379.     TPcentroid.append(TP)
1380.     plt.figure()
1381.     plt.plot(P,'-*')
1382.     plt.show()
1383.
1384.
1385.     # In[25]:
1386.
1387.
1388.     defuzzify_method = defuzzMethods[2]
1389.     Pinitial = initialConditions[2]
1390.     P = np.zeros(50)
1391.     P[0] = Pinitial
1392.     DH = np.zeros(50)
1393.     EP = []
1394.     TP = []
1395.     for t in range(1, timeHorizon):
1396.         EPt = (P[t - 1] - PO)
1397.         if EPt > 15:
1398.             EPt = 15
1399.         elif EPt < -15:
1400.             EPt = -15
```

```
        EPt *= 1.0
        EP.append(EPt)
        if t == 1:
            TPt = 0
        else:
            TPt = P[t - 1] - P[t - 2]
            if TPt > 15:
                TPt = 15
            elif TPt < -15:
                TPt = -15
        TPt *= 1.0
        TP.append(TPt)

        # Given Rules:
        rule_1 = fuzzy_rule([EPt], Ng, [TPt], NgUPp, Domain, Pg)
        rule_2 = fuzzy_rule([EPt], NgUNm, [TPt], NgUNp, Domain, P
    m)
        rule_3 = fuzzy_rule([EPt], Np, [TPt], NpUPi, Domain, Pm)
        rule_4 = fuzzy_rule([EPt], Ni, [TPt], NgUNm, Domain, Pm)
        rule_5 = fuzzy_rule([EPt], Ni, [TPt], PmUPg, Domain, Np)
        rule_6 = fuzzy_rule([EPt], NiUPi, [TPt], Ce, Domain, Ce)
        rule_7 = fuzzy_rule([EPt], Pi, [TPt], NgUNm, Domain, Pp)
        rule_8 = fuzzy_rule([EPt], Pi, [TPt], PmUPg, Domain, Nm)
        rule_9 = fuzzy_rule([EPt], Pp, [TPt], NiUPp, Domain, Nm)
        rule_10 = fuzzy_rule([EPt], PmUPg, [TPt], PpUPg, Domain,
    Nm)
        rule_11 = fuzzy_rule([EPt], Pg, [TPt], NpUPg, Domain, Ng)
        rule_12 = fuzzy_rule([EPt], Ni, [TPt], Pp, Domain, Ce)
        rule_13 = fuzzy_rule([EPt], Ni, [TPt], Np, Domain, Pp)
        rule_14 = fuzzy_rule([EPt], Pi, [TPt], Np, Domain, Ce)
        rule_15 = fuzzy_rule([EPt], Pi, [TPt], Pp, Domain, Np)
        rule_16 = fuzzy_rule([EPt], NgUNp, [TPt], PmUPg, Domain,
    Pg)
        rule_17 = fuzzy_rule([EPt], PpUPg, [TPt], NgUNm, Domain,
    Ng)

        rule_map=[rule_1, rule_2, rule_3, rule_4, rule_5, rule_6,
     rule_7, rule_8,
                  rule_9, rule_10, rule_11, rule_12, rule_13, rul
    e_14, rule_15, rule_16, rule_17]
        COGs = []
        for rule in rule_map:
            COGs.append(COG(Domain, rule))
        rule_1 = fuzzy_rule_antecedent([EPt], Ng, [TPt], NgUPp)
        rule_2 = fuzzy_rule_antecedent([EPt], NgUNm, [TPt], NgUNp
    )
        rule_3 = fuzzy_rule_antecedent([EPt], Np, [TPt], NpUPi)
        rule_4 = fuzzy_rule_antecedent([EPt], Ni, [TPt], NgUNm)
        rule_5 = fuzzy_rule_antecedent([EPt], Ni, [TPt], PmUPg)
        rule_6 = fuzzy_rule_antecedent([EPt], NiUPi, [TPt], Ce)
        rule_7 = fuzzy_rule_antecedent([EPt], Pi, [TPt], NgUNm)
        rule_8 = fuzzy_rule_antecedent([EPt], Pi, [TPt], PmUPg)
        rule_9 = fuzzy_rule_antecedent([EPt], Pp, [TPt], NiUPp)
        rule_10 = fuzzy_rule_antecedent([EPt], PmUPg, [TPt], PpUP
    g)
        rule_11 = fuzzy_rule_antecedent([EPt], Pg, [TPt], NpUPg)
        rule_12 = fuzzy_rule_antecedent([EPt], Ni, [TPt], Pp)
```

```
1450.         rule_13 = fuzzy_rule_antecedent([EPt], Ni, [TPt], Np)
1451.         rule_14 = fuzzy_rule_antecedent([EPt], Pi, [TPt], Np)
1452.         rule_15 = fuzzy_rule_antecedent([EPt], Pi, [TPt], Pp)
1453.         rule_16 = fuzzy_rule_antecedent([EPt], NgUNp, [TPt], PmUP
   g)
1454.         rule_17 = fuzzy_rule_antecedent([EPt], PpUPg, [TPt], NgUN
   m)
1455.
1456.         rule_map=[rule_1, rule_2, rule_3, rule_4, rule_5, rule_6,
    rule_7, rule_8,
1457.                   rule_9, rule_10, rule_11, rule_12, rule_13, rul
   e_14, rule_15, rule_16, rule_17]
1458.         num = 0
1459.         den = 0
1460.         for i in range(0, len(rule_map)):
1461.             cog = COGs[i][0]
1462.             mu = rule_map[i][0]
1463.             num += cog*mu
1464.             den += mu
1465.         aggregated_rules = [num/den]
1466.
1467.         dHt = defuzz(Domain, aggregated_rules, defuzzify_method)
1468.         if dHt[0] > 15:
1469.             dHt[0] = 15
1470.         elif dHt[0] < -15:
1471.             dHt[0] = -15
1472.         DH[t] = dHt[0]
1473.         dP = dHt[0] * K
1474.         P[t] = P[t - 1] + dP
1475.
1476.     Pheights.append(P)
1477.     EPheights.append(EP)
1478.     TPheights.append(TP)
1479.     plt.figure()
1480.     plt.plot(P,'-*')
1481.     plt.show()
1482.
1483.
1484.     # In[26]:
1485.
1486.
1487.     fig = plt.figure()
1488.     plt.plot(Pmom[2], label = 'MoM')
1489.     plt.plot(Pcentroid[2], label = 'Centroid')
1490.     plt.plot(Pheights[2], label = 'Heights')
1491.     plt.legend()
1492.     plt.xlabel('Tiempo')
1493.     plt.ylabel('Presión')
1494.     plt.title('Respuesta del controlador con distintos métodos de
   desdifusión')
1495.     fig.savefig('from600.png')
1496.     plt.show()
1497.
1498.
1499.     # In[27]:
1500.
1501.
```

```
    fig = plt.figure()
    plt.plot(EPmom[0], TPmom[0])
    plt.xlabel('EP'); plt.ylabel('TP')
    plt.xlim(-15.3, 15.3); plt.ylim(-15.3, 15.3)
    plt.title('Método MoM - Pi = 500')
    fig.savefig('mom500.png')
    plt.show()
    fig = plt.figure()
    plt.plot(EPmom[1], TPmom[1])
    plt.xlabel('EP'); plt.ylabel('TP')
    plt.xlim(-15.3, 15.3); plt.ylim(-15.3, 15.3)
    plt.title('Método MoM - Pi = 550')
    fig.savefig('mom550.png')
    plt.show()
    fig = plt.figure()
    plt.plot(EPmom[2], TPmom[2])
    plt.xlabel('EP'); plt.ylabel('TP')
    plt.xlim(-15.3, 15.3); plt.ylim(-15.3, 15.3)
    plt.title('Método MoM - Pi = 600')
    fig.savefig('mom600.png')
    plt.show()


    # In[28]:


    fig = plt.figure()
    plt.plot(EPcentroid[0], TPcentroid[0])
    plt.xlabel('EP'); plt.ylabel('TP')
    plt.xlim(-15.3, 15.3); plt.ylim(-15.3, 15.3)
    plt.title('Método Centroide - Pi = 500')
    fig.savefig('centroid500.png')
    plt.show()
    fig = plt.figure()
    plt.plot(EPcentroid[1], TPcentroid[1])
    plt.xlabel('EP'); plt.ylabel('TP')
    plt.xlim(-15.3, 15.3); plt.ylim(-15.3, 15.3)
    plt.title('Método Centroide - Pi = 550')
    fig.savefig('centroid550.png')
    plt.show()
    fig = plt.figure()
    plt.plot(EPcentroid[2], TPcentroid[2])
    plt.xlabel('EP'); plt.ylabel('TP')
    plt.xlim(-15.3, 15.3); plt.ylim(-15.3, 15.3)
    plt.title('Método Centroide - Pi = 600')
    fig.savefig('centroid600.png')
    plt.show()


    # In[29]:


    fig = plt.figure()
    plt.plot(EPheights[0], TPheights[0])
    plt.xlabel('EP'); plt.ylabel('TP')
    plt.xlim(-15.3, 15.3); plt.ylim(-15.3, 15.3)
    plt.title('Método Alturas - Pi = 500')
```

```
    fig.savefig('heights500.png')
    plt.show()
    fig = plt.figure()
    plt.plot(EPheights[1], TPheights[1])
    plt.xlabel('EP'); plt.ylabel('TP')
    plt.xlim(-15.3, 15.3); plt.ylim(-15.3, 15.3)
    plt.title('Método Alturas - Pi = 550')
    fig.savefig('heights550.png')
    plt.show()
    fig = plt.figure()
    plt.plot(EPheights[2], TPheights[2])
    plt.xlabel('EP'); plt.ylabel('TP')
    plt.xlim(-15.3, 15.3); plt.ylim(-15.3, 15.3)
    plt.title('Método Alturas - Pi = 600')
    fig.savefig('heights600.png')
    plt.show()


    # In[30]:


    defuzzify_method = defuzzMethods[1] ## <--- Printeo de
variable rule_map. Se puede ver la primera
    Pinitial = initialConditions[0]     ## regla en activarse. En
este bloque se pueden tomar lo dos
    P = np.zeros(50)                    ## primeros métodos de
defusificación y cualquiera de ,los Pinitial
    P[0] = Pinitial
    DH = np.zeros(50)
    EP = []
    TP = []
    for t in range(1, timeHorizon):
        EPt = (P[t - 1] - PO)
        if EPt > 15:
            EPt = 15
        elif EPt < -15:
            EPt = -15
        EPt *= 1.0
        EP.append(EPt)
        if t == 1:
            TPt = 0
        else:
            TPt = P[t - 1] - P[t - 2]
            if TPt > 15:
                TPt = 15
            elif TPt < -15:
                TPt = -15
        TPt *= 1.0
        TP.append(TPt)

        # Given Rules:
        rule_1 = fuzzy_rule([EPt], Ng, [TPt], NgUPp, Domain, Pg)
        rule_2 = fuzzy_rule([EPt], NgUNm, [TPt], NgUNp, Domain, P
m)
        rule_3 = fuzzy_rule([EPt], Np, [TPt], NpUPi, Domain, Pm)
        rule_4 = fuzzy_rule([EPt], Ni, [TPt], NgUNm, Domain, Pm)
        rule_5 = fuzzy_rule([EPt], Ni, [TPt], PmUPg, Domain, Np)
```

```
1612.         rule_6 = fuzzy_rule([EPt], NiUPi, [TPt], Ce, Domain, Ce)
1613.         rule_7 = fuzzy_rule([EPt], Pi, [TPt], NgUNm, Domain, Pp)
1614.         rule_8 = fuzzy_rule([EPt], Pi, [TPt], PmUPg, Domain, Nm)
1615.         rule_9 = fuzzy_rule([EPt], Pp, [TPt], NiUPp, Domain, Nm)
1616.         rule_10 = fuzzy_rule([EPt], PmUPg, [TPt], PpUPg, Domain,
   Nm)
1617.         rule_11 = fuzzy_rule([EPt], Pg, [TPt], NpUPg, Domain, Ng)
1618.         rule_12 = fuzzy_rule([EPt], Ni, [TPt], Pp, Domain, Ce)
1619.         rule_13 = fuzzy_rule([EPt], Ni, [TPt], Np, Domain, Pp)
1620.         rule_14 = fuzzy_rule([EPt], Pi, [TPt], Np, Domain, Ce)
1621.         rule_15 = fuzzy_rule([EPt], Pi, [TPt], Pp, Domain, Np)
1622.         rule_16 = fuzzy_rule([EPt], NgUNp, [TPt], PmUPg, Domain,
   Pg)
1623.         rule_17 = fuzzy_rule([EPt], PpUPg, [TPt], NgUNm, Domain,
   Ng)
1624.
1625.         rule_map=[rule_1, rule_2, rule_3, rule_4, rule_5, rule_6,
    rule_7, rule_8,
1626.                   rule_9, rule_10, rule_11, rule_12, rule_13, rul
   e_14, rule_15, rule_16, rule_17]
1627.         print(rule_map)
1628.         aggregated_rules = aggregation(rule_map)
1629.         dHt = defuzz(Domain, aggregated_rules, defuzzify_method)
1630.         if dHt[0] > 15:
1631.             dHt[0] = 15
1632.         elif dHt[0] < -15:
1633.             dHt[0] = -15
1634.         DH[t] = dHt[0]
1635.         dP = dHt[0] * K
1636.         P[t] = P[t - 1] + dP
1637.
1638.
1639.     # In[31]:
1640.
1641.
1642.     defuzzify_method = defuzzMethods[2] ## <--- Printeo de
   variable rule_map. Se puede ver la primera
1643.     Pinitial = initialConditions[0]     ## regla en activarse. En
   este bloque se puede tomar el último
1644.     P = np.zeros(50)                    ## de los métodos de
   defusificación y cualquiera de ,los Pinitial
1645.     P[0] = Pinitial
1646.     DH = np.zeros(50)
1647.     EP = []
1648.     TP = []
1649.     for t in range(1, timeHorizon):
1650.         EPt = (P[t - 1] - PO)
1651.         if EPt > 15:
1652.             EPt = 15
1653.         elif EPt < -15:
1654.             EPt = -15
1655.         EPt *= 1.0
1656.         EP.append(EPt)
1657.         if t == 1:
1658.             TPt = 0
1659.         else:
1660.             TPt = P[t - 1] - P[t - 2]
```

```
1661.             if TPt > 15:
1662.                 TPt = 15
1663.             elif TPt < -15:
1664.                 TPt = -15
1665.         TPt *= 1.0
1666.         TP.append(TPt)
1667.
1668.         # Given Rules:
1669.         rule_1 = fuzzy_rule([EPt], Ng, [TPt], NgUPp, Domain, Pg)
1670.         rule_2 = fuzzy_rule([EPt], NgUNm, [TPt], NgUNp, Domain, P
    m)
1671.         rule_3 = fuzzy_rule([EPt], Np, [TPt], NpUPi, Domain, Pm)
1672.         rule_4 = fuzzy_rule([EPt], Ni, [TPt], NgUNm, Domain, Pm)
1673.         rule_5 = fuzzy_rule([EPt], Ni, [TPt], PmUPg, Domain, Np)
1674.         rule_6 = fuzzy_rule([EPt], NiUPi, [TPt], Ce, Domain, Ce)
1675.         rule_7 = fuzzy_rule([EPt], Pi, [TPt], NgUNm, Domain, Pp)
1676.         rule_8 = fuzzy_rule([EPt], Pi, [TPt], PmUPg, Domain, Nm)
1677.         rule_9 = fuzzy_rule([EPt], Pp, [TPt], NiUPp, Domain, Nm)
1678.         rule_10 = fuzzy_rule([EPt], PmUPg, [TPt], PpUPg, Domain,
    Nm)
1679.         rule_11 = fuzzy_rule([EPt], Pg, [TPt], NpUPg, Domain, Ng)
1680.         rule_12 = fuzzy_rule([EPt], Ni, [TPt], Pp, Domain, Ce)
1681.         rule_13 = fuzzy_rule([EPt], Ni, [TPt], Np, Domain, Pp)
1682.         rule_14 = fuzzy_rule([EPt], Pi, [TPt], Np, Domain, Ce)
1683.         rule_15 = fuzzy_rule([EPt], Pi, [TPt], Pp, Domain, Np)
1684.         rule_16 = fuzzy_rule([EPt], NgUNp, [TPt], PmUPg, Domain,
    Pg)
1685.         rule_17 = fuzzy_rule([EPt], PpUPg, [TPt], NgUNm, Domain,
    Ng)
1686.
1687.         rule_map=[rule_1, rule_2, rule_3, rule_4, rule_5, rule_6,
     rule_7, rule_8,
1688.                   rule_9, rule_10, rule_11, rule_12, rule_13, rul
    e_14, rule_15, rule_16, rule_17]
1689.         COGs = []
1690.         for rule in rule_map:
1691.             COGs.append(COG(Domain, rule))
1692.         rule_1 = fuzzy_rule_antecedent([EPt], Ng, [TPt], NgUPp)
1693.         rule_2 = fuzzy_rule_antecedent([EPt], NgUNm, [TPt], NgUNp
    )
1694.         rule_3 = fuzzy_rule_antecedent([EPt], Np, [TPt], NpUPi)
1695.         rule_4 = fuzzy_rule_antecedent([EPt], Ni, [TPt], NgUNm)
1696.         rule_5 = fuzzy_rule_antecedent([EPt], Ni, [TPt], PmUPg)
1697.         rule_6 = fuzzy_rule_antecedent([EPt], NiUPi, [TPt], Ce)
1698.         rule_7 = fuzzy_rule_antecedent([EPt], Pi, [TPt], NgUNm)
1699.         rule_8 = fuzzy_rule_antecedent([EPt], Pi, [TPt], PmUPg)
1700.         rule_9 = fuzzy_rule_antecedent([EPt], Pp, [TPt], NiUPp)
1701.         rule_10 = fuzzy_rule_antecedent([EPt], PmUPg, [TPt], PpUP
    g)
1702.         rule_11 = fuzzy_rule_antecedent([EPt], Pg, [TPt], NpUPg)
1703.         rule_12 = fuzzy_rule_antecedent([EPt], Ni, [TPt], Pp)
1704.         rule_13 = fuzzy_rule_antecedent([EPt], Ni, [TPt], Np)
1705.         rule_14 = fuzzy_rule_antecedent([EPt], Pi, [TPt], Np)
1706.         rule_15 = fuzzy_rule_antecedent([EPt], Pi, [TPt], Pp)
1707.         rule_16 = fuzzy_rule_antecedent([EPt], NgUNp, [TPt], PmUP
    g)
```

```
        rule_17 = fuzzy_rule_antecedent([EPt], PpUPg, [TPt], NgUNm)

        rule_map=[rule_1, rule_2, rule_3, rule_4, rule_5, rule_6, rule_7, rule_8,
                  rule_9, rule_10, rule_11, rule_12, rule_13, rule_14, rule_15, rule_16, rule_17]
        print(rule_map)
        num = 0
        den = 0
        for i in range(0, len(rule_map)):
            cog = COGs[i][0]
            mu = rule_map[i][0]
            num += cog*mu
            den += mu
        aggregated_rules = [num/den]

        dHt = defuzz(Domain, aggregated_rules, defuzzify_method)
        if dHt[0] > 15:
            dHt[0] = 15
        elif dHt[0] < -15:
            dHt[0] = -15
        DH[t] = dHt[0]
        dP = dHt[0] * K
        P[t] = P[t - 1] + dP


    # In[32]:


    ## Las pruebas de los 2 bloques anteriores indican que las primeras reglas en activarse
    ## son por lo general la 1 y la 11, seguidas de la 17 en el caso de defusificación por
    ## método de las alturas

    ## Por lo tanto se repetirá la primera experiencia borrando estas reglas.


    # In[33]:


    PO = 560
    K = 0.6
    timeHorizon = 50
    initialConditions = [500.0, 550.0, 600.0]
    defuzzMethods = [MOM, COG, Heights]
    Pmom = []
    Pcentroid = []
    Pheights = []
    EPmom = []
    TPmom = []
    EPcentroid = []
    TPcentroid = []
    EPheights = []
    TPheights = []
```

```
1759.
1760.
1761.    # In[34]:
1762.
1763.
1764.    defuzzify_method = defuzzMethods[0]
1765.    Pinitial = initialConditions[0]
1766.    P = np.zeros(50)
1767.    P[0] = Pinitial
1768.    DH = np.zeros(50)
1769.    EP = []
1770.    TP = []
1771.    for t in range(1, timeHorizon):
1772.        EPt = (P[t - 1] - PO)
1773.        if EPt > 15:
1774.            EPt = 15
1775.        elif EPt < -15:
1776.            EPt = -15
1777.        EPt *= 1.0
1778.        EP.append(EPt)
1779.        if t == 1:
1780.            TPt = 0
1781.        else:
1782.            TPt = P[t - 1] - P[t - 2]
1783.            if TPt > 15:
1784.                TPt = 15
1785.            elif TPt < -15:
1786.                TPt = -15
1787.        TPt *= 1.0
1788.        TP.append(TPt)
1789.
1790.        # Given Rules:
1791.        #rule_1 = fuzzy_rule([EPt], Ng, [TPt], NgUPp, Domain, Pg)
1792.        rule_2 = fuzzy_rule([EPt], NgUNm, [TPt], NgUNp, Domain, P
   m)
1793.        rule_3 = fuzzy_rule([EPt], Np, [TPt], NpUPi, Domain, Pm)
1794.        rule_4 = fuzzy_rule([EPt], Ni, [TPt], NgUNm, Domain, Pm)
1795.        rule_5 = fuzzy_rule([EPt], Ni, [TPt], PmUPg, Domain, Np)
1796.        rule_6 = fuzzy_rule([EPt], NiUPi, [TPt], Ce, Domain, Ce)
1797.        rule_7 = fuzzy_rule([EPt], Pi, [TPt], NgUNm, Domain, Pp)
1798.        rule_8 = fuzzy_rule([EPt], Pi, [TPt], PmUPg, Domain, Nm)
1799.        rule_9 = fuzzy_rule([EPt], Pp, [TPt], NiUPp, Domain, Nm)
1800.        rule_10 = fuzzy_rule([EPt], PmUPg, [TPt], PpUPg, Domain,
   Nm)
1801.        rule_11 = fuzzy_rule([EPt], Pg, [TPt], NpUPg, Domain, Ng)
1802.        rule_12 = fuzzy_rule([EPt], Ni, [TPt], Pp, Domain, Ce)
1803.        rule_13 = fuzzy_rule([EPt], Ni, [TPt], Np, Domain, Pp)
1804.        rule_14 = fuzzy_rule([EPt], Pi, [TPt], Np, Domain, Ce)
1805.        rule_15 = fuzzy_rule([EPt], Pi, [TPt], Pp, Domain, Np)
1806.        rule_16 = fuzzy_rule([EPt], NgUNp, [TPt], PmUPg, Domain,
   Pg)
1807.        rule_17 = fuzzy_rule([EPt], PpUPg, [TPt], NgUNm, Domain,
   Ng)
1808.
1809.        rule_map=[rule_2, rule_3, rule_4, rule_5, rule_6, rule_7,
    rule_8,
```

```
                    rule_9, rule_10, rule_11, rule_12, rule_13, rul
   e_14, rule_15, rule_16, rule_17]
        aggregated_rules = aggregation(rule_map)
        #dHt = COG(Domain, aggregated_rules)
        dHt = defuzz(Domain, aggregated_rules, defuzzify_method)
        if dHt[0] > 15:
            dHt[0] = 15
        elif dHt[0] < -15:
            dHt[0] = -15
        DH[t] = dHt[0]
        dP = dHt[0] * K
        P[t] = P[t - 1] + dP

    Pmom.append(P)
    EPmom.append(EP)
    TPmom.append(TP)
    plt.figure()
    plt.plot(P,'-*')
    plt.show()


    # In[35]:


    defuzzify_method = defuzzMethods[1]
    Pinitial = initialConditions[0]
    P = np.zeros(50)
    P[0] = Pinitial
    DH = np.zeros(50)
    EP = []
    TP = []
    for t in range(1, timeHorizon):
        EPt = (P[t - 1] - PO)
        if EPt > 15:
            EPt = 15
        elif EPt < -15:
            EPt = -15
        EPt *= 1.0
        EP.append(EPt)
        if t == 1:
            TPt = 0
        else:
            TPt = P[t - 1] - P[t - 2]
            if TPt > 15:
                TPt = 15
            elif TPt < -15:
                TPt = -15
        TPt *= 1.0
        TP.append(TPt)

        # Given Rules:
        #rule_1 = fuzzy_rule([EPt], Ng, [TPt], NgUPp, Domain, Pg)
        rule_2 = fuzzy_rule([EPt], NgUNm, [TPt], NgUNp, Domain, P
   m)
        rule_3 = fuzzy_rule([EPt], Np, [TPt], NpUPi, Domain, Pm)
        rule_4 = fuzzy_rule([EPt], Ni, [TPt], NgUNm, Domain, Pm)
        rule_5 = fuzzy_rule([EPt], Ni, [TPt], PmUPg, Domain, Np)
```

```
        rule_6 = fuzzy_rule([EPt], NiUPi, [TPt], Ce, Domain, Ce)
        rule_7 = fuzzy_rule([EPt], Pi, [TPt], NgUNm, Domain, Pp)
        rule_8 = fuzzy_rule([EPt], Pi, [TPt], PmUPg, Domain, Nm)
        rule_9 = fuzzy_rule([EPt], Pp, [TPt], NiUPp, Domain, Nm)
        rule_10 = fuzzy_rule([EPt], PmUPg, [TPt], PpUPg, Domain,
   Nm)
        rule_11 = fuzzy_rule([EPt], Pg, [TPt], NpUPg, Domain, Ng)
        rule_12 = fuzzy_rule([EPt], Ni, [TPt], Pp, Domain, Ce)
        rule_13 = fuzzy_rule([EPt], Ni, [TPt], Np, Domain, Pp)
        rule_14 = fuzzy_rule([EPt], Pi, [TPt], Np, Domain, Ce)
        rule_15 = fuzzy_rule([EPt], Pi, [TPt], Pp, Domain, Np)
        rule_16 = fuzzy_rule([EPt], NgUNp, [TPt], PmUPg, Domain,
   Pg)
        rule_17 = fuzzy_rule([EPt], PpUPg, [TPt], NgUNm, Domain,
   Ng)

        rule_map=[rule_2, rule_3, rule_4, rule_5, rule_6, rule_7,
    rule_8,
                  rule_9, rule_10, rule_11, rule_12, rule_13, rul
   e_14, rule_15, rule_16, rule_17]
        aggregated_rules = aggregation(rule_map)
        dHt = defuzz(Domain, aggregated_rules, defuzzify_method)
        if dHt[0] > 15:
            dHt[0] = 15
        elif dHt[0] < -15:
            dHt[0] = -15
        DH[t] = dHt[0]
        dP = dHt[0] * K
        P[t] = P[t - 1] + dP

    Pcentroid.append(P)
    EPcentroid.append(EP)
    TPcentroid.append(TP)
    plt.figure()
    plt.plot(P,'-*')
    plt.show()


    # In[36]:


    defuzzify_method = defuzzMethods[2]
    Pinitial = initialConditions[0]
    P = np.zeros(50)
    P[0] = Pinitial
    DH = np.zeros(50)
    EP = []
    TP = []
    try:
        for t in range(1, timeHorizon):
            EPt = (P[t - 1] - PO)
            if EPt > 15:
                EPt = 15
            elif EPt < -15:
                EPt = -15
            EPt *= 1.0
            EP.append(EPt)
```

```
            if t == 1:
                TPt = 0
            else:
                TPt = P[t - 1] - P[t - 2]
                if TPt > 15:
                    TPt = 15
                elif TPt < -15:
                    TPt = -15
            TPt *= 1.0
            TP.append(TPt)

            # Given Rules:
            #rule_1 = fuzzy_rule([EPt], Ng, [TPt], NgUPp, Domain,
   Pg)
            rule_2 = fuzzy_rule([EPt], NgUNm, [TPt], NgUNp, Domai
   n, Pm)
            rule_3 = fuzzy_rule([EPt], Np, [TPt], NpUPi, Domain,
   Pm)
            rule_4 = fuzzy_rule([EPt], Ni, [TPt], NgUNm, Domain,
   Pm)
            rule_5 = fuzzy_rule([EPt], Ni, [TPt], PmUPg, Domain,
   Np)
            rule_6 = fuzzy_rule([EPt], NiUPi, [TPt], Ce, Domain,
   Ce)
            rule_7 = fuzzy_rule([EPt], Pi, [TPt], NgUNm, Domain,
   Pp)
            rule_8 = fuzzy_rule([EPt], Pi, [TPt], PmUPg, Domain,
   Nm)
            rule_9 = fuzzy_rule([EPt], Pp, [TPt], NiUPp, Domain,
   Nm)
            rule_10 = fuzzy_rule([EPt], PmUPg, [TPt], PpUPg, Doma
   in, Nm)
            rule_11 = fuzzy_rule([EPt], Pg, [TPt], NpUPg, Domain,
    Ng)
            rule_12 = fuzzy_rule([EPt], Ni, [TPt], Pp, Domain, Ce
   )
            rule_13 = fuzzy_rule([EPt], Ni, [TPt], Np, Domain, Pp
   )
            rule_14 = fuzzy_rule([EPt], Pi, [TPt], Np, Domain, Ce
   )
            rule_15 = fuzzy_rule([EPt], Pi, [TPt], Pp, Domain, Np
   )
            rule_16 = fuzzy_rule([EPt], NgUNp, [TPt], PmUPg, Doma
   in, Pg)
            rule_17 = fuzzy_rule([EPt], PpUPg, [TPt], NgUNm, Doma
   in, Ng)

            rule_map=[rule_2, rule_3, rule_4, rule_5, rule_6, rul
   e_7, rule_8,
                      rule_9, rule_10, rule_11, rule_12, rule_13,
    rule_14, rule_15, rule_16, rule_17]
            COGs = []
            for rule in rule_map:
                COGs.append(COG(Domain, rule))
            #rule_1 = fuzzy_rule_antecedent([EPt], Ng, [TPt],
   NgUPp)
```

```
1953.             rule_2 = fuzzy_rule_antecedent([EPt], NgUNm, [TPt], N
   gUNp)
1954.             rule_3 = fuzzy_rule_antecedent([EPt], Np, [TPt], NpUP
   i)
1955.             rule_4 = fuzzy_rule_antecedent([EPt], Ni, [TPt], NgUN
   m)
1956.             rule_5 = fuzzy_rule_antecedent([EPt], Ni, [TPt], PmUP
   g)
1957.             rule_6 = fuzzy_rule_antecedent([EPt], NiUPi, [TPt], C
   e)
1958.             rule_7 = fuzzy_rule_antecedent([EPt], Pi, [TPt], NgUN
   m)
1959.             rule_8 = fuzzy_rule_antecedent([EPt], Pi, [TPt], PmUP
   g)
1960.             rule_9 = fuzzy_rule_antecedent([EPt], Pp, [TPt], NiUP
   p)
1961.             rule_10 = fuzzy_rule_antecedent([EPt], PmUPg, [TPt],
   PpUPg)
1962.             rule_11 = fuzzy_rule_antecedent([EPt], Pg, [TPt], NpU
   Pg)
1963.             rule_12 = fuzzy_rule_antecedent([EPt], Ni, [TPt], Pp)
1964.             rule_13 = fuzzy_rule_antecedent([EPt], Ni, [TPt], Np)
1965.             rule_14 = fuzzy_rule_antecedent([EPt], Pi, [TPt], Np)
1966.             rule_15 = fuzzy_rule_antecedent([EPt], Pi, [TPt], Pp)
1967.             rule_16 = fuzzy_rule_antecedent([EPt], NgUNp, [TPt],
   PmUPg)
1968.             rule_17 = fuzzy_rule_antecedent([EPt], PpUPg, [TPt],
   NgUNm)
1969.
1970.             rule_map=[rule_2, rule_3, rule_4, rule_5, rule_6, rul
   e_7, rule_8,
1971.                       rule_9, rule_10, rule_11, rule_12, rule_13,
    rule_14, rule_15, rule_16, rule_17]
1972.             num = 0
1973.             den = 0
1974.             for i in range(0, len(rule_map)):
1975.                 cog = COGs[i][0]
1976.                 mu = rule_map[i][0]
1977.                 num += cog*mu
1978.                 den += mu
1979.             aggregated_rules = [num/den]
1980.
1981.             dHt = defuzz(Domain, aggregated_rules, defuzzify_meth
   od)
1982.             if dHt[0] > 15:
1983.                 dHt[0] = 15
1984.             elif dHt[0] < -15:
1985.                 dHt[0] = -15
1986.             DH[t] = dHt[0]
1987.             dP = dHt[0] * K
1988.             P[t] = P[t - 1] + dP
1989.     except Exception:
1990.         pass
1991.
1992.     Pheights.append(P)
1993.     EPheights.append(EP)
1994.     TPheights.append(TP)
```

```
    plt.figure()
    plt.plot(P,'-*')
    plt.show()


    # In[37]:


    fig = plt.figure()
    plt.plot(Pmom[0], label = 'MoM')
    plt.plot(Pcentroid[0], label = 'Centroid')
    plt.legend()
    plt.xlabel('Tiempo')
    plt.ylabel('Presión')
    plt.title('Respuesta del controlador con distintos métodos de
desdifusión')
    fig.savefig('from500-noR1.png')
    plt.show()


    # In[38]:


    defuzzify_method = defuzzMethods[0]
    Pinitial = initialConditions[1]
    P = np.zeros(50)
    P[0] = Pinitial
    DH = np.zeros(50)
    EP = []
    TP = []
    for t in range(1, timeHorizon):
        EPt = (P[t - 1] - PO)
        if EPt > 15:
            EPt = 15
        elif EPt < -15:
            EPt = -15
        EPt *= 1.0
        EP.append(EPt)
        if t == 1:
            TPt = 0
        else:
            TPt = P[t - 1] - P[t - 2]
            if TPt > 15:
                TPt = 15
            elif TPt < -15:
                TPt = -15
        TPt *= 1.0
        TP.append(TPt)

        # Given Rules:
        #rule_1 = fuzzy_rule([EPt], Ng, [TPt], NgUPp, Domain, Pg)
        rule_2 = fuzzy_rule([EPt], NgUNm, [TPt], NgUNp, Domain, P
m)
        rule_3 = fuzzy_rule([EPt], Np, [TPt], NpUPi, Domain, Pm)
        rule_4 = fuzzy_rule([EPt], Ni, [TPt], NgUNm, Domain, Pm)
        rule_5 = fuzzy_rule([EPt], Ni, [TPt], PmUPg, Domain, Np)
        rule_6 = fuzzy_rule([EPt], NiUPi, [TPt], Ce, Domain, Ce)
```

```
2050.         rule_7 = fuzzy_rule([EPt], Pi, [TPt], NgUNm, Domain, Pp)
2051.         rule_8 = fuzzy_rule([EPt], Pi, [TPt], PmUPg, Domain, Nm)
2052.         rule_9 = fuzzy_rule([EPt], Pp, [TPt], NiUPp, Domain, Nm)
2053.         rule_10 = fuzzy_rule([EPt], PmUPg, [TPt], PpUPg, Domain,
   Nm)
2054.         rule_11 = fuzzy_rule([EPt], Pg, [TPt], NpUPg, Domain, Ng)
2055.         rule_12 = fuzzy_rule([EPt], Ni, [TPt], Pp, Domain, Ce)
2056.         rule_13 = fuzzy_rule([EPt], Ni, [TPt], Np, Domain, Pp)
2057.         rule_14 = fuzzy_rule([EPt], Pi, [TPt], Np, Domain, Ce)
2058.         rule_15 = fuzzy_rule([EPt], Pi, [TPt], Pp, Domain, Np)
2059.         rule_16 = fuzzy_rule([EPt], NgUNp, [TPt], PmUPg, Domain,
   Pg)
2060.         rule_17 = fuzzy_rule([EPt], PpUPg, [TPt], NgUNm, Domain,
   Ng)
2061.
2062.         rule_map=[rule_2, rule_3, rule_4, rule_5, rule_6, rule_7,
    rule_8,
2063.                   rule_9, rule_10, rule_11, rule_12, rule_13, rul
   e_14, rule_15, rule_16, rule_17]
2064.         aggregated_rules = aggregation(rule_map)
2065.         #dHt = COG(Domain, aggregated_rules)
2066.         dHt = defuzz(Domain, aggregated_rules, defuzzify_method)
2067.         if dHt[0] > 15:
2068.             dHt[0] = 15
2069.         elif dHt[0] < -15:
2070.             dHt[0] = -15
2071.         DH[t] = dHt[0]
2072.         dP = dHt[0] * K
2073.         P[t] = P[t - 1] + dP
2074.
2075.     Pmom.append(P)
2076.     EPmom.append(EP)
2077.     TPmom.append(TP)
2078.     plt.figure()
2079.     plt.plot(P,'-*')
2080.     plt.show()
2081.
2082.
2083.     # In[39]:
2084.
2085.
2086.     defuzzify_method = defuzzMethods[1]
2087.     Pinitial = initialConditions[1]
2088.     P = np.zeros(50)
2089.     P[0] = Pinitial
2090.     DH = np.zeros(50)
2091.     EP = []
2092.     TP = []
2093.     for t in range(1, timeHorizon):
2094.         EPt = (P[t - 1] - PO)
2095.         if EPt > 15:
2096.             EPt = 15
2097.         elif EPt < -15:
2098.             EPt = -15
2099.         EPt *= 1.0
2100.         EP.append(EPt)
2101.         if t == 1:
```

```
2102.             TPt = 0
2103.         else:
2104.             TPt = P[t - 1] - P[t - 2]
2105.             if TPt > 15:
2106.                 TPt = 15
2107.             elif TPt < -15:
2108.                 TPt = -15
2109.         TPt *= 1.0
2110.         TP.append(TPt)
2111.
2112.         # Given Rules:
2113.         #rule_1 = fuzzy_rule([EPt], Ng, [TPt], NgUPp, Domain, Pg)
2114.         rule_2 = fuzzy_rule([EPt], NgUNm, [TPt], NgUNp, Domain, P
   m)
2115.         rule_3 = fuzzy_rule([EPt], Np, [TPt], NpUPi, Domain, Pm)
2116.         rule_4 = fuzzy_rule([EPt], Ni, [TPt], NgUNm, Domain, Pm)
2117.         rule_5 = fuzzy_rule([EPt], Ni, [TPt], PmUPg, Domain, Np)
2118.         rule_6 = fuzzy_rule([EPt], NiUPi, [TPt], Ce, Domain, Ce)
2119.         rule_7 = fuzzy_rule([EPt], Pi, [TPt], NgUNm, Domain, Pp)
2120.         rule_8 = fuzzy_rule([EPt], Pi, [TPt], PmUPg, Domain, Nm)
2121.         rule_9 = fuzzy_rule([EPt], Pp, [TPt], NiUPp, Domain, Nm)
2122.         rule_10 = fuzzy_rule([EPt], PmUPg, [TPt], PpUPg, Domain,
   Nm)
2123.         rule_11 = fuzzy_rule([EPt], Pg, [TPt], NpUPg, Domain, Ng)
2124.         rule_12 = fuzzy_rule([EPt], Ni, [TPt], Pp, Domain, Ce)
2125.         rule_13 = fuzzy_rule([EPt], Ni, [TPt], Np, Domain, Pp)
2126.         rule_14 = fuzzy_rule([EPt], Pi, [TPt], Np, Domain, Ce)
2127.         rule_15 = fuzzy_rule([EPt], Pi, [TPt], Pp, Domain, Np)
2128.         rule_16 = fuzzy_rule([EPt], NgUNp, [TPt], PmUPg, Domain,
   Pg)
2129.         rule_17 = fuzzy_rule([EPt], PpUPg, [TPt], NgUNm, Domain,
   Ng)
2130.
2131.         rule_map=[rule_2, rule_3, rule_4, rule_5, rule_6, rule_7,
    rule_8,
2132.                   rule_9, rule_10, rule_11, rule_12, rule_13, rul
   e_14, rule_15, rule_16, rule_17]
2133.         aggregated_rules = aggregation(rule_map)
2134.         dHt = defuzz(Domain, aggregated_rules, defuzzify_method)
2135.         if dHt[0] > 15:
2136.             dHt[0] = 15
2137.         elif dHt[0] < -15:
2138.             dHt[0] = -15
2139.         DH[t] = dHt[0]
2140.         dP = dHt[0] * K
2141.         P[t] = P[t - 1] + dP
2142.
2143.     Pcentroid.append(P)
2144.     EPcentroid.append(EP)
2145.     TPcentroid.append(TP)
2146.     plt.figure()
2147.     plt.plot(P,'-*')
2148.     plt.show()
2149.
2150.
2151.     # In[40]:
2152.
```

```
2153.
2154.     defuzzify_method = defuzzMethods[2]
2155.     Pinitial = initialConditions[1]
2156.     P = np.zeros(50)
2157.     P[0] = Pinitial
2158.     DH = np.zeros(50)
2159.     EP = []
2160.     TP = []
2161.     try:
2162.         for t in range(1, timeHorizon):
2163.             EPt = (P[t - 1] - PO)
2164.             if EPt > 15:
2165.                 EPt = 15
2166.             elif EPt < -15:
2167.                 EPt = -15
2168.             EPt *= 1.0
2169.             EP.append(EPt)
2170.             if t == 1:
2171.                 TPt = 0
2172.             else:
2173.                 TPt = P[t - 1] - P[t - 2]
2174.                 if TPt > 15:
2175.                     TPt = 15
2176.                 elif TPt < -15:
2177.                     TPt = -15
2178.             TPt *= 1.0
2179.             TP.append(TPt)
2180.
2181.             # Given Rules:
2182.             #rule_1 = fuzzy_rule([EPt], Ng, [TPt], NgUPp, Domain,
    Pg)
2183.             rule_2 = fuzzy_rule([EPt], NgUNm, [TPt], NgUNp, Domai
    n, Pm)
2184.             rule_3 = fuzzy_rule([EPt], Np, [TPt], NpUPi, Domain,
    Pm)
2185.             rule_4 = fuzzy_rule([EPt], Ni, [TPt], NgUNm, Domain,
    Pm)
2186.             rule_5 = fuzzy_rule([EPt], Ni, [TPt], PmUPg, Domain,
    Np)
2187.             rule_6 = fuzzy_rule([EPt], NiUPi, [TPt], Ce, Domain,
    Ce)
2188.             rule_7 = fuzzy_rule([EPt], Pi, [TPt], NgUNm, Domain,
    Pp)
2189.             rule_8 = fuzzy_rule([EPt], Pi, [TPt], PmUPg, Domain,
    Nm)
2190.             rule_9 = fuzzy_rule([EPt], Pp, [TPt], NiUPp, Domain,
    Nm)
2191.             rule_10 = fuzzy_rule([EPt], PmUPg, [TPt], PpUPg, Doma
    in, Nm)
2192.             rule_11 = fuzzy_rule([EPt], Pg, [TPt], NpUPg, Domain,
     Ng)
2193.             rule_12 = fuzzy_rule([EPt], Ni, [TPt], Pp, Domain, Ce
    )
2194.             rule_13 = fuzzy_rule([EPt], Ni, [TPt], Np, Domain, Pp
    )
2195.             rule_14 = fuzzy_rule([EPt], Pi, [TPt], Np, Domain, Ce
    )
```

```
            rule_15 = fuzzy_rule([EPt], Pi, [TPt], Pp, Domain, Np
)
            rule_16 = fuzzy_rule([EPt], NgUNp, [TPt], PmUPg, Doma
in, Pg)
            rule_17 = fuzzy_rule([EPt], PpUPg, [TPt], NgUNm, Doma
in, Ng)

            rule_map=[rule_2, rule_3, rule_4, rule_5, rule_6, rul
e_7, rule_8,
                      rule_9, rule_10, rule_11, rule_12, rule_13,
 rule_14, rule_15, rule_16, rule_17]
            COGs = []
            for rule in rule_map:
                COGs.append(COG(Domain, rule))
            #rule_1 = fuzzy_rule_antecedent([EPt], Ng, [TPt],
NgUPp)
            rule_2 = fuzzy_rule_antecedent([EPt], NgUNm, [TPt], N
gUNp)
            rule_3 = fuzzy_rule_antecedent([EPt], Np, [TPt], NpUP
i)
            rule_4 = fuzzy_rule_antecedent([EPt], Ni, [TPt], NgUN
m)
            rule_5 = fuzzy_rule_antecedent([EPt], Ni, [TPt], PmUP
g)
            rule_6 = fuzzy_rule_antecedent([EPt], NiUPi, [TPt], C
e)
            rule_7 = fuzzy_rule_antecedent([EPt], Pi, [TPt], NgUN
m)
            rule_8 = fuzzy_rule_antecedent([EPt], Pi, [TPt], PmUP
g)
            rule_9 = fuzzy_rule_antecedent([EPt], Pp, [TPt], NiUP
p)
            rule_10 = fuzzy_rule_antecedent([EPt], PmUPg, [TPt],
PpUPg)
            rule_11 = fuzzy_rule_antecedent([EPt], Pg, [TPt], NpU
Pg)
            rule_12 = fuzzy_rule_antecedent([EPt], Ni, [TPt], Pp)
            rule_13 = fuzzy_rule_antecedent([EPt], Ni, [TPt], Np)
            rule_14 = fuzzy_rule_antecedent([EPt], Pi, [TPt], Np)
            rule_15 = fuzzy_rule_antecedent([EPt], Pi, [TPt], Pp)
            rule_16 = fuzzy_rule_antecedent([EPt], NgUNp, [TPt],
PmUPg)
            rule_17 = fuzzy_rule_antecedent([EPt], PpUPg, [TPt],
NgUNm)

            rule_map=[rule_2, rule_3, rule_4, rule_5, rule_6, rul
e_7, rule_8,
                      rule_9, rule_10, rule_11, rule_12, rule_13,
 rule_14, rule_15, rule_16, rule_17]
            num = 0
            den = 0
            for i in range(0, len(rule_map)):
                cog = COGs[i][0]
                mu = rule_map[i][0]
                num += cog*mu
                den += mu
            aggregated_rules = [num/den]
```

```
2233.
2234.             dHt = defuzz(Domain, aggregated_rules, defuzzify_meth
   od)
2235.             if dHt[0] > 15:
2236.                 dHt[0] = 15
2237.             elif dHt[0] < -15:
2238.                 dHt[0] = -15
2239.             DH[t] = dHt[0]
2240.             dP = dHt[0] * K
2241.             P[t] = P[t - 1] + dP
2242.     except Exception:
2243.         pass
2244.
2245.     Pheights.append(P)
2246.     EPheights.append(EP)
2247.     TPheights.append(TP)
2248.     plt.figure()
2249.     plt.plot(P,'-*')
2250.     plt.show()
2251.
2252.
2253.     # In[41]:
2254.
2255.
2256.     fig = plt.figure()
2257.     plt.plot(Pmom[1], label = 'MoM')
2258.     plt.plot(Pcentroid[1], label = 'Centroid')
2259.     plt.legend()
2260.     plt.xlabel('Tiempo')
2261.     plt.ylabel('Presión')
2262.     plt.title('Respuesta del controlador con distintos métodos de
   desdifusión')
2263.     fig.savefig('from550-noR1.png')
2264.     plt.show()
2265.
2266.
2267.     # In[42]:
2268.
2269.
2270.     defuzzify_method = defuzzMethods[0]
2271.     Pinitial = initialConditions[2]
2272.     P = np.zeros(50)
2273.     P[0] = Pinitial
2274.     DH = np.zeros(50)
2275.     EP = []
2276.     TP = []
2277.     for t in range(1, timeHorizon):
2278.         EPt = (P[t - 1] - PO)
2279.         if EPt > 15:
2280.             EPt = 15
2281.         elif EPt < -15:
2282.             EPt = -15
2283.         EPt *= 1.0
2284.         EP.append(EPt)
2285.         if t == 1:
2286.             TPt = 0
2287.         else:
```

```
2288.             TPt = P[t - 1] - P[t - 2]
2289.             if TPt > 15:
2290.                 TPt = 15
2291.             elif TPt < -15:
2292.                 TPt = -15
2293.         TPt *= 1.0
2294.         TP.append(TPt)
2295.
2296.         # Given Rules:
2297.         rule_1 = fuzzy_rule([EPt], Ng, [TPt], NgUPp, Domain, Pg)
2298.         rule_2 = fuzzy_rule([EPt], NgUNm, [TPt], NgUNp, Domain, P
   m)
2299.         rule_3 = fuzzy_rule([EPt], Np, [TPt], NpUPi, Domain, Pm)
2300.         rule_4 = fuzzy_rule([EPt], Ni, [TPt], NgUNm, Domain, Pm)
2301.         rule_5 = fuzzy_rule([EPt], Ni, [TPt], PmUPg, Domain, Np)
2302.         rule_6 = fuzzy_rule([EPt], NiUPi, [TPt], Ce, Domain, Ce)
2303.         rule_7 = fuzzy_rule([EPt], Pi, [TPt], NgUNm, Domain, Pp)
2304.         rule_8 = fuzzy_rule([EPt], Pi, [TPt], PmUPg, Domain, Nm)
2305.         rule_9 = fuzzy_rule([EPt], Pp, [TPt], NiUPp, Domain, Nm)
2306.         rule_10 = fuzzy_rule([EPt], PmUPg, [TPt], PpUPg, Domain,
   Nm)
2307.         #rule_11 = fuzzy_rule([EPt], Pg, [TPt], NpUPg, Domain,
   Ng)
2308.         rule_12 = fuzzy_rule([EPt], Ni, [TPt], Pp, Domain, Ce)
2309.         rule_13 = fuzzy_rule([EPt], Ni, [TPt], Np, Domain, Pp)
2310.         rule_14 = fuzzy_rule([EPt], Pi, [TPt], Np, Domain, Ce)
2311.         rule_15 = fuzzy_rule([EPt], Pi, [TPt], Pp, Domain, Np)
2312.         rule_16 = fuzzy_rule([EPt], NgUNp, [TPt], PmUPg, Domain,
   Pg)
2313.         rule_17 = fuzzy_rule([EPt], PpUPg, [TPt], NgUNm, Domain,
   Ng)
2314.
2315.         rule_map=[rule_1, rule_2, rule_3, rule_4, rule_5, rule_6,
    rule_7, rule_8,
2316.                   rule_9, rule_10, rule_12, rule_13, rule_14, rul
   e_15, rule_16, rule_17]
2317.         aggregated_rules = aggregation(rule_map)
2318.         #dHt = COG(Domain, aggregated_rules)
2319.         dHt = defuzz(Domain, aggregated_rules, defuzzify_method)
2320.         if dHt[0] > 15:
2321.             dHt[0] = 15
2322.         elif dHt[0] < -15:
2323.             dHt[0] = -15
2324.         DH[t] = dHt[0]
2325.         dP = dHt[0] * K
2326.         P[t] = P[t - 1] + dP
2327.
2328.     Pmom.append(P)
2329.     EPmom.append(EP)
2330.     TPmom.append(TP)
2331.     plt.figure()
2332.     plt.plot(P,'-*')
2333.     plt.show()
2334.
2335.
2336.     # In[43]:
2337.
```

```
2338.
2339.     defuzzify_method = defuzzMethods[1]
2340.     Pinitial = initialConditions[2]
2341.     P = np.zeros(50)
2342.     P[0] = Pinitial
2343.     DH = np.zeros(50)
2344.     EP = []
2345.     TP = []
2346.     for t in range(1, timeHorizon):
2347.         EPt = (P[t - 1] - PO)
2348.         if EPt > 15:
2349.             EPt = 15
2350.         elif EPt < -15:
2351.             EPt = -15
2352.         EPt *= 1.0
2353.         EP.append(EPt)
2354.         if t == 1:
2355.             TPt = 0
2356.         else:
2357.             TPt = P[t - 1] - P[t - 2]
2358.             if TPt > 15:
2359.                 TPt = 15
2360.             elif TPt < -15:
2361.                 TPt = -15
2362.         TPt *= 1.0
2363.         TP.append(TPt)
2364.
2365.         # Given Rules:
2366.         rule_1 = fuzzy_rule([EPt], Ng, [TPt], NgUPp, Domain, Pg)
2367.         rule_2 = fuzzy_rule([EPt], NgUNm, [TPt], NgUNp, Domain, P
    m)
2368.         rule_3 = fuzzy_rule([EPt], Np, [TPt], NpUPi, Domain, Pm)
2369.         rule_4 = fuzzy_rule([EPt], Ni, [TPt], NgUNm, Domain, Pm)
2370.         rule_5 = fuzzy_rule([EPt], Ni, [TPt], PmUPg, Domain, Np)
2371.         rule_6 = fuzzy_rule([EPt], NiUPi, [TPt], Ce, Domain, Ce)
2372.         rule_7 = fuzzy_rule([EPt], Pi, [TPt], NgUNm, Domain, Pp)
2373.         rule_8 = fuzzy_rule([EPt], Pi, [TPt], PmUPg, Domain, Nm)
2374.         rule_9 = fuzzy_rule([EPt], Pp, [TPt], NiUPp, Domain, Nm)
2375.         rule_10 = fuzzy_rule([EPt], PmUPg, [TPt], PpUPg, Domain,
    Nm)
2376.         #rule_11 = fuzzy_rule([EPt], Pg, [TPt], NpUPg, Domain,
    Ng)
2377.         rule_12 = fuzzy_rule([EPt], Ni, [TPt], Pp, Domain, Ce)
2378.         rule_13 = fuzzy_rule([EPt], Ni, [TPt], Np, Domain, Pp)
2379.         rule_14 = fuzzy_rule([EPt], Pi, [TPt], Np, Domain, Ce)
2380.         rule_15 = fuzzy_rule([EPt], Pi, [TPt], Pp, Domain, Np)
2381.         rule_16 = fuzzy_rule([EPt], NgUNp, [TPt], PmUPg, Domain,
    Pg)
2382.         rule_17 = fuzzy_rule([EPt], PpUPg, [TPt], NgUNm, Domain,
    Ng)
2383.
2384.         rule_map=[rule_1, rule_2, rule_3, rule_4, rule_5, rule_6,
     rule_7, rule_8,
2385.                   rule_9, rule_10, rule_12, rule_13, rule_14, rul
    e_15, rule_16, rule_17]
2386.         aggregated_rules = aggregation(rule_map)
2387.         dHt = defuzz(Domain, aggregated_rules, defuzzify_method)
```

```
        if dHt[0] > 15:
            dHt[0] = 15
        elif dHt[0] < -15:
            dHt[0] = -15
        DH[t] = dHt[0]
        dP = dHt[0] * K
        P[t] = P[t - 1] + dP

    Pcentroid.append(P)
    EPcentroid.append(EP)
    TPcentroid.append(TP)
    plt.figure()
    plt.plot(P,'-*')
    plt.show()


    # In[44]:


    defuzzify_method = defuzzMethods[2]
    Pinitial = initialConditions[2]
    P = np.zeros(50)
    P[0] = Pinitial
    DH = np.zeros(50)
    EP = []
    TP = []
    try:
        for t in range(1, timeHorizon):
            EPt = (P[t - 1] - PO)
            if EPt > 15:
                EPt = 15
            elif EPt < -15:
                EPt = -15
            EPt *= 1.0
            EP.append(EPt)
            if t == 1:
                TPt = 0
            else:
                TPt = P[t - 1] - P[t - 2]
                if TPt > 15:
                    TPt = 15
                elif TPt < -15:
                    TPt = -15
            TPt *= 1.0
            TP.append(TPt)

            # Given Rules:
            rule_1 = fuzzy_rule([EPt], Ng, [TPt], NgUPp, Domain,
   Pg)
            rule_2 = fuzzy_rule([EPt], NgUNm, [TPt], NgUNp, Domai
   n, Pm)
            rule_3 = fuzzy_rule([EPt], Np, [TPt], NpUPi, Domain,
   Pm)
            rule_4 = fuzzy_rule([EPt], Ni, [TPt], NgUNm, Domain,
   Pm)
            rule_5 = fuzzy_rule([EPt], Ni, [TPt], PmUPg, Domain,
   Np)
```

```
            rule_6 = fuzzy_rule([EPt], NiUPi, [TPt], Ce, Domain,
   Ce)
            rule_7 = fuzzy_rule([EPt], Pi, [TPt], NgUNm, Domain,
   Pp)
            rule_8 = fuzzy_rule([EPt], Pi, [TPt], PmUPg, Domain,
   Nm)
            rule_9 = fuzzy_rule([EPt], Pp, [TPt], NiUPp, Domain,
   Nm)
            rule_10 = fuzzy_rule([EPt], PmUPg, [TPt], PpUPg, Doma
   in, Nm)
            #rule_11 = fuzzy_rule([EPt], Pg, [TPt], NpUPg,
   Domain, Ng)
            rule_12 = fuzzy_rule([EPt], Ni, [TPt], Pp, Domain, Ce
   )
            rule_13 = fuzzy_rule([EPt], Ni, [TPt], Np, Domain, Pp
   )
            rule_14 = fuzzy_rule([EPt], Pi, [TPt], Np, Domain, Ce
   )
            rule_15 = fuzzy_rule([EPt], Pi, [TPt], Pp, Domain, Np
   )
            rule_16 = fuzzy_rule([EPt], NgUNp, [TPt], PmUPg, Doma
   in, Pg)
            rule_17 = fuzzy_rule([EPt], PpUPg, [TPt], NgUNm, Doma
   in, Ng)

            rule_map=[rule_1, rule_2, rule_3, rule_4, rule_5, rul
   e_6, rule_7, rule_8,
                      rule_9, rule_10, rule_12, rule_13, rule_14,
    rule_15, rule_16, rule_17]
            COGs = []
            for rule in rule_map:
                COGs.append(COG(Domain, rule))
            rule_1 = fuzzy_rule_antecedent([EPt], Ng, [TPt], NgUP
   p)
            rule_2 = fuzzy_rule_antecedent([EPt], NgUNm, [TPt], N
   gUNp)
            rule_3 = fuzzy_rule_antecedent([EPt], Np, [TPt], NpUP
   i)
            rule_4 = fuzzy_rule_antecedent([EPt], Ni, [TPt], NgUN
   m)
            rule_5 = fuzzy_rule_antecedent([EPt], Ni, [TPt], PmUP
   g)
            rule_6 = fuzzy_rule_antecedent([EPt], NiUPi, [TPt], C
   e)
            rule_7 = fuzzy_rule_antecedent([EPt], Pi, [TPt], NgUN
   m)
            rule_8 = fuzzy_rule_antecedent([EPt], Pi, [TPt], PmUP
   g)
            rule_9 = fuzzy_rule_antecedent([EPt], Pp, [TPt], NiUP
   p)
            rule_10 = fuzzy_rule_antecedent([EPt], PmUPg, [TPt],
   PpUPg)
            #rule_11 = fuzzy_rule_antecedent([EPt], Pg, [TPt],
   NpUPg)
            rule_12 = fuzzy_rule_antecedent([EPt], Ni, [TPt], Pp)
            rule_13 = fuzzy_rule_antecedent([EPt], Ni, [TPt], Np)
            rule_14 = fuzzy_rule_antecedent([EPt], Pi, [TPt], Np)
```

```
2472.             rule_15 = fuzzy_rule_antecedent([EPt], Pi, [TPt], Pp)
2473.             rule_16 = fuzzy_rule_antecedent([EPt], NgUNp, [TPt],
   PmUPg)
2474.             rule_17 = fuzzy_rule_antecedent([EPt], PpUPg, [TPt],
   NgUNm)
2475.
2476.             rule_map=[rule_1, rule_2, rule_3, rule_4, rule_5, rul
   e_6, rule_7, rule_8,
2477.                       rule_9, rule_10, rule_12, rule_13, rule_14,
    rule_15, rule_16, rule_17]
2478.             num = 0
2479.             den = 0
2480.             for i in range(0, len(rule_map)):
2481.                 cog = COGs[i][0]
2482.                 mu = rule_map[i][0]
2483.                 num += cog*mu
2484.                 den += mu
2485.             aggregated_rules = [num/den]
2486.
2487.             dHt = defuzz(Domain, aggregated_rules, defuzzify_meth
   od)
2488.             if dHt[0] > 15:
2489.                 dHt[0] = 15
2490.             elif dHt[0] < -15:
2491.                 dHt[0] = -15
2492.             DH[t] = dHt[0]
2493.             dP = dHt[0] * K
2494.             P[t] = P[t - 1] + dP
2495.     except Exception:
2496.         pass
2497.
2498.     Pheights.append(P)
2499.     EPheights.append(EP)
2500.     TPheights.append(TP)
2501.     plt.figure()
2502.     plt.plot(P,'-*')
2503.     plt.show()
2504.
2505.
2506.     # In[45]:
2507.
2508.
2509.     fig = plt.figure()
2510.     plt.plot(Pmom[2], label = 'MoM')
2511.     plt.plot(Pcentroid[2], label = 'Centroid')
2512.     plt.legend()
2513.     plt.xlabel('Tiempo')
2514.     plt.ylabel('Presión')
2515.     plt.title('Respuesta del controlador con distintos métodos de
   desdifusión')
2516.     fig.savefig('from600-noR11.png')
2517.     plt.show()
2518.
2519.
```