Trabajo práctico N°2: Razonamiento

Gino Avanzini Emiliano Cabrino Adrián Cantaloube Gonzalo Fernández

Desarrollo y análisis de una base de conocimiento para una planta de reducción de presión de gas

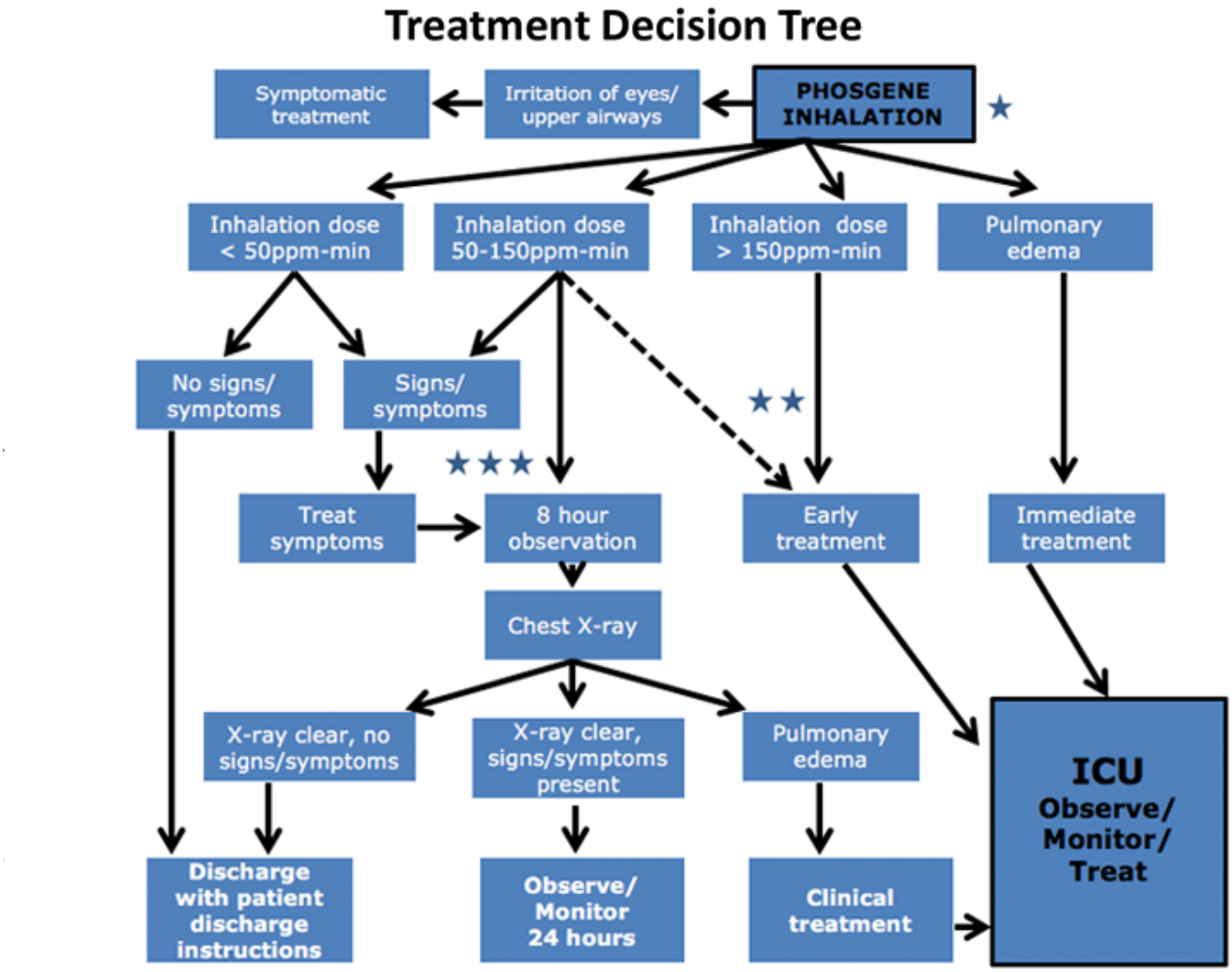
Utilizamos el lenguaje Prolog para generar la base de conocimientos según los lineamientos de la gráfica de acción de la planta.

De forma sencilla, en el archivo "ejercicio1bis.pl" ingresamos las reglas axiomáticas y los ground facts. Las primeras las escribimos de tal manera que Prolog acceda a su algoritmo interno de backtracking permitiendo verificar, primero si el estado en el que nos encontramos analizando es válido, para luego buscar el axioma superior a él (de cumplirse la condición) y verificar el nuevo estado para saber si detenerse y entregar la orden o continuar buscando de forma automática. Los estados se podrán modificar dinamicamente utilizando los comandos assert y retract, modificándolos según lo que ocurra en la planta. Una vez cerrado el programa estos cambios en las condiciones de los estados se perderán.

Se presentó una variante de este problema (el archivo "ejercicio1.pl") sin el uso del backtracking en donde el usuario puede acceder a cualquier parte del árbol formulando la pregunta e indicando si esta es verdadera o falsa según lo que suceda en la planta. El programa le dirá que acción tomar. Esto puede ser o una orden o que se verifique otro axioma. Aquí no serán necesarios los comandos assert y retract ya que los estados se han indicado como variables que el usuario debe ingresar al hacer la pregunta, o sea, que no forman parte de los ground facts. En este enfoque también se agregaron valores de variables como thickness o max regulating pressure para distintos tipos de válvulas (sv y rv) como se sugiere en el enunciado del ejercicio.

Además se realizó otro ejercicio con el uso del árbol de decisión para tratamiento médico ante la inhalación de fosgeno[1]. La base de conocimiento se encuentra en el archivo *treatment.pl*

[1]PHOSGENE: Information on Options for First Aid and Medical Treatment, American Chemistry Council.



★ If information to estimate the actual exposure is not available or not clear, or if there is liquid phosgene or phosgene in solvent exposure to the facial area, then it would be prudent to assume that an exposure of 150 ppm-min or greater occurred.

★★ The dotted line indicated that treatment at levels as low as 50ppm-min may be considered.
Note: Exposure level at which treatment is warranted is undetermined.

★★★ For inhalation doses <50 ppm-min only significant, especially respiratory symptoms need to be observed for 8 hours, and not minor irritant or subjective symptoms.

Notes: "Early Treatment" options when indicated (before pulmonary edema develops), may include sedation, steroids, positive pressure ventilation & *N*-acetylcysteine.
Extracorporeal Membrane Oxygenation (ECMO), as a support option, is a consideration for life threatening pulmonary edema.

Fig. 1. Árbol de decisión para tratamiento por inhalación de fosgeno

Análisis del modelo físico del péndulo invertido

Las constantes del modelo físico son las siguientes:

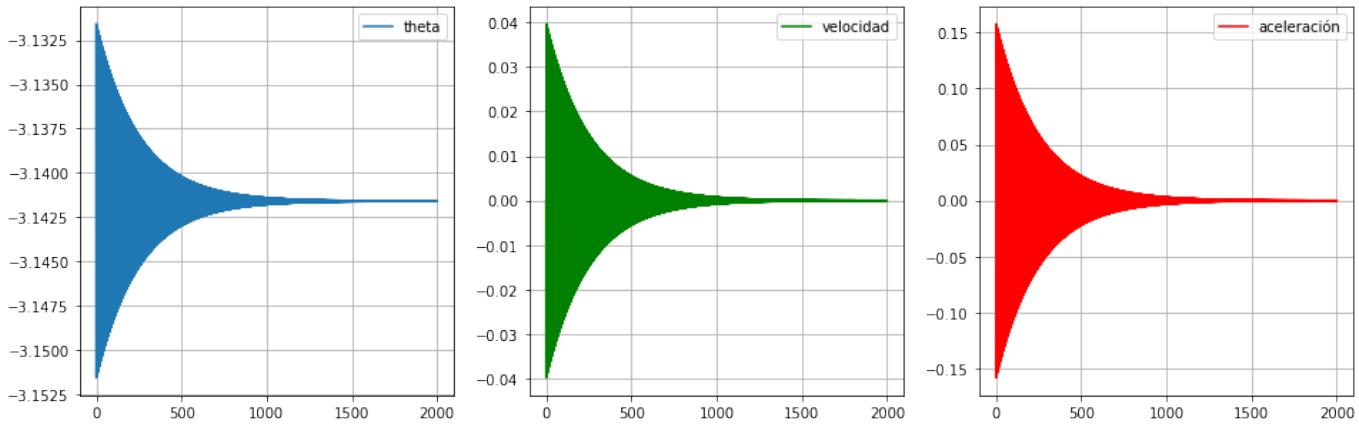
```
from math import sin, cos, pow, pi

#-------------------------------------------------
# Constantes del modelo
#-------------------------------------------------
g = 9.81    # [m/s^2]   Aceleración de la gravedad
F = 0       # [N]       Fuerza externa
m = 0.2     # [Kg]      Masa del péndulo
l = 0.5     # [m]       Longitud del péndulo
M = 2       # [Kg]      Masa del carro
#-------------------------------------------------
```

A continuación se calcula el modelo en el tiempo, sin aplicar ningún tipo de control:

```
def update(x, dt, F):

    x_t = x

    num = g * sin(x[0]) + cos(x[0]) * (- F - m * l * pow(x[1], 2) * sin(x[0])) / (M + m)
    den = l * (4/3 - m * pow(cos(x[0]), 2) / (M + m))

    x_t[2] = num / den

    x_t[1] = x[1] + x[2]*dt
    x_t[0] = x[0] + x[1]*dt + x[2]*pow(dt, 2)/2

    return x_t

MAX_ITER = 2000

dt = 0.001
t = 0

x = [-pi + 0.01, 0, 0]

pos = []
vel = []
acel = []
time = []

while(t < MAX_ITER):

    pos.append(x[0])
    vel.append(x[1])
    acel.append(x[2])

    x = update(x, dt, 0)

    time.append(t)

    t += dt
```

Las gráficas de ángulo θ, velocidad angular y aceleración angular en el tiempo son:

```
from matplotlib import pyplot as plt

fig, (ax0, ax1, ax2) = plt.subplots(1, 3, figsize=(16, 5))

ax0.plot(time, pos, label="theta")
ax0.grid()
ax0.legend(loc="upper right")

ax1.plot(time, vel, color='g', label="velocidad")
ax1.grid()
ax1.legend(loc="upper right")

ax2.plot(time, acel, color='r', label="aceleración")
ax2.grid()
ax2.legend(loc="upper right")

plt.show()
```



A pesar de que el modelo no considera ningún tipo de amortiguamiento, puede observarse cómo el sistema "pierde energía". Esto es debido a que el análisis es discreto, y a lo largo de las iteraciones se acumula un error cada vez que el péndulo llega a los extremos, produciendo un efecto similar al de un amortiguamiento.

Definición de las variables linguísticas y los conjuntos borrosos de entrada

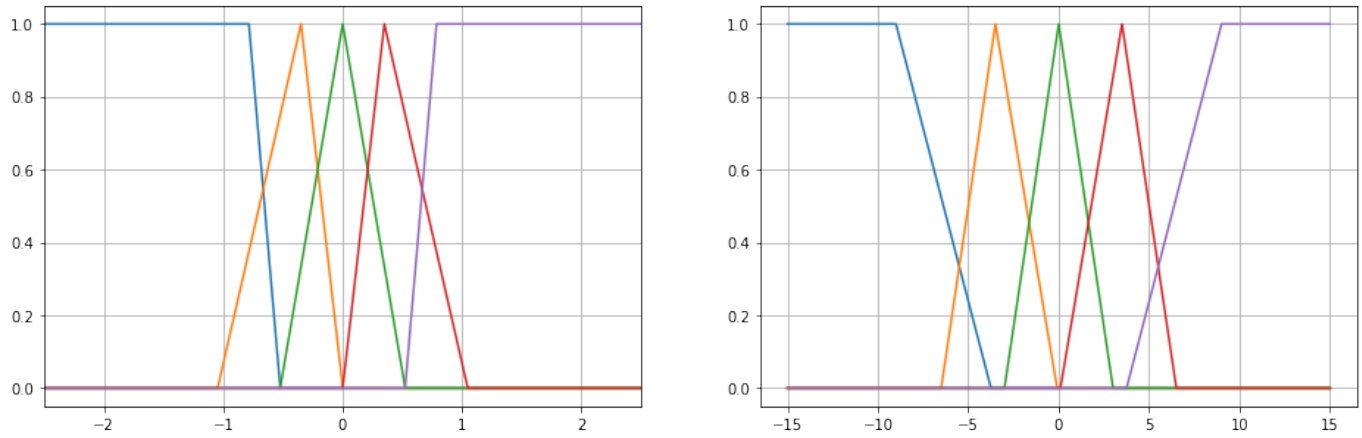
Se elige como variables de entrada el ángulo θ y la velocidad angular. Ambos conjuntos borrosos pueden tomar valores *MN*, muy negativo, *N*, negativo, *Z*, cero, *P*, positivo y *MP*, muy positivo.

```
from numpy import arange
from matplotlib import pyplot as plt
from math import pi
from fuzzy_logic import generate_profile

T_STEP = 0.001      # [rad]
V_STEP = 0.005      # [rad/s]
A_STEP = 0.01
F_STEP = 0.05

theta = []
v = []

ang_vel = 15
force_mag = 10

theta.append(arange(-2*pi, 2*pi+T_STEP, T_STEP))
```

```
v.append(arange(-ang_vel, ang_vel+V_STEP, V_STEP))
for i in range(0, len(v[0])):
    v[0][i] = round(v[0][i], 3)

# Generación de conjuntos borrosos de entrada

# Conjunto borroso de theta:
theta.append({})
theta[1]['MN'] = generate_profile(-pi/4, theta[0], max=-pi/6)
theta[1]['N'] = generate_profile(-pi/9, theta[0], min=-pi/3, max=0)
theta[1]['Z'] = generate_profile(0, theta[0], min=-pi/6, max=pi/6)
theta[1]['P'] = generate_profile(pi/9, theta[0], min=0, max=pi/3)
theta[1]['MP'] = generate_profile(pi/4, theta[0], min=pi/6)

# Conjunto borroso de velocidad angular:
v.append({})
v[1]['MN'] = generate_profile(-9, v[0], max=-3.75)
v[1]['N'] = generate_profile(-3.5, v[0], min=-6.5, max=-0.03*pi)
v[1]['Z'] = generate_profile(0, v[0], min=-3, max=3)
v[1]['P'] = generate_profile(3.5, v[0], min=0.03*pi, max=6.5)
v[1]['MP'] = generate_profile(9, v[0], min=3.75)
```

Obteniendose los siguientes conjuntos borrosos para cada variable:

```
fig, (ax0, ax1) = plt.subplots(1, 2, figsize=(16, 5))
for i in theta[1].values():
    ax0.plot(theta[0], i, label=i)
ax0.grid()
ax0.set_xlim(-2.5, 2.5)

for i in v[1].values():
    ax1.plot(v[0], i, label=i)
ax1.grid()

plt.show()
```



Definición de las variable linguísticas y los conjuntos borrosos de salida

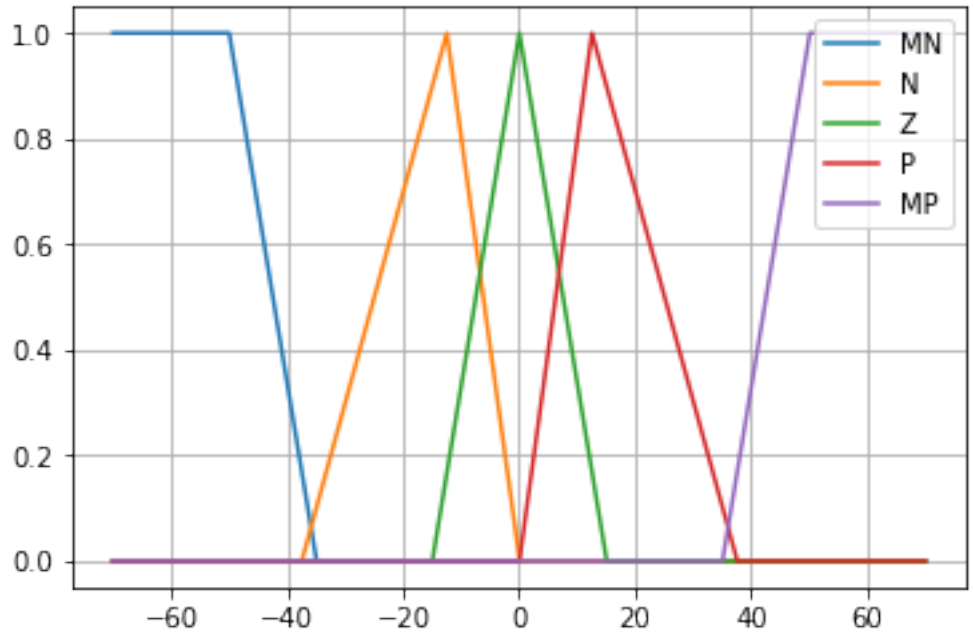
Se elige como variable de salida del sistema la fuerza de acción F sobre el carro (base del péndulo invertido). Los diferentes conjuntos se denominan igual, solo cambia el rango de valores reales que puede tomar:

```
F = []
F.append(arange(-7*force_mag, 7*force_mag+F_STEP, F_STEP))
```

```
# Generación de conjuntos borrosos de salida
F.append({})
F[1]['MN'] = generate_profile(-5*force_mag, F[0], max=-3.5*force_mag)
F[1]['N'] = generate_profile(-1.25*force_mag, F[0], min=-3.75*force_mag, max=0)
F[1]['Z'] = generate_profile(0, F[0], min=-1.5*force_mag, max=1.5*force_mag)
F[1]['P'] = generate_profile(1.25*force_mag, F[0], min=0, max=3.75*force_mag)
F[1]['MP'] = generate_profile(5*force_mag, F[0], min=3.5*force_mag)
```

Obteniendose los siguientes conjuntos borrosos para la variable F:

```
for i in theta[1]:
    plt.plot(F[0], F[1][i], label=i)
plt.grid()
plt.legend(loc="upper right")
```



Implementación de la inferencia borrosa mediante base de reglas tipo if-then

Las base de reglas que se estableció para la inferencia borrosa se expone en la siguiente tabla:

	MN	N	Z	P	MP
MN	MN	MN	MN	N	Z
N	MN	MN	N	Z	P
Z	MN	N	Z	P	MP
P	N	Z	P	MP	MP
MP	Z	P	MP	MP	MP

la cual está implementada en python como un diccionario de diccionarios. Esto debe leerse con la primera key del diccionario como el conjunto de theta y cada key dentro de el diccionario de theta como los conjuntos de theta punto. El valor resultante es un string que caracteriza el conjunto borroso de F. Por ejemplo, R[‘Z’][‘MP’] == ‘MP’ lo que significa que si theta es cero y theta punto es muy positiva entonces la fuerza debe ser muy positiva

```
R = {
    'MN': {'MN': 'MN', 'N': 'MN', 'Z': 'MN', 'P': 'N', 'MP': 'Z'},
    'N': {'MN': 'MN', 'N': 'MN', 'Z': 'N', 'P': 'Z', 'MP': 'P'},
    'Z': {'MN': 'MN', 'N': 'N', 'Z': 'Z', 'P': 'P', 'MP': 'MP'},
    'P': {'MN': 'N', 'N': 'Z', 'Z': 'P', 'P': 'MP', 'MP': 'MP'},
```

```
    'MP': {'MN': 'Z', 'N': 'P', 'Z': 'MP', 'P': 'MP', 'MP': 'MP'}
    }
```

Obtención de la fuerza dadas las condiciones iniciales

Dadas ciertas condiciones iniciales probamos lo que hace el controlador difuso para un delta_t. Primero obtenemos los valores de pertenencia de las condiciones iniciales a cada conjunto borroso de su variable lingüística.

```
from fuzzy_logic import fuzzifier, defuzzifier
from numpy import zeros

cond_inic = [pi/6, pi/4]

value = zeros(3)
value[0] = cond_inic[0]
value[1] = cond_inic[1]
value[2] = 0

Force = 0

mu_theta = fuzzifier(value[0], theta)
mu_thetadot = fuzzifier(value[1], v)

print(mu_theta)
print(mu_thetadot)
```

```
{'MN': 0, 'N': 0, 'Z': 0.0, 'P': 0.7507163323782235, 'MP': 0.0}
{'MN': 0, 'N': 0, 'Z': 0.7366666666666667, 'P': 0.20411160058737152, 'MP': 0}
```

Luego se realiza el cálculo de los antecedentes de cada regla en las que intervienen los valores no nulos de pertenencia que acabamos de calcular.

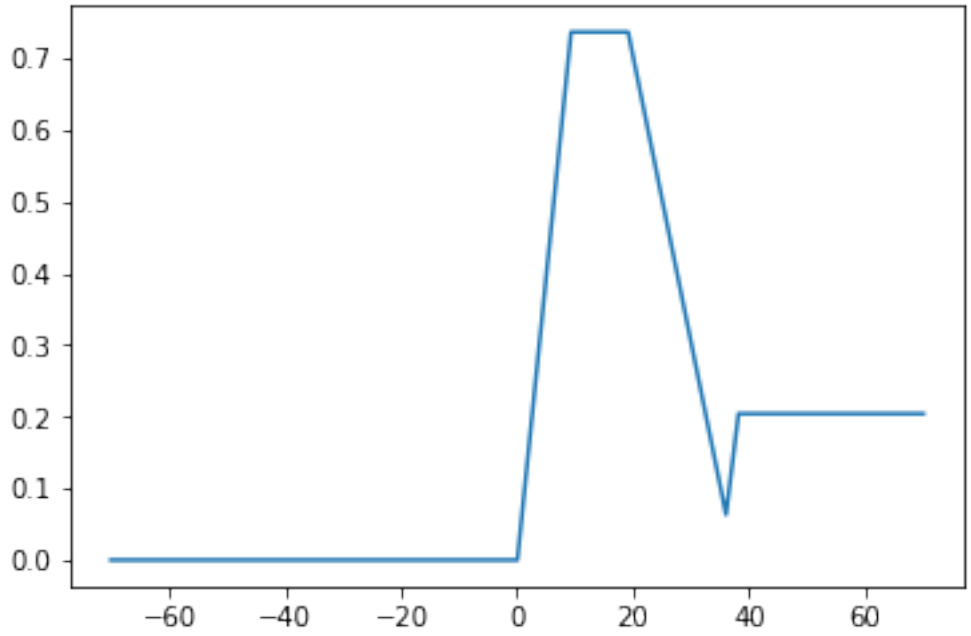
```
v_antec = {'MN': 0, 'N': 0, 'Z':0, 'P': 0, 'MP': 0}

for theta_r in R.keys():
    for thetadot_r in R[theta_r].keys():
        if (mu_thetadot[thetadot_r] == 0) or (mu_theta[theta_r] == 0):
            continue

        antec = min(mu_theta[theta_r], mu_thetadot[thetadot_r])

        if (antec > v_antec[R[theta_r][thetadot_r]]):
            v_antec[R[theta_r][thetadot_r]] = antec
```

Posteriormente ealizamos la implicación de cada regla. Esto genera el truncamiento de los conjuntos borrosos de salida de la variable fuerza. Finalmente encontramos el valor nítido de fuerza usando el desborrosificador de centro de gravedad conjunto truncado.

```
# Implicación/truncado de conjuntos de salida:
F_out = zeros(len(F[0]))

for i in F[1].keys():
    for j in range(0, len(F[0])):

        min_temp = min(F[1][i][j], v_antec[i])

        if (min_temp > F_out[j]):
            F_out[j] = min_temp

plt.plot(F[0], F_out)
```

```
force = defuzzifier(F_out, F[0])

print("fuerza=",force)
```

```
fuerza= 27.149999999994478
```



Controlador difuso

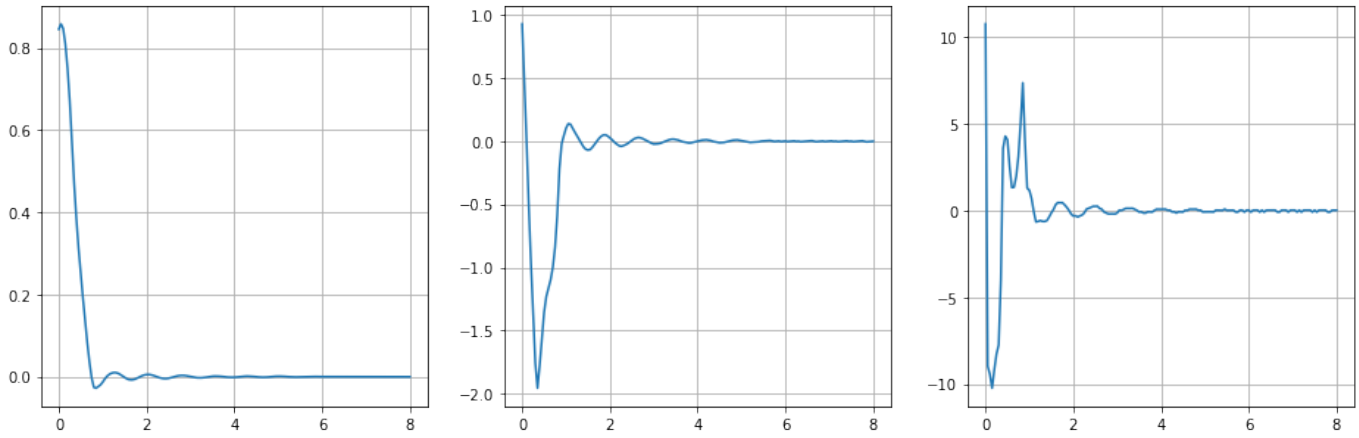
Luego de verificar el funcionamiento del algoritmo difuso pasamos a realizar un control de posición el cual se basa en ejecutar lo antes explicado iterativamente y calculando las nuevas posiciones y velocidades angulares con el modelo del péndulo invertido también ya descripto.

```
from fuzzy_logic import fuzzy_control
from math import pi
from numpy import zeros

cond_inic = [pi/4, pi/8]

x = zeros(3)
x[0] = cond_inic[0]
x[1] = cond_inic[1]
x[2] = 0

pos = []
vel = []
acel = []
time = []

Force = 0
force_hist = []
t = 0
dt = 0.05


while(t < 8):

    x = update(x, dt, Force)
```

```
    pos.append(x[0])
    vel.append(x[1])
    acel.append(x[2])

    Force = fuzzy_control([x[0], x[1]], theta, v, R, F)
    force_hist.append(Force)
    time.append(t)

    t += dt
```

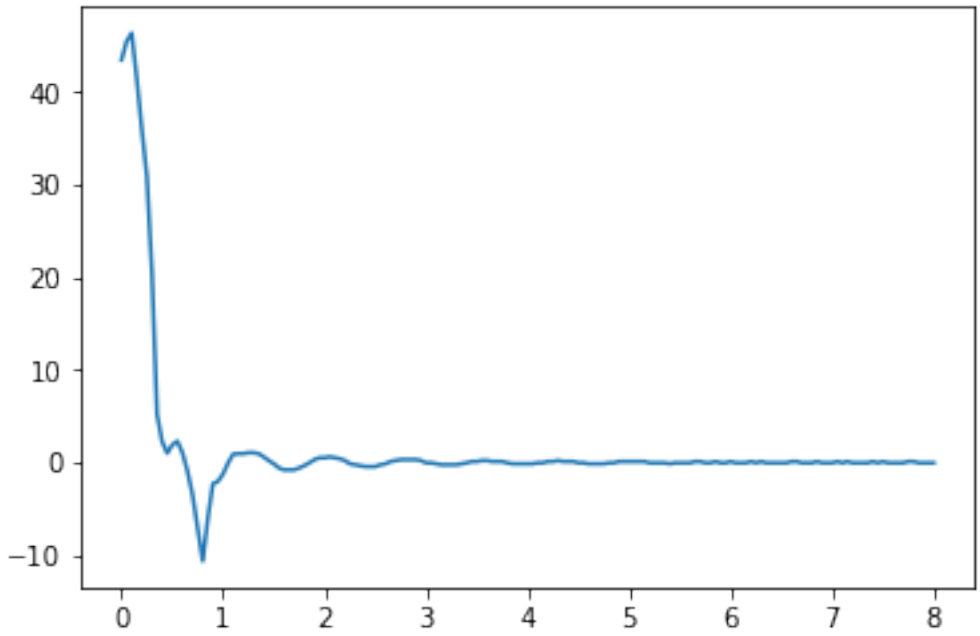
Para un ángulo inicial de pi/4 rad y una velocidad angular inicial de pi/8 rad/s obtenemos estas respuestas de posición, velocidad y aceleración angulares.

```
fig, (ax0, ax1, ax2) = plt.subplots(1, 3, figsize=(16, 5))
ax0.plot(time, pos)
ax0.grid()
ax1.plot(time, vel)
ax1.grid()
ax2.plot(time, acel)
ax2.grid()
```



Además el perfil de fuerzas es el siguiente

```
plt.plot(time, force_hist)
plt.show()
```



Levantar el péndulo

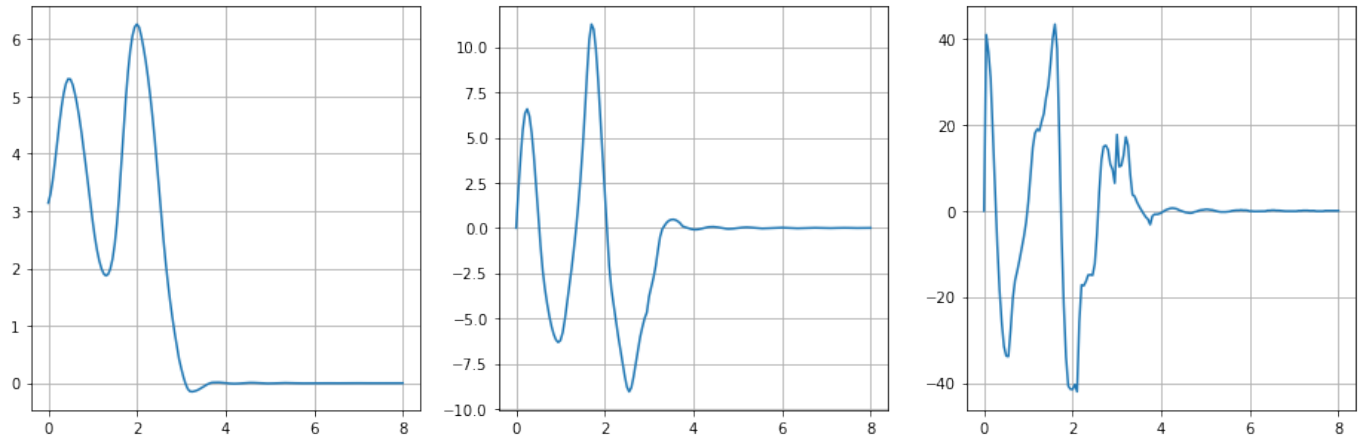
También existe la posibilidad de levantar el péndulo desde una posición inicial vertical, hacia abajo (pi radianes).

```
from fuzzy_logic import fuzzy_control
from math import pi
from numpy import zeros

cond_inic = [pi, 0]

x = zeros(3)
x[0] = cond_inic[0]
x[1] = cond_inic[1]
x[2] = 0

pos = []
vel = []
acel = []
time = []

Force = 0
force_hist = []
t = 0
dt = 0.05


while(t < 8):

    x = update(x, dt, Force)

    pos.append(x[0])
    vel.append(x[1])
    acel.append(x[2])

    Force = fuzzy_control([x[0], x[1]], theta, v, R, F)
    force_hist.append(Force)
```

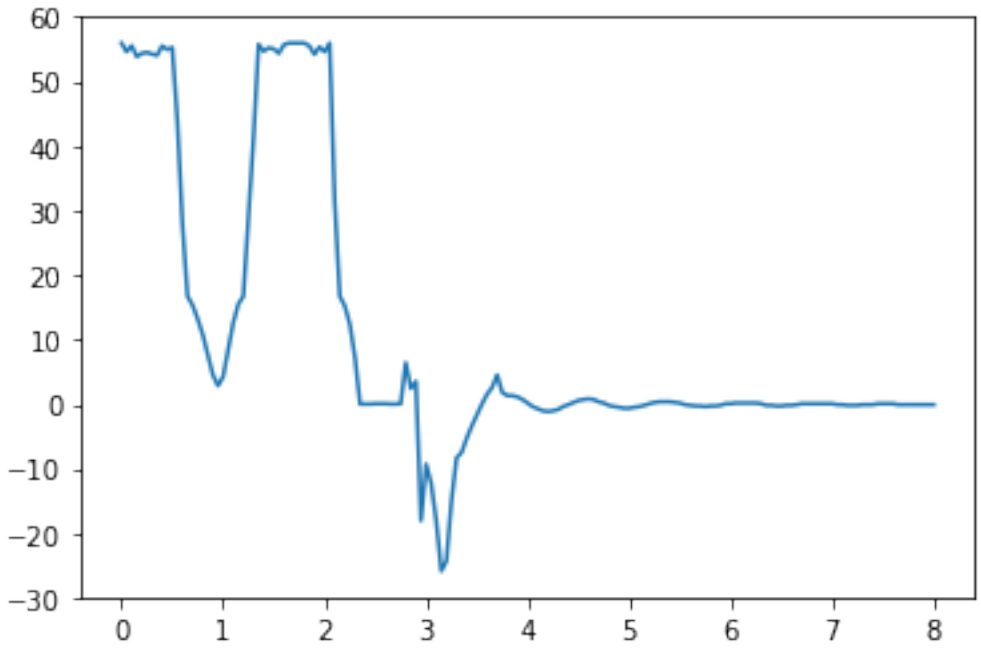
```
    time.append(t)

    t += dt
```

Y las curvas obtenidas de ángulo, velocidad, aceleración y fuerza son las siguientes

```
fig, (ax0, ax1, ax2) = plt.subplots(1, 3, figsize=(16, 5))
ax0.plot(time, pos)
ax0.grid()
ax1.plot(time, vel)
ax1.grid()
ax2.plot(time, acel)
ax2.grid()
```



```
plt.plot(time, force_hist)
plt.show()
```



Ejercicios de planificación con Fast Downward

Dominio de transporte aéreo de cargas

La consigna consiste en modelar en PDDl el dominio de transporte aéreo de cargas y definir algunas instancias del problema para luego encontrar soluciones con el planificador Fast Downward.

El modelado del problema fue basado en el ejemplo de planificación visto en clase.

En el archivo domain3a.pddl se definió el dominio del problema, con los predicados *LOAD*, *PLANE* y *AIRPORT* para definir variables y *in* y *at* para definir estados de las variables. Las acciones definidas fueron:

- *fly*, parámetros *p*, *from* y *to*. Acción en la que el avión *p* vuela del aeropuerto *from* al aeropuerto *to*.
- *load*, parámetros *c*, *p* y *a*. Acción de cargar *c*, en el avión *p*, en el aeropuerto *a*.
- *unload*, parámetros *c*, *p* y *a*. Acción de descargar *c*, en el avión *p*, en el aeropuerto *a*.

En el archivo task3a.pddl se dan las condiciones iniciales del problema en específico, instanciando dos aviones, dos aeropuertos y dos cargas diferentes; y el estado objetivo donde las cargas deben estar en los aeropuertos diferentes al que comienzan.

Solución:

```
(load c2 p2 jfk)
(fly p2 jfk sf0)
(unload c2 p2 sf0)
(load c1 p2 sf0)
(fly p2 sf0 jfk)
(unload c1 p2 jfk)
; cost = 6 (unit cost)
```

Dificultades: No se pudo lograr que el planificador logre paralelizar el problema para que contemple en la solución el uso de ambos aviones (reduciendo el costo temporal). Se concluyó que dado que el planificador Fast Downward solo soporta hasta PDDL 2, no se puede realizar esto en dicha versión del lenguaje.

Planificación de Procesos Asistida por Computadora

La consigna consiste en modelar en PDDL y definir una instancia del problema de planificación de Procesos Asistida por Computadora (CAPP, Computer-Aided Process Planning).

En el archivo domain3b.pddl se definió el dominio del problema, con los predicados *CUERPO*, *HERRAMIENTA*, *ARMARIO*, *HUSILLO*, *DIRECCIÓN*, *AGUJERO* y *RANURA* para definir variables y *en* y *vinculo* (para relacionar un tipo de operación con determinada herramienta) para definir estados de las variables. Las acciones definidas fueron:

- *cambio-herramienta*, parámetros *h1*, *h2*, *a* y *m*. Acción en la que se reemplaza las herramientas *h1* y *h2* que están en el armario *a* y el husillo de la máquina *m* respectivamente.
- *agujerear*, parámetros *p*, *h*, *m* y *a*. Acción de realizar un agujero tipo *a* en el cuerpo *p* con la herramienta *h* en el husillo de la máquina *m*.
- *fresar*, parámetros *p*, *h*, *m* y *r*. Acción de realizar una ranura tipo *r* en el cuerpo *p* con la herramienta *h* en el husillo de la máquina *m*.

En el archivo task3b.pddl se dan las condiciones iniciales del problema en específico, instanciando tres herramientas para realizar tres tipos de gujeros diferentes (brocas), tres tipos de herramientas para realizar tres tipos de ranuras (fresas) y la materia prima; y el estado objetivo donde ya se encuentra el producto final con los tres agujeros y ranuras realizados.

Solución:

```
(fresar p f1 m r1)
(cambio-herramienta b1 f1 a m)
(agujerear p b1 m h1)
(cambio-herramienta b2 b1 a m)
(agujerear p b2 m h2)
(cambio-herramienta b3 b2 a m)
```

```
(agujerear p b3 m h3)
(cambio-herramienta f2 b3 a m)
(fresar p f2 m r2)
(cambio-herramienta f3 f2 a m)
(fresar p f3 m r3)
; cost = 11 (unit cost)
```

Dificulatades: Dado que el planificador Fast Downward soporta PDDL solo hasta segunda versión, no se pudo implementar restricciones blandas de precedencia entre las tareas, pero se expresaría de la siguiente manera:

```
(:goal
...
    (preference precendencia
        (sometime-after (en p r2) (en p r1))
        (sometime-after (en p h1) (en p r1))
    )

...
)
```

Explicitando la preferencia de ejecución de dos tareas en un determinado orden, y estableciendo un métrica de la forma:

```
(:metric (minimize (is-violated precedencia)))
```