PROYECTO 4 Ejemplo avanzado para sistemas de control difuso (proceso químico industrial)

Para un sistema de control difuso típico es común el mantener una determinada variable cerca de un valor especifico. Por ejemplo, puede ser necesario mantener relativamente constante la temperatura de un proceso químico industrial. Para ello, el sistema suele conocer dos cosas:

- El error, o desviación del valor ideal
- La forma en que cambia el error. Esta es la primera derivada matemática; la llamaremos delta

A partir de estos dos valores podemos construir un sistema que actué de forma adecuada. Ahora bien configurar el sistema de control implica utilizar una nueva API del sistema de control para este problema ya que seria demasiado complicado modelarlo manualmente

```
import numpy as np
import skfuzzy.control as ctrl

# El universo disperso hace los calculos mas rapidos, sin sacrificar la precisión
# Aqui solo se incluyen los puntos criticos; hacerlo de mayor resolucion es innecesario

universo = np.linspace(-2, 2, 5)

# Crear las 3 variables difusas, dos entradas una salida

error = ctrl.Antecedent(universo, 'error')
delta = ctrl.Antecedent(universo, 'delta')
salida = ctrl.Consequent(universo, 'salida')

#  Aqui usamos la conveniencia 'automf' para poblar las variables difusas con terminos.
# El kwarg opcional 'names=' nos permite especificar los nombre de nuestros terminos

names = ['nb', 'ns', 'ze', 'ps', 'pb']
error.automf(names = names)
delta.automf(names = names)
salida.automf(names = names)
```

Definir reglas complejas

Este sistema tiene un complicado conjunto de reglas totalmente conectadas que se definen a continuación (nota salida podría llamarse outtput si como pasa con delta o error ser una variable privativa de la función)

```
# definicion del conjunto de reglas complejas

regla0 = ctrl.Rule(antecedent = ((error['nb'] & delta['nb']) |
        (error['ns'] & delta['nb']) |
        (error['nb'] & delta['ns'] )), consequent = output['nb'], label = 'rule nb')

regla1 = ctrl.Rule(antecedent = ((error['nb'] & delta['ze']) |
        (error['nb'] & delta['ps']) |
        (error['ns'] & delta['ns']) |
        (error['ns'] & delta['ze']) |
        (error['ze'] & delta['ns']) |
        (error['ze'] & delta['nb']) |
        (error['ps'] & delta['nb'] )), consequent = output['ns'], label = 'rule ns')

regla2 = ctrl.Rule(antecedent = ((error['nb'] & delta['pb']) |
        (error['ns'] & delta['ps']) |
        (error['ze'] & delta['ze']) |
        (error['ps'] & delta['ns']) |
        (error['pb'] & delta['nb'] )), consequent = output['ze'], label = 'rule ze')

regla3 = ctrl.Rule(antecedent = ((error['ns'] & delta['pb']) |
        (error['ze'] & delta['pb']) |
        (error['ze'] & delta['ps']) |
        (error['ps'] & delta['ps']) |
        (error['ps'] & delta['ze']) |
        (error['pb'] & delta['ze']) |
        (error['pb'] & delta['ns'] )), consequent = output['ps'], label = 'rule ps')

regla4 = ctrl.Rule(antecedent = ((error['ps'] & delta['pb']) |
        (error['pb'] & delta['pb']) |
        (error['pb'] & delta['ps'] )), consequent = output['pb'], label = 'rule pb')
```

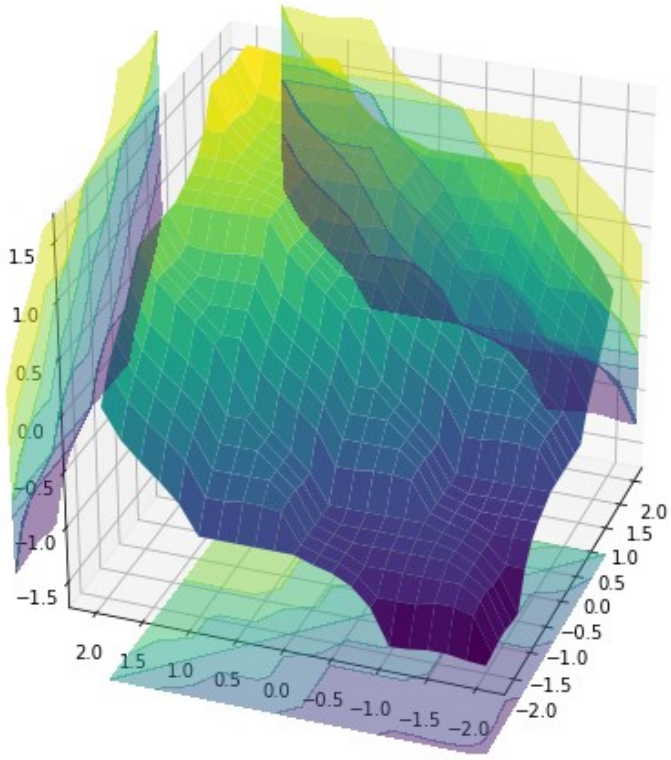
A pesar del extenso conjunto de reglas, el nuevo marco del sistema de control difuso se ejecutara en mili-segundos. A continuación añadimos estas reglas a un nuevo ControlSystem y definimos un ControlSystemSimulation para ejecutarlo.

```
# A continuacion añadimos estas reglas a un nuevo ControlSystem y definimos un ControlSystemSimulation para ejecutarlo.

sistema = ctrl.ControlSystem(rules = [regla0, regla1, regla2, regla3, regla4])

#Mas tarde tenemos la intencion de ejecutar este sistema con un conjunto de 21 X 21 entradas,
#por lo que permitimos muchas mas que una unica ejecucion, antes de que los resultados se vacien.
#Las ejecuciones posteriores volverian un 1/8 del tiempo

sim = ctrl.ControlSystemSimulation(sistema, flush_after_run = 21 * 21 + 1)
```

Ver el espacio de control

Con el útil uso de Matplotlib y simulaciones repetidas, podemos observar el aspecto de toda la superficie del sistema de control en tres dimensiones

```python
# Con el util uso de Matplotlib y simulaciones repetidas,
# podemos observar el aspecto de toda la superficie del sistema de control en tres dimensiones
# podemos simular a mayor resolucion con total precision

upsampled = np.linspace(-2, 2, 21)
x, y = np.meshgrid(upsampled, upsampled) # matrices de coordenadas
# z = np.zeros_like                       # matrices cuyos componentes son ceros
z = np.zeros((21, 21))                    # Inicializa z como una matriz de ceros de tamaño 21x21

# Recorre el sistema 21 X 21 veces para recoger la superficie de control
for i in range(21):
    for j in range(21):
        sim.input['error'] = x[i, j]
        sim.input['delta'] = y[i, j]
        sim.compute()
        output_value = sim.output['output']  # Almacena el valor de output en una variable
        z[i, j] = output_value  # Asigna el valor almacenado en z[i, j]

# realizando el plot con las librerias listadas
fig = plt.figure(figsize=(8, 8))
ax = fig.add_subplot(111, projection='3d')

surf = ax.plot_surface(x, y, z, rstride = 1, cstride = 1, cmap = 'viridis', linewidth = 0.4, antialiased = True)

cset = ax.contourf(x, y, z, zdir = 'z', offset = -2.5, cmap = 'viridis', alpha = 0.5) # sombreado bajo la grafica que proyecta lo
cset = ax.contourf(x, y, z, zdir = 'x', offset = 3, cmap = 'viridis', alpha = 0.5) # que pasa en los dos ejes
cset = ax.contourf(x, y, z, zdir = 'y', offset = 3, cmap = 'viridis', alpha = 0.5)

ax.view_init(30, 200) #Un valor de 30 significa que la cámara está elevada a 30 grados por encima del plano horizontal1.
                      #Un valor de 200 significa que la cámara ha girado 200 grados en sentido antihorario desde el eje x1.
```



Reflexiones finales

En este ejemplo se han utilizado una serie de técnicas nuevas y avanzadas que pueden resultar útiles en el diseño practico de sistemas difusos:

- Un sistema altamente disperso(máximamente disperso)
- Control de nombres de términos generados por automf
- Un conjunto de reglas largo y lógicamente complicado, en el que se respeta el orden de las operaciones
- Control del vaciado de la cache al crear un ControlSystemSimulation, que puede ajustarse en función de las restricciones de memoria.
- Ejecución repetida de un ControlSystemSimulation
- Creación y visualización de una superficie de control 3D