模糊控制及其應用

作業一

系級：電機碩一

學號：107521067

姓名：李慶鴻

題目：

1. 請用定理12.2設計一個模糊規則庫來近似一個非線性函數 ，其中，且準確度設定為，並畫出
2. 請用定理12.3設計一個模糊規則庫來近似一個非線性函數 ，其中，且準確度設定為，並畫出

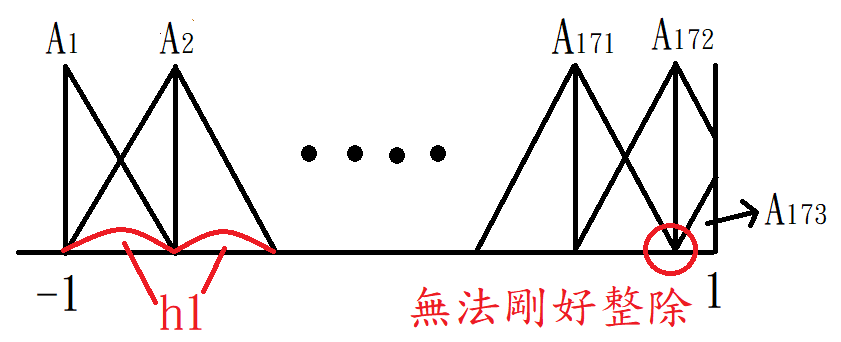
以上每小題請附上(1)程式碼(請用python或matlab)，並需說明(2)共有幾個fuzzy sets且有幾條規則，(3)每條規則的form 是如何? 劃出  與分別的圖及比較。

答案：

兩小題都是用同一種非線性方程式g(x)，因此首先是建立正確的x和g(x)的值，由題目中可以看到，而取樣點為每間隔0.01便取一點，故x值總共取200點，再代入函數g(x)進行計算，求得正確的函數g(x)曲線。

1. 根據課本上定理12.2，接者要計算g(x)對x作一次微分的最大值，經過計算後約為8.5925，題目中已經有給準確度設定必須為，因此根據公式：

可以計算出約為0.0116，以此為間隔對x進行劃分，可以劃出172個區域，因為無法剛好整除，故要再加上頭尾，因此假設總共有173個fuzzy sets，並且每個fuzzy set對應到一個規則，所以總共有173個規則。



圖：Fuzzy sets分布

* 規則數：

：若x是，則y是

：若x是，則y是

：若x是，則y是

.

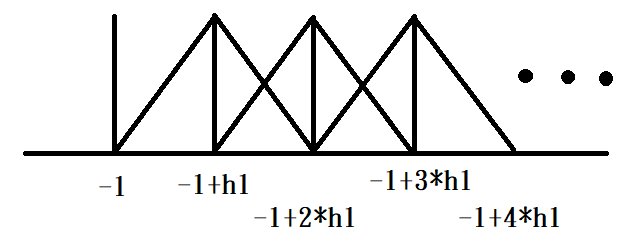
.

.

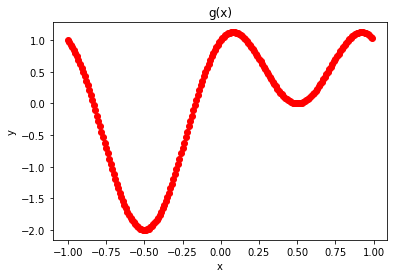
：若x是，則y是

：若x是，則y是

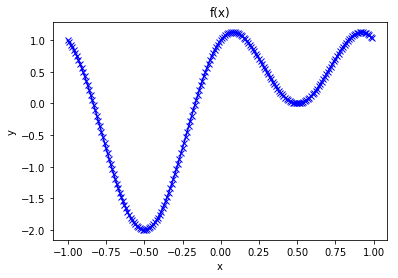
* 每條規則的形式皆為等腰三角形，三角形的底皆為2倍的，中央點為從最負點-1開始，以為間隔往最正端+1推，如下圖所示：



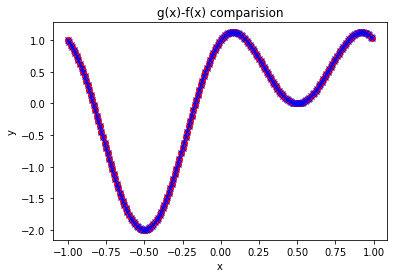
* 結果圖：
* 原始函數g(x)



* Fuzzy逼近之函數f(x)



* 兩者比較



1. 根據課本上定理12.3，接者要計算g(x)對x作二次微分的最大值，經過計算後約為49.34802，題目中已經有給準確度設定必須為，因此根據公式：

可以計算出約為0.12732，以此為間隔對x進行劃分，可以劃出16個區域，因為無法剛好整除，故要再加上頭尾，因此假設總共有17個fuzzy sets，並且每個fuzzy set對應到一個規則，所以總共有17個規則。

* 規則數：

：若x是，則y是

：若x是，則y是

：若x是，則y是

.

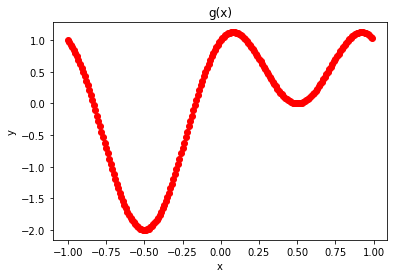
.

.

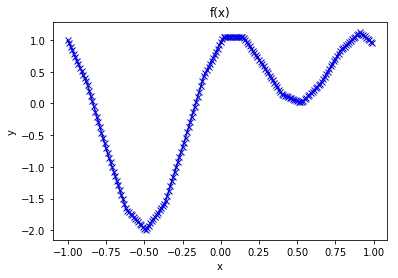
：若x是，則y是

：若x是，則y是

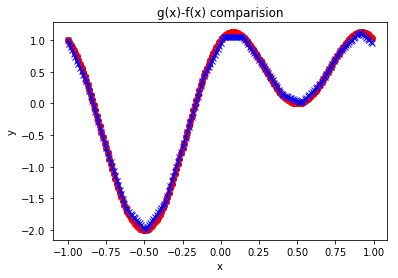
* 而Fuzzy set的分布、形式皆以(a)小題相同，採用等腰三角形的形式。
* 結果圖：
  + 原始函數g(x)



* + Fuzzy逼近之函數f(x)



* + 兩者比較



附錄：程式碼

1. **小題**

import numpy as np #載入NumPy函式庫

import matplotlib.pyplot as plt #載入Matpltlib函式庫

import math #載入math函式庫

#定義原始方程式

def fn(x):

y = math.sin(math.pi\*x)+math.cos(2\*math.pi\*x)

return y;

#原始方程式之微分

def fn\_diff(x):

y = -2\*math.pi\*math.sin(2\*math.pi\*x) + math.pi\*math.cos(math.pi\*x)

return y;

#使用Fuzzy逼近的計算

def fuzzy\_calculate(x,X\_degree):

#for j in range(len(x)):

for i in range(len(X\_degree)):

if X\_degree[i] > x:

break;

Fire\_R1\_value = 1-((x-X\_degree[i-1])/(X\_degree[i]-X\_degree[i-1]))

Fire\_R2\_value = 1-((x-X\_degree[i])/(X\_degree[i-1]-X\_degree[i]))

molecule = fn(X\_degree[i-1])\*Fire\_R1\_value +

fn(X\_degree[i])\*Fire\_R2\_value

Denominator = Fire\_R1\_value + Fire\_R2\_value

Final\_fn = molecule / Denominator

return Final\_fn;

#建立x軸之個數

def x\_number(start,stop,step):

x = {}

x['x'] = np.arange(start,stop,step)

x['x\_num'] = int((stop - start)/step)

return x;

#建立原始方程式正確的輸出值

def g\_function(x,x\_num):

g = np.zeros(x\_num)

for i in range(len(x)):

g[i] = fn(x[i])

return g;

#尋找原始方程式微分一次後之最大值

def g\_diff(x,x\_num):

g\_diff = np.zeros(x\_num)

for i in range(len(x)):

temp = fn\_diff(x[i])

g\_diff[i] = abs(temp)

g\_diff\_max = max(g\_diff)

return g\_diff\_max;

#主程式

start = -1

stop = 1

step = 0.01

Epsilon = 0.1

x = x\_number(start,stop,step)

g = g\_function(x['x'],x['x\_num'])

g\_diff\_max = g\_diff(x['x'],x['x\_num'])

h1 = Epsilon/g\_diff\_max #算出h1

#算出需要多少規則數

Total\_rule\_num = 2/h1

Total\_rule\_num2 = int(Total\_rule\_num)

if Total\_rule\_num > Total\_rule\_num2:

Total\_rule\_num2 = Total\_rule\_num2+2

Total\_rule\_num = Total\_rule\_num2

#建立各規則的歸屬值最大值H(A)=1所對應的x值

X\_degree = np.zeros((Total\_rule\_num,1))

for i in range(Total\_rule\_num):

X\_degree[i] = -1+i\*h1

#計算Fuzzy逼近之函數值

fuzzy\_fn = np.zeros(len(x['x']))

x\_temp = x['x']

for j in range(len(x\_temp)):

fuzzy\_fn[j] = fuzzy\_calculate(x\_temp[j],X\_degree)

#畫圖

fig = plt.figure()

plt.plot(x['x'],g, c='r',marker='o')

plt.plot(x['x'],fuzzy\_fn, c='b',marker='x')

plt.xlabel('x')

plt.ylabel('y')

plt.title('g(x)-f(x) comparision');

plt.show()

print('OK')

-----------------------------------------------------------------------------------------------------------------

1. **小題**

import numpy as np #載入NumPy函式庫

import matplotlib.pyplot as plt #載入Matpltlib函式庫

import math #載入math函式庫

#定義原始方程式

def fn(x):

y = math.sin(math.pi\*x)+math.cos(2\*math.pi\*x)

return y;

#原始方程式之二次微分

def fn\_diff(x):

y = -math.pi\*math.pi\*math.sin(math.pi\*x) + 4\*math.pi\*math.pi\*math.cos(2\*math.pi\*x)

return y;

#使用Fuzzy逼近的計算

def fuzzy\_calculate(x,X\_degree):

#for j in range(len(x)):

for i in range(len(X\_degree)):

if X\_degree[i] > x:

break;

Fire\_R1\_value = 1-((x-X\_degree[i-1])/(X\_degree[i]-X\_degree[i-1]))

Fire\_R2\_value = 1-((x-X\_degree[i])/(X\_degree[i-1]-X\_degree[i]))

molecule = fn(X\_degree[i-1])\*Fire\_R1\_value + fn(X\_degree[i])\*Fire\_R2\_value

Denominator = Fire\_R1\_value + Fire\_R2\_value

Final\_fn = molecule / Denominator

return Final\_fn;

#建立x軸之個數

def x\_number(start,stop,step):

x = {}

x['x'] = np.arange(start,stop,step)

x['x\_num'] = int((stop - start)/step)

return x;

#建立原始方程式正確的輸出值

def g\_function(x,x\_num):

g = np.zeros(x\_num)

for i in range(len(x)):

g[i] = fn(x[i])

return g;

#尋找原始方程式微分二次後之最大值

def g\_diff(x,x\_num):

g\_diff = np.zeros(x\_num)

for i in range(len(x)):

temp = fn\_diff(x[i])

g\_diff[i] = abs(temp)

g\_diff\_max = max(g\_diff)

return g\_diff\_max;

#主程式

start = -1

stop = 1

step = 0.01

Epsilon = 0.1

x = x\_number(start,stop,step)

g = g\_function(x['x'],x['x\_num'])

g\_diff\_max = g\_diff(x['x'],x['x\_num'])

#算出h1

h1\_square = Epsilon/g\_diff\_max\*8

h1 = h1\_square\*\*(1/2)

#算出需要多少規則

Total\_rule\_num = 2/h1

Total\_rule\_num2 = int(Total\_rule\_num)

if Total\_rule\_num > Total\_rule\_num2:

Total\_rule\_num2 = Total\_rule\_num2+2

Total\_rule\_num = Total\_rule\_num2

#建立各規則的歸屬值最大值H(A)=1所對應的x值

X\_degree = np.zeros((Total\_rule\_num,1))

for i in range(Total\_rule\_num):

X\_degree[i] = -1+i\*h1

#計算Fuzzy逼近之函數值

fuzzy\_fn = np.zeros(len(x['x']))

x\_temp = x['x']

for j in range(len(x\_temp)):

fuzzy\_fn[j] = fuzzy\_calculate(x\_temp[j],X\_degree)

#畫圖

fig = plt.figure()

plt.plot(x['x'],g, c='r',marker='o')

plt.plot(x['x'],fuzzy\_fn, c='b',marker='x')

plt.xlabel('x')

plt.ylabel('y')

plt.title('g(x)-f(x) comparision');

plt.show()

print('OK')