بدنه اصلی کد شامل یک کلاس است که در سه بخش زیر مختصرا توضیح داده خواهد شد:

```
class Wangmendel():

    def __init__(self,x1_train,x2_train,y_train,x1_test,x2_test,y_test,no_mf):
        self.X1 = x1_train
        self.X2 = x2_train
        self.Y= y_train
        self.X11= x1_test
        self.X22= x2_test
        self.Y22= y_test
        self.number_mf= no_mf

    def generate_mf(self):
        self.x_range = np.linspace(min(self.X1),max(self.X1), self.number_mf)
        self.y_range = np.linspace(min(self.Y), max(self.Y), self.number_mf)
        self.X1_mf=[]
        self.X2_mf=[]
        self.Y_mf=[]
        self.center=[]
        for i in range(len(self.x_range) -3):
            self.X1_mf.append(fuzz.trapmf(self.x_range, [self.x_range[i],self.x_range[i+1],self.x_range[i+2],self.x_rang
            self.X2_mf.append(fuzz.trapmf(self.x_range, [self.x_range[i],self.x_range[i+1],self.x_range[i+2],self.x_rang
            self.Y_mf.append(fuzz.trapmf(self.y_range, [self.y_range[i],self.y_range[i+1],self.y_range[i+2],self.y_range
            self.center.append( (self.y_range[i+1]+self.y_range[i+2])/2)
        # plot X1
        for i in (self.X1_mf):
            plt.plot(self.x_range,i,color='coral')
        plt.xlabel('X1')
        plt.ylabel('membership function u(x1)')
        plt.show()
        # plot X2
        for i in (self.X2_mf):
            plt.plot(self.x_range,i,color='gold')
        plt.xlabel('X2')
        plt.ylabel('membership function u(x2)')
        plt.show()
        # plot Y
        for i in (self.Y_mf):
            plt.plot(self.y_range,i,color='navy')
        plt.xlabel('Y')
        plt.ylabel('membership function u(y)')
        plt.show()
```

پس از تعریف تابع __init__ برای گرفتن مقادیر ورودی تابع generate_mf نوشته شده است که از داده های ورودی استفاده میکند و به تعداد پارامتر ورودی کلاس با استفاده از کتابخانه skfuzzy تابع عضویت ذوزنقه ایی تولید میکند. بعد از تولید mfها آن هارا جداگانه برای سه متغیر X۱,X۲,Y رسم میکنیم.

```
def generate_rule(self):
    rule_2000 = []
    for i in range(len(self.X1)):
        listX1 , listX2 ,listY = [],[],[]
        for j in range(len(self.X1_mf)):
            listX1.append(fuzz.interp_membership(self.x_range,self.X1_mf[j],self.X1[i]))
            listX2.append(fuzz.interp_membership(self.x_range,self.X2_mf[j],self.X2[i]))
            listY.append(fuzz.interp_membership(self.y_range,self.Y_mf[j],self.Y[i]))

        x1Max , x2max , ymax = np.argmax(listX1) , np.argmax(listX2) , np.argmax(listY)
        degree = listX1[x1Max] * listX2[x2max] * listY[ymax]
        rule_2000.append([x1Max,x2max , ymax , degree])

    print('primitive Rules:')
    print('we have',len(rule_2000),'rules!','\n')

   # eliminate duplicates and conflicts here!!!!
    rule_2000.sort(key= lambda rule_2000:rule_2000[0],reverse=True)
    rule_2000.sort(key= lambda rule_2000:rule_2000[1],reverse=True)
    rule_2000.sort(key= lambda rule_2000:rule_2000[2],reverse=True)
    rule_2000.sort(key= lambda rule_2000:rule_2000[3],reverse=True)

    f_rule=[]
    alternative=[]
    alternative.append(rule_2000[0][0:-1]) # -1 for drop degree of rule!
    f_rule.append(rule_2000[0])
    for i in rule_2000[1:]:
        a= i[0]
        b= i[1]
        c= i[2]
        d= i[3]
        if [a,b,c]in alternative:
            continue
        elif [a,b,c]not in alternative:
            alternative.append(i[0:-1])
            f_rule.append(i)


    self.f_rule2=[]
    self.f_rule2.append(f_rule[0])
    alternative2=[]
    alternative2.append(f_rule[0][0:-2])
    for z in f_rule[1:]:
        a= z[0]
        b= z[1]
        c= z[2]
        d= z[3]
        if [a,b] in alternative2:
            continue
        elif [a,b] not in alternative2:
            self.f_rule2.append(z)
            alternative2.append(z[0:-2])
    print('finall rules: ')
    print('we have',len(self.f_rule2),'non conflict and non duplicate rule' )
    K=pd.DataFrame(self.f_rule2)
    print(K)
```

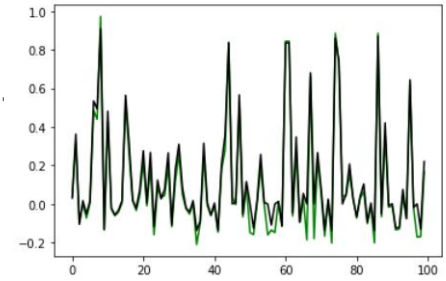
در این جا برای ۲۰۰۰عدد داده ورودی ۲۰۰۰ قانون (حداکثر تعداد) تولید میکنیم که برای این کار ابتدا تک تک داده ها را با توابع عضویت برخورد میدهیم و در هرتابع عضویتی که داده مقدار حداکثر داشته باشد شماره آن تابع عضویت برای آن عدد گرفته میشود. درمرحله بعد قوانین تکراری حذف میشوند و در نهایت قوانین بدون تکرار به تعدادی قانون بدون برخورد(non conflict) تبدیل میشوند(حداکثر به تعداد ضرب توابع عضویت های ورودی).

```
    def train_result(self):
        rule_2 = np.array(self.f_rule2,dtype=int)

        Y_predict = []

        for i in range(len(self.X1)):

            list1 , list2  = [],[]
            for j in range(len(self.X1_mf)):
                list1.append(fuzz.interp_membership(self.x_range,self.X1_mf[j],self.X1[i]))
                list2.append(fuzz.interp_membership(self.x_range,self.X2_mf[j],self.X2[i]))

            sorat,makhraj = 0,0
            for k in range(len(rule_2)):
                sorat += list1[rule_2[k][0]] * list2[rule_2[k][1]] * self.center[rule_2[k][2]]
                makhraj += list1[rule_2[k][0]] * list2[rule_2[k][1]]
            if makhraj==0:
                Y_predict.append(0)
            else:
                Y_predict.append(sorat/makhraj)

        mse_train = sum((self.Y - Y_predict) **2) / (2*len(self.Y))

        # draw plot
        plt.figure()
        plt.plot(range(100),self.Y[0:100],color= 'green')
        plt.plot(range(100) , Y_predict[0:100],color='black')
        plt.show()
        print('the train error is :',mse_train)
    def test_result(self):
        rule_2 = np.array(self.f_rule2,dtype=int)
        Y_test = []

        listRules = []
        for i in range(len(self.X11)):

            list1 , list2 ,listY = [],[],[]
            for j in range(len(self.X1_mf)):
                list1.append(fuzz.interp_membership(self.x_range,self.X1_mf[j],self.X11[i]))
                list2.append(fuzz.interp_membership(self.x_range,self.X2_mf[j],self.X22[i]))

            sorat,makhraj = 0,0
            for k in range(len(rule_2)):
                sorat += list1[rule_2[k][0]] * list2[rule_2[k][1]] * self.center[rule_2[k][2]]
                makhraj += list1[rule_2[k][0]] * list2[rule_2[k][1]]
            if makhraj==0:
                Y_test.append(0)
            else:
                Y_test.append(sorat/makhraj)

        mse_test = sum((self.Y22 - Y_test) **2) / (2*len(self.Y22))
```

در نهایت پس از تولید قوانین برای سنجیدن کیفیت قاتون های ساخته شده این قانون ها را یکبار روی داده های آموزش ویکبار روی داده های تست اعمال میکنیم که به ترتیب مقادیر خروجی Y_predict و Y_test ساخته میشوند، سپس با کمک مقادیر اصلی(ورودی) و مقادیر به دست آمده، خطا (mse) قابل اندازه گیری است.

## train part

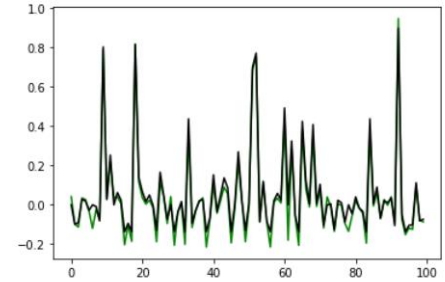
In [11]:
```
1 wang.train_result()
```




```
the train error is : 0.0009316010821081626
```

## test part

In [12]:
```
1 wang.test_result()
```




```
the test error is : 0.0010295196981456164
```