# 1 代码说明

本文设计了一个主程序：main\_sensor\_fusion，和一个函数程序：cal\_fuse。主程序里面包含主干部分和绘图部分，函数程序包含数据生成函数gen，检测概率计算函数cal，非0逻辑矩阵函数No\_zero\_value，单传感器判决函数fus\_seq，多传感融合函数fusion。

# 2 附录I(主程序源代码)

下面给出程序源代码，完整Python文件见附件。

# -\*- coding: utf-8 -\*-

# ！/usr/bin/python3.9.7

# @Time   : 2022.4.18

# @Author : Yue Gaofeng

# @version: V2.0

# @Des    : NP\_frame-based to learn optimal fusion and draw ROC curve. Muddle headed.

from time import \*

import cal\_fus

import numpy as np

import matplotlib.pyplot as plt

import random

if \_\_name\_\_ == "\_\_main\_\_":

    begin\_time = time()                                  # Measure processing time of program

    N, epoch = 6, 10000                                  # Set sensor number and length of individual sequence

    M = 20                                               # Set length of measurement sequences

    Sigma = [1,1.1,1.2,1.3,1.4,1.5]                      # Noise variances of six sensors

    Yita=0.01                                            # Slice operation

    Pdset = np.linspace(0,1,int(1/Yita+1))               # False alarm probability（PF） constraint value

    Alpha = 0.3                                          # PF of local sensor

    Chi\_mtx = np.zeros([N,epoch,M])                      # Finish data initialization

    Chi\_mtx = cal\_fus.gen(N,Chi\_mtx,Sigma,epoch,M)       # Generate dataset

    print('Sequence fusion is working!!!')

    Z\_mtx = cal\_fus.fus\_seq(Chi\_mtx,Sigma,Pdset,M,\

                            epoch,N)                     # fusing sequences of same sensor

    Pd\_single = cal\_fus.cal(Z\_mtx,Sigma,Pdset,N,epoch)   # Compute dection probability

    print('Mutiple-sonsor fusion is working!!!')

    pd\_N,pf\_N = cal\_fus.fusion(Pdset,Alpha,Pd\_single,N)  # N sensors' data fusion

    pd\_N\_2, pf\_N\_2 = cal\_fus.fusion(Pdset,Alpha,\

                                Pd\_single, int(N/2))     # N/2 sensors' data fusion

    plt.figure()                                         # Drawing ROC curve

    for i in range(N):                                   # Sensor 1-7 ROC curve

        colour = ''.join(random.sample('0123456789ABCDEF', 6))

        plot1 = plt.plot(Pdset,Pd\_single[i,:],'#'+colour,linestyle='-.')

    plot7 = plt.plot(pd\_N\_2,pf\_N\_2,'r','-')              # Three sensors fusion ROC curve'

    plot8 = plt.plot(pd\_N,pf\_N,'b','-')                  # Six sensors fusion ROC curve'

    plt.grid(linestyle='-.')

    plt.xlabel('False alarm probability', fontsize=14)

    plt.ylabel('Detection probability', fontsize=14)

    plt.title('ROC Curve Analysis', fontsize=14)

    plt.xlim([-0.02, 1.0])

    plt.ylim([-0.02, 1.05])

    plt.xticks([0.0,0.2,0.4,0.6,0.8,1.0], fontsize=14)   # Set range of X-axis

    plt.yticks([0.0,0.2,0.4,0.6,0.8,1.0], ['0.0','0.2',\

                '0.4','0.6','0.8','1.0'], fontsize=14)   # Set range of Y-axis

    plt.legend(['Sensor1','Sensor2','Sensor3','Sensor4','Sensor5',\

                   'Sensor6','Three sensors fusion','Six sensors fusion'],\

                       fontsize = 14, loc='best')

    end\_time = time()

    run\_time = end\_time - begin\_time

    print('Working is over! Running time:', run\_time)

    #plt.savefig('Fig1.png',dpi = 600)                   # Save figure

    plt.show()

# -\*- coding: utf-8 -\*-

# ！/usr/bin/python3

# @Time   : 2022.3.18

# @Author : Yue Gaofeng

# @version: V2.0

# @Des    : Generate raw data and calculate detection probability of simple sensor

#         : Mutiple-sensor fusion. So hard!!!

import itertools

from math import sqrt

from re import A

from turtle import shape

import numpy as np

from scipy.stats import norm

from tqdm import tqdm

def gen(N, Chi\_mtx, Sigma, epoch, M):                      # Initialization and generate data

    for i in range(N):

        for j in range(epoch):

            A, B = 2, sqrt(Sigma[i])                      # A is amplitude and B is standard deviation

            Chi\_mtx[i,j,:] = np.random.normal(A, B, M)    # Obtaining measurement matrix

    return Chi\_mtx

def cal(Z\_mtx, Sigma, Pdset, N, epoch):               # Find the PD of mutiple-sensors given PF

    L = len(Pdset); Gamma = np.zeros([N,L])

    pd\_single = np.zeros([N, L], dtype=float)         # Number of one (0-1 distribution)

    for i in tqdm(range(N)):

        for j in range(L):

            Gamma[i, j] = norm.ppf(1-Pdset[j], 0, sqrt(Sigma[i])) # Inverse of CDF and get PD given PF

            s = 0.0

            for k in range(epoch):

                if Z\_mtx[i,k] > Gamma[i,j]: s+=1.0   # Likelihood ratio judgment, caculative sum

                else: s+=0.0

                pd\_single[i, j] = s/(k+1)            # Calculate

    return pd\_single

def No\_zero\_value(value\_list, logical\_list):  # Saving non-zero elements but delete corresponding to 0

    L = len(value\_list)

    back\_list = []

    for i in range(0, L):

        if logical\_list[i] == 1: back\_list.append(value\_list[i])  # Save where index is true

    return back\_list

def fus\_seq(Chi\_mtx,Sigma,Pdset,M,epoch,N):     # fusing sequences of same sensor

    L = len(Pdset)

    Gamma = np.zeros([1,L])

    pd\_m = np.zeros([N, epoch], dtype=float)

    for i in tqdm(range(N)):

        Chi\_mtx = np.resize(Chi\_mtx[i],(epoch,M))

        for j in range(L):

            Gamma[0, j] = norm.ppf(1-Pdset[j], 0, sqrt(Sigma[i])) # Inverse of CDF and get PD given PF

            for m in range(M):

                s = 0.0

                for k in range(epoch):

                    if Chi\_mtx[k,m] > Gamma[0,j]: s+=1.0   # Likelihood ratio judgment, caculative sum

                    else: s+=0.0

                    pd\_m[i,k] = s/epoch                   # Calculate

    return pd\_m

def fusion(Pdset,Alpha,Pd\_single,N):        # Diffrent sensor fusion

    L = len(Pdset)                          # Initialize local variables , L = 101

    Pro\_if = np.ones([3,N], dtype=float)

    pd\_tmp = np.ones([1,N], dtype=float)

    delta = np.ones([1,2\*\*N], dtype=float)

    p01\_newp01 = np.ones([4,2\*\*N], dtype=float)

    Arrange = np.array([i for i in itertools.product([0, 1], repeat = N)])   # Arrange combinations

    for i in range(2\*\*N):                                 # Deal various arrange

        for j in range(N):                                #To address N sensors

            No\_zero = ((Pdset[0:L] == Alpha).astype(int)).tolist()  # Logical metrix

            pd\_tmp[0,j] = Pd\_single[j, No\_zero.index(max(No\_zero))] # Get index of max value

            if  Arrange[i,j] == 0:                        # Conditional probability(Con\_pro)

                Pro\_if[0,j] = (1-pd\_tmp[0,j])/(1-Alpha)   # Miss probability: beta/(1-alpha)

                Pro\_if[1,j] =  1-Alpha                    # H0: Con\_pro of single sensor likelihood ratio

                Pro\_if[2,j] = 1-pd\_tmp[0,j]               # H1: Con\_pro of single sensor likelihood ratio

            else:

                Pro\_if[0,j] = pd\_tmp[0,j]/Alpha

                Pro\_if[1,j] = Alpha

                Pro\_if[2,j] = pd\_tmp[0,j]

            delta[0,i]  = np.prod(Pro\_if[0,:])            # Multi-sensor union likelihood ratio

            p01\_newp01[0,i]  = np.prod(Pro\_if[1,:])       # H0: Con\_pro of Multi-sensor union likelihood ratio

            p01\_newp01[1,i]  = np.prod(Pro\_if[2,:])       # H1: Con\_pro of Multi-sensor union likelihood ratio

    idx = delta.argsort()                                 # Sored from max to min to likelihood

    p01\_newp01[2] = p01\_newp01[0,idx[0,:]]                # Re-ordering

    p01\_newp01[3] = p01\_newp01[1,idx[0,:]]

    b2\_c2 = np.zeros([2, 2\*\*N], dtype=float)

    pf\_pd = np.zeros([2,L], dtype=float)

    b2\_c2[0,:] = np.cumsum(p01\_newp01[2,:])             # Cumulative sum to form novel array

    b2\_c2[1,:] = np.cumsum(p01\_newp01[3,:])             # Cumulative sum by element in row

    b2\_new = [0]+list(b2\_c2[0,:])+[1]

    c2\_new = [0]+list(b2\_c2[1,:])+[1]

    for i in range(L):

        metric = ((b2\_new <= Pdset[i]).astype(int))            # Logical Matrix

        back\_list = No\_zero\_value(b2\_new, metric)              # Delete the column corresponding to 0

        pf\_pd[0,i], id = max(back\_list), np.argmax(back\_list)  # Returns index and maximum

        pf\_pd[1,i] = c2\_new[id]                                # Returns pd sorted by idex list

    return pf\_pd[1,:].tolist(),pf\_pd[0,:].tolist()