```python
# Imports
import numpy as np
import scipy as sp
import pandas as pd
from scipy.stats import truncnorm
import scipy.stats as st
import ultranest


##################################################

#Functions
## Multi-indexing Function
def IndexPixEp(df):
    df['npixel'] = df['healpix'].copy()
    df['Epoch_index']= df['Epochs'].copy()
    df.set_index(['npixel',  'Epoch_index'], inplace=True) # Indexing 2 levels
    df.sort_index(inplace=True)  # ascending order of index (healpix)
    return(df)
## Minimum (68%) confidence interval function
def minconf(SampledNXSV,numberofbins):
    logh, loge = np.histogram(SampledNXSV, bins=numberofbins)      #logh: counts␣
 ↪in each bin, loge: bin size or bin intervals
    logbins = np.logspace( np.log10(np.min(SampledNXSV)), np.log10(np.
 ↪max(SampledNXSV)), len(loge))  # Constructing the bins

    lgv = loge[0:-1]+(loge[1:]-loge[0:-1])/2.0     # USEFUL to constrain SENSIBLE␣
 ↪upper index values
    # (loge[1:]-loge[0:-1]) : difference of neighbouring bin intervals
    # (loge[1:]-loge[0:-1])/2.0 : half the difference between neighbouring bin␣
 ↪intervals
    # lgv : shifting every bin interval by its semidistance to the neighbouring␣
 ↪one
    imode = np.argmax(logh)       # returns indices of the max element of the␣
 ↪array
    il, iu = intPDF(lgv, logh, imode)   # returns the lower and upper INDEX␣
 ↪values that contain most massive 68%
    if(il==imode):       # if lowest index is the index with highest counts
        fl = 0.000001          # np.min(SampledNXSV)  # lowest sampled value
        mode = 0.000001          # lgv[imode]         # value in the middle of␣
 ↪highest bin
        fu = lgv[iu]    # value in the middle of the upper bin
    else:
        mode = lgv[imode]  #value in the middle of the highest bin
        fl = lgv[il]
        fu = lgv[iu]
```

```
    return mode, fl, fu
## Integrating the probability density function (from histogram)
def intPDF(lgflux, prob, imode, CL=0.6826):
    probN = prob / prob.sum()  # Normalising the binned counts collection
    iu = imode                 # Upper & lower index equates to index of bin
 ↪with highest counts
    il = imode
    C = probN[imode]           # Number of counts in bin with highest counts
    while True:
        #print il, iu, C, lgflux[il], lgflux[iu], lgflux[imode], lgflux[0]
        if(C>CL):              # if fraction of counts by itself is higher than
 ↪68% we break
            break
        if( il-1>=0 and iu+1<lgflux.size-1 ):  #stepping one index lower and one
 ↪higher (sensible index values)
            if(probN[il-1]>=probN[iu+1]):  # if fraction of counts in one bin
 ↪lower is >= one bin higher
                C = C + probN[il-1]        # we add to the normalised counts C
 ↪the counts of lower bin
                il = il - 1                # we proceed one index lower that
 ↪before
            else:                          # if fraction of counts in one bin
 ↪lower is < one bin higher
                C = C + probN[iu+1]        # we add to the normalised counts C
 ↪the counts of higher bin
                iu = iu + 1                #and step one higher/ BIN COUNTS
 ↪behave as AREA of pdf
        if(il-1<0 and iu+1<lgflux.size-1): # if non-sensical Lower Index value,
 ↪then proceed to higher bins
            C = C + probN[iu+1]
            iu = iu + 1
        if(il-1>=0 and iu+1>=lgflux.size-1):  # if non-sensical Upper Index
 ↪value, then proceed to lower bins
            C = C + probN[il-1]
            il = il - 1
    return il, iu;                # return the INDEX values

###################################################

# Data & Indexing
df_PN_spectro = pd.read_csv('/home/mado/XMM-XXL data/Reduced_PN_spectro.csv')
df_PN_photo = pd.read_csv('/home/mado/XMM-XXL data/Reduced_PN_photo.csv')
df_PN_bin3epo = pd.read_csv('/home/mado/XMM-XXL data/Binned_PN_3epoch.csv')
df_PN_bin3epo = IndexPixEp(df_PN_bin3epo)

###################################################
```

```
# Bayesian

Chains_NXSV_SN3 = {}
Chains_CR_SN3 = {}
index_pixel =0

## Prior
def Prior(cube):      # USE INVERSE CDF of prior  (inverse cdf of log uniform)
    theta = cube.copy()
    theta[0] = 10**(cube[0] * (np.log10(0.3) - np.log10(0.000001)) + np.log10(0.
↪000001))
    theta[1] = 10**(cube[1] * (np.log10(15) - np.log10(0.000001)) + np.log10(0.
↪000001))
    return theta

## Log Likelihood of each source (Importance sampling)
for pxli in  df_PN_bin3epo[df_PN_bin3epo['SNR_band6_stacked_PN']>3].index.
↪get_level_values('npixel').unique():
    # pxli= healpix index - each index is one AGN source
    df_working = df_PN_bin3epo[df_PN_bin3epo['SNR_band6_stacked_PN']>3].
↪loc[(pxli)].reset_index() # slice of dataframe
    # Integration points
    LargeN = 1000
    # Parametres' names
    param_names = ['mu', 'NormXSV']

    def LogLike(theta):  # df = df_PN_bin3epo for most calculations␣
↪integrationpoints = 50000 for our calc
        CR_mean = theta[0]
        sigmaTot =np.sqrt( theta[1]* (CR_mean**2))
        low = 0.0
        high = CR_mean +10*sigmaTot
        ProposedCRvaluesPos =  np.random.normal(CR_mean, sigmaTot, size=LargeN)
        ProposedCRvaluesPos[ProposedCRvaluesPos <0]=0.0
        LikelihoodSource = np.zeros(shape=(len(df_working)))
        for ObservIndex in range(len(df_working)):
            # Lightcurve point values
            texp = df_working.band6_exposure[ObservIndex]
            frac = df_working.eef[ObservIndex]
            Bgr = df_working.band6_bck_counts[ObservIndex]
            Ni=  df_working.band6_src_counts[ObservIndex]
            # Lambda linspace
            ProposedLambdaValues = ProposedCRvaluesPos*texp*frac + Bgr
            x = ProposedLambdaValues
            sample = st.poisson.pmf(Ni, x) #the poisson quantity in I.S.␣
↪summation
```

```
            S = sample.sum()  # the sum in I.S. formula
            I = S/LargeN     # I.S. estimation of integral
            if I>0:
                LikelihoodSource[ObservIndex]= np.log(I)    # log likelihood of␣
 ↪each lightcurve point
            else:
                LikelihoodSource[ObservIndex]= -200

        TotLsource =  np.sum(LikelihoodSource)
        return TotLsource

    sampler = ultranest.ReactiveNestedSampler(param_names, LogLike, Prior)
    result = sampler.run()
    sampler.print_results()

    SampledCRs = (result['samples'][:,0]).tolist()
    SampledNXSVs = (result['samples'][:,1]).tolist()
    hlpx = df_working.healpix[0]
    Ones_CR = np.ones(shape= len(SampledCRs))
    Ones_NXSV = np.ones(shape= len(SampledNXSVs))

    A1 = ( hlpx * Ones_CR).tolist()
    B1 = SampledCRs

    A2 = ( hlpx * Ones_NXSV).tolist()
    B2 = SampledNXSVs

    Chains_CR_SN3[index_pixel] = pd.DataFrame({'healpix': A1, 'CHAIN_MeanCR':␣
 ↪B1} )
    Chains_NXSV_SN3[index_pixel] = pd.DataFrame({'healpix': A2, 'CHAIN_NXSV':␣
 ↪B2} )

    index_pixel = index_pixel +1

df_MeanCR_chain = pd.concat(Chains_CR_SN3)
df_NXSV_chain = pd.concat(Chains_NXSV_SN3)

###################################################

# Handling chains
## Mean Count Rate
Dict_Bayesian_CR_SN3 = []
for pxli in  df_MeanCR_chain.index.get_level_values('npixel').unique():  #␣
 ↪iterate sources
    df_working = df_MeanCR_chain.loc[(pxli)].reset_index()              #␣
 ↪dataframe of source
```

```
    SampledCR = df_working.CHAIN_MeanCR.to_numpy()                          # array␣
 ↪chain

    A1 = np.mean(SampledCR)                          # Mean
    B1 = np.std(SampledCR)                           # Standard Deviation
    C1 = np.std(SampledCR)/(np.sqrt(len(SampledCR)))  # Standard Error of Mean

    Dict_Bayesian_CR_SN3.append({'healpix': pxli, 'Mean_MeanCR': A1,␣
 ↪'StDev_MeanCR':B1, 'StEr_MeanCR':C1})

MeanCR_SN3_Bayes = pd.DataFrame(Dict_Bayesian_CR_SN3, columns =['healpix',␣
 ↪'Mean_MeanCR', 'StDev_MeanCR', 'StEr_MeanCR'])
## NXSV
Dict_Bayesian_NXSV_SN3 = []

df_work = df_NXSV_chain.loc[(pxli)].reset_index()    # dataframe of source
SampledNXSV = df_work.CHAIN_NXSV.to_numpy()

for pxli in  df_NXSV_chain.index.get_level_values('npixel').unique():  # iterate␣
 ↪sources
    df_working = df_NXSV_chain.loc[(pxli)].reset_index()               #␣
 ↪dataframe of source
    SampledNXSV = df_working.CHAIN_NXSV.to_numpy()                          # array␣
 ↪chain
    mode, fl, fu = minconf(SampledNXSV,500)

    A2 = np.mean(SampledNXSV)                            # Mean
    B2 = np.std(SampledNXSV)                             # Standard Deviation
    C2 = np.std(SampledNXSV)/(np.sqrt(len(SampledNXSV)))  # Standard Error of␣
 ↪Mean
    D2 = np.median(SampledNXSV)                          # Median
    E2 = np.quantile(SampledNXSV,[0.16,0.5,0.84])[0]     # Lower value of␣
 ↪Confidence Interval 68%
    F2 = np.quantile(SampledNXSV,[0.16,0.5,0.84])[2]     # Upper value of␣
 ↪Confidence Interval 68%
    G2 = mode      # Mode
    H2 = fl        # low value of CI 68%
    I2 = fu        # upper value of CI 68%

    Dict_Bayesian_NXSV_SN3.append({'healpix': pxli, 'Mean_NXSV': A2,␣
 ↪'StDev_NXSV':B2, 'StEr_NXSV':C2, 'Median_NXSV':D2, 'Low68CI_NXSV_np':E2,␣
 ↪'Up68CI_NXSV_np':F2, 'Mode_NXSV':G2, 'Low68CI_NXSV':H2, 'Up68CI_NXSV':I2})
```

```
NXSV_SN3_Bayes_B = pd.DataFrame(Dict_Bayesian_NXSV_SN3, columns =['healpix',␣
↪'Mean_NXSV', 'StDev_NXSV', 'StEr_NXSV', 'Median_NXSV', 'Low68CI_NXSV_np',␣
↪'Up68CI_NXSV_np', 'Mode_NXSV', 'Low68CI_NXSV', 'Up68CI_NXSV'])
NXSV_SN3_Bayes_B
```

[ ]:

```

```