oof_bk.py

def func(x,p):

输入：
X，表示线性函数的自变量
P，表示线性函数的系数

func 函数实现了简单的线性函数，其中 a 和 b 是通过参数向量 p 传递进来的。
这种线性函数的形式为：y = a * x + b，其中 y 是输出值，x 是输入值。

输出：
返回线性函数计算得到的结果，即 a * x + b

总结：这段代码的目的是解析一个 XML 文件，该文件包含有关天文观测的配置参数。具体而言，它包括了一些与观测相关的信息，观测（SRC）、接收机（RCV）、频率（FREQ）、赤经（SAH）、赤纬（DEC）、校准（TCAL）、观测者（OBSER）、观测带宽（OBSW）、本振频率（LO），以及最大（狭（MAXL）等

obsConfigReader.py

class obsXmlReader:
def __init__(self,fileName):

输入：
fileName：一个要读取的 XML 文件路径的字符串。

类包含一个构造函数 __init__，该构造函数接收一个参数 fileName，表示要读取的 XML 文件的路径。构造函数将该路径存储在类的属性 self.fileName 中。

def loadObsXml(self):

输入：
self：类的实例。
self.fileName：类的实例属性，表示要加载的 XML 文件的路径。

打开 XML 文件，如果文件无法打开，则返回加载失败的信息。使用 Qt 的 QDomDocument 类解析 XML 文件内容。从 XML 文件的根节点开始，提取其中的各种天文观测参数，包括源名(SRC)、接收机类型(RCV)、频率(FREQ)等等。将提取的参数存储在类的实例属性中，例如 self.SRC、self.RCV、self.obw 等等。最后，将加载的参数封装成一个元组，其中包含源名、接收机类型、观测带宽、赤经、赤纬、温度校准(TCAL)、观测者(OBSER)、频率通道末端标识(bankEnd)、频率通道对应的中心频率(restFreq)、局放观测中心频率(Lo)以及最大延迟(maxL)等信息。如果 XML 文件中有多个频率通道，这里会生成一个列表 self.bankEnd，表示每个通道的标识。

输出：
如果成功加载 XML 文件，返回一个元组 (True, (obs_parameters))，其中 obs_parameters 是包含加载的天文观测参数的元组。
如果加载失败，返回一个元组 (False, ())。

总结：这个程序主要是处理天文观测数据和射电望远镜bk校准数据，最终将它们合并成一个数据集

$f(x) = a \cdot e^{-\frac{(x-m)^2}{2\sigma^2}}$

其中：
$f(x)$ 是函数的值。
a 是振幅（峰值）。
x 是自变量。
m 是均值，表示函数的中心。
σ 是标准差，控制曲线的宽度。

mergeData.py

def fun(x, p):

输入：
x：输入参数
p包含了k、b、a、m、n(k,b是线性函数系数，a,m,n是高斯函数的系数)

这个函数的目的是实现一个数学模型，包含线性部分和高斯函数部分。它可以用于拟合实际数据，其中 x 是输入，p 包含了需要优化的参数。通过调整参数，可以使用这个函数来拟合某种模式的数据。

输出：
r = k*x + b + a*np.exp(-(x-m)**2/(2*n**2))线性函数加高斯函数计算所得的r

def residuals(p,x,z):

输入：
x：输入参数
p包含了k、b、a、m、n(k,b是线性函数系数，a,m,n是高斯函数的系数)
z:实际观测到的数据

这个函数的主要目的是计算拟合后的值与实际观测数据 z 之间的差异。在拟合问题中，优化算法通常试图通过调整参数 p 以最小化这些残差。因此，residuals 函数通常被用作最小二乘法等优化算法的目标函数。在最小二乘法中，优化的目标是最小化残差的平方和。

输出：
残差，即观测数据 z 与通过拟合模型 fun 计算得到的预测值之间的差异。
residuals=z-fun(x,p)

class MergeData:
def __init__(self,antF,dtF,rcvTy,Amp,Lo,Bw,Sta,\
Freq,telF = "BB",multibeam = False):

输入：
antF：天线文件？
dtF：数据文件？
rcvTy：接收机类型。
Amp：放大器增益。
Lo：本振。
Bw：带宽。
Sta：站点。
Freq：频率。
telF：望远镜文件？（可选参数，默认为字符串 "BB"）。
multibeam：是否是多波束观测（可选参数，默认为 False）

定义了一个MergeData类通过构造函数 __init__，将输入的参数值分配给类的实例变量，以便在后续的代码中使用。

输出：
antF：天线文件？
dtF：数据文件？
rcvTy：接收机类型。
Amp：放大器增益。
Lo：本振。
Bw：带宽。
Sta：站点。
Freq：频率。
telF：望远镜文件？
multibeam：是否是多波束观测。

def getMergeData(self):

输入：类实例存在的数据

getMergeData 方法的主要功能是调用类中的 fitData 方法，并返回其结果。

输出：通过调用 self.fitData() 方法的返回值

def fitData(self):

输入：
使用了 self.mergeTelBk() 方法的返回值。
方位角 (mAz)、俯仰角 (mEl)、记录 (record)、射电望远镜的水平角度 (aH)、垂直角度 (aV)、左右旋的 (ITsys、IData、lmCal 和 rTsys、rData、rmCal)。

调用了 self.mergeTelBk() 方法，获取了一些数据，包括方位角 (mAz)、俯仰角 (mEl)、记录 (record)、射电望远镜的水平角度 (aH)、垂直角度 (aV)，左右旋的 (ITsys、IData、lmCal 和 rTsys、rData、rmCal)。根据站点信息（self.Sta）选择了合适的数据作为拟合的输入。例如如果站点是 "AZ"，则选择了 aH 作为 fx，使用 self.subFitData 方法对左右旋的数据进行了拟合，得到拟合结果。
构建了三个元组，包含了原始数据、左右旋的拟合数据最终返回这个包含三个元组的元组

输出：
返回一个包含三个元组的元组：(rawData, lFit, rFit)。
rawData：包含了一系列原始的天文和射电望远镜的数据，包括方位角 (mAz)、俯仰角 (mEl)、记录 (record)、射电望远镜的水平角度 (aH)、垂直角度 (aV)，左右旋的 (ITsys、IData、lmCal 和 rTsys、rData、rmCal)。
lFit：包含了与左旋相关的拟合信息，包括 lRMS（误差）、lhpbw（半功率波束宽度）、lpter（主瓣峰值电平？）、lta（副瓣峰值电平？）、lfitData（拟合后的左旋数据）。
rFit：包含了与右旋相关的拟合信息，包括 rRMS、rhpbw、rpter、rta、rfitData

总结：这个程序主要是处理并存储了记录日期Tm、偏移量dx、俯仰角El Az、方位角

rawTelDataReader.py

class TelRawData:
def __init__(self,dataFile):
self.dataFile = dataFile

定义了一个类 TelRawData，该类的初始化方法将传入的数据文件路径或名称存储在类的实例变量 self.dataFile 中。

def getTelData(self,Fg):

输入：
self: 这是一个类方法，所以第一个参数是类实例本身。
Fg: 一个表示条件的参数，用于筛选数据，fg = datalist[6] 这一行代码是从 datalist 列表中提取第 7 个元素，并将其赋值给变量 fg。

打开指定路径的文件（self.dataFile ）存储了文件路径）。初始化时间、方位角，俯仰角，x轴偏移和y轴偏移量的空列表用于存储文件提取的数据。然后遍历文件的每一行。对每一行进行处理，将其拆分成数组列表。提取数据列表中的各个数据包括日期时间、方位角（azimuth）、俯仰角（elevation）、偏移量（dx、dy）、以及一个标志（fg）。根据条件（Fg是否等于 fg）筛选出符合条件的数据。将从datatime中提取的年、月、日、时、分、秒使用datib库计算将日期时间转换为%Y-%m-%d%H:%M:%S这种格式，并将所有的数据存储到对应的列表中。最后，将这些列表转换为 NumPy 数组，并返回这些数组的元组。

输出：
记录日期（Tm）、偏移量X（dx\）、偏移量Y（dy\）、方位角（Az）、俯仰角（El）

总结：这个程序主要是处理计算了左右旋的系统温度Tsys、左右旋的校准Tcal、左右旋的增益、左右旋的校准均值和左右选开关频率的均值

rawBkDataReader.py

class DIBASRawData
def __init__(self,dataFile,rcvTy,Amp,Lo,Bw,nWin,\
maxTm,minTm):

输入：一系列参数
dataFile: 数据文件的路径
rcvTy: 接收类型
Amp: 放大器的增益（Low、High、No）
Lo: 本振
Bw: 带宽
nWin: 窗口数量
maxTm:最大积分时间？
minTm:最小积分时间？

这个类的目的是创建一个数据对象，其中包含一系列属性，这些属性代表了不同的数据信息。类的实例化需要提供一些参数，这些参数在初始化方法（__init__方法是一个特殊的方法，用于初始化类的实例）中被用于设置实例的属性。

输出：类的实例属性
dataFile: 数据文件的路径
rcvTy: 接收类型
Amp: 放大器的增益（Low、High、No）
Lo: 本振
Bw: 带宽
nWin: 窗口数量？
LG 和 RG: 左右旋增益，两个属性都被初始化为 1.0
centFreq: 中心频率，初始化为 0.0
一系列空列表
Record：记录时间，
lonx, loffx,:左旋开关的功率？
ronx, roffx:右旋开关的功率？
lData,rData:左右旋的数据
一些浮点数变量
lmCal, rmCal:左右旋校准因子
ITsys, rTsys:左右旋系统温度
maxTm:最大积分时间？
minTm:最小积分时间？

def load8kHead(self)

输入：
self.dataFile: 一个成员变量，表示 FITS文件的路径。
self.nWin: 类的一个成员变量，用作索引来获取subFreq中的值。
self.Lo:类的一个成员变量，用作计算self.centFreq的一部分。

创建一个存储子频率的空列表。使用pyfits库打开指定路径的FITS文件从文件头中提取相关信息，包括频道数，参考像素坐标、观测带宽、通道带宽以及 **系统参数信息**。
将提取的信息赋值给类的成员变量。计算中心频率 self.centFreq，使用 self.nWin 作为索引从 subFreq 中获取值，并加上 self.Lo。可计算出中心频率的值。

输出：
self.nchan: FITS文件头中提取的整数；通道数。
self.crpix: FITS文件头中提取的浮点数；像素中心的坐标。
self.obsbw: FITS文件头中提取的浮点数；观测带宽（以GHz为单位）。
self.chbw: FITS文件头中提取的浮点数；通道带宽（以GHz为单位）。
self.centFreq: 通过计算得到的中心频率。

def getRedData(self):

输入：
self.record: 类的成员变量；记录
self.ITsys: 类的成员变量；左旋系统温度
self.lData:类的成员变量；左旋数据
self.lmCal: 类的成员变量；左旋校准数据。
self.rTsys:类的成员变量；右旋系统温度
self.rData:类的成员变量；右旋数据
self.rmCal: 类的成员变量；右旋校准数据

将类的成员变量组织成一个包含三个元组的元组。第一个元组包含了类的记录，第二个元组包含了左旋相关数据的 Tsys、Data 和 mCal。第三个元组包含了右旋相关数据的 Tsys、Data 和 mCal。返回这个包含三个元组的元组作为方法的输出。

输出：
返回一个元组，其中包含三个元组。
第一个元组包含一个值，即self.record的值。
第二个元组包含三个值，分别是 self.ITsys、self.lData 和 self.lmCal 的值。
第三个元组包含三个值，分别是 self.rTsys、self.rData 和 self.rmCal 的值。

def getTsys(self):

输入：
self.lofx, self.lonx：左旋开关功率，这些变量是类的成员变量；
self.roffx, self.ronx：右旋开关功率，这些变量是类的成员变量；
self.LG, self.RG: 左右旋增益，这些变量也是类的成员变量

初始化四个空列表 (LOFFOFF, LOFFON, ROFFOFF, ROFFON左右旋开关频率？）用于存储数据。定义用于切片的索引范围 (rp0, lp1, rp0, rp1)。将原始数据切片成四个不同的数组。将切片得到的数据组合到相应的列表(LOFFOFF 和 LOFFON、ROFFOFF 和 ROFFON)。然后计算它们的均值，继续计算 LOFFP 和 ROFFP，它们是 LOFFON 和 ROFFON 以及 LOFFOFF 和 ROFFOFF 的平均值。最后用公式：self.ITsys = LOFFP * self.LG； self.rTsys = ROFFP * self.RG 左右旋的系统温度，分别赋值给 self.ITsys 和 self.rTsys。输出左右旋系统温度值。

输出：
self.ITsys, self.rTsys: 这两个变量是类的成员变量，表示左右旋的系统温度。

def getCalData(self):

输入：
self.lonx, self.loffx: 左旋开关数据。
self.ronx, self.roffx: 右旋开关数据。
self.LG, self.RG: 左右旋的增益。
self.ITsys, self.rTsys: 左右旋的系统噪声温度。（有一个条件语句判断如果 self.Amp 的值等于 No就会将系统噪声温度重新设置为1.0。）

这段代码的目的是处理左右旋的数据。它首先计算左右旋开关数据的平均值并乘以相应的增益，然后通过左右旋数据减去左右旋系统温度对数据进行校准，如果 self.Amp 的值等于 No就会将系统噪声温度重新设置为 1.0。

输出：
self.lData, self.rData: 经过处理和校准的左右旋数据

def getAvgBkData(self):

输入：调用了 self.getRawBkData() 方法，获取原始背景(?)数据。
record: 表示记录或观测的序号或标识。每个记录对应于一个数据集。?
integ: 表示积分数据，即整个观测过程中的积分值。
lon: 表示左旋开的数据。
loff: 表示左旋关的数据。
ron: 表示右旋开的数据。
roff: 表示右旋关的数据。
使用类的成员变量 self.Bw（频宽）、self.chbw（通道宽度）、self.crpix（通道中心索引）

调用 self.getRawBkData() 获取原始背景数据。然后用带宽除以通道带宽得到积分通道数；使用类的成员变量 self.crpix计算积分通道中心像素（指示数据集中的中心位置）？。然后通过通道数和中心通道计算出积分通道范围的起始和结束值。循环处理每个记录，计算左右旋开关频率数据的均值。需要前面得到的索引范围并将其添加到相应的列表中，将列表转换为 NumPy 数组。使用 plt.plot 绘制左侧和右侧的均值曲线，以进行可视化。

输出：
对原始背景数据的处理结果，左右旋开关频率均值数据，使用 Matplotlib进行左右旋的可视化
，通过 plt.plot 绘制了两条曲线。

def getAvgTcalData(self):

self.crpix：中心像素？
self.centFreq：中心频率

输入：调用了 self.getRawTcalData() 方法，获取原始 Tcal 数据。使用类的成员变量， self.Bw（频宽）、self.chbw（通道宽度）和 self.crpix（通道中心索引）。

调用 self.getRawTcalData() 获取原始 Tcal 数据。计算积分通道数和积分中心通道像素，然后通过通道数和中心通道计算出积分通道范围的起始和结束值。然后使用 NumPy 的 mean() 方法计算左右旋的 Tcal 数据在指定积分通道范围内的均值。将计算得到的均值分别存储在类的成员变量 self.lmCal 和 self.rmCal 中。

输出：
存储在类的成员变量 self.lmCal 和 self.rmCal 中的左右旋的 Tcal 数据的均值。

def getAvgGain(self):

输入：
self.Amp: 表示是否有放大器。如果为 'No'，将左右旋的增益设置为1.0。
self.lonx, self.loffx, self.ronx, self.roffx: 左右旋开关的频率，用于计算增益。
self.lmCal, self.rmCal: 左右旋的校准数据，用于将观测数据转换为物理单位（定标）

首先，判断放大器的影响，如果 self.Amp 为 'No'，将左右旋的增益设置为1.0。
如果self.Amp不为 'No'，计算左右旋的增益。增益的计算方式左右旋开频率减去左右旋关的频率的差再除以校准数据除以校准数据。然后计算左侧和右侧增益的均值。取左侧和右侧均值的倒数，即计算逆增益。逆增益用于后续的数据校正，以将观测数据转换为物理单位

输出：
self.LG, self.RG（左右旋的增益值，计算左右旋开频率减去右旋关的频率的差再除以校准数据）的均值的倒数并存储在类的成员变量self.LG 和 self.RG 中

def subFitData(self,px,py,pol):

输入：
px：水平轴的输入数据。
py：垂直轴的输入数据。
pol：极化

计算标准差 sigma，通过一系列常数和类实例中的频率属性计算而得。将输入数据 px 和 py 转换为 NumPy 数组。print输出输入的 px 和 py。初始化全局变量 OFF 为 0。根据条件修改全局变量 OFF 的值，具体而言，如果 rcvTy 为 "Q"，且 multibeam 为 True，pol 为 'L'，且 Sta 为 "AZ"，则将 OFF 设置为 84.0。类似地，如果 rcvTy 为 "K"，且 multibeam 为 True，pol 为 'L'，且 Sta 为 "AZ"，则将 OFF 设置为 150.0。根据条件修改 px 的值，如果 Sta 为 "AZ"，且 multibeam 为 True，且 pol 为 'L'，则将 px 的值加上 OFF。定义初始拟合参数 Init（其中包括零阶项、一阶项、py 的最大值、零点处的偏移，以及之前计算得到的 Sigma）。调用 leastsq 函数进行非线性最小二乘拟合，得到拟合结果 r。然后计算均方根误差 rms：对平均平方残差取平方根计算得到均方根误差。波束主瓣峰值电平 ta：从拟合结果 r 中提取第三个元素，波束中心位置 Pc:从拟合结果 r 中提取第四个元素，半功率波束宽度 Hpbw:从拟合结果 r 中提取第五个元素，即拟合的高斯函数的标准差。乘以 2.35，以将标准差转换为高斯函数的半功率波束宽度。math.fabs() 用于获取绝对值，确保宽度为正值。调用函数 fun(px, r) 获取通过拟合得到的数据。print与拟合结果相关的信息。返回包含拟合结果的元组 (rms, hpbw, pc, ta, fitd)。

输出：
返回一个包含五个值的元组 (rms, hpbw, pc, ta, fitd)。
rms：均方根误差。
hpbw：半功率波束宽度。
pc：波束中心的位置。?
ta：波束主瓣的峰值电平。?
fitd：通过拟合得到的数据。

def mergeTelBk(self):

输入：
self.antF: 天线数据文件。
self.telF: 天文观测类型，默认为"BB"。

初始化用于存储数据的列表，包括左右旋数据、左右旋校准数据、天线角度，左右旋Tsys、记录。创建 TelRawData 类的实例，获取天文观测数据，包括时间 (Tm)、水平数据 (dH)、垂直数据 (dV)、方位角 (Az) 和俯仰角 (El)。计算观测数据的最大和最小时间并使用 interp1d 函数对水平和垂直数据进行时间插值。
创建 DIBASRawData 类的实例，获取射电望远镜校准数据。执行一系列方法，包括加载头文件、获取平均bk数据、获取平均 Tcal 数据、获取平均增益、获取 Tsys、获取校准数据以及获取修正数据。将校准结果的时间、左右旋的 Tsys、左右旋的数据、左右旋的天线校准、水平数据、垂直数据添加到相应的列表中。将各个列表的第一个元素提取出来，形成一个完整的数据元组。返回包含合并后数据的元组。

输出：
返回一个包含并后数据的元组，包括：
meanAz: 方位角均值。
meanEl: 俯仰角均值。
ftm: 合并后的时间数据。
faH: 水平角度。
faV: 垂直角度。
(ITSys, fIdata, fImCal):处理整合后的左旋系统温度、数据、校准数据。
(frTsys, frdata, frmCal):处理整合后的右旋系统温度、数据、校准数据。

def getRawBkData(self):

输入：
self.dataFile: 存储数据的文件路径。
self.nWin: 用于索引的窗口编号。
self.maxTm, self.minTm: 时间范围的上下限

初始化变量 pData （列表：pData 包含了 FITS 文件中第 6 个扩展的数据） 和 integ（列表：用于存储积分值）
，print提示信息表明正在打开数据文件。然后打开 FITS 文件并获取以下数据
record = pData.field(0)：从 pData 中提取第 0 列（字段）的数据，并将其存储在名为 record 的变量中。这个变量用于存储记录的时间信息？
frecord = []：初始化一个空列表 frecord，这个列表将用于存储有效记录的时间信息？
row = len(pData)：获取 pData 的行数，即数据表中记录的数量，将其存储在变量 row 中。
lon = []；loff = []；ron = []；roff = []：初始化四个空列表，分别用于存储左右旋开关的频率。
然后遍历数据行，提取左右旋的背景数据（ I0、L1...）以及左右旋的**有用**信息（I0F、I1F）。计算时间差和积分值通过左右旋频率计算得到。当前循环迭代中处理的记录的时间：
record[idx]=record[idx] + delTm（时间差）然后过滤掉超出时间范围的数据。记录有效的记录和左右旋的数据 ：ld0 = l0 * 1.0 / l0F * 1e-4。处理可能的异常值。存储左右旋的开关的数据：lon.append(ld0)。返回结果 (frecord, integ, lon, loff, ron, roff)。

输出：
(frecord, integ, lon, loff, ron, roff): 一个包含有效记录、积分值以及左右通道的背景数据的元组

def getRawTcalData(self):

输入：
self.rcvTy: 类的成员变量，作为文件名的一部分。通过 self.rcvTy + '.dat' 构造文件名，并尝试打开该文件。
self.Amp: 类的成员变量，表示放大器的影响，可能为 'High'、'Low' 或 'No'。
self.nchan: 类的成员变量，表示通道的数量。
self.centFreq: 类的成员变量，表示中心频率。
self.crpix: 类的成员变量，表示中心像素。
self.chbw: 类的成员变量，表示通道带宽。

打开文件：通过 open(self.rcvTy + '.dat', 'r') 打开以 self.rcvTy 命名的文件，并将文件对象存储在 File 变量中。读取文件内容：循环迭代文件的每一行，将每一行的数据提取出来。数据处理：将每行数据拆分为一组数据，然后将这些数据分别添加到不同的列表中
（f：存储频率数据
lh：存储高放大器模式下左旋的校准值
ll：存储低放大器模式下左旋的校准值
rh：存储高放大器模式下右旋的校准值
rl：存储低放大器模式下右旋的校准值
ln：存储无放大器模式下左旋的校准值
rn：存储无放大器模式下旋通道的校准值），
创建NumPy数组：使用 NumPy 将列表转换为数组，分别为 x、y、z。根据条件插值：根据 self.Amp 的值，选择使用 lh、ll 或 ln 进行插值，得到两个插值函数 flm 和 frm。然后创建频率数组：使用 w 计算频率数组。对频率数组进行插值：使用插值函数 flm 和 frm 对频率数组进行插值，得到 ltCal 和 rtCal。

输出：ltCal: 左旋在不同频率下的温度标定值。
rtCal: 右旋在不同频率下的温度标定值。