```
import sys,os,math
from numpy import *
import time
import datetime,time

sys.path.append("/Users/kuntaro/00.Develop/Prog/02.Python/Libs/")
from ReflWidthStill import *
from ReadMtz import *
from GaussFitXY import *

import iotbx.mtz
from libtbx import easy_mp
import scipy.spatial

class ProfileMaker:
        def __init__(self,still_mtz):
                self.still_mtz=still_mtz

        def init(self):
                start_time=time.time()
                ## Open MOSFLM MTZ file
                self.smtz=ReadMtz(self.still_mtz)
                self.smtz.getSymmOption()

                ## Extract intensity related cctbx.array
                self.stiI = self.smtz.getIntensityArray()

                ## M/ISYMM
                self.s_isyms=self.smtz.getColumn("M_ISYM").data()%256

                ## resolution
                self.s_d=self.stiI.d_spacings().data()

                ## Batch number
                self.s_ba=self.smtz.getColumn("BATCH").data()

                # Detector area
                self.s_xa=self.smtz.getColumn("XDET").data()
                self.s_ya=self.smtz.getColumn("YDET").data()

                # If neede, make it possible 140129 KH
                # Detector area setting
                #self.da=DetectorArea(3072,8,4)
                #self.da.init()

                print "%10d reflections were read from %s"%(len(self.s_ba),self.still_mtz)
                self.nrefl=len(self.s_ba)
                end_time=time.time()
                print "Init: %10.1f sec"%(end_time-start_time)

        def isSameRefl(self,i1,i2):
                # HKL information of the first index
                hkl1=self.HKL[i1]
                isym1=self.ISYM[i1]
                # HKL information of the second index
                hkl2=self.HKL[i2]
                isym2=self.ISYM[i2]

                if hkl1==hkl2 and isym1==isym2:
                        return True
                else:
                        return False

        def prepInfo(self,matfile,startphi=35.0,stepphi=0.1,
                wl=1.24,divv=0.02,divh=0.02,mosaic=0.3,dispersion=0.0002):

                self.matfile=matfile
                self.wl=wl
                self.divv=0.02
                self.divh=0.02
                self.mosaic=0.3
                self.dispersion=0.0002

                start_time=time.time()
```

```
                # PHISTART and PHISTEP
                phi0=startphi

                # List of parameters
                self.HKL=[]
                self.Q=[]
                self.RLP=[]
                self.PHI=[]
                self.I=[]
                self.SIGI=[]
                self.ISYM=[]
                self.BATCH=[]
                self.PINFO=[]

                idx=0
                for (hkl1,sI,ssigI),isym,batch,d in zip(self.stiI,
                                                        self.s_isyms,self.s_ba,self.s_d):
                        # Initial batch number
                        if idx==0:
                                batch0=batch

                        # Convertion HKL -> original HKL in MOSFLM
                        ohkl=self.smtz.getOriginalIndex(hkl1,isym)
                        self.HKL.append(ohkl)
                        self.I.append(sI)
                        self.SIGI.append(ssigI)
                        self.ISYM.append(isym)
                        self.BATCH.append(batch)

                        # Rotation angle
                        phi1=phi0+(batch-batch0)*stepphi

                        idx+=1

                        # Collect info to array
                        pinfo=ohkl,phi1,sI,ssigI
                        self.PINFO.append(pinfo)

                a_a=array(self.PINFO)

                self.RLPQ=[]
                self.RLPQ=easy_mp.pool_map(
                        fixed_func=self.getRefInfo,
                        args=self.PINFO,
                        processes=8)

                print "RLPQ=%5d"%len(self.RLPQ)
                for rlpq in self.RLPQ:
                        rlp,q=rlpq
                        self.RLP.append(rlp)
                        self.Q.append(q)

                print "Processed %5d reflections"%idx
                end_time=time.time()
                print "Prep %10.1f sec"%(end_time-start_time)

        def getRefInfo(self,pinfo):
                # Required class for RLP coodrinate calculation
                rws=ReflWidthStill(self.matfile,self.divv,self.divh,self.mosaic,self.dispersion,self
.wl)
                ohkl,phi1,sI,ssigI=pinfo
                rws.setHKL(ohkl,phi1)
                rws.calcDELEPS()
                # RLP coordinate
                rlp=rws.getRLP()
                q=rws.calcQ()
                return rlp,q

        def bunch(self):
                start_time=time.time()
                # independent reflection list
                self.reflist_i=[]
```

```
		# Working list
		lwork=[]

		# Initial condition
		save_i=0

		# Count reflections
		n_alone=0

		# Processing
		for i in range(1,self.nrefl):
			# check if saved reflection and this one is 'same' reflection
			# (not including 'equivalent'
			# DEBUGGING
			#print i,self.HKL[i],self.ISYM[i]

			if self.isSameRefl(i,save_i):
				lwork.append(i)
			else:
				if len(lwork)==1:
					#print "HKL is one",save_i,self.HKL[save_i]
					n_alone+=1

				# Reflection which fills conditions to estimate
				# intensity profile
				#else:
					#self.makeProfile(lwork)
					#print lwork

			# save information
				save_i=i
				self.reflist_i.append(lwork)
				lwork=[]
				lwork.append(i)

		print "%10d reflections are stored."%len(self.reflist_i)
		print "%10d reflections are rejected because observation was once"%n_alone
		end_time=time.time()
		print "Bunch %10.1f sec"%(end_time-start_time)

	def calcRLPdist(self,rlp1,rlp2):
		#print rlp1,rlp2
		vector=rlp1-rlp2
		dist=linalg.norm(vector)
		return dist

	def process_multi_gauss(self,dstar_thresh,nproc):
		# In order to make a 'tree' for fast calculation
		# Firstly, this routine makes the numpy array of RLP
		# codes for each reflection
		# Each reflection is stored into self.reflist_i as indices
		# grouped from MTZ file
		# Ex) in self.reflist_i
		# [0] 1,2,3,4,5
		# [1] 6,7,8
		# [2] 9
		# [3] 10,11,12
		# ...
		# ...
		# each index in each component represents the index
		# of reflections sorted in MTZ file

		start_time=time.time()

		# Finaly this list will be converted to numpy array
		rlp3d_list=[]

		################
		# Multi-processing Guassian fitting of each reflection
		# self.Profile is a list for strroring Gaussian function
		# parameters (index is same with self.reflist_i)
		################
		print "Gauss fitting starts: %s"%datetime.datetime.now()
		self.Profile=easy_mp.pool_map(
```

```
                        fixed_func=self.gaussFit,
                        args=self.reflist_i,
                        processes=nproc)
                print "Gauss fitting ends: %s"%datetime.datetime.now()

                ################
                # processing each reflection to extract RLP coordinate
                # of the first index
                ################
                num_ng=0
                for each_refl in self.reflist_i:
                        rlp_code=self.RLP[each_refl[0]]
                        rlp3d_list.append(rlp_code)

                # Convertion of the list to numpy.array
                rlp3d=array(rlp3d_list)

                # Making the tree for all RLPs
                self.tree=scipy.spatial.cKDTree(rlp3d)

                #print num_ng,num_ok  #s10.mtz Results = 5759 8534 140204

                self.PROC=[]
                # Grouping near reflection list
                for rlp in rlp3d:
                        proclist=[]
                        dist,idx=self.tree.query(
                                rlp,k=300,p=1,distance_upper_bound=dstar_thresh)
                        # Bunch of processing
                        for (d,i) in zip(dist,idx):
                                if d==float('inf'):
                                        break
                                else:
                                        proclist.append(i)
                        self.PROC.append(proclist)

                # self.PROC has a same reflection index with
                # self.reflist_i

                end_time=time.time()
                print "ProcessMulti %10.1f sec"%(end_time-start_time)

        def gaussFit(self,iwork):
                xlist=[]
                ylist=[]

                index=iwork[0]

                # if a number of reflections is 2
                # Gauss fitting is not conducted
                if len(iwork) <= 2:
                        #print "Gaussian fitting cannot be done #refls=%5d!"%len(iwork)
                        return -999,-999,-999

                for i in iwork:
                        xlist.append(self.Q[i])
                        ylist.append(self.I[i])

                # Gaussian fitting
                g=GaussFitXY(xlist,ylist)

                try:
                        params=g.simpleFit()
                        A,mean,sigma,base=params
                except:
                        A,mean,sigma,base=-999,-999,-999,-999

                #print "========"
                #print "GaussFit End"
                #print "========"

                return A,mean,sigma

if __name__ == "__main__":
```

```
matfile="still_10.mat"
divv=0.02
divh=0.02
mosaic=0.3
dispersion=0.0002


starttime=time.time()
# ARG1 = MOSFLM MTZ file
h=ProfileMaker(sys.argv[1])
nproc=int(sys.argv[2])

h.init()
h.prepInfo(matfile,startphi=35.0,stepphi=0.1)
h.bunch()
h.process_multi_gauss(0.035,nproc)
endtime=time.time()
total_time=endtime-starttime
print total_time

#h.process()
#h.process2_test(0.035)
```