```
import sys
import iotbx.mtz

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

# Take common reflections
def commonalize(*Is):
        new_Is = []
        Is0 = Is[0]
        for I in Is[1:]:
                Is0, I = Is0.common_sets(I, assert_is_similar_symmetry=False)
                new_Is.append(I)
        Is = []
        for I in new_Is:
                I = I.common_set(Is0, assert_is_similar_symmetry=False)
                assert len(Is0.data()) == len(I.data())
                Is.append(I)

        return [Is0,] + Is
# commonalize()

# One liner function for extracting Intensity related columns in MTZ file
get_I_arrays = lambda x: filter(lambda y: y.is_xray_intensity_array(), x)

#def run(ref_mtz, frame_mtz,matfile,tbatch,pthresh):
def     run(ref_mtz, frame_mtz, matfile, tbatch, pthresh, mosaic, phistep):
        # MTZ file reading
        ref_arrays = iotbx.mtz.object(ref_mtz).as_miller_arrays()
        frame_arrays = iotbx.mtz.object(frame_mtz).as_miller_arrays(merge_equiva
lents=False)

        # Obtain all of the symmetry operation from FRAME MTZ
        ops = [op.inverse().r() for op in iotbx.mtz.object(frame_mtz).space_grou
p().all_ops()]

        # Extract intensity related cctbx.array
        ref_I = get_I_arrays(ref_arrays)[0]
        frame_I = get_I_arrays(frame_arrays)[0]

        # Extract M/ISYM
        m_isym = filter(lambda a:"M_ISYM" in a.info().labels, frame_arrays)[0]

        # Extract FRACTIONCALC
        fracc = filter(lambda a:"FRACTIONCALC" in a.info().labels, frame_arrays
)[0]
        sel=fracc.data() > pthresh
        fracc = fracc.select(sel)

        # Extract BATCH number
        batch = filter(lambda a: "BATCH" in a.info().labels, frame_arrays)[0]
        sel= batch.data()==tbatch
        batch = batch.select(sel)

        print "ORIGINAL ref I:", ref_mtz, ref_I.info().label_string(),len(ref_I.da
ta())
        print "ORIGINAL frm I:", frame_mtz, frame_I.info().label_string(),len(fram
e_I.data())
        print "ORIGINAL M/ISYM:", frame_mtz, m_isym.info().label_string(),len(m_
isym.data())

        # Take common sets of these
        batch,ref_I,frame_I,m_isym,fracc = commonalize(batch,ref_I,frame_I,m_isy
m,fracc)
        assert len(ref_I.data()) == len(frame_I.data()) == len(m_isym.data()) ==
 len(fracc.data())

        print "CHOSEN ref I:", len(ref_I.data())
        print "CHOSEN frm I:", len(frame_I.data())
```



```
        print "CHOSEN M/ISYM:", len(m_isym.data())

        # delete FULL/PARTIAL flag
        isyms = m_isym.data()%256

        # scale factor
        scale=1.0
        tmp=frame_I.customized_copy(data=frame_I.data()*scale)

        # Preparation for diffraction width
        rwbe=ReflWidthBothEdge(matfile,0.02,0.02,mosaic,0.0002,phistep)

        for (hkl1,rI,rsigI),(hkl2,fI,fsigI),isym,frac in zip(ref_I,tmp,isyms,fra
cc.data()):
                assert hkl1 == hkl2

                # Calculate original index
                sign = -1 if isym%2 == 0 else 1
                ohkl = hkl1*ops[int((isym-1)/2)]
                ohkl = tuple(map(lambda x:int(x*sign), ohkl))

                h=ohkl[0]
                k=ohkl[1]
                l=ohkl[2]
                pobs=fI/rI

                phi=0.1*float(tbatch-1)
                rwbe.setHKL(ohkl,0.0)
                rwbe.calcDELEPS()
                pcalc=rwbe.calcPartiality()

                print "%5d%5d%5d %12.1f%12.1f%12.7f%12.7f%12.7f"%(h,k,l,rI,fI,pobs,fra
c,pcalc)
                #print hkl1,ohkl,rI,fI

        print "Done."

if __name__ == "__main__":

        if len(sys.argv)!=7:
                print "REFMTZ FRAMEMTZ MATFILE BATCH PTHRESH MOSAIC OSCSTEP"

        ref_mtz = sys.argv[1]
        frame_mtz = sys.argv[2]
        mat_file=sys.argv[3]
        batch=int(sys.argv[4])
        pthresh=float(sys.argv[5])
        mosaic=float(sys.argv[6])
        phistep=float(sys.argv[7])

        run(ref_mtz, frame_mtz, mat_file, batch, pthresh, mosaic, phistep)
```