```java
package klib.refl;
import klib.mat.*;
import java.math.*;

public class ReflCondition {

        // Main components
        Matrix s0=null;
        Matrix e1=null;
        Matrix e2=null;
        Matrix e3=null;
        Matrix xyz1=null;
        Matrix xyz2=null;
        Matrix xyzsol=null;
        double dtor=4.0D*Math.atan(1.0D)/180D;
        double lorentz_factor;
        double dstar,dstar2,dst4;
        //      Experimental conditions
        double divv,divh,mosaic;
        double dispersion;      // energy dispersion
        //      Divergence parameters
        double divergence;
        // distance from ES to XRLP at start/end oscillation
        double del1,del2,adel1,adel2;
        double ymid,yms;        // a middle point of XRLP1 and XRLP2 and its squ
are

        // XRLP coordinates in Reciprocal space
        double x1,y1,z1;
        double x2,y2,z2;
        double xys;

        // oscillation information
        // phistart: start oscillation angle of phi(input)
        // phiend  : final oscillation angle of phi(input)
        // phi: the anglular point at which XRLP and E.S cross
        double oscwidth;
        double phistart,phiend;
        double phi;

        // for 'condition()'
        double cea,ceb,cec,ceabsq;

        boolean isInit=false;                   // initialization
        boolean isSolved=false;                 // solving phi
        boolean isCalculated=false;             // calculation of DELEPSs/Pcalc
        boolean isLorentzAvailable=false;       // calculation of Lorentz
        boolean isDelAvailable=false;           // calculation of DEL1/DEL2
        boolean isRspotAvailable=false;         // calculation of RSPOT
        boolean isDiffWidthAvailable=false;     // calculation of DIFFWIDTH
        boolean isNframeAvailable=false;        // calculation of diffraction wi
dth (frame)

        // reciprocal coordinates of xyz
        double[][] rc1;
        double[][] rc2;

        // reflection width
        // rspot: angle width of reflection
        // phiw: width of reflection
        // phis: angle at which a reflection start to reflect
        // phie: angle at which a reflection completely go through E.S.
        // istart: frame on which this reflection is observed at the first
        // iend  : frame on which this reflection is observed at the end
        double rspot;
        double phiw,phis,phie;
        double width_max=3.0; // UNIT: degree
        int iwidth;     // diffraction frames on oscillation
        int istart,iend;
```



```java
        // For cusp check
        double csmin,csmin2;
        int cuspflag=0;
        boolean inCusp=false;

        // epsilons to estimate partiality
        double deleps1,deleps2;

        // partiality of this reflection
        double part;

        // xyz1: reciprocal coordinate of the reflection at 'phistart'
        // xyz2: reciprocal coordinate of the reflection at 'phiend'
        // !!!NOTE!!!
        // mosaic: half width of mosaic spread in RADIANS.
        //===================
        // !!! Angular parameters should be converted from DEGREE to RADIANS bef
ore this routine !!!
        // !!! Except for phistart/phiend (They are in DEGREE)
        //===================
        public ReflCondition(Matrix xyz1,Matrix xyz2,double phistart,double phie
nd,double divv,double divh,double mosaic,double dispersion) {
                this.xyz1=xyz1;
                this.xyz2=xyz2;
                this.phistart=phistart;
                this.phiend=phiend;
                this.divv=divv;
                this.divh=divh;
                this.mosaic=mosaic;
                this.dispersion=dispersion;
        }
        private boolean init() {
        //      S0 vector (anti-parallel to the x-ray)
                double[][] tmp=new double[4][2];
                tmp[1][1]=-1;
                s0=new Matrix(tmp);
        //      E3 vector (z axis for rotation)
                double[][] tmp2=new double[4][2];
                tmp2[3][1]=1;
                e3=new Matrix(tmp2);
        //      E2 vector (E3xRLP/|E3xRLP|
                e2=e3.cross(xyz1);
                double e2_length=e2.length();
                e2=e2.scale(1.0/e2_length);
        //      E1 vector (see the equation)
                e1=e2.cross(e3);
        //      Receive actual double arrays of matrices xyz1 & xyz2
                rc1=xyz1.matrix();
                rc2=xyz2.matrix();
        //      For XRLP1
                x1=rc1[1][1];
                y1=rc1[2][1];
                z1=rc1[3][1];
                xys=Math.pow(x1,2)+Math.pow(y1,2);
                //System.out.printf("Simple XRLP=%12.8f%12.8f%12.8f\n",x1,y1,z1)
;
        //      For XRLP2
                x2=rc2[1][1];
                y2=rc2[2][1];
                z2=rc2[3][1];
        //      Preparation for d* and related values
                dstar=xyz1.length();
                dstar2=dstar*dstar;
                dst4=dstar2*dstar2*0.25D;
        //      Flag
                isInit=true;
                return(isInit);
        }
```

```java
	/* Estimation of deleps1,deleps2. this is the objective of this CLASS
	 */
	private boolean calc_deleps() {
		//	Is spot cut both ends?
		double sign;
		// Calculate distance of edge of spot from sphere at end of rota
tion
		double dist2=adel2-rspot;

		// Test if spot is cut at beginning of rotation
		// Set DELEPS to a negative value
		// NOTE: sign change depending on whether Y is +ve/-ve
		if(adel1-rspot<0.0) {
			if(y1<0.0) {
				sign=-1;
			} else sign=1;
			deleps1=-(sign*del1/rspot+1.0D)*0.5D;
		//	Spot cut at beginning -check for cut at both ends
			if(dist2<0.0) {
				if(y2<0.0) {
					sign=-1;
				}
				deleps2=(1.0D-sign*del2/rspot)*0.5D;
			}
		} else if(dist2<0.0) {
			if(y2<0.0) sign=-1;
			else sign=1;
			deleps2=(1.0D-sign*del2/rspot)*0.5D;
		}
		//System.out.printf("DELEPS1/DELEPS2=%12.8f%12.8f\n",
			//deleps1,deleps2);
	//	Flag
		isRspotAvailable=true;
		return(true);
	}
	// calculating partialities
	private boolean calc_part() {
		// check if this reflection is
		// 1. Already on Ewald sphere
		// 2. Cut at both ends
		// 3. Finally on the Ewald sphere

		double dist2=adel2-rspot;

		// Case 1
		if(adel1-rspot<0.0) {
			// Case 2
			if(dist2<0.0) {
				double tmp=0.5D*(1.0-Math.cos(deleps1*Math.PI));
				part=tmp-(1.0D-0.5D*(1.0-Math.cos(deleps2*Math.P
I)));
				return(true);
			}
			else {
				part=0.5D*(1.0-Math.cos(deleps1*Math.PI));
				return(true);
			}
		} else if(dist2<0.0) {
			part=0.5D*(1.0D-Math.cos(deleps2*Math.PI));
			return(true);
		} else {
			System.out.printf("Something wrong\n");
			part=0.0D;
			return(false);
		}
	}

	/*
```



```java
	   "Radius of spot is same in plane perp to rot"
	   "Note that one should use the horizontal crossfire term here"
	*/
	private boolean cuspcheck() {
		csmin=0.5D*dstar2+rspot;
		csmin2=dst4+dstar2*rspot;
	/*
	   C---- What should csimin test be ? The spot can still appear on image
	   C     even if the centre of the rlp never cuts the sphere (ie lies in
	   C     the cusp region) providing any part of the rlp touches the sphe
re.
	   C     In this case, the test on line below is the right one, but this
	   C     seems to overpredict in practice, so limit it to case where the
	   C     centre of the rlp must intersect the sphere.
	*/
		if(xys<=csmin2) {
		//	A part of spot is in cusp region
			cuspflag=-3;
		} else if(xys<dst4) {
		//	Whole spot is included in cusp region
			cuspflag=-4;
			inCusp=true;
		}
		return(inCusp);
	}

	// Reflection width and start/end phi of diffraction
	private boolean diffWidth() {
	//	Full width of reflection condition (UNIT:radians)
		phiw=2.0D*rspot*lorentz_factor/dtor;
		phis=phi-0.5D*phiw;
		phie=phis+phiw;
	//	Flag
		isDiffWidthAvailable=true;
		return(true);
	}

	/*
	Diffraction condition
	CEA.cos(phic)+CEB.sin(phic)=CEC

	1) CEA=(XRLP.E1)*(E1.S0)
	2) CEB=(XRLP.E1)*(E2.S0)
	3) CEC=0.5*(XRLP.E1)^2+0.5*(XRLP.E3)^2-(XRLP.E3)*(E3.S0)
	4) CEABSQ=CEA^2+CEB^2 (=0.0 -> XRLP on rotation axis)
	*/
	private boolean solvePhi() {
		if(!isInit) this.init();
		double xyz1_dot_e1=xyz1.dot(e1);
		double xyz1_dot_e3=xyz1.dot(e3);
		double e1_dot_s0=e1.dot(s0);
		double e2_dot_s0=e2.dot(s0);
		double e3_dot_s0=e3.dot(s0);
	//	CEA,CEB,CEC
		cea=xyz1_dot_e1*e1_dot_s0;
		ceb=xyz1_dot_e1*e2_dot_s0;
		cec=0.5D*Math.pow(xyz1_dot_e1,2.0D)+0.5D*(Math.pow(xyz1_dot_e3,2
.0D))-(xyz1_dot_e3*e3_dot_s0);
		//System.out.printf("CEA,CEB,CEC:%12.8f %12.8f %12.8f\n",cea,ceb
,cec);
		ceabsq=Math.pow(cea,2.0D)+Math.pow(ceb,2.0D);
		//System.out.printf("CEAB**2=%15.8f\n",ceabsq);
	//	2 solutions
		double arg1=cec/Math.sqrt(ceabsq);
		//System.out.printf("ARG1=%12.8f\n",arg1);
		if(arg1>1.0) arg1=1.0;
		else if(arg1<-1.0) arg1=-1.0;
		double t1=Math.acos(arg1);
		double t2=Math.atan2(ceb,cea);
```

```java
		// 1st solution
		double phic=(t1+t2)/dtor;
		double phia=phistart+phic;
		// 2nd solution
		phic=(-t1+t2)/dtor;
		double phib=phistart+phic;
		//System.out.printf("T1/T2=%12.8f %12.8f\n",t1,t2);
		//System.out.printf("PHIA/PHIB=%12.8f %12.8f\n",phia,phib);
		//System.out.printf("PHIC:%12.8f\n",phic/dtor);
		xyzsol=new RotMat(phic*dtor).rotmat().prod(xyz1,true);
	//	Phi range of this oscillation
		oscwidth=Math.abs(phiend-phistart);
		double midphi=0.5D*(phistart+phiend);
	//	Choosing a phi solution nearby a current oscillation
		double diff1=Math.abs(phia-midphi);
		double diff2=Math.abs(phib-midphi);
		if(diff1<diff2) phi=phia;
		else phi=phib;
		//System.out.printf("Solved PHI= %12.8f\n",phi);
	//	Flag
		isSolved=true;
		return(true);
	}

	private boolean distEStoRLP() {
		if(!isSolved) this.solvePhi();
	/*	Ewald sphere
		(x+1)^2 + y^2 + z^2 = 1.0
	*/
	//	XRLP at start phi
		double x1_2=(x1+1.0D)*(x1+1.0D);
		double y1_2=y1*y1;
		double z1_2=z1*z1;
	//	XRLP at start phi
		double x2_2=(x2+1.0D)*(x2+1.0D);
		double y2_2=y2*y2;
	//	Middle point of XRP1 and 2 on Y axis
	//	!! NOTE !! Z conpoment is Z1_2 in del2
		ymid=(y1+y2)*0.5D;
		yms=ymid*ymid;

		del1=Math.sqrt(x1_2+y1_2+z1_2)-1.0D;
		del2=Math.sqrt(x2_2+y2_2+z1_2)-1.0D;
		//System.out.printf("DEL1/DEL2=(%12.8f,%12.8f)\n",del1,del2);
	//	Absolute values of del1/del2
		adel1=Math.abs(del1);
		adel2=Math.abs(del2);
		//System.out.printf("ADEL1/ADEL2=(%12.5f,%12.5f)\n",adel1,adel2)
;
	//	Flag
		isDelAvailable=true;
		return(true);
	}

	private boolean calcLorentz() {
		Matrix tmp1=e3.cross(s0);
	//
	//	Lorentz factor is estimated at the beginning rotation angle
	//
		double arg2=xyzsol.dot(tmp1);
		//System.out.printf("T3=%12.8f\n",arg2);
		lorentz_factor=Math.abs(1.0D/arg2);
		//System.out.printf("Lorentz factor=%12.8f\n",lorentz_factor);
	//	Flag
		isLorentzAvailable=true;
		return(true);
	}

	private boolean numframes() {
```



```java
	//	Estimating max frames of this reflections
		int maxframes=(int)(width_max/oscwidth);

	//	Find start BATCH in which this reflection is observed
		for(int i=1;i<=maxframes+1;i++) {
			if(phistart-(i-1)*oscwidth<=phis) {
				istart=i;
				break;
			}
		}
		for(int i=1;i<=maxframes+1;i++) {
			if(phiend+(i-1)*oscwidth>=phie) {
				iend=i;
				break;
			}
		}
		iwidth=istart-1+iend-1+1;
	//	Flag
		return(true);
	}

	// Estimating spot radius of a reflection
	private boolean rspot() {
		double delcor=0.0001D;
	//	Preparation of parameters required for divergence calculation
		double esynh=delcor*dstar2+z1*divh;
		double esynv=divv*ymid;
	//	Divergence calculation
		divergence=Math.sqrt((Math.pow(esynh,2.0)+Math.pow(esynv,2.0))/(
yms+z1*z1));
	//	Radius of reflection spot
		rspot=(divergence+mosaic)*Math.sqrt(dstar2-dst4)+0.25D*dispersio
n*dstar2;
		//System.out.printf("divv=%12.8f  divh=%12.8f\n",divv,divh);
		//System.out.printf("DIVERGE=%12.8f SPOT RADIUS %12.8f\n",diverg
ence,rspot);
		//System.out.printf("RSPOT=%12.5f\n",rspot);
		return(true);
	}

	public boolean ppp() {
		init();
		solvePhi();
		distEStoRLP();
		calcLorentz();
		rspot();
		diffWidth();
		numframes();
		if(cuspcheck()) {
			System.out.printf("The reflection is in a cusp region.\n");
			this.deleps1=0.0;
			this.deleps2=0.0;
			this.part=0.0;
		} else {
			calc_deleps();
			calc_part();
		}
	//	Calculation
		isCalculated=true;
		return(true);
	}

	public double pcalc() {
		if(isCalculated==false) ppp();
		return(this.part);
	}

	public double deleps1() {
		if(isCalculated==false) ppp();
```

```java
                return(this.deleps1);
        }
        public double deleps2() {
                if(isCalculated==false) ppp();
                return(this.deleps2);
        }

        public double lorentz() {
                if(isLorentzAvailable==false) {
                        init();
                        solvePhi();
                        distEStoRLP();
                        calcLorentz();
                }
                return(this.lorentz_factor);
        }

        public double phi() {
                if(!isSolved) {
                        init();
                        solvePhi();
                }
                return(phi);
        }

        public double getDiffWidth() {
                distEStoRLP();
                calcLorentz();
                rspot();
                return(rspot);
        }

        public int numFrames() {
                if(!isInit) this.init();
                if(!isSolved) this.solvePhi();
                if(!isDelAvailable) this.distEStoRLP();
                if(!isLorentzAvailable) this.calcLorentz();
                if(!isRspotAvailable) this.rspot();
                if(!isDiffWidthAvailable) this.diffWidth();
                if(!isNframeAvailable) this.numframes();
                numframes();

                return(iwidth);
        }

        public static void main(String[] args) {
                Matrix amat=new Amatrix(args[0]).amat();
//              Double
                double tmp[][]=new double[4][2];
                tmp[1][1]=-12;
                tmp[2][1]=-16;
                tmp[3][1]=-19;
                Matrix hkltmp=new Matrix(tmp);

                Matrix rm=new RotMat(Math.toRadians(0.01)).rotmat();
                Matrix arot=rm.prod(amat,true);

                Matrix conv=new Axis(amat,arot).transmat();
                conv.showMat();
                Matrix convamat=conv.prod(amat,true);

                Matrix xtmp=convamat.prod(hkltmp,true);
                xtmp.showMat();
                ReflCondition rc=new ReflCondition(xtmp,0.0,0.01);
                rc.ppp();
        }
}
```