

Playing with XDS predictions (using Numpy)

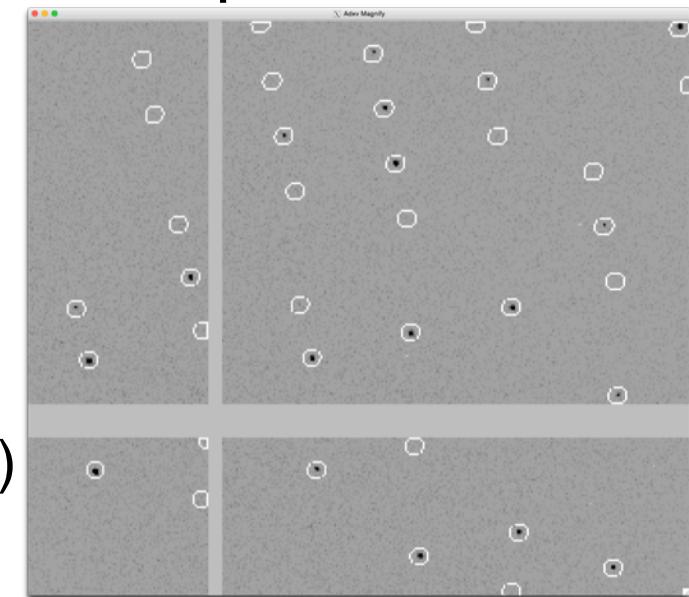
Keitaro Yamashita
RIKEN SPring-8 Center

2016.08.25-28 ECACOMSIG
ECM30 Computing School in Loßburg-Wittendorf

Craving for predictions in XDS

- In data processing, it is extremely important to see if prediction matches actual spot positions
- In XDS, FRAME.cbf is provided to check the predictions on the last integrated frame

FRAME.cbf
(displayed with Adxv)



- What if we want to see predictions on an arbitrary frame?
 - Run INTEGRATE for the specific frame
 - Alternative way: **calculate predictions by ourselves** based on XPARM.XDS (indexing result)
..because this is Computing School.

Tools required in this tutorial

- Python2.7 with CCTBX and Numpy (1.8 or higher)
 - If you have phenix or ccp4, you may use `phenix.python` or `cctbx.python` (phenix-1.10)
- XDS
- generate_XDS.INP
- Adxv

Slides, links, and source code are available on the Github

<https://github.com/keitaroyam/ECACOMSIG-ECM30>

Short URL: <http://git.io/v6zbx>

Reference paper

Acta Cryst. (2010). D66, 133–144. doi:10.1107/S0907444909047374

research papers

Acta Crystallographica Section D

**Biological
Crystallography**

ISSN 0907-4449

Integration, scaling, space-group assignment and post-refinement

Wolfgang Kabsch

Max-Planck-Institut für Medizinische Forschung,
Abteilung Biophysik, Jahnstrasse 29,
69120 Heidelberg, Germany

Correspondence e-mail:
wolfgang.kabsch@mpimf-heidelberg.mpg.de

Important steps in the processing of rotation data are described that are common to most software packages. These programs differ in the details and in the methods implemented to carry out the tasks. Here, the working principles underlying the data-reduction package *XDS* are explained, including the new features of automatic determination of spot size and reflecting range, recognition and assignment of crystal symmetry and a highly efficient algorithm for the determination of correction/scaling factors.

Received 19 August 2009
Accepted 9 November 2009

A version of this paper will be published as a chapter in the new edition of Volume F of *International Tables for Crystallography*.

Related sections in this tutorial:

2.1 Coordinate systems and parameters

2.2 Spot prediction

2.3 Standard spot shape

Test data

<https://zenodo.org/record/10271#.V6vChZOLRBz>

The screenshot shows the Zenodo website interface. At the top, there's a navigation bar with links for Search, Communities, Browse, Upload, Get started, Sign In, and Sign Up. The main content area displays a dataset record for "Thaumatin / Diamond Light Source I04 user training" published on 02 June 2014. The dataset is marked as "Open access". The record details include:

- Publication date: 02 June 2014
- DOI: [10.5281/zenodo.10271](https://doi.org/10.5281/zenodo.10271)
- Keywords: thaumatin, x-ray diffraction, diamond light source
- Collections: Communities, Datasets, Open Access
- License (for files): Creative Commons CCZero
- Uploaded by: graeme (on 03 June 2014)

Below the record details, there's a "New to Zenodo?" section with a "Sign Up" button and a link to learn more about features and benefits. The page also lists the files uploaded to the dataset, including "th_8_2.tar.bz2" (335.2 MB) with a "Download" button. A dropdown menu for citation styles is visible at the bottom.

Detector: PILATUS 6M Prosport+, S/N 60-0100 Diamond
2013-07-03T09:49:57.347
Pixel_size 172e-6 m x 172e-6 m
Silicon sensor, thickness 0.000320 m
Exposure_time 0.0645000 s
Exposure_period 0.0670000 s
Tau = 199.1e-09 s
Count_cutoff 161977 counts
Threshold_setting: 6329 eV
Gain_setting: mid gain (vrf = -0.200)
N_excluded_pixels = 1629
Excluded_pixels: badpix_mask.tif
Flat_field: (nil)
Trim_file: p6m0100_E12658_T6329_vrf_m0p20.bin
Image_path: /ramdisk/2013/mx4014-3/20130703/thaumatin/

Name	Date	Size
th_8_2.tar.bz2	03 Jun 2014	335.2 MB

Prepare indexing result

```
mkdir th_8_2
wget https://zenodo.org/record/10271/files/th\_8\_2.tar.bz2
tar xvf th_8_2.tar.bz2
mkdir xds
cd xds
generate_XDS.INP "../th_8_2_0???.cbf"
vi XDS.INP # Edit SPOT_RANGE= 1 27
# JOB= XYCORR INIT COLSPOT IDXREF
xds_par # Run XDS
```

```
***** REFINED SOLUTION BASED ON INDEXED REFLECTIONS IN SUBTREE # 1 *****

REFINED VALUES OF DIFFRACTION PARAMETERS DERIVED FROM      2988 INDEXED SPOTS
REFINED PARAMETERS: BEAM ORIENTATION CELL
STANDARD DEVIATION OF SPOT POSITION (PIXELS)      0.52
STANDARD DEVIATION OF SPINDLE POSITION (DEGREES)    0.19
SPACE GROUP NUMBER      1
UNIT CELL PARAMETERS      57.913      57.829      150.063     89.848     89.989     89.813
REC. CELL PARAMETERS     0.017267    0.017293    0.006664    90.152    90.011    90.187
COORDINATES OF UNIT CELL A-AXIS      -15.071     -53.744     -15.439
COORDINATES OF UNIT CELL B-AXIS      19.947      9.553     -53.433
COORDINATES OF UNIT CELL C-AXIS      135.413     -49.842      41.207
CRYSTAL MOSAICITY (DEGREES)      0.200
LAB COORDINATES OF ROTATION AXIS  1.000000   0.000000   0.000000
DIRECT BEAM COORDINATES (REC. ANGSTROEM)  0.001382   0.000691   1.024327
DETECTOR COORDINATES (PIXELS) OF DIRECT BEAM      1227.43    1194.51
DETECTOR ORIGIN (PIXELS) AT          1225.35    1193.47
CRYSTAL TO DETECTOR DISTANCE (mm)      265.27
LAB COORDINATES OF DETECTOR X-AXIS  1.000000   0.000000   0.000000
LAB COORDINATES OF DETECTOR Y-AXIS  0.000000   1.000000   0.000000
```

Coordinate systems in XDS

Goniostat system $\{\mathbf{m}_1, \mathbf{m}_2, \mathbf{m}_3\}$

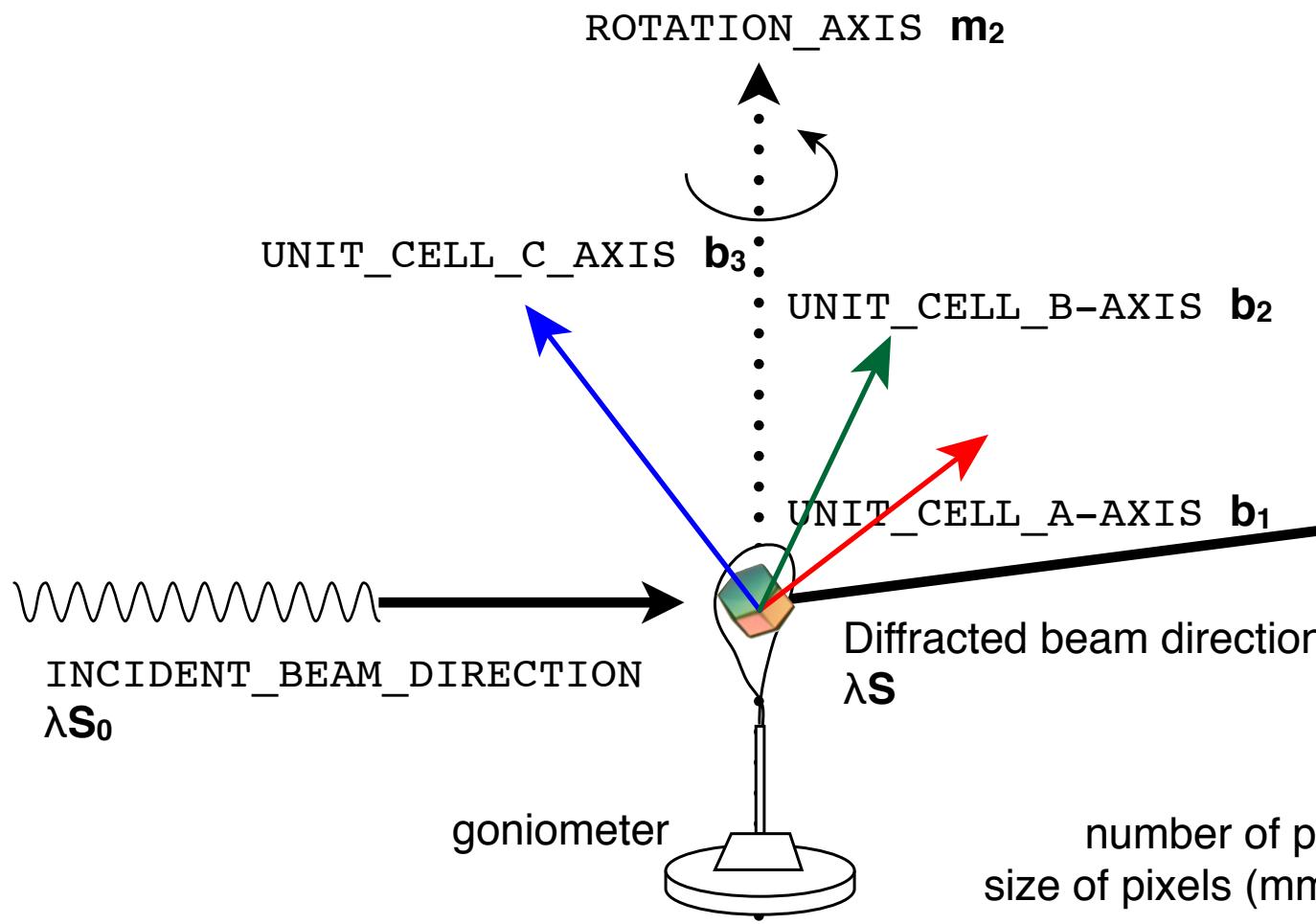
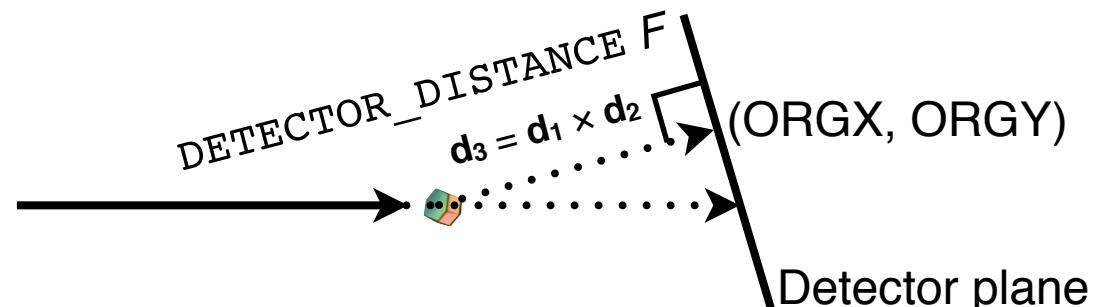
$$\mathbf{m}_1 = \mathbf{m}_2 \times \mathbf{S}_0 / |\mathbf{m}_2 \times \mathbf{S}_0|$$

$$\mathbf{m}_3 = \mathbf{m}_1 \times \mathbf{m}_2$$

Detector system $\{\mathbf{d}_1, \mathbf{d}_2, \mathbf{d}_3\}$

Crystal coordinate system $\{\mathbf{b}_1, \mathbf{b}_2, \mathbf{b}_3\}$

Its reciprocal basis $\{\mathbf{b}_1^*, \mathbf{b}_2^*, \mathbf{b}_3^*\}$

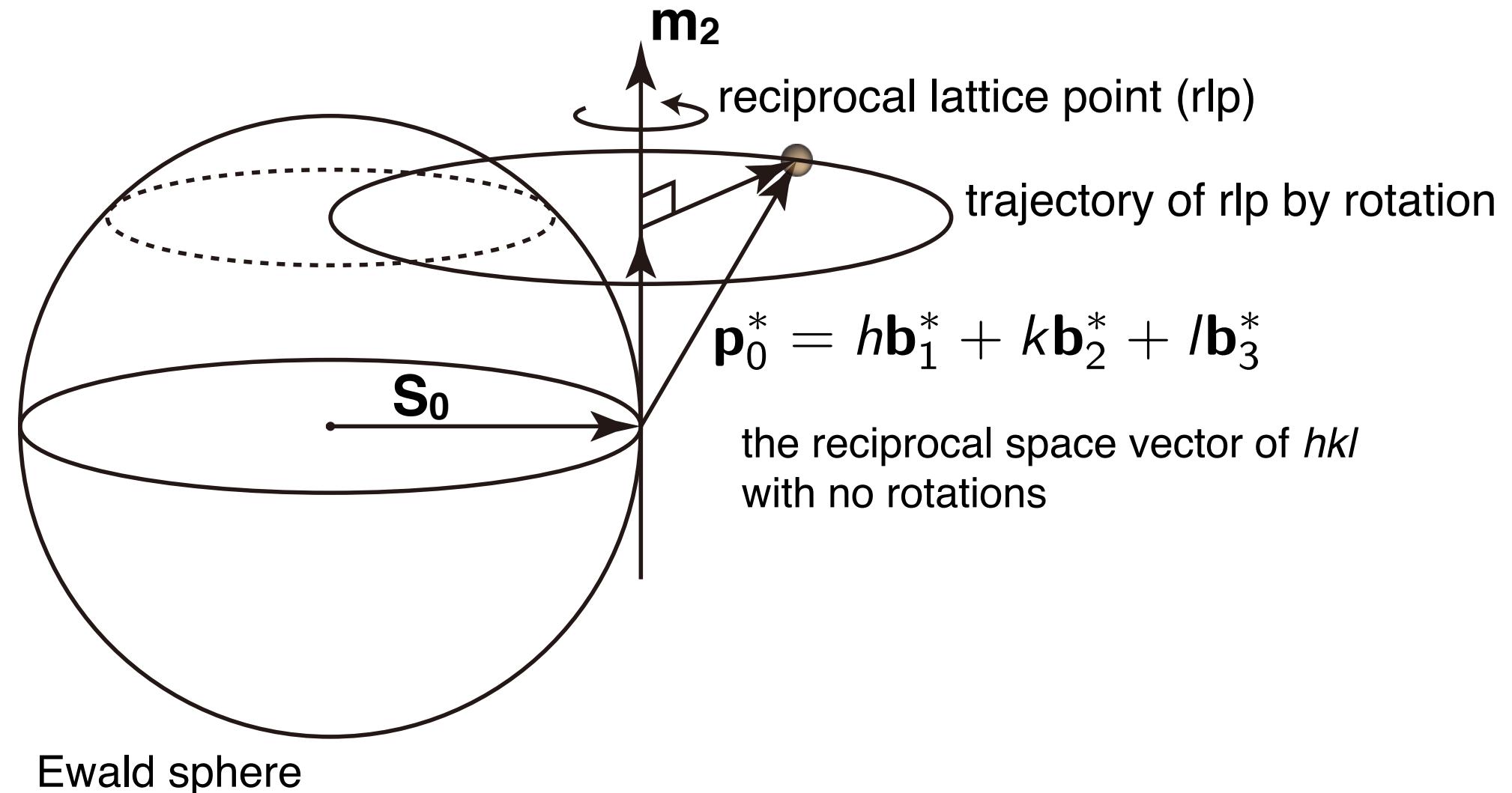


XPARM.XDS

STARTING_FRAME	STARTING_ANGLE	OSCILLATION_RANGE	
XPARM.XDS	VERSION May 1, 2016	BUILT=20160617	
1	0.0000	0.1500	1.000000 0.000000 0.000000 ROTATION_AXIS
X-RAY_WAVELENGTH	0.976250	0.001331	0.000727 1.024327 INCIDENT_BEAM_DIRECTION
SPACE_GROUP_NUMBER	1 57.9046	57.8227	150.0966 89.926 90.023 89.805 UNIT_CELL_CONSTANTS
	-15.079662	-53.735268	-15.429407 = UNIT_CELL_A-AXIS
	19.930243	9.538352	-53.434731 = UNIT_CELL_B-AXIS
	135.399963	-49.820015	41.398369 = UNIT_CELL_C-AXIS
ORGX, ORGY	1 2463	2527	0.172000 0.172000 NX, NY, QX, QY
	1225.349976	1193.469971	265.269989 DETECTOR_DISTANCE
	1.000000	0.000000	0.000000 = DIRECTION_OF_DETECTOR_X-AXIS
	0.000000	1.000000	0.000000 = DIRECTION_OF_DETECTOR_Y-AXIS
	0.000000	0.000000	1.000000 (Detector normal)
	1	2463	1 2527 segment information
	0.00	0.00	1.000000 0.000000 0.000000 1.000000 0.000000

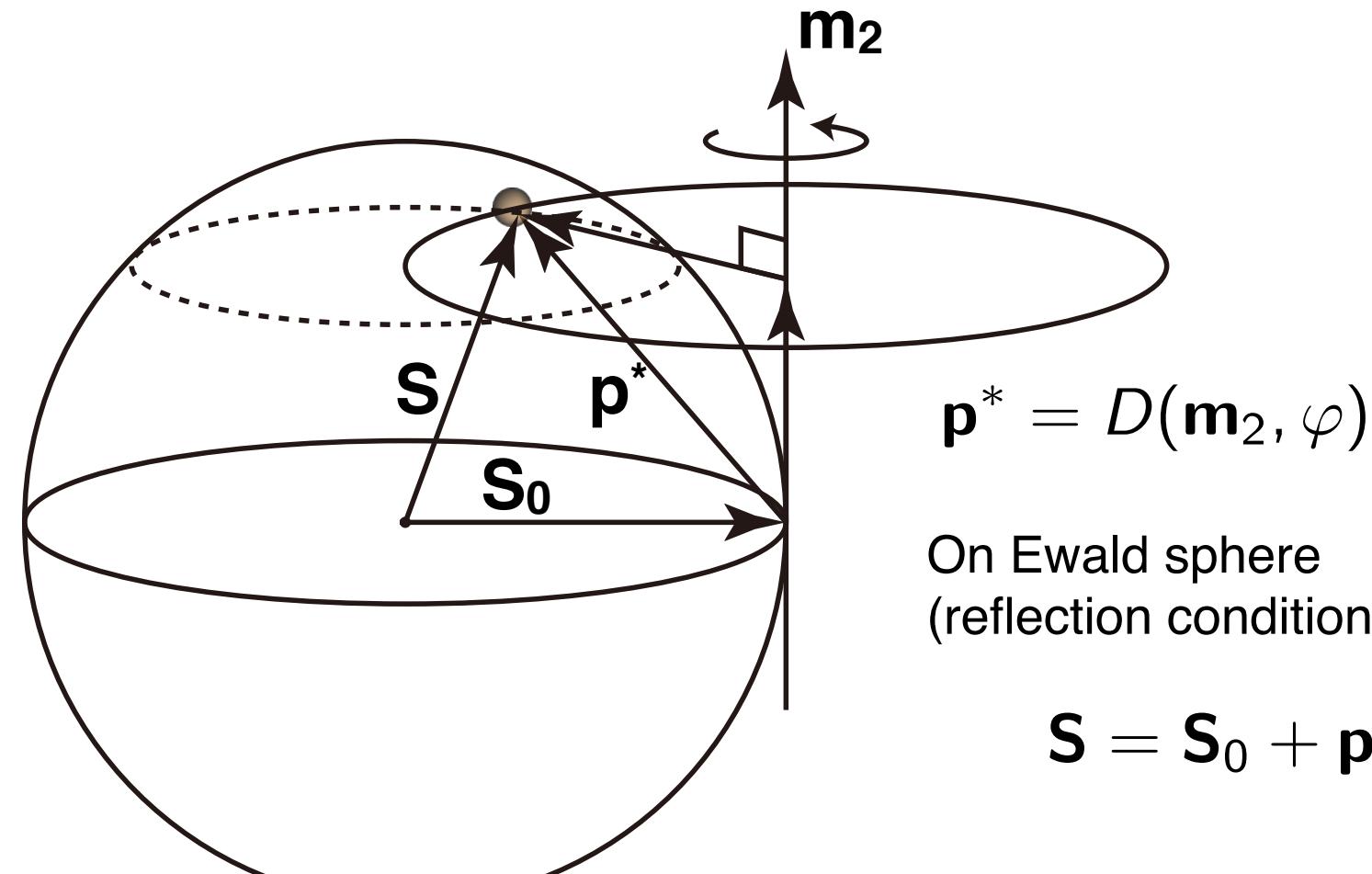
How to predict when reflection occurs

At initial state (no rotations)



How to predict when reflection occurs

After φ rotation



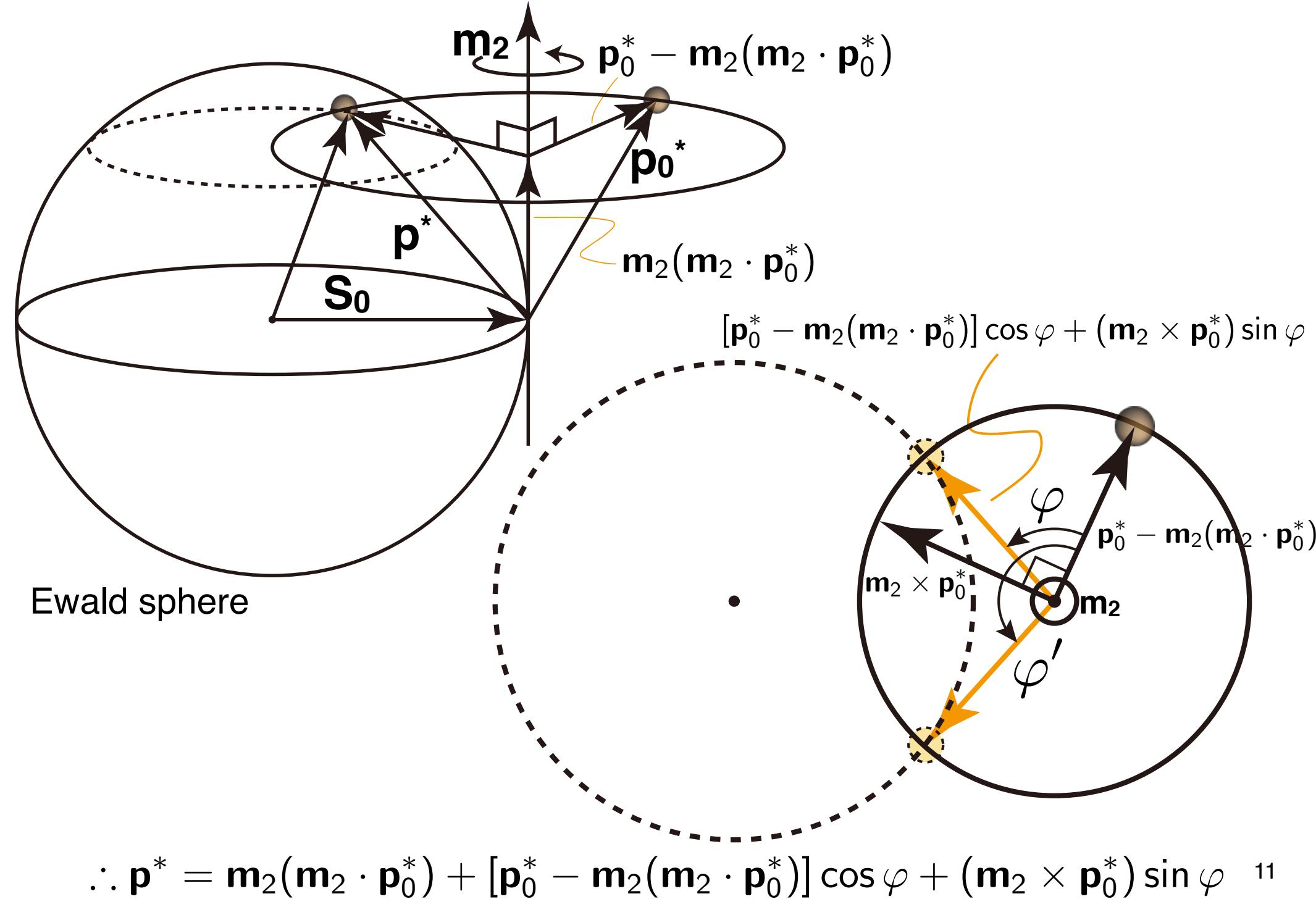
$$\mathbf{p}^* = D(\mathbf{m}_2, \varphi) \mathbf{p}_0^*$$

On Ewald sphere
(reflection condition satisfied)

$$\mathbf{S} = \mathbf{S}_0 + \mathbf{p}^*$$

Ewald sphere

How to predict when reflection occurs



How to predict when reflection occurs

Express \mathbf{p}_0^* with $\mathbf{m}_1, \mathbf{m}_2, \mathbf{m}_3$ basis

$$\mathbf{p}_0^* = \mathbf{m}_1(\mathbf{m}_1 \cdot \mathbf{p}_0^*) + \mathbf{m}_2(\mathbf{m}_2 \cdot \mathbf{p}_0^*) + \mathbf{m}_3(\mathbf{m}_3 \cdot \mathbf{p}_0^*)$$

$$\mathbf{p}^* = \mathbf{m}_2(\mathbf{m}_2 \cdot \mathbf{p}_0^*) + [\mathbf{p}_0^* - \mathbf{m}_2(\mathbf{m}_2 \cdot \mathbf{p}_0^*)] \cos \varphi + (\mathbf{m}_2 \times \mathbf{p}_0^*) \sin \varphi$$

$$\begin{aligned} & \mathbf{m}_1(\mathbf{m}_1 \cdot \mathbf{p}_0^*) + \mathbf{m}_3(\mathbf{m}_3 \cdot \mathbf{p}_0^*) & \mathbf{m}_2 \times \mathbf{m}_1(\mathbf{m}_1 \cdot \mathbf{p}_0^*) + \mathbf{m}_2 \times \mathbf{m}_3(\mathbf{m}_3 \cdot \mathbf{p}_0^*) \\ & & = -\mathbf{m}_3(\mathbf{m}_1 \cdot \mathbf{p}_0^*) + \mathbf{m}_1(\mathbf{m}_3 \cdot \mathbf{p}_0^*) \end{aligned}$$

$$\begin{aligned} & = \mathbf{m}_1[(\mathbf{m}_1 \cdot \mathbf{p}_0^*) \cos \varphi + (\mathbf{m}_3 \cdot \mathbf{p}_0^*) \sin \varphi] + \mathbf{m}_2(\mathbf{m}_2 \cdot \mathbf{p}_0^*) \\ & \quad + \mathbf{m}_3[(\mathbf{m}_3 \cdot \mathbf{p}_0^*) \cos \varphi - (\mathbf{m}_1 \cdot \mathbf{p}_0^*) \sin \varphi] \end{aligned}$$

$$= \mathbf{m}_1(\mathbf{m}_1 \cdot \mathbf{p}^*) + \mathbf{m}_2(\mathbf{m}_2 \cdot \mathbf{p}^*) + \mathbf{m}_3(\mathbf{m}_3 \cdot \mathbf{p}^*) \quad \text{expressed with } \mathbf{m}_1, \mathbf{m}_2, \mathbf{m}_3 \text{ basis}$$

$$\therefore \begin{cases} (\mathbf{m}_1 \cdot \mathbf{p}_0^*) \cos \varphi + (\mathbf{m}_3 \cdot \mathbf{p}_0^*) \sin \varphi = \mathbf{m}_1 \cdot \mathbf{p}^* \\ (\mathbf{m}_3 \cdot \mathbf{p}_0^*) \cos \varphi - (\mathbf{m}_1 \cdot \mathbf{p}_0^*) \sin \varphi = \mathbf{m}_3 \cdot \mathbf{p}^* \end{cases}$$

$$\iff \begin{cases} \cos \varphi = [(\mathbf{m}_1 \cdot \mathbf{p}_0^*)(\mathbf{m}_1 \cdot \mathbf{p}^*) + (\mathbf{m}_3 \cdot \mathbf{p}_0^*)(\mathbf{m}_3 \cdot \mathbf{p}^*)]/\rho^2 \\ \sin \varphi = [(\mathbf{m}_3 \cdot \mathbf{p}_0^*)(\mathbf{m}_1 \cdot \mathbf{p}^*) - (\mathbf{m}_1 \cdot \mathbf{p}_0^*)(\mathbf{m}_3 \cdot \mathbf{p}^*)]/\rho^2 \end{cases}$$

$$\rho^2 = (\mathbf{m}_1 \cdot \mathbf{p}_0^*)^2 + (\mathbf{m}_3 \cdot \mathbf{p}_0^*)^2 = |\mathbf{p}_0^* - \mathbf{m}_2(\mathbf{m}_2 \cdot \mathbf{p}_0^*)|^2 = \mathbf{p}_0^{*2} - (\mathbf{m}_2 \cdot \mathbf{p}_0^*)^2$$

(squared distance from \mathbf{p}_0^* to rotation axis)

How to predict when reflection occurs

Express \mathbf{p}_0^* with $\mathbf{m}_1, \mathbf{m}_2, \mathbf{m}_3$ basis

$$\mathbf{p}_0^* = \mathbf{m}_1(\mathbf{m}_1 \cdot \mathbf{p}_0^*) + \mathbf{m}_2(\mathbf{m}_2 \cdot \mathbf{p}_0^*) + \mathbf{m}_3(\mathbf{m}_3 \cdot \mathbf{p}_0^*)$$

$$\mathbf{p}^* = \mathbf{m}_2(\mathbf{m}_2 \cdot \mathbf{p}_0^*) + [\mathbf{p}_0^* - \mathbf{m}_2(\mathbf{m}_2 \cdot \mathbf{p}_0^*)] \cos \varphi + (\mathbf{m}_2 \times \mathbf{p}_0^*) \sin \varphi$$

$$\begin{aligned} & \mathbf{m}_1(\mathbf{m}_1 \cdot \mathbf{p}_0^*) + \mathbf{m}_3(\mathbf{m}_3 \cdot \mathbf{p}_0^*) & \mathbf{m}_2 \times \mathbf{m}_1(\mathbf{m}_1 \cdot \mathbf{p}_0^*) + \mathbf{m}_2 \times \mathbf{m}_3(\mathbf{m}_3 \cdot \mathbf{p}_0^*) \\ & & = -\mathbf{m}_3(\mathbf{m}_1 \cdot \mathbf{p}_0^*) + \mathbf{m}_1(\mathbf{m}_3 \cdot \mathbf{p}_0^*) \end{aligned}$$

$$\begin{aligned} & = \mathbf{m}_1[(\mathbf{m}_1 \cdot \mathbf{p}_0^*) \cos \varphi + (\mathbf{m}_3 \cdot \mathbf{p}_0^*) \sin \varphi] + \mathbf{m}_2(\mathbf{m}_2 \cdot \mathbf{p}_0^*) \\ & \quad + \mathbf{m}_3[(\mathbf{m}_3 \cdot \mathbf{p}_0^*) \cos \varphi - (\mathbf{m}_1 \cdot \mathbf{p}_0^*) \sin \varphi] \end{aligned}$$

$$= \mathbf{m}_1(\mathbf{m}_1 \cdot \mathbf{p}^*) + \mathbf{m}_2(\mathbf{m}_2 \cdot \mathbf{p}^*) + \mathbf{m}_3(\mathbf{m}_3 \cdot \mathbf{p}^*) \quad \text{expressed with } \mathbf{m}_1, \mathbf{m}_2, \mathbf{m}_3 \text{ basis}$$

$$\therefore \begin{cases} (\mathbf{m}_1 \cdot \mathbf{p}_0^*) \cos \varphi + (\mathbf{m}_3 \cdot \mathbf{p}_0^*) \sin \varphi = \mathbf{m}_1 \cdot \mathbf{p}^* \\ (\mathbf{m}_3 \cdot \mathbf{p}_0^*) \cos \varphi - (\mathbf{m}_1 \cdot \mathbf{p}_0^*) \sin \varphi = \mathbf{m}_3 \cdot \mathbf{p}^* \end{cases} \quad \text{We don't know!}$$

$$\iff \begin{cases} \cos \varphi = [(\mathbf{m}_1 \cdot \mathbf{p}_0^*)(\mathbf{m}_1 \cdot \mathbf{p}^*) + (\mathbf{m}_3 \cdot \mathbf{p}_0^*)(\mathbf{m}_3 \cdot \mathbf{p}^*)]/\rho^2 \\ \sin \varphi = [(\mathbf{m}_3 \cdot \mathbf{p}_0^*)(\mathbf{m}_1 \cdot \mathbf{p}^*) - (\mathbf{m}_1 \cdot \mathbf{p}_0^*)(\mathbf{m}_3 \cdot \mathbf{p}^*)]/\rho^2 \end{cases}$$

$$\rho^2 = (\mathbf{m}_1 \cdot \mathbf{p}_0^*)^2 + (\mathbf{m}_3 \cdot \mathbf{p}_0^*)^2 = |\mathbf{p}_0^* - \mathbf{m}_2(\mathbf{m}_2 \cdot \mathbf{p}_0^*)|^2 = \mathbf{p}_0^{*2} - (\mathbf{m}_2 \cdot \mathbf{p}_0^*)^2$$

(squared distance from \mathbf{p}_0^* to rotation axis)

How to predict when reflection occurs

When Laue equations satisfied

$$\mathbf{S} = \mathbf{S}_0 + \mathbf{p}^*, \quad \mathbf{S}^2 = \mathbf{S}_0^2 \implies \mathbf{p}^{*2} = -2\mathbf{S}_0 \cdot \mathbf{p}^* = \mathbf{p}_0^{*2}$$

Express \mathbf{S}_0 and \mathbf{p}^* with $\mathbf{m}_1, \mathbf{m}_2, \mathbf{m}_3$ basis

$$-2 \left[\underline{\mathbf{m}_1(\mathbf{m}_1 \cdot \mathbf{S}_0)} + \mathbf{m}_2(\mathbf{m}_2 \cdot \mathbf{S}_0) + \mathbf{m}_3(\mathbf{m}_3 \cdot \mathbf{S}_0) \right] \left[\mathbf{m}_1(\mathbf{m}_1 \cdot \mathbf{p}^*) + \underline{\mathbf{m}_2(\mathbf{m}_2 \cdot \mathbf{p}^*)} + \mathbf{m}_3(\mathbf{m}_3 \cdot \mathbf{p}^*) \right] = 0$$
$$= \mathbf{m}_2 \cdot \mathbf{p}_0^*$$

$$= -2 \left[(\mathbf{m}_2 \cdot \mathbf{S}_0)(\mathbf{m}_2 \cdot \mathbf{p}_0^*) + (\mathbf{m}_3 \cdot \mathbf{S}_0)(\mathbf{m}_3 \cdot \mathbf{p}^*) \right] = \mathbf{p}^{*2}$$

$$\therefore \mathbf{m}_3 \cdot \mathbf{p}^* = \frac{-\mathbf{p}^{*2}/2 - (\mathbf{m}_2 \cdot \mathbf{S}_0)(\mathbf{m}_2 \cdot \mathbf{p}_0^*)}{\mathbf{m}_3 \cdot \mathbf{S}_0}$$

$\mathbf{m}_2 \cdot \mathbf{p}^* = \mathbf{m}_2 \cdot \mathbf{p}_0^*$ (as the component along rotation axis does not change by rotation)

$$\mathbf{m}_1 \cdot \mathbf{p}^* = \pm \sqrt{\mathbf{p}_0^{*2} - (\mathbf{m}_2 \cdot \mathbf{p}^*)^2 - (\mathbf{m}_3 \cdot \mathbf{p}^*)^2} \quad (\text{as } \mathbf{m}_1, \mathbf{m}_2, \mathbf{m}_3 \text{ are orthonormal basis})$$

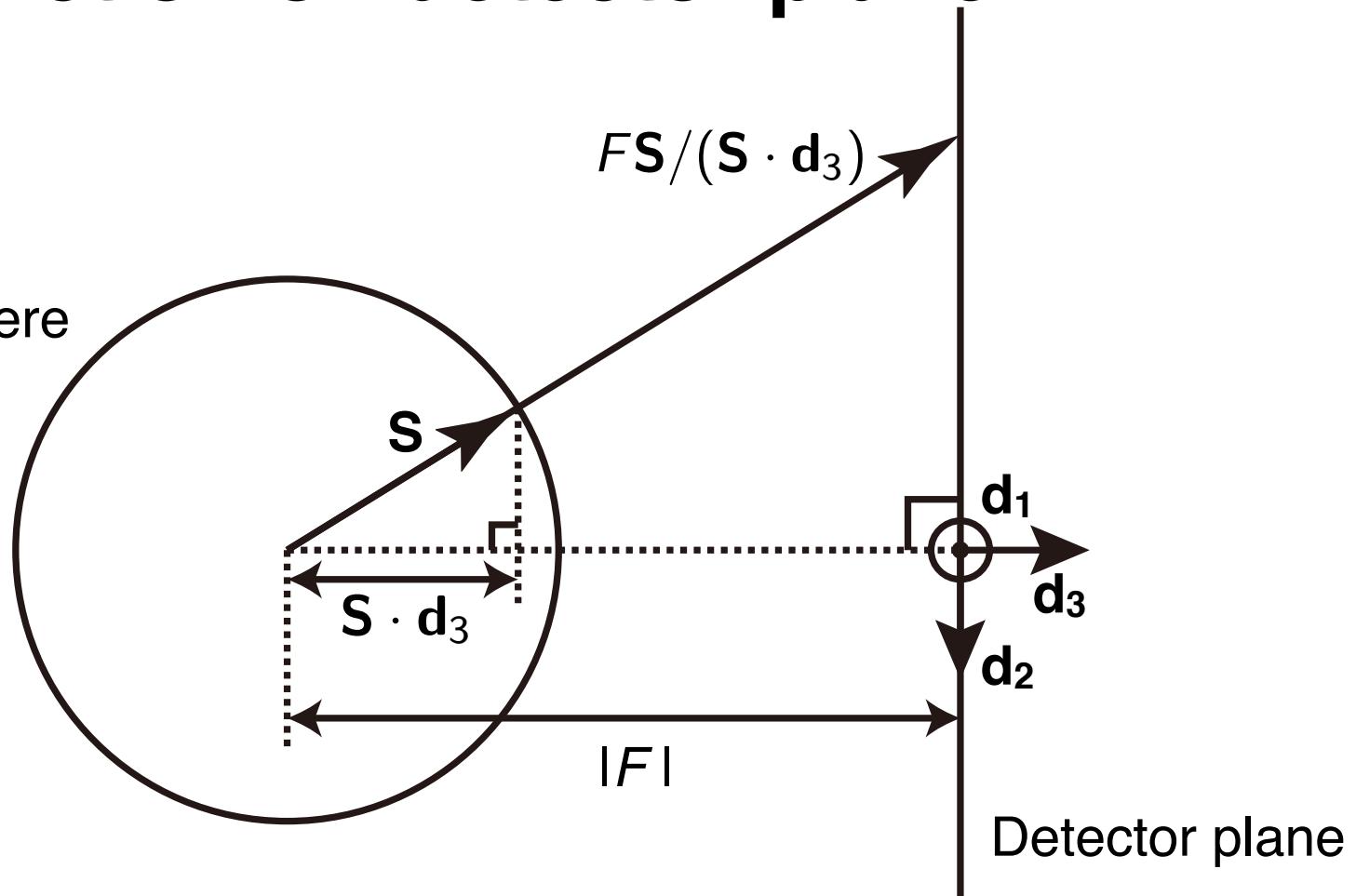
Finally \mathbf{p}^* has been obtained as components along $\mathbf{m}_1, \mathbf{m}_2, \mathbf{m}_3$!

But Laue equations have no solutions if $\rho^2 < (\mathbf{m}_3 \cdot \mathbf{p}^*)^2$ or $\mathbf{p}_0^{*2} > 4\mathbf{S}_0^2$

(blind region) (out of limiting sphere)

Prediction on detector plane

Ewald sphere
Beam intersects the
detector only when
 $F(\mathbf{S} \cdot \mathbf{d}_3) > 0$

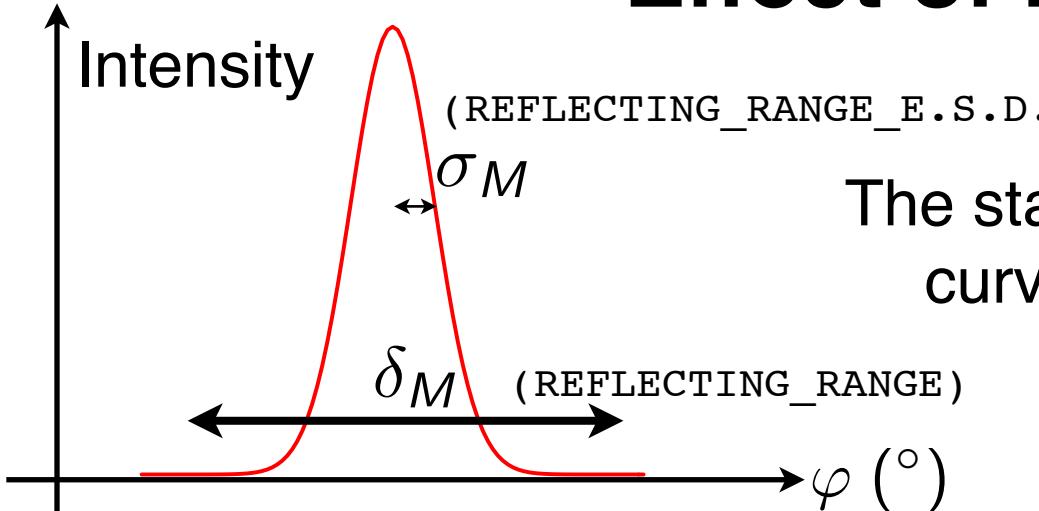


Expressing with $\mathbf{d}_1, \mathbf{d}_2, \mathbf{d}_3$ basis

$$\begin{aligned} F\mathbf{S}/(\mathbf{S} \cdot \mathbf{d}_3) &= F\mathbf{d}_1(\mathbf{S} \cdot \mathbf{d}_1)/(\mathbf{S} \cdot \mathbf{d}_3) + F\mathbf{d}_2(\mathbf{S} \cdot \mathbf{d}_2)/(\mathbf{S} \cdot \mathbf{d}_3) + F\mathbf{d}_3 \\ &= (X - X_0)\mathbf{d}_1 + (Y - Y_0)\mathbf{d}_2 + F\mathbf{d}_3 \end{aligned}$$

$$\therefore \begin{cases} X = X_0 + F(\mathbf{S} \cdot \mathbf{d}_1)/(\mathbf{S} \cdot \mathbf{d}_3) \\ Y = Y_0 + F(\mathbf{S} \cdot \mathbf{d}_2)/(\mathbf{S} \cdot \mathbf{d}_3) \end{cases} \quad (X_0 = \text{ORGX}, Y_0 = \text{ORGY})$$

Effect of mosaicity

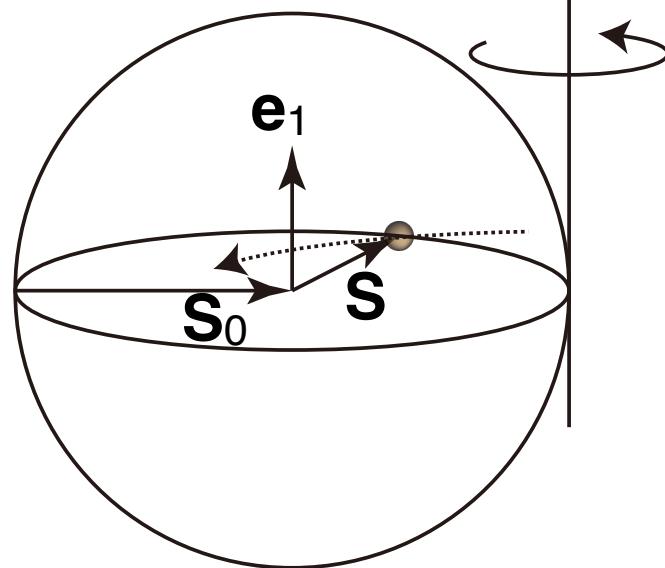


The standard deviation of Gaussian rocking curve is defined as “mosaicity” in XDS

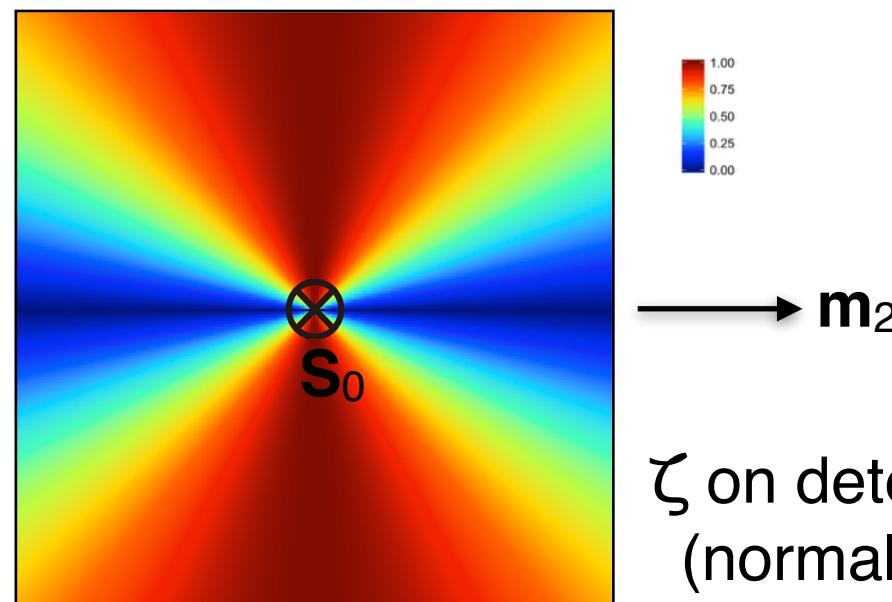
However, the reflecting range is elongated by the factor $1/|\zeta|$

$$\zeta = \mathbf{e}_1 \cdot \mathbf{m}_2 \quad (\mathbf{e}_1 = \mathbf{S} \times \mathbf{S}_0 / |\mathbf{S} \times \mathbf{S}_0|)$$

A reciprocal lattice point crosses the Ewald sphere by the shortest route when $\mathbf{m}_2 // \mathbf{e}_1$



Ewald sphere



Mathematical summary

Given hkl , the reciprocal lattice vector \mathbf{p}_0^* at initial geometry is

$$\mathbf{p}_0^* = h\mathbf{b}_1^* + k\mathbf{b}_2^* + l\mathbf{b}_3^*$$

After φ rotation around \mathbf{m}_2 vector, reflection condition is satisfied

$$\mathbf{p}^* = D(\mathbf{m}_2, \varphi)\mathbf{p}_0^*, \quad \mathbf{S} = \mathbf{S}_0 + \mathbf{p}^*$$

Here φ can be obtained by

$$\varphi = \tan^{-1} \frac{[(\mathbf{m}_3 \cdot \mathbf{p}_0^*)(\mathbf{m}_1 \cdot \mathbf{p}^*) - (\mathbf{m}_1 \cdot \mathbf{p}_0^*)(\mathbf{m}_3 \cdot \mathbf{p}^*)]/\rho^2}{[(\mathbf{m}_1 \cdot \mathbf{p}_0^*)(\mathbf{m}_1 \cdot \mathbf{p}^*) + (\mathbf{m}_3 \cdot \mathbf{p}_0^*)(\mathbf{m}_3 \cdot \mathbf{p}^*)]/\rho^2}$$

with

$$\left\{ \begin{array}{l} \mathbf{m}_3 \cdot \mathbf{p}^* = \frac{-\mathbf{p}^{*2}/2 - (\mathbf{m}_2 \cdot \mathbf{S}_0)(\mathbf{m}_2 \cdot \mathbf{p}_0^*)}{\mathbf{m}_3 \cdot \mathbf{S}_0} \\ \mathbf{m}_2 \cdot \mathbf{p}^* = \mathbf{m}_2 \cdot \mathbf{p}_0^* \\ \mathbf{m}_1 \cdot \mathbf{p}^* = \pm \sqrt{\mathbf{p}_0^{*2} - (\mathbf{m}_2 \cdot \mathbf{p}^*)^2 - (\mathbf{m}_3 \cdot \mathbf{p}^*)^2} \end{array} \right.$$

Then, the coordinate of the spot on the detector is

$$\left\{ \begin{array}{l} X = X_0 + F(\mathbf{S} \cdot \mathbf{d}_1)/(\mathbf{S} \cdot \mathbf{d}_3) \\ Y = Y_0 + F(\mathbf{S} \cdot \mathbf{d}_2)/(\mathbf{S} \cdot \mathbf{d}_3) \end{array} \right.$$

Standard deviation of the reflecting range is

$$\sigma_M/|\zeta|, \quad \zeta = \mathbf{m}_2 \cdot (\mathbf{S} \times \mathbf{S}_0)/|\mathbf{S} \times \mathbf{S}_0|$$

Ready?

Let us implement a code to calculate the list of (X, Y, φ)

1. Generate a complete set of Miller indices using CCTBX
2. Calculate (X, Y, φ) by matrix calculations using NumPy
3. For a given frame number, generate .adx file which includes (X, Y) coordinates for viewing with Adxv

NumPy

“Python is fast if you don’t use python”

Pascal (2012) CCP4BB

<https://www.mail-archive.com/ccp4bb@jiscmail.ac.uk/msg24791.html>

- Python is very powerful and easy to use, but slow!
 - For example, to calculate predictions, if we use for-loop for each hkl , it would be extremely slow
- NumPy is a free (open-source) package for scientific computing with Python
 - N -dimensional array object, linear algebra, FFT, ...
 - NumPy data types and functions are implemented in C
 - Very fast if we write the code using NumPy functions

Linear algebra with NumPy

```
In [1]: import numpy
```

```
In [2]: a = numpy.array([0.,1.,2.])
```

```
In [3]: b = numpy.array([3.,4.,5.])
```

$$a = \begin{pmatrix} 0 \\ 1 \\ 2 \end{pmatrix}, b = \begin{pmatrix} 3 \\ 4 \\ 5 \end{pmatrix}$$

```
In [4]: b[0]
```

```
Out[4]: 3.0
```

Index starts with zero

```
In [5]: a*b
```

```
Out[5]: array([ 0., 4., 10.]) = \begin{pmatrix} a_0 b_0 \\ a_1 b_1 \\ a_2 b_2 \end{pmatrix}
```

Multiplications of elements

```
In [6]: numpy.dot(a,b)
```

```
Out[6]: 14.0
```

$$\sum_i a_i b_i = \mathbf{a} \cdot \mathbf{b} \quad \text{Inner product}$$

```
In [7]: numpy.cross(a,b)
```

```
Out[7]: array([-3., 6., -3.]) = \mathbf{a} \times \mathbf{b} \quad \text{Cross product}
```

```
In [8]: numpy.linalg.norm(a)
```

```
Out[8]: 2.2360679774997898
```

```
In [9]: numpy.dot(a, a)
```

```
Out[9]: 5.0
```

```
In [10]: numpy.sum(a**2)
```

```
Out[10]: 5.0
```

$$\left. \begin{array}{l} \mathbf{a} \cdot \mathbf{a} = \sum_i a_i^2 = |\mathbf{a}|^2 \\ \mathbf{a} \cdot \mathbf{b} = \sum_i a_i b_i \\ \mathbf{a} \times \mathbf{b} \end{array} \right\}$$

Linear algebra with NumPy

```
In [11]: A = numpy.array([[0.,1.,2.], [3.,4.,5.], [6.,7.,8.]])
```

```
In [12]: A
```

```
Out[12]: array([[ 0.,  1.,  2.],  
                 [ 3.,  4.,  5.],  
                 [ 6.,  7.,  8.]])
```

$$A = \begin{pmatrix} 0 & 1 & 2 \\ 3 & 4 & 5 \\ 6 & 7 & 8 \end{pmatrix}$$

```
In [13]: A.transpose()
```

```
Out[13]: array([[ 0.,  3.,  6.],  
                 [ 1.,  4.,  7.],  
                 [ 2.,  5.,  8.]])
```

$$A^t = \begin{pmatrix} 0 & 3 & 6 \\ 1 & 4 & 7 \\ 2 & 5 & 8 \end{pmatrix}$$

```
In [14]: A[1,2]
```

```
Out[14]: 5.0
```

```
In [15]: A[1,:]
```

```
Out[15]: array([ 3.,  4.,  5.])
```

```
In [16]: A[:,1]
```

```
Out[16]: array([ 1.,  4.,  7.])
```

```
In [17]: A[1,:] = numpy.array([90, 91, 92])
```

```
In [18]: A
```

```
Out[18]: array([[ 0.,  1.,  2.],  
                 [ 90.,  91.,  92.],  
                 [ 6.,  7.,  8.]])
```

```
In [19]: A = numpy.array([[0.,1.,2.], [3.,4.,5.], [6.,7.,8.]])
```

A[1,:]

A[:,1]

A = $\begin{pmatrix} 0 & 1 & 2 \\ 3 & 4 & 5 \\ 6 & 7 & 8 \end{pmatrix}$

A[1,:]

Replace the slice

Linear algebra with NumPy

```
In [20]: A*A
```

```
Out[20]:
```

```
array([[ 0.,  1.,  4.],  
       [ 9., 16., 25.],  
       [36., 49., 64.]])
```



Multiplications of elements

```
In [21]: numpy.dot(A, A)
```

```
Out[21]:
```

```
array([[ 15.,  18.,  21.],  
       [ 42.,  54.,  66.],  
       [ 69.,  90., 111.]])
```



Matrix multiplication

```
In [22]: numpy.dot(A, a)
```

```
Out[22]: array([ 5., 14., 23.])
```

$$Aa = \begin{pmatrix} 0 & 1 & 2 \\ 3 & 4 & 5 \\ 6 & 7 & 8 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \\ 2 \end{pmatrix}$$

```
In [23]: numpy.dot(a, A)
```

```
Out[23]: array([ 15., 18., 21.])
```

$$a^t A = \begin{pmatrix} 0 & 1 & 2 \end{pmatrix} \begin{pmatrix} 0 & 1 & 2 \\ 3 & 4 & 5 \\ 6 & 7 & 8 \end{pmatrix}$$

```
In [24]: numpy.dot(A.transpose(), a)    \because (AB)^t = B^t A^t
```

```
Out[24]: array([ 15., 18., 21.])
```



Dimension of array (matrix)

```
In [25]: A.shape
```

```
Out[25]: (3, 3)
```

```
In [26]: a.shape
```

```
Out[26]: (3,)
```

Linear algebra with NumPy

```
In [27]: B = numpy.array(range(15)).reshape(5,3)
```

```
In [28]: B
```

```
Out[28]:
```

```
array([[ 0,  1,  2],  
       [ 3,  4,  5],  
       [ 6,  7,  8],  
       [ 9, 10, 11],  
      [12, 13, 14]])
```

$$B = \begin{pmatrix} 0 & 1 & 2 \\ 3 & 4 & 5 \\ 6 & 7 & 8 \\ 9 & 10 & 11 \\ 12 & 13 & 14 \end{pmatrix}$$

```
In [29]: numpy.sum(B)
```

sum of all elements

```
Out[29]: 105
```



$$\begin{pmatrix} 0 & 1 & 2 \\ 3 & 4 & 5 \\ 6 & 7 & 8 \\ 9 & 10 & 11 \\ 12 & 13 & 14 \end{pmatrix} \xrightarrow{\text{sum}} \text{sum}$$
$$\begin{pmatrix} 0 & 1 & 2 \\ 3 & 4 & 5 \\ 6 & 7 & 8 \\ 9 & 10 & 11 \\ 12 & 13 & 14 \end{pmatrix} \xrightarrow{\text{sum}} \text{sum}$$

```
In [30]: numpy.sum(B, axis=1)
```

```
Out[30]: array([ 3, 12, 21, 30, 39])
```

```
In [31]: B_lensq = numpy.sum(B**2, axis=1)
```

```
In [32]: B_lensq
```

```
Out[32]: array([ 5, 50, 149, 302, 509])
```

```
In [33]: sel = B_lensq > 100
```

```
In [34]: sel
```

```
Out[34]: array([False, False, True, True, True], dtype=bool)
```

```
In [35]: B(sel)
```

Take selected rows of B

```
Out[35]:  
array([[ 6,  7,  8],  
       [ 9, 10, 11],  
      [12, 13, 14]])
```

Linear algebra with NumPy

```
In [36]: d = numpy.array([0,0,1.])
In [37]: C = numpy.cross(B, d)
In [38]: C
Out[38]:
```



```
array([[ 1.,  0.,  0.],
       [ 4., -3.,  0.],
       [ 7., -6.,  0.],
       [10., -9.,  0.],
       [13., -12.,  0.]])
```

each row = $B[i,:] \times d$

```
In [39]: numpy.linalg.norm(C, axis=1)
Out[39]: array([ 1.          ,  5.          ,  9.21954446, 13.45362405, 17.69180601])
```

```
In [40]: C / numpy.linalg.norm(C, axis=1)
```

```
ValueError                                Traceback (most recent call last)
<ipython-input-40-56f89923ab30> in <module>()
----> 1 C / numpy.linalg.norm(C, axis=1)
```

ValueError: operands could not be broadcast together with shapes (5,3) (5,)

```
In [41]: C / numpy.linalg.norm(C, axis=1).reshape(C.shape[0], 1)
```

```
Out[41]:
```



```
array([[ 1.          ,  0.          ,  0.          ],
       [ 0.8         , -0.6         ,  0.          ],
       [ 0.7592566 , -0.65079137,  0.          ],
       [ 0.74329415, -0.66896473,  0.          ],
       [ 0.73480344, -0.6782801 ,  0.          ]])
```

each row = $B[i,:] \times d / |B[i,:] \times d|$

How fast?

Test with matrix multiplication:

$$\begin{pmatrix} h_1 & k_1 & l_1 \\ & \vdots & \\ h_N & k_N & l_N \end{pmatrix} \begin{pmatrix} \mathbf{b}_1^{*t} \\ \mathbf{b}_2^{*t} \\ \mathbf{b}_3^{*t} \end{pmatrix} = \begin{pmatrix} \mathbf{p}_0^{*t} \\ \vdots \\ \vdots \end{pmatrix}$$

```
In [1]: import numpy
```

```
In [2]: h = numpy.random.randint(-100,100,3*1000000).reshape(1000000,3)
```

```
In [3]: a = numpy.array([[-0.00451148, -0.01603847, -0.00454565],  
...: [ 0.00595611,  0.00291487, -0.01597257],  
...: [ 0.00600637, -0.00221528,  0.00184483]])
```

```
In [4]: def mul(A, b):  
...:     ret = numpy.empty(A.shape)  
...:     for i in xrange(A.shape[0]):  
...:         for j in xrange(A.shape[1]):  
...:             ret[i,j] = sum(map(lambda x: x[0]*x[1], zip(A[i,:], b[:,j])))  
...:     return ret  
...:
```

```
In [5]: %timeit numpy.dot(h, a)  
10 loops, best of 3: 23.6 ms per loop
```

```
In [6]: %timeit mul(h, a)  
1 loops, best of 3: 21.8 s per loop
```



~900× faster
(slow when for-loop was used)

Generate a complete list of miller indices

```
In : from cctbx import crystal
In : from cctbx import miller
In : import numpy
In : xs = crystal.symmetry(unit_cell=(100,100,100,90,90,90), space_group="p1")
In : miller_set = miller.build_set(xs, anomalous_flag=True, d_min=3.0)
In : miller_set.indices() # flex.miller_index object
Out: <cctbx_array_family_flex_ext.miller_index at 0x10c448050>
In : miller_set.indices()[0]
Out: (-33, -4, 1)
In : miller_set.indices().size() # number of indices
Out: 155330
In : h = numpy.array(miller_set.indices()) # N×3 matrix
In : h
Out:
array([[[-33, -4, 1],
       [ 33,  4, -1],
       [-33, -4, 2],
       ...,
       [-33, -4, -1],
       [ 33,  4,  2],
       [-33, -4, -2]]])
```

“Main” function of the program

```
if __name__ == "__main__":
    import sys

    xparm_in = sys.argv[1]
    d_min = float(sys.argv[2])
    sigma_m = float(sys.argv[3])
    frames = map(int, sys.argv[4:])

    preds = Predictions()
    print "Reading XPARM.XDS.."
    preds.read_xparm(xparm_in)

    indices = preds.prep_indices(d_min)

    print "Calculating the predicted centroids.."
    preds.calc_centroids(indices)

    for frame in frames:
        print "Calculating the predictions on frame %d.." % frame
        pindices, pdata = preds.get_predicted_positions(sigma_m, frame)

        ofs = open("prediction_{:6d}.adx".format(frame), "w")
        for (h,k,l), (x, y, phi, zeta) in zip(pindices, pdata):
            ofs.write("{} {} {} {} {}\n".format(x, y, h, k, l))

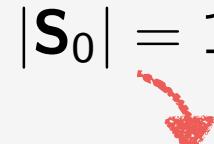
        ofs.close()
```

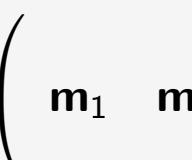
Parsing XPARM.XDS

```

def read_xparm(self, xpin):
    fin = open(xpin)
    assert "XPARM.XDS" in fin.readline()

    # Line 2
    sp = fin.readline().split()
    self.starting_frame = int(sp[0])
    self.starting_angle, self.osc_range = float(sp[1]), float(sp[2])
    m2 = numpy.array(map(float, sp[3:6]))  m2

    # Line 3
    sp = fin.readline().split()
    self.wavelength = float(sp[0])  |S0 | = 1/ $\lambda$ 
    incident_beam = numpy.array(map(float, sp[1:4]))
    self.s0 = incident_beam / numpy.linalg.norm(incident_beam) / self.wavelength

    m = numpy.empty(dtype=numpy.float, shape=(3,3))
    m[:,1] = m2 / numpy.linalg.norm(m2)
    m[:,0] = numpy.cross(m[:,1], self.s0) 
    m[:,0] /= numpy.linalg.norm(m[:,0])
    m[:,2] = numpy.cross(m[:,0], m[:,1])
    self.m_matrix = m 

```

XPARM.XDS	VERSION	May 1, 2016	BUILT=20160617					
1	0.0000	0.1500	1.000000	0.000000	0.000000			
0.976250		0.001331		0.000727		1.024327		
1	57.9046	57.8227	150.0966	89.926	90.023	89.805		
-15.079662		-53.735268		-15.429407				
19.930243		9.538352		-53.434731				
135.399963		-49.820015		41.398369				
1	2463	2527	0.172000		0.172000			
1225.349976		1193.469971		265.269989				
1.000000		0.000000		0.000000				
0.000000		1.000000		0.000000				
0.000000		0.000000		1.000000				
1		1	2463	1	2527			
0.00	0.00	0.00	1.000000	0.000000	0.000000	0.000000	1.000000	0.000000

$$\begin{pmatrix} \mathbf{m}_1 & \mathbf{m}_2 & \mathbf{m}_3 \end{pmatrix} = \begin{pmatrix} \frac{\mathbf{m}_2 \times \mathbf{S}_0}{|\mathbf{m}_2 \times \mathbf{S}_0|} & \mathbf{m}_2 & \mathbf{m}_1 \times \mathbf{m}_2 \end{pmatrix}$$

Parsing XPARM.XDS

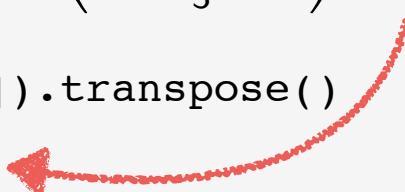
```
# Line 4
sp = fin.readline().split()
self.crystal_symmetry = crystal.symmetry(unit_cell=map(float, sp[1:7]),
                                             space_group_symbol=int(sp[0]))
```

Line 5,6,7 real space vectors

```
a_axis = map(float, fin.readline().split())
b_axis = map(float, fin.readline().split())
c_axis = map(float, fin.readline().split())
```

$$\begin{pmatrix} \mathbf{b}_1^{*t} \\ \mathbf{b}_2^{*t} \\ \mathbf{b}_3^{*t} \end{pmatrix} = \begin{pmatrix} \mathbf{b}_1 & \mathbf{b}_2 & \mathbf{b}_3 \end{pmatrix}^{-1}$$

```
b_mat = numpy.array([a_axis, b_axis, c_axis]).transpose()
self.astar_matrix = numpy.linalg.inv(b_mat)
```



```
# Line 8 detector dimensions and pixel size
sp = fin.readline().split()
self.nxy = map(int, sp[1:3]) # NX, NY
self.qxy = map(float, sp[3:5]) # QX, QY
```

Parsing XPARM.XDS

```

XPARM.XDS      VERSION May 1, 2016   BUILT=20160617
    1        0.0000    0.1500  1.000000  0.000000  0.000000
    0.976250    0.001331    0.000727    1.024327
    1      57.9046    57.8227  150.0966  89.926  90.023  89.805
   -15.079662   -53.735268   -15.429407
   19.930243     9.538352   -53.434731
  135.399963   -49.820015   41.398369
    1        2463     2527  0.172000  0.172000
1225.349976  1193.469971  265.269989
    1.000000    0.000000    0.000000
    0.000000    1.000000    0.000000
    0.000000    0.000000    1.000000
    1            1       2463        1      2527
  0.00      0.00     0.00  1.000000  0.000000  0.000000  0.000000  1.000000  0.000000

```

```
# Line 9 ORGX,Y & detector distance
sp = fin.readline().split()
self.orgxy = map(float, sp[:2])
self.detector_F = float(sp[2])

# Line 10,11,12
d1 = map(float, fin.readline().split())
d2 = map(float, fin.readline().split())
d3 = map(float, fin.readline().split())

self.d_matrix = numpy.array([d1,d2,d3]).transpose()
```

$$\begin{pmatrix} \mathbf{d}_1 & \mathbf{d}_2 & \mathbf{d}_3 \end{pmatrix}$$

Implementation of centroid calculation

```
def calc_centroids(self, indices):
    h = numpy.array(indices) # hkl in each row
    m, d, a, F = self.m_matrix, self.d_matrix, self.astar_matrix, self.detector_F
    orgx, orgy = self.orgxy
    qx, qy = self.qxy
    s0 = self.s0
    p0s = numpy.dot(h, a)
    p0s_m = numpy.dot(p0s, m)
    s0_m = numpy.dot(m.transpose(), s0)
    p0s_lensq = numpy.sum(p0s**2, axis=1)
```

$$\begin{pmatrix} h_1 & k_1 & l_1 \\ \vdots & \ddots & \vdots \\ h_N & k_N & l_N \end{pmatrix} \begin{pmatrix} \mathbf{b}_1^{*t} \\ \mathbf{b}_2^{*t} \\ \mathbf{b}_3^{*t} \end{pmatrix} = \begin{pmatrix} \mathbf{p}_0^{*t} \\ \vdots \\ \vdots \end{pmatrix}$$
$$\begin{pmatrix} \mathbf{p}_0^{*t} \\ \vdots \end{pmatrix} \begin{pmatrix} \mathbf{m}_1 & \mathbf{m}_2 & \mathbf{m}_3 \end{pmatrix} = \begin{pmatrix} (\mathbf{m}_1 \cdot \mathbf{p}_0^*) & (\mathbf{m}_2 \cdot \mathbf{p}_0^*) & (\mathbf{m}_3 \cdot \mathbf{p}_0^*) \\ \vdots & \ddots & \vdots \end{pmatrix}$$
$$\begin{pmatrix} \mathbf{m}_1^t \\ \mathbf{m}_2^t \\ \mathbf{m}_3^t \end{pmatrix} \cdot \mathbf{s}_0 = \begin{pmatrix} \mathbf{m}_1 \cdot \mathbf{s}_0 \\ \mathbf{m}_2 \cdot \mathbf{s}_0 \\ \mathbf{m}_3 \cdot \mathbf{s}_0 \end{pmatrix}$$
$$\begin{pmatrix} \mathbf{p}_0^{*2} \\ \vdots \end{pmatrix}$$

Implementation of centroid calculation



$$\left(\begin{array}{ccc} (\mathbf{m}_1 \cdot \mathbf{p}^*) & (\mathbf{m}_2 \cdot \mathbf{p}^*) & (\mathbf{m}_3 \cdot \mathbf{p}^*) \\ & \vdots & \end{array} \right) \quad \left\{ \begin{array}{l} \mathbf{m}_3 \cdot \mathbf{p}^* = \frac{-\mathbf{p}^{*2}/2 - (\mathbf{m}_2 \cdot \mathbf{s}_0)(\mathbf{m}_2 \cdot \mathbf{p}_0^*)}{\mathbf{m}_3 \cdot \mathbf{s}_0} \\ \mathbf{m}_2 \cdot \mathbf{p}^* = \mathbf{m}_2 \cdot \mathbf{p}_0^* \\ \mathbf{m}_1 \cdot \mathbf{p}^* = \pm \sqrt{\mathbf{p}_0^{*2} - (\mathbf{m}_2 \cdot \mathbf{p}^*)^2 - (\mathbf{m}_3 \cdot \mathbf{p}^*)^2} \end{array} \right.$$

```
ps_m = numpy.empty(p0s_m.shape)
ps_m[:,2] = (-0.5*p0s_lensq - s0_m[1]*p0s_m[:,1]) / s0_m[2]
ps_m[:,1] = p0s_m[:,1]
ps_m[:,0] = p0s_lensq - ps_m[:,1]**2 - ps_m[:,2]**2 # sqrt after check

sel_ok = ps_m[:,0] > 0 # No solution (blind region) if < 0
h, p0s_m, ps_m = h[sel_ok], p0s_m[sel_ok], ps_m[sel_ok]
ps_m[:,0] = numpy.sqrt(ps_m[:,0])

self.predicted_hkl = numpy.empty((0, 3), dtype=numpy.int) # h,k,l
self.predicted_data = numpy.empty((0, 4)) # x, y, phi, zeta
```

Implementation of centroid calculation

```

for p1sign in (+1, -1):
    ps_m[:, 0] *= p1sign
    phi = numpy.arctan2(p0s_m[:, 2]*ps_m[:, 0]-p0s_m[:, 0]*ps_m[:, 2],
                         p0s_m[:, 0]*ps_m[:, 0]+p0s_m[:, 2]*ps_m[:, 2])
    e1_m = numpy.cross(ps_m, s0_m)
    e1_m /= numpy.linalg.norm(e1_m, axis=1).reshape(e1_m.shape[0], 1)
    zeta = e1_m[:, 1]
    
$$\zeta = \begin{pmatrix} (\mathbf{m}_1 \cdot \mathbf{e}_1) & (\mathbf{m}_2 \cdot \mathbf{e}_2) & (\mathbf{m}_3 \cdot \mathbf{e}_3) \\ \vdots & \vdots & \vdots \end{pmatrix}$$

    s = s0 + numpy.dot(ps_m, m.transpose())
    s_d = numpy.dot(s, d)
    
$$s_d = \begin{pmatrix} (\mathbf{d}_1 \cdot \mathbf{S}) & (\mathbf{d}_2 \cdot \mathbf{S}) & (\mathbf{d}_3 \cdot \mathbf{S}) \\ \vdots & \vdots & \vdots \end{pmatrix}$$

    sel_ok = F*s_d[:, 2] > 0
    s_d, h_ok, phi, zeta = s_d[sel_ok], h[sel_ok], phi[sel_ok], zeta[sel_ok]

    xdet = orgx + F*s_d[:, 0]/s_d[:, 2]/qx
    ydet = orgy + F*s_d[:, 1]/s_d[:, 2]/qy
    
$$\begin{cases} X = X_0 + F(\mathbf{S} \cdot \mathbf{d}_1)/(\mathbf{S} \cdot \mathbf{d}_3) \\ Y = Y_0 + F(\mathbf{S} \cdot \mathbf{d}_2)/(\mathbf{S} \cdot \mathbf{d}_3) \end{cases}$$


    self.predicted_hkl = numpy.row_stack([self.predicted_hkl, h_ok])
    self.predicted_data = numpy.row_stack([self.predicted_data,
                                           numpy.column_stack([xdet, ydet, phi, zeta])])

```

32

Select predictions on specific frame

```
def get_predicted_positions(self, sigma_m, frame, esd_factor=3.5):
    phi = self.starting_angle + self.osc_range*(frame-self.starting_frame+0.5)
    print "Phi at frame %d = %.3f" % (frame, phi)

    phi, sigma_m = numpy.deg2rad([phi, sigma_m])

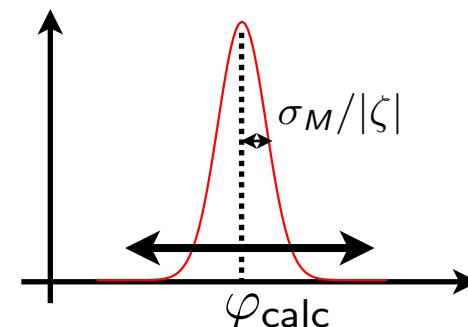
    phi_calc = self.predicted_data[:,2]
    zeta = self.predicted_data[:,3]

    phi_diff = numpy.fmod(phi_calc - phi, 2.*numpy.pi)
    phi_diff[phi_diff < -numpy.pi] += 2.*numpy.pi
    phi_diff[phi_diff > numpy.pi] -= 2.*numpy.pi

    sel = numpy.abs(phi_diff) < esd_factor * sigma_m / numpy.abs(zeta)
    return self.predicted_hkl[sel], self.predicted_data[sel]
```

Make the difference in $[-\pi, \pi]$ range

Predict if $|\varphi_{\text{calc}} - \varphi_{\text{frame}}| < 3.5\sigma_M/|\zeta|$

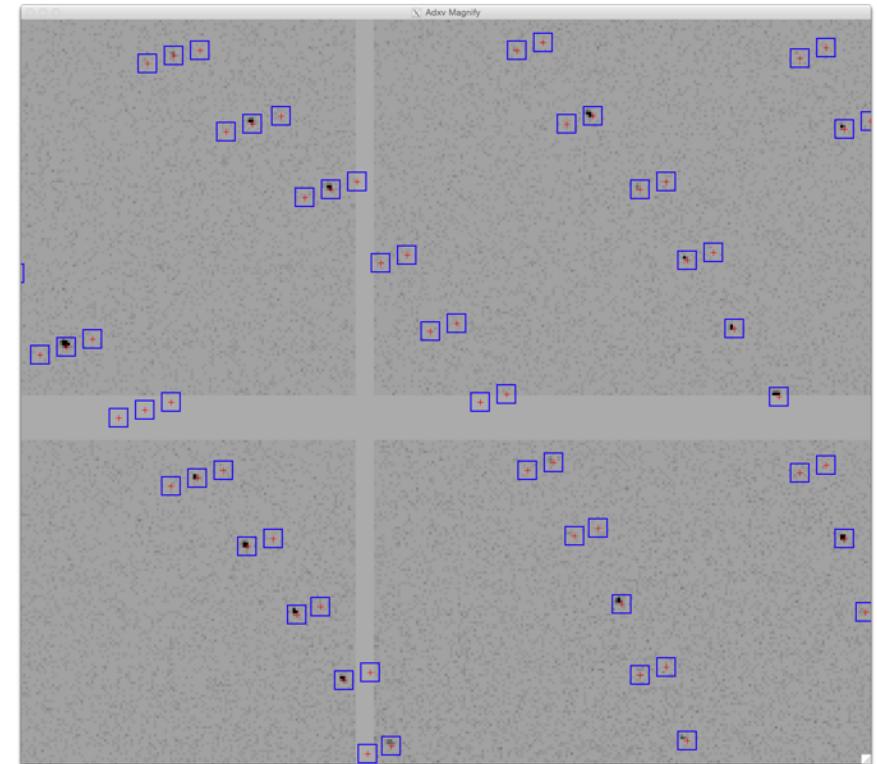
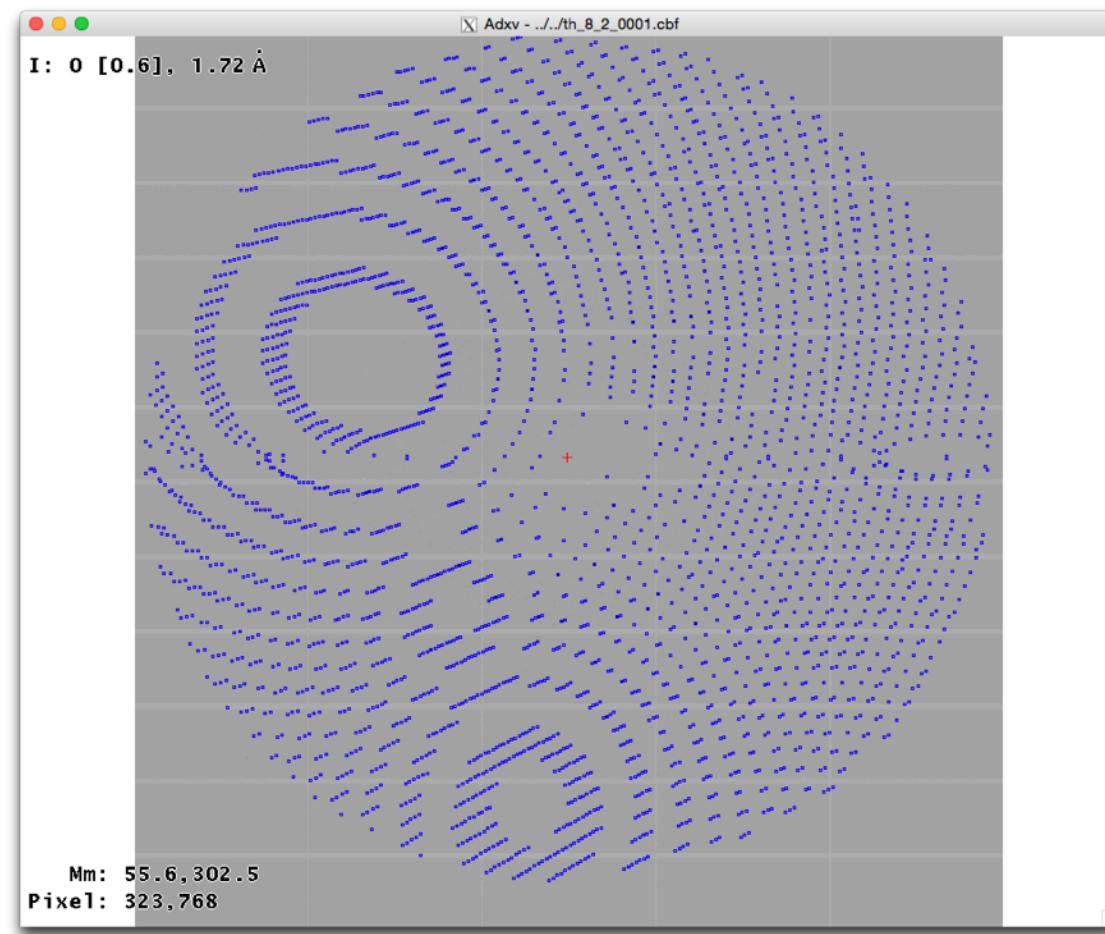


Result

```
$ phenix.python ./prediction.py ./XPARM.XDS 1.5 0.06928 1
```

```
Reading XPARM.XDS..  
Calculating the predicted centroids..  
Calculating the predictions on frame 1..  
Phi at frame 1 = 0.075
```

```
$ adxv ../../th_8_2_0001.cbf prediction_00001.adx
```

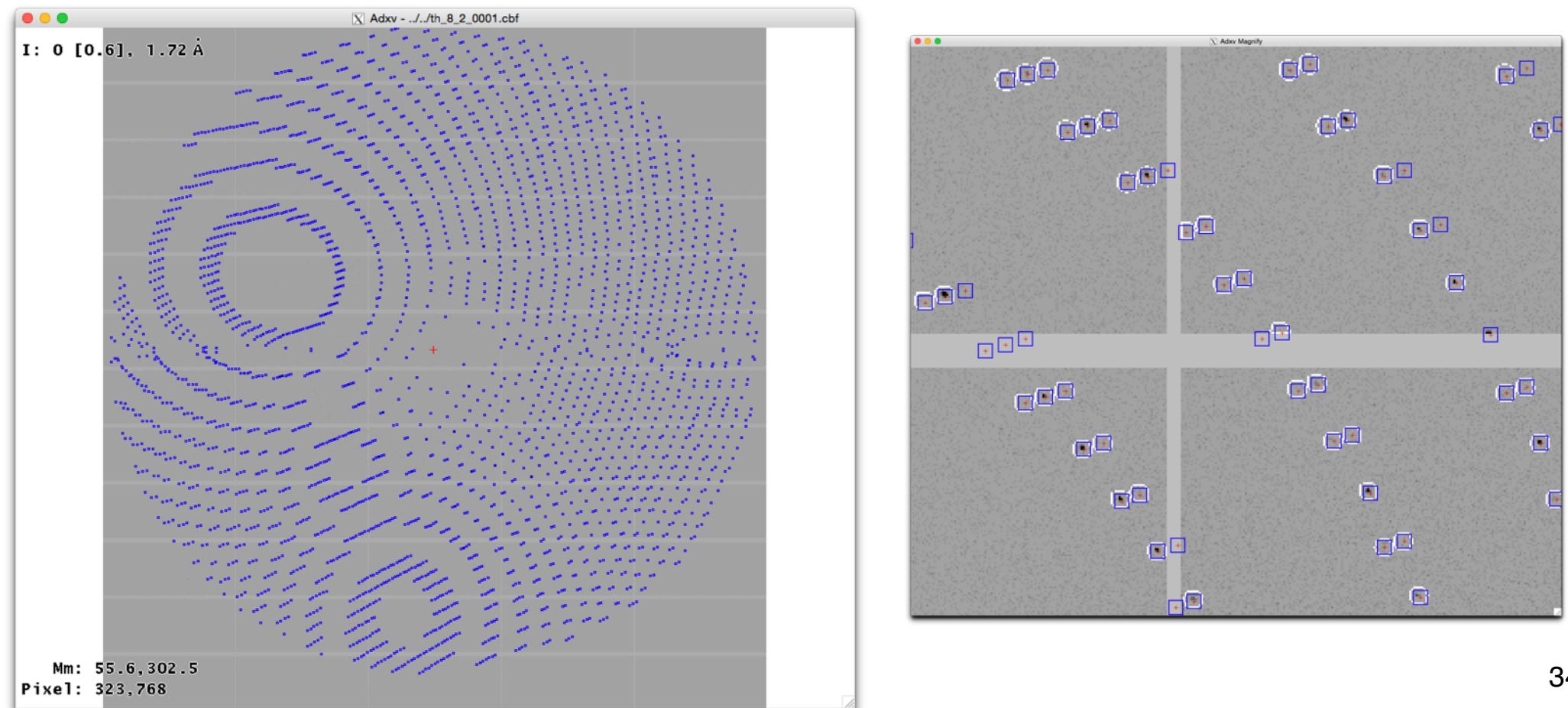


Result

```
$ phenix.python ./prediction.py ./XPARM.XDS 1.5 0.06928 1
```

```
Reading XPARM.XDS..  
Calculating the predicted centroids..  
Calculating the predictions on frame 1..  
Phi at frame 1 = 0.075
```

```
$ adxv ../../th_8_2_0001.cbf prediction_00001.adx
```



Final remarks

- We wrote a Python code to calculate reflection centroids and make a prediction on specific frames
- There are relatively large errors in “Lorentz zone”
 - We only calculated (x, y, φ) of the centroids of reflections
 - To know more accurate spot positions on a frame, we may need to model individual reflection routes as XDS ver. Dec 31, 2011 (and newer) does

Acknowledgement

I thank Takanori Nakane
for discussions and useful blog articles*

* http://d.hatena.ne.jp/biochem_fan/20150727/1437982461