A Python Tool for Optical Ray Tracing
DESI-2745

Mike Lampton, UCB SSL

May 16, 2017

1 Resources and Previous Studies

Lampton, M., DESI-2745 companion sourcecode, demo files, test files (.zip)
Sholl, M., DESI-0329 v15 Description of E22 Corrector, Section 4.2
Arizona Optical Systems Status Report, 10 January 2017 "The Dimple"
AOS C3-Front Sag Table "AOS-C3-data-7Feb2017.dat"
Lampton, M., "Consequences of the C3 Dimple," DESI-2642 v3 Feb 2017
Lampton, M., BEAM FOUR Users Guide v.190 Jan 2016 Stellar Software
G.H.Spencer and M.Murty, "General ray-tracing procedure," JOSA v52 No 6 (1962).

2 Introduction

As part of the validation process for accepting DESI's as-built corrector lens package, it is valuable to have a variety of optical ray tracing tools available that can be used to cross-check each other and that can perform three frequently needed tasks:

- Evaluate the nominally aligned corrector under various settings (focus; ADC prisms...);
- Evaluate the mis-aligned corrector under those same settings;
- Evaluate the corrector performance with as-built lens profile test data.

For this purpose I provide a Python-based optical ray tracer. Its foundation is intended to be highly traceable. Most functions derive from the original Spencer and Murty formulation, and have many features in common with the the open-source code functions in the BEAM FOUR Java-language optical ray tracer. Deviations from that flowdown are described in this writeup, and commented directly within the source code.

At this writing, the current source code release carries a file name of **RT22.py** and is co-posted with this writeup on DESI-DOC. This will increment as further work is posted.

The most important segments of this writeup are: **input data**, **internal processing**, and **output**. Following those are short sections about test results, a list of Zernike polynomials, and a summary of the Hermite cubic interpolation method. I conclude with a short demo of the Echo-22 corrector illustrating the effects of the as-built surface map of its C3Front. The

source code RT122.py starts with a brief Table of Contents, saying where its components are to be found.

3 Input Data

To perform an optical ray trace, three kinds of initial data are needed, and these are traditionally furnished as three input files. For easiest communication to/from spreadsheets, this Python tool assumes all input files are in "CSV" (comma separted values) format: plain ASCII text, with fields separated with commas. Except for the commas, these closely follow the BEAM FOUR file conventions and indeed a tiny utility "B4toCSV.py" is provided with this posting.

- An optics table: a list of the surfaces that will interact with, in the right order.
- A ray start table: a list of the ray starts with which the optic is to be illuminated.
- A glass table: to give the refractive index of each glass, with a wavelength provided by the ray start table, and the name of the glass of the medium approaching each surface.
- Optionally, one or more sag maps describing measured surface deviations.

3.1 Optics Table

Every optical surface has a vertex location (X, Y, Z) in space, and an orientation specified by the three angles (Tilt, Pitch, Roll). Missing specifications default to zero. Other input parameter specifications determine the shapes and size of the optical surfaces. Specifically the parameters that are currently supported these:

- Location of the surface (X,Y,Z) and orientation (Tilt,Pitch,Roll) in lab coordinates;
- Outer and inner diameters, distinguished by "Diam" vs "diam";
- Curvature "Curv" and asphericity "Asph" for conic sections of revolution;
- Polynomial coefficients A1 ... A14;
- Zernike polynomials Zern0 ... Zern36.

Here's an example of an optics table describing a Newtonian telescope and eyepiece.

```
   9  , surfaces                NEWTON.OPT.CSV
Index ,Diam,diam, X   ,   Z   ,Tilt,Pitch,  Curv   ,Asph  , Mirror?           ,
      , 2.1, 0.4,     ,  1.0  ,    ,     ,         ,      ,iris & obstruction ,
      , 2.2,    ,     , 10.0  ,    ,     ,  -0.050 , -1.0 ,MIR primary        ,
      , 0.7,    ,     ,  2.0  ,    , 45  ,         ,      ,mirror- 45deg      ,
      , 0.2,    , 2.0 ,  2.0  ,    , 90  ,         ,      ,iris               ,
      , 0.6,    , 3.0 ,  2.0  ,    , 90  ,   0.0   ,      ,lens               ,
BK7   , 0.6,    , 3.1 ,  2.0  ,    , 90  , -1.254  ,      ,lens               ,
      , 0.6,    , 3.4 ,  2.0  ,    , 90  ,  0.417  ,      ,lens               ,
SF12  , 0.6,    , 3.5 ,  2.0  ,    , 90  ,   0.0   ,      ,lens               ,
      , 1.0,    , 4.0 ,  2.0  ,    , 90  ,         ,      ,eye                ,
```

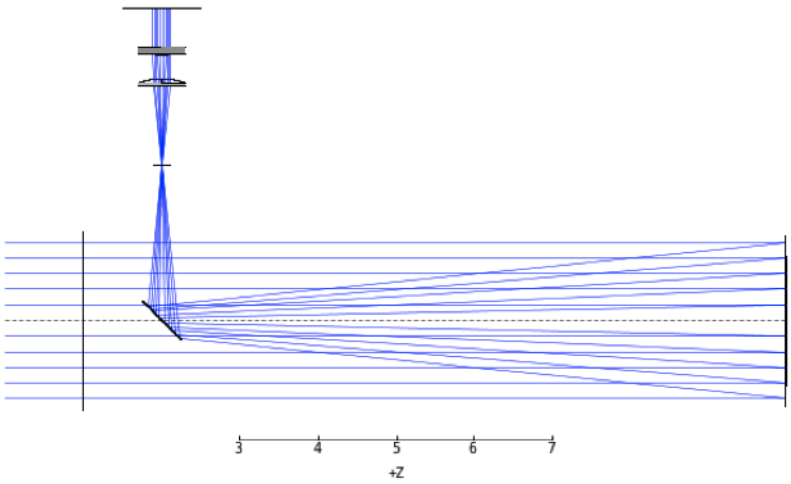

Figure 1: Side view of the Newton optic described here; artwork is a BEAM FOUR layout.

In these tables, the top line or title line is ignored except for its initial guide number that controls how many of the table's data records are to be read and parsed. The second line shows the field headers – these can be abbreviated since in most cases just the initial one or two characters are used. In an optics table, the successive data records define the sequence of optical surfaces in the same order that light will visit them. Refractive index data defaults to 1.00 meaning air; the specified index refers to the medium in which light approaches the surface. Numerical entries specify the refractive index, while glass names are acceptable provided that a glass table (below) is furnished listing those glasses.

3.2 Ray Table

Mathematically, a ray has a point of origin (X0, Y0, Z0) and an initial unit vector direction (U0, V0, W0). In addition to these six numbers, each ray is usually supplied with its lab wavelength, usually in microns. The header "@wave" identifies the wavelength field. So seven numbers are used to describe each ray start. Wavelengths need not be numerical, if they can be looked up in the glass table provided.

```
  10  , RAYS        NEWTON.RAY.CSV
   X0 ,   Y0 , U0 ,@wave, Note    ,
  1.0 ,  0   , 0  ,   C , OK 9    ,
  0.8 ,  0   , 0  ,   C , OK 9    ,
  0.6 ,  0   , 0  ,   C , OK 9    ,
  0.4 ,  0   , 0  ,   C , OK 9    ,
  0.2 ,  0   , 0  ,   C , OK 9    ,
 -0.2 ,  0   , 0  ,   C , OK 9    ,
 -0.4 ,  0   , 0  ,   C , OK 9    ,
 -0.6 ,  0   , 0  ,   C , OK 9    ,
 -0.8 ,  0   , 0  ,   C , OK 9    ,
 -1.0 ,  0   , 0  ,   C , OK 9    ,
```

In BEAM FOUR a variety of ray output data can be chosen to be included in each ray table. Numerical outputs in this Python tool are not part of its ray tables, but are gathered from the runrays() function described below.

3.3 Glass Table

The glass table is a simple lookup table. It lists each glass down the page, and for each glass lists its refraction index across the page, according to the wavelength. An example is shown here. It uses the Fraunhofer line identifier letters, plus HeNe.

```
11  , entries            GLASS.MED.CSV
Material                ,   C      ,   D      ,   F      ,  HeNe    ,      ,
silica                  ,  1.456369,  1.458404,  1.463128,  1.457020, 67.8 ,
BK7                     ,  1.514323,  1.516728,  1.522377,  1.515089, 64.2 ,
K5                      ,  1.519816,  1.522411,  1.528600,  1.520640, 59.5 ,
SK16                    ,  1.617273,  1.620320,  1.627558,  1.618241, 60.3 ,
SF12                    ,  1.642710,  1.648143,  1.661868,  1.644410, 33.8 ,
LAK8                    ,  1.708975,  1.712885,  1.722220,  1.710215, 53.8 ,
sapphire                ,  1.764943,  1.768138,  1.775580,  1.765963, 72.2 ,
SF11                    ,  1.775986,  1.784458,  1.806455,  1.778621, 25.6 ,
```

3.4 Sag Table

Optical manufacturers often choose to supply surface metrology in the form of a sag map or sag table: a list of vertical surface deviations from the nominal ideal surface, with values shown on an NxN grid of equally spaced surface locations. These data typically originate in an interferometric fringe map, from which a regular sampling grid evaluates the integrated surface deviations from nominal. One such table is the C3-Front table provided to DESI by our contractor, AOS; its filename is "AOS-C3-data-7Feb2017.dat" and it consists of a single header line followed by 68121 records (= 261 x 261 entries). These present a square raster scan, zeros around the corners, but nonzero data throughout the central 261 point circular diameter field. At 3mm per data point, this circle spans 780mm.

```
261 261 3.000000 3.000000 0
0.000000e+00 0 0 0 1
0.000000e+00 0 0 0 1
. . . . .
```

Files like this are trivial to read and plot in Python. This code snippet does the trick:

```
def getSag(filename):
    try:
        saglist = np.loadtxt(filename, skiprows=1, usecols=(0,))
    except IOError:
        print 'sag list is unavailable: ', filename
        quit()
    sagtable = np.reshape(saglist, (261,261))
    plt.imshow(sagtable)
    plt.colorbar()
    plt.title(name)
    plt.show()
```

It reads the text file into an array of numbers, reformats those numbers into a 2D array, and plots them using the popular MatPlotLib graphics library. That plot is shown here.

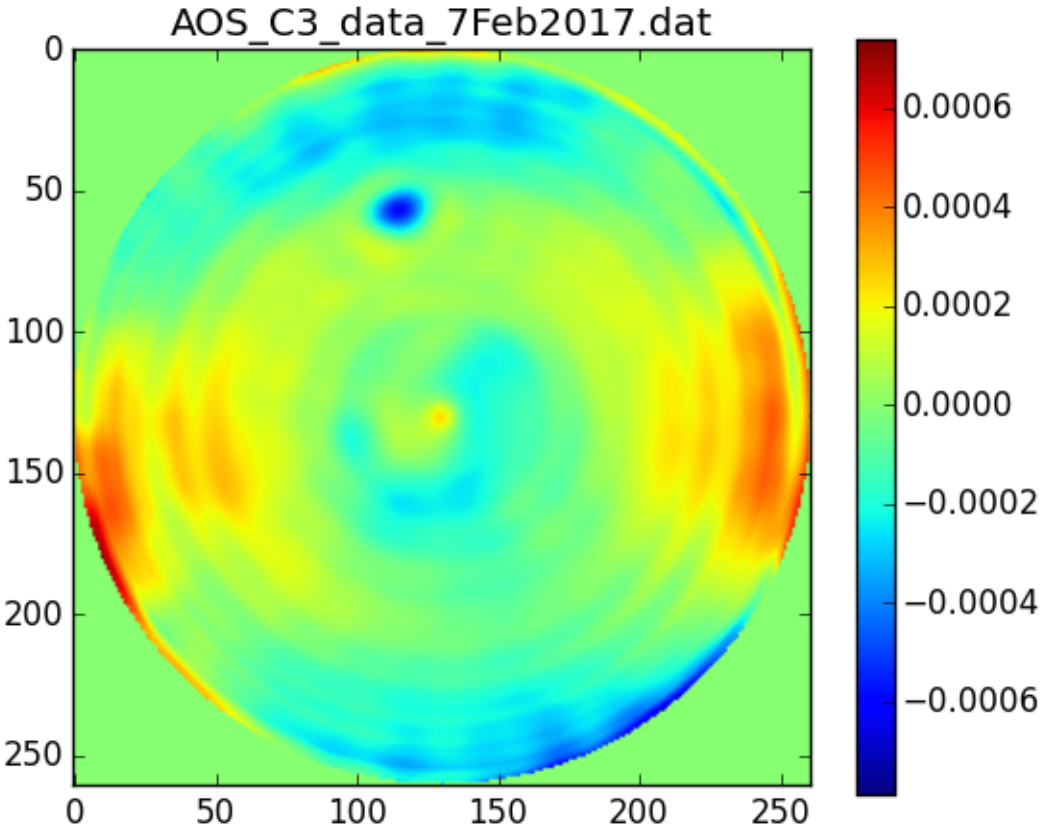

Figure 2: Sag file converted to 2D and plotted.

4 Internal Processing

4.1 Data Capture

The first stage of internal processing is to extract and organize the informtion provided by the optics table, the ray start table, the media table, and if present, the sag tables. These data are represented internally as a family of lists; the "O" lists come from the Optics table, and so forth. In addition, these list data are parsed and loaded into a set of full-double-precision numpy arrays are created to speed the individual ray processing. The calls are unremarkable:

```
getOpticsCSV(myOptFileName)
getRaysCSV(myRayFileName)
getMediaCSV(myMedFileName)
getSag(mySagFileName)
```

4.2 Ray Propagation

Within this numerical environment, the second stage of processing is to run the rays. There is an outer loop, that selects successive ray start numbers (1 through Nrays inclusive) to load and start, and there is an inner loop that steps one ray start through the successive optical surfaces numbered 1 to Nsurfs inclusive. Codewise such loops are straightforward. Here's a summary of the two methods, not showing the escape clauses for error codes:

```
def runAllRays():
    for iray in range(1, Nrays+1):
        displayInput(iray)
        howfar = runOneRay(iray)
        if howfar == Nsurfs:  # success!
            displayOutput(iray, howfar)

def runOneRay(iray):
    howfar = 0
    for jtarget in range(1, Nsurfs+1):
        errcode = labtovx(iray, jtarget)
        errcode = intercept(iray, jtarget)
        errcode = validate(iray, jtarget)
        errcode = redirect(iray, jtarget)
        errcode = vxtolab(iray, jtarget)
        howfar = jtarget
    return howfar
```

Here are some details on each of the five steps.

4.2.1 LabToVx

Two coordinate systems are carried: the lab frame, in which ray starts are expressed and which each ray segment process furnishes as a concluding step. The other frame is the local frame, whose origin is the vertex of the target surface under study. Its orientation is given by the declared angular triplet (Tilt, Pitch, Roll) and its location (X, Y, Z) in lab frame coordinates. The actual work is done in local vertex coordinates because, in this frame, the math is simpler. This routine **LabToVx** does not move the ray, but merely converts the lab frame coordinates from the previous surface into its coordinates of the present surface. Or, if this is the initial surface, the lab coordinates are taken from the list of declared lab-frame ray starts. This transformation cannot fail, so its error code is always "OK." The reorientations are facilitated by having precomputed the unitary Euler matrix transformations for each optical surface. These are created as part of input processing of the Optics definition file.

4.2.2 Intercept

Intercept is where half the work is done. From the ray's local frame origin at the previous surface, where will this ray intercept the target surface? For a simple plano surface the equation set is linear and there is just one root: a legal intercept if the ray travel distance is positive, otherwise a failure of type **backwards**. For a conic section of revolution there can be zero, one, or two roots, and again we must discard any with negative intercept distances, and also discard any that lie on the phantom hyperboloid sheet or the backside ellipsoid

sheet. When there are two valid intercepts, I choose the one with shorter propagation distance. Higher surfaces (polynomial, Zernike, and sag surfaces) have no general closed-form analytic intercept solution. I use an iterative solver; specifically I use the conic solution as the starting point for an iterative Newton root finder. In practice this converges quickly when the higher correction terms (polynomial departures, surface sag) are small compared to the conic intercept distance. **Intercept** propagates the ray, in local coordinates, to the new found intercept location in local (vertex) coordinates.

4.2.3 Validate

Surfaces may have declared outer and inner diameters. The **Validate** function simply passes rays whose intercepts lie between these two diameters, and fails the others.

4.2.4 Redirect

The other half of the work is to determine a valid ray's new direction. **Redirect** is a switchyard that uses the surface action type to decide which of several redirectors to apply. The surface actions are these, in order of increasing complexity:

- IrisAction (requires nothing)
- RetroAction (requires nothing)
- MirrorAction (needs the local surface normal vector)
- LensAction (needs local surface normal vector and refractive indices)

IrisAction has nothing to do: **Validate** has done the work.

RetroAction has one simple task. It reverses all three components of the unit ray direction vector: $\vec{U}_{out} = -\vec{U}_{in}$. It is completely unphysical: it violates the principle of stationary phase. But it is very useful in redirecting backward rays that originate at a pupil, causing them to return to that pupil with full validation.

MirrorAction has the task of reversing the normal component of the local unit ray direction vector. To accomplish this, I evaluate the local unit surface normal vector direction $\vec{N}$ and apply the vector dot product: $\vec{U}_{out} = \vec{U}_{in} - 2\,\vec{N}\,(\vec{U}_{in} \cdot \vec{N})$.

LensAction must provide a new ray direction unit vector. Seen as an electromagnetic boundary value matching issue, the wave vector $k_{parallel}$ on the incoming and outgoing sides of the surface must be equal, hence

$$n_{in}\,\vec{U}_{in,parallel} = n_{out}\,\vec{U}_{out,parallel} \tag{1}$$

Here the refractive indices are n_{in} and n_{out} and the parallel vectors are the surface components of the unit ray directions. With $\vec{N}$ the local unit normal, $\vec{U}_{in,parallel} = \vec{N} \times \vec{U}_{in} \times \vec{N}$ we get

$$\vec{U}_{out,parallel} = \mu\,\vec{U}_{in,parallel} = \mu\,\vec{N} \times \vec{U}_{in} \times \vec{N} \tag{2}$$

where $\mu = n_{in}/n_{out}$. If this parallel portion of the outgoing unit vector exceeds 1.0, there is no solution and a total-internal-reflection TIR failure must be declared. Otherwise, $\vec{U}_{perp}$

must complete the square: $\vec{U}_{out,perp} = \vec{N}\ (1 - U^2_{out,parallel})^{1/2}$. This component's sign must be chosen to be the transmitted wave, which is to say that its component along the surface normal must agree with the incoming vector's sign. Finally, $\vec{U}_{out} = \vec{U}_{out,parallel} + \vec{U}_{out,perp.}$

4.2.5 VxToLab

The final step at each surface is to convert the local frame ray properties to lab frame. Like **LabToVx** this routine is nothing more that a combination of a translation and a rotation.

5 Output

The present releases have primitive output functions, simply giving screen text listings for the final labframe ray coordinates of each successful ray's trace. It is expected that a subset of these output data will be gathered and organized into useful graphic products depending on the specific research task.

One example of graphical output is the spot diagram. This is a plotted collection of the (X,Y) coordinates of those rays that have successfully navigated the optic. To set up the input for a spot diagram it is important to illuminate the pupil uniformly with rays, and then gather the successful ray completions into a list of valid coordinates.

The routine *Circles.py* produces a CSV file of ray starts (X,Y) in pupil coordinates, organized into concentric circles that fill a circular pupil of given radius. This ray generator is posted among the working files under this DESI number.

The code snippet below shows how to gather the good ray Xpoints and Ypoints into lists, and then when all ray starts have been tried, produce a plot showing the distribution of the successful rays. To see them in their final focal surface coordinates, call out surface number Nsurfs from Rarray[]. To see the successful rays elsewhere in the system, retain the qualifier **if howfar==Nsurfs** but extract the Xpoints and Ypoints from elsewhere in Rarray[ ]. The optics surface definition file here is E22.OPT.CSV, also posted.

```
xpoints = [ ]  # empty list to gather good ray X values
ypoints = [ ]  # empty list to gather good ray Y values

def runAllRays():
    for iray in range(1, Nrays+1):
        howfar = runOneRay(iray)
        if howfar == Nsurfs:
            xpoints.append(Rarray[iray, Nsurfs, RX])
            ypoints.append(Rarray[iray, Nsurfs, RY])

#---analyze and show the good rays---------

runAllRays()
Xrms = np.std(xpoints)
Yrms = np.std(ypoints)
print "Xrms, Yrms = {:12.6f}{:12.6f}".format(Xrms, Yrms)

plt.plot(xpoints, ypoints, 'ro')
plt.axis('equal')
plt.title(myRay)
plt.show()
```

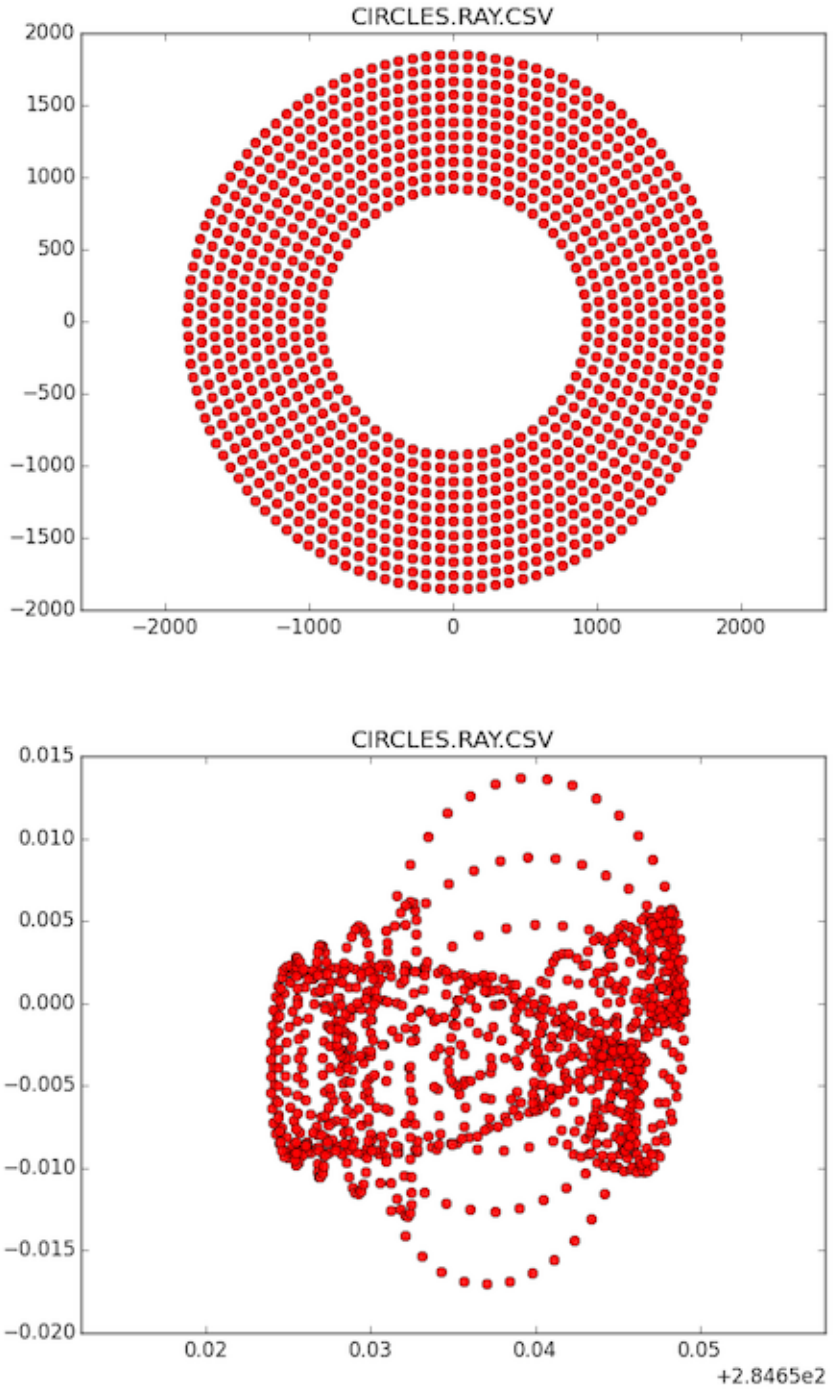

Figure 3: Two examples of output plots: Circular ray pattern arrives at primary mirror (top) and is imaged onto the focal surface (bottom). Units are mm. The central pupil blockage, due to the prime focus assembly, has eliminated all rays that lie near the center of the pupil. Of twenty concentric rings and 1261 ray starts, only 990 rays survive their journey.

6 Testing

Hyperbolic Lens: One kind of optical software test is a so-called **null test** in which the result is known in advance. Typically the optic is very simple: a conic section, or succession of conic sections, with defined separations and curvatures. A hyperbolic lens can produce an unaberrated on-axis spot (Descartes, R., ca 1637). A hyperbolic refractor null test is listed here. To work, $Curvature = Power/(n-1)$ as for any plano-convex lens, and also $Asphericity = -n^2$.

```
   3    ,  surfaces                    HYPER.OPT
 Index  ,   Z     ,Curve, Asph  , Diameter, Lens?   ,
        ,  1.00   ,     ,       ,   2.2   ,  lens   ,
 1.5    ,  2.00   ,  -2 , -2.25 ,   2.2   ,  lens   ,
        ,  3.00   ,     ,       ,   2.2   ,  film   ,

*** Output: howfar, X,Y,Z:   3      0.0000000000000000      0.00000000      3.00000000
*** Output: howfar, X,Y,Z:   3     -0.0000000000000003      0.00000000      3.00000000
*** Output: howfar, X,Y,Z:   3     -0.0000000000000004      0.00000000      3.00000000
*** Output: howfar, X,Y,Z:   3      0.0000000000000007      0.00000000      3.00000000
*** Output: howfar, X,Y,Z:   3     -0.0000000000000004      0.00000000      3.00000000
```

It is seen that this refraction calculation is good to about fifteen decimal places, which is to be expected for the usual double precision math operations done by Python and Numpy. I have run about half a dozen null tests on RT113.py and RT118.py, all with good results.

Concave Paraboloid, Four Ways: A concave parabolic mirror can perfectly image incoming parallel light (Diocles, ca. 200B.C.) and therefore provides another null test that can be used to compare the numerical results of different surface representations and computational methods. A paraboloid can be specified in four alternative ways:

- as a conic section of revolution whose ray intercepts and slopes are analytic formulas. To obtain a focal length of 4000mm, the theoretical curvature $C = -1/(2 \cdot FL)$ which is -0.000125 reciprocal mm. The asphericity of a paraboloid is -1. With these optical parameters, a perfect focus should be achieved.
- as a polynomial of revolution (using A2) using an iterative rootfinder and analytic slope. Theoretically the required coefficient $A2 = -1/(4 \cdot FL)$ which is -0.0000625 reciprocal mm. With this optical parameter, an ideal focus should be achieved.
- as a sum of Zern0 and Zern3 using an iterative rootfinder and analytic slope. At an edge radius R=390mm, the theoretically required coefficients are $Zern0 = Zern3 = -R^2/(8 \cdot FL)$ which is -4.753125 mm for this test case. With these two parameters set, an ideal image should be achieved.
- as a 261x261 grid of sag numbers, precomputed to be a paraboloid, using a Hermite bicubic interpolator for both the intercept and the surface slope. I used a map whose edge height = 1mm, for which the theoretical $SagCoef = R^2/(4 \cdot FL)$ which is -9.50625 mm for this test case having FL=4000mm and Diam=780mm.

Here is the ray start table followed by the (X, Y, U, V) results for each of the four methods.

```
    8  , rays     PARAB.RAY.CSV for 4000mm focal length
 @wave ,  XOmm  , YOmm   ,  ZOmm ,
       ,  350   ,    0   ,   0   ,
       ,  100   ,    0   ,   0   ,
       ,   50   ,    0   ,   0   ,
       ,    0   ,    0   ,   0   ,
       ,  -50   ,    0   ,   0   ,
       , -150   ,    0   ,   0   ,
       , -350   ,    0   ,   0   ,
       ,    0   ,  350   ,   0   ,

-----CONIC SECTION PARABOLOID: C = -0.000125, ASPH = -1.00-------------
 X,Y,U,V   -0.00000000000006    0.00000000000000   -0.08733283948692    0.00000000000000
 X,Y,U,V   -0.00000000000001    0.00000000000000   -0.02499609436026    0.00000000000000
 X,Y,U,V    0.00000000000002    0.00000000000000   -0.01249951173782    0.00000000000000
 X,Y,U,V    0.00000000000000    0.00000000000000    0.00000000000000    0.00000000000000
 X,Y,U,V   -0.00000000000002    0.00000000000000    0.01249951173782    0.00000000000000
 X,Y,U,V   -0.00000000000003    0.00000000000000    0.03748682103948    0.00000000000000
 X,Y,U,V    0.00000000000006    0.00000000000000    0.08733283948692    0.00000000000000
 X,Y,U,V    0.00000000000000   -0.00000000000006    0.00000000000000   -0.08733283948692

-----POLYNOMIAL  PARABOLOID:  A2 = -0.0000625-----------
 X,Y,U,V   -0.00000000000006    0.00000000000000   -0.08733283948692    0.00000000000000
 X,Y,U,V   -0.00000000000001    0.00000000000000   -0.02499609436026    0.00000000000000
 X,Y,U,V    0.00000000000002    0.00000000000000   -0.01249951173782    0.00000000000000
 X,Y,U,V    0.00000000000000    0.00000000000000    0.00000000000000    0.00000000000000
 X,Y,U,V   -0.00000000000002    0.00000000000000    0.01249951173782    0.00000000000000
 X,Y,U,V   -0.00000000000003    0.00000000000000    0.03748682103948    0.00000000000000
 X,Y,U,V    0.00000000000006    0.00000000000000    0.08733283948692    0.00000000000000
 X,Y,U,V    0.00000000000000   -0.00000000000006    0.00000000000000   -0.08733283948692

-----ZERNIKE PARABOLOID: ZERN0 = ZERN3 = -4.753125mm-------------
 X,Y,U,V    0.00000000000000    0.00000000000000   -0.08733283948692    0.00000000000000
 X,Y,U,V    0.00000000000001    0.00000000000000   -0.02499609436026    0.00000000000000
 X,Y,U,V    0.00000000000004    0.00000000000000   -0.01249951173782    0.00000000000000
 X,Y,U,V    0.00000000000000    0.00000000000000    0.00000000000000    0.00000000000000
 X,Y,U,V   -0.00000000000004    0.00000000000000    0.01249951173782    0.00000000000000
 X,Y,U,V   -0.00000000000003    0.00000000000000    0.03748682103948    0.00000000000000
 X,Y,U,V    0.00000000000000    0.00000000000000    0.08733283948692    0.00000000000000
 X,Y,U,V    0.00000000000000    0.00000000000000    0.00000000000000   -0.08733283948692

----------SAGMAP PARABOLOID: SAG = -9.50625mm----------
 X,Y,U,V   -0.00000000000506    0.00000000000000   -0.08733283948692    0.00000000000000
 X,Y,U,V    0.00000000000107    0.00000000000000   -0.02499609436026    0.00000000000000
 X,Y,U,V   -0.00000000000046    0.00000000000000   -0.01249951173782    0.00000000000000
 X,Y,U,V    0.00000000000000    0.00000000000000    0.00000000000000    0.00000000000000
 X,Y,U,V    0.00000000000050    0.00000000000000    0.01249951173782    0.00000000000000
 X,Y,U,V   -0.00000000000031    0.00000000000000    0.03748682103948    0.00000000000000
 X,Y,U,V   -0.00000000000341    0.00000000000000    0.08733283948692    0.00000000000000
 X,Y,U,V    0.00000000000000   -0.00000000000506    0.00000000000000   -0.08733283948692
```

The input ray heights are a few hundred mm, and the focal errors are $\sim$ 1E-13mm, so we are seeing about 15 digits accuracy. Although the sag map was calculated to 16 digits, the sag map paraboloid focus is poorer, only about 14 digits. The added errors are due to its interpolated slopes being obtained by subtracting nearly-equal neighbors spaced only 3mm.

Tortous Test As a check on getting all the angular transformations right, it is good to run an optical system that exercises all its angles. The files DISK3.OPT.CSV and DISK3.RAY.CSV accomplish this.

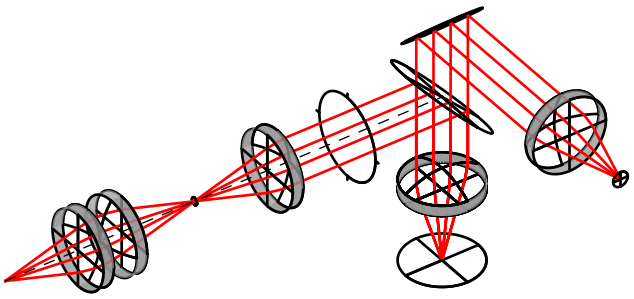

Figure 4: Artwork from BEAM FOUR shows the DISK3 configuration with legs in the X, Y, and Z directions. The *.CSV* listings below show the BEAM FOUR files, converted to *.CSV* format. The final listing below is the output from RT118.py. Top: the surfaces. Middle: the B4 output. Bottom: the RT118 output. Agreement is satisfactory.

```
  18  , surfaces     DISK3.OPT.CSV
index ,  X   ,  Y  ,  Z  ,  Curv  , mirr? ,Diam ,Tilt ,Pitch, Asph  ,
      ,   0  ,  0  , 1.0 ,  0.0   ,       ,  1  ,     ,     ,       ,
 1.7  ,   0  ,  0  , 1.3 , -1.2145,       ,  1  ,     ,     , -0.638,
      ,   0  ,  0  , 1.3 ,  1.2145,       ,  1  ,     ,     , -0.638,
 1.7  ,   0  ,  0  , 1.6 ,  0.0   ,       ,  1  ,     ,     ,       ,
      ,   0  ,  0  , 2.6 ,        , iris  , 0.1 ,     ,     ,       ,
      ,   0  ,  0  , 3.6 ,  0.0   ,       ,  1  ,     ,     ,       ,
 1.7  ,   0  ,  0  , 3.9 , -1.2145,       ,  1  ,     ,     , -0.638,
      ,   0  ,  0  , 4.7 ,        , iris  , 1.1 ,     ,     ,       ,
      ,   0  ,  0  , 6.0 ,        , mirror, 1.4 ,     ,  45 ,       ,
      , -1.0 ,  0  , 6.0 , -1.2145,       ,  1  ,     ,  90 , -0.638,
 1.7  , -1.3 ,  0  , 6.0 ,  0.0   ,       ,  1  ,     ,  90 ,       ,
      , -2.3 ,  0  , 6.0 ,        ,  mir  ,  1  ,     ,  90 ,       ,
      , -1.3 ,  0  , 6.0 ,  0.0   ,       ,  1  ,     ,  90 ,       ,
 1.7  , -1.0 ,  0  , 6.0 , -1.2145,       ,  1  ,     ,  90 , -0.638,
      ,   1  ,  0  , 6.0 ,        , mirror,  1  ,  90 ,  45 ,       ,
      ,   1  , 2.2 , 6.0 ,  1.2145,       ,  1  , -90 ,   0 , -0.638,
 1.7  ,   1  , 2.5 , 6.0 ,  0.0   ,       ,  1  , -90 ,   0 ,       ,
      ,   1  , 3.5 , 6.0 ,        , focus , 0.2 , -90 ,   0 ,       ,

 4  ,   rays                DISK3.RAY.CSV
   U0   ,@,    XFINAL     ,    YFINAL      ,    ZFINAL      ,
  0.3   , ,  1.00000000000,  3.500000000000,  6.000079881986,
  0.1   , ,  1.00000000000,  3.500000000000,  5.999996895969,
 -0.1   , ,  1.00000000000,  3.500000000000,  6.000003104031,
 -0.3   , ,  1.00000000000,  3.500000000000,  5.999920118014,


*** Output: howfar,  XYZ: 18    1.000000000000    3.500000000000      6.000079881986
*** Output: howfar,  XYZ: 18    1.000000000000    3.500000000000      5.999996895969
*** Output: howfar,  XYZ: 18    1.000000000000    3.500000000000      6.000003104031
*** Output: howfar,  XYZ: 18    1.000000000000    3.500000000000      5.999920118014
```

Echo-22: Another class of test is to run more elaborate optical systems, whose results can be validated by direct comparison with other ray tracers, looking for areas of disagreement. Echo-22 is the development name for the prime focus corrector adopted by the DESI team. It comprises six lenses including two counter-rotating Atmospheric Dispersion Corrector lens-prisms made of N-BK7 borosilicate glass. Its other lenses are made of fused silica. Two of the glass surfaces and the focal surface are aspherical with polynomial-of-revolution coefficients.

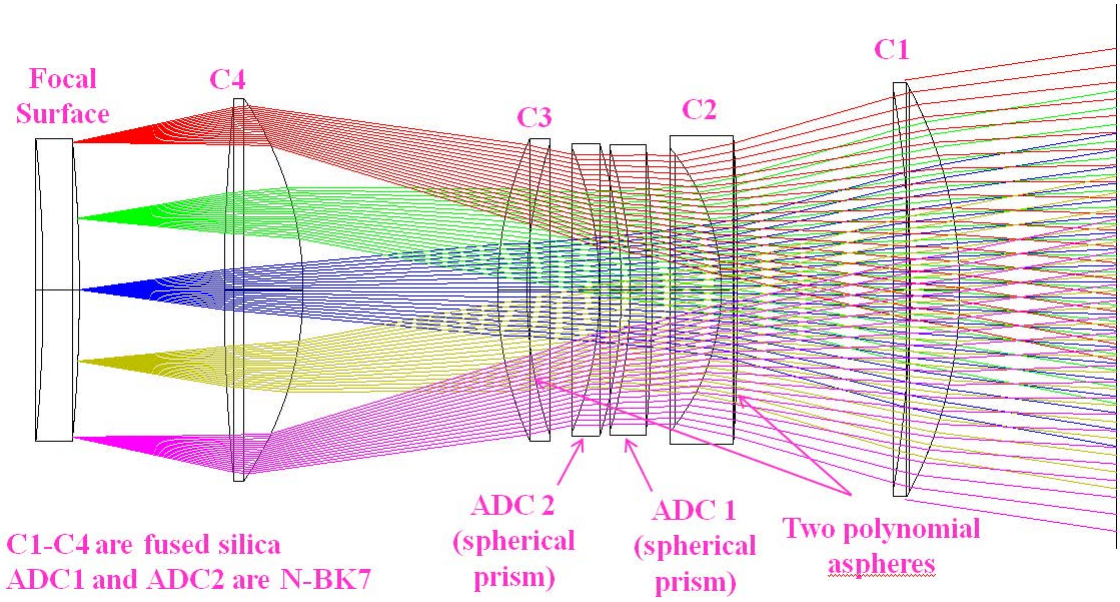

Figure 5: Side view layout of the E-22 corrector, from Sholl, DESI-0329v15. Its entrance aperture is 1140 mm diameter and its focal surface is 830mm in diameter.

Table 4 Optical prescription of E22

#	Type	Comment	ROC (mm)	Thickness	Glass	Diam. (mm)	Conic	Wedge	A4	A6	A8	A10
7	STANDARD	M1	-21336.00	-7200.0	MIRROR	3801.73	-1.098					
8	STANDARD			-1500.0								
10	STANDARD	555 CA C1_1	-1184.80	-136.4	SILICA	1140						
11	STANDARD	543 CA C1_2	-3295.57	0.0		1140						
12	STANDARD	Dummy C1_2		-475.3								
14	EVENASPH	410 CA C2_1	-12626.63	-45.0	SILICA	850			-1.7305E-10	1.5353E-16	3.6754E-22	-9.7376E-28
15	STANDARD	375 CA C2_2	-612.42	0.0		850						
16	STANDARD	Dummy C2_2		-190.0								
19	IRREGULA	375 CA ADC1_1	-4589.20	-60.0	N-BK7	800						
20	IRREGULA	377 CA ADC1_2	-1370.96	0.0		800		0.247				
22	COORDBRK			-25.0								
25	IRREGULA	375 CA ADC2_1	-1392.10	-60.0	N-BK7	804		0.250				
26	IRREGULA	375 CA ADC2_2	-1049.48	0.0		804						
28	COORDBRK			-200.0								
30	EVENASPH	390 CA C3_1	1340.70	-80.0	SILICA	834			1.2876E-10	-2.4724E-16	4.4388E-22	-7.3773E-27
31	STANDARD	402 CA C3_2	1026.99	0.0		834						
32	STANDARD	Dummy C3_2		-536.5								
34	STANDARD	502 CA C4_1	-934.08	-216.9	SILICA	1034						
35	STANDARD	502 CA C4_1	5187.21	-399.2		1034						
36	EVENASPH	Focal Plane	-4977.99	0.0		812			-2.9648E-10	3.4523E-15	-1.8042E-20	3.2571E-26

This E-22 corrector with the Mayall 4-m telescope satisfies a number of performance requirements. It provides a sky field of view 3.2 degrees in diameter, and an f/4 final focal ratio Its working wavelength range is 360-980nm, providing image spot sizes about 0.5 arcseconds in diameter – desirably smaller than the typical one arcsecond seeing at the Mayall, and the optical fiber size (107 microns or 1.45 arcseconds diameter).

Here, detailed comparison of RT118.py runs with BEAM FOUR release 1.99 shows excellent agreement; about 30 such comparisons have been examined. The **E22** model uses 17 surfaces, of which three are polynomial and two are tilted. The ADC was set to zero. Units are millimeters.

The upper output listing shown below is from BEAM FOUR rev 199 whose ray table includes both ray start information and ray trace results. The lower listing is from RT118, showing just its XYZ results. (The output value "howfar" is 17 when each ray has run to the end of its prescribed Nsurfs list in its .OPT table.) The XYZ results agree to 10^{-12} mm which compared to ray heights $\sim 1000mm$ is about fifteen digits of accuracy which I regard as entirely acceptable.

```
    6   rays    E22nominal.RAY
   X0       Y0    Z0    U0     W0  @wave notes    Xfinal          Yfinal               Zfinal
-------:-------:-----:---.--:-----:-----:------:----.------------:---.------------:-------.------------:-
 -1000 : -1000 : -200: -0.02: -1.0:0.620: OK17 : 284.698118281231:  0.003575073326: -11133.181007245894:
 -1000 :  1000 : -200: -0.02: -1.0:0.620: OK17 : 284.696003501763: -0.007655214750: -11133.180872354333:
     0 : -1000 : -200: -0.02: -1.0:0.620: OK17 : 284.690787236541: -0.005141562260: -11133.180539621955:
     0 :  1000 : -200: -0.02: -1.0:0.620: OK17 : 284.689419541034: -0.000580927597: -11133.180452379130:
  1000 : -1000 : -200: -0.02: -1.0:0.620: OK17 : 284.677207774968: -0.008275979435: -11133.179673464743:
  1000 :  1000 : -200: -0.02: -1.0:0.620: OK17 : 284.676784431515:  0.001739692056: -11133.179646455230:

*** Output: howfar,  XYZ: 17  284.698118281231    0.003575073326 -11133.181007245894
*** Output: howfar,  XYZ: 17  284.696003501763   -0.007655214750 -11133.180872354333
*** Output: howfar,  XYZ: 17  284.690787236541   -0.005141562260 -11133.180539621953
*** Output: howfar,  XYZ: 17  284.689419541034   -0.000580927597 -11133.180452379131
*** Output: howfar,  XYZ: 17  284.677207774968   -0.008275979435 -11133.179673464743
*** Output: howfar,  XYZ: 17  284.676784431515    0.001739692056 -11133.179646455230
```

7 Conic Sections, from Beam Four Users Guide

Conic sections are mathematical curves (parabolas, hyperbolas, circles etc.) that satisfy quadratic algebraic expressions. Geometrically they are equivalent to the intersection of a cone with a plane, hence the name. When a conic section is rotated about an axis, it sweeps out a surface in three dimensions (paraboloid, hyperboloid, sphere or ellipsoid). Surfaces of this type are very useful in optics. The parameters that represent conic sections depend on the coordinate system chosen. The table below can be used to convert a conic section description in one coordinate system to another. BEAM FOUR, in common with the optics industry, uses the vertex-origin Cartesian system.

	Center-origin Cartesian Coords	**Vertex-origin Cartesian Coords**	**Focus-origin Polar Coords**
Equation:	$Ax^2 + Bz^2 = 1$	$z = \frac{Cx^2}{1+\sqrt{1-SC^2x^2}}$	$r = \frac{a(1-e^2)}{1+e\cdot\cos\theta}$
Semimajor z axis:	$1/\sqrt{B}$	$1/CS$	a
Semiminor x axis:	$1/\sqrt{A}$	$1/(C\sqrt{S})$	$a\sqrt{1-e^2}$
Asphericity Asph:	(B/A) - 1	S - 1	$-e^2$
Shape S:	B/A	$Asph$+1	$1-e^2$
Eccentricity e:	$\sqrt{1-B/A}$	$\sqrt{1-S}$	e
Curvature C = 1/radius	$\pm A/\sqrt{B}$	C	$1/a(1-e^2)$
Locations of foci:	$z = \pm\sqrt{1/B - 1/A}$	$z = 1/C(1\pm\sqrt{1-S})$	$r = 0,\ r = 2ae$
Distance between foci:	$2\sqrt{A-B}/\sqrt{AB}$	$2\sqrt{1-S}/CS$	$2ae$
Oblate ellipsoid:	$B>A>0$	$S>1$, Asph>0	fails
Sphere:	$B=A>0$	$S=1$, Asph=0	$e=0$
Prolate ellipsoid:	$A>B>0$	$0<S<1$, -1<Asph<0	$0<e<1$
Paraboloid:	fails	$S=0$, Asph= -1	$e=1$
Hyperboloid:	$A<0<B$	$S<0$, Asph< -1	$e>1$

Beware of the term "conic constant" and the symbols K, k and κ. These descriptors are not well standardized. Some authors (e.g. Fischer & Tadic-Galeb) use them to mean asphericity (i.e.departure from a sphere). Others (e.g. Spencer & Murty; Wetherell & Rimmer) use κ to describe departure from a paraboloid. W.J.Smith (p.445) uses "p" for departure from a paraboloid. In BEAM FOUR, we use the descriptor "Asphericity" (abbreviated "Asph") to indicate departure from a sphere. We use "Shape" (abbreviated "S") to indicate departure from a paraboloid, so that S = Asph+1. Use either Shape or Asphericity, but not both, in your optics tables.

8 Rotation Sequences and our ADC

The orientation of an object in three dimensions can be described in a variety of ways. One such description is as follows. Start with the object coordinate system coinciding with the lab frame, so that x'=X, y'=Y, and z'=Z. Then first tilt the object about the x axis defining a tilt angle T. Then, pitch it about its new y axis, defining a pitch angle P. Finally roll it about its new z axis, defining a roll angle R. For axisymmetric surfaces, roll has no effect.

An alternative orientation specification is to use all lab frame angles X, Y, Z. In terms of these, the successive $Tilt, Pitch, Roll$ angles become

$$Pitch = arcsin(CX \cdot SY \cdot CZ + SX \cdot SZ) = arcsin(SX \cdot SZ) \tag{3}$$

$$Tilt = arctan\frac{SX \cdot CZ - CX \cdot SY \cdot SZ}{CX \cdot CY} = arctan\frac{SX \cdot CZ}{CX} \tag{4}$$

$$Roll = arctan\frac{CX \cdot SZ - SX \cdot SY \cdot CZ}{CY \cdot CZ} = arctan\frac{CX \cdot SZ}{CY \cdot CZ} \tag{5}$$

where the CX, CY, CZ, SX, SY, SZ are the lab frame X and Z cosines and sines. The simplified expressions are for $SY = 0$, appropriate for our ADC prisms that are tilted but not pitched. If the DESI ADC prisms are axisymmetric, roll does nothing and is not needed. However, with realistic surface maps, ADC roll will have to be taken into account. For a working example of ADC angles at work in the Echo-22 corrector, look at surfaces 10 and 11 in the optics file E22-ADC45.OPT.CSV which is one of the file accompanying this writeup.

The short table below shows a spreadsheet listing the ADC surface tilts and pitches that implement ADC roll settings of 0 deg, 30 deg, etc. The counterrotation of the two prisms has the sense that for a given ADC Z-axis angle, A1 turns by +Z and A2 turns by -Z. With this sense, and the +X direction of the telescope being upward (towards the zenith) then the effect of the Earth's atmosphere is to elevate the apparent direction of incoming blue light, shifting the blue image towards the -X focal plane direction, i.e. downward. When the ADC gets cranked in, it pushes this blue light back upward in the +X focal plane coordinate, correcting for the atmospheric dispersion.

ADC: Z deg	A1back tilt: +Z	A1back pitch: +Z	A2front tilt: -Z	A2front pitch: -Z
0	0.246515	0	0.250123	0
30	0.21349	0.12326	0.21661	-0.12506
45	0.17431	0.17431	0.17686	-0.17686
60	0.12326	0.21349	0.12506	-0.21661
90	0.00000	0.24652	0.00000	-0.25012

Figure 5 below illustrates the ADC action at a single location in the field. In "E22.OPT" I make use of a retro illuminator, where rays originate near the PM, go backwards to a retro reflector at the dome, then are returned to the pupil with (U,V,W) having signs reversed from their origin. The outgoing backwards light angles (U,V) = (+0.028, 0) become incoming descending rays that appear to originate at positive sky angles (above the optical Z axis) and are imaged near the bottom of the focal surface. The files being used here are E22-ADC45.OPT.CSV, E22trichromatic.RAY.CSV, and E22.MED.CSV. This ray table provides

about 331 ray starts per waveband, or 993 rays total, of which roughly 750 rays total survive their journey through the Echo-22 obstacle course, chiefly the prime focus assembly blockage and the vignetting at the peripheries of the corrector lenses. My U=+0.028 is an ascending ray emitted (backwards, towards the sky) from the pupil, which after retro reflection becomes a descending ray, and is therefore imaged at the bottom or -X side of the focal surface.

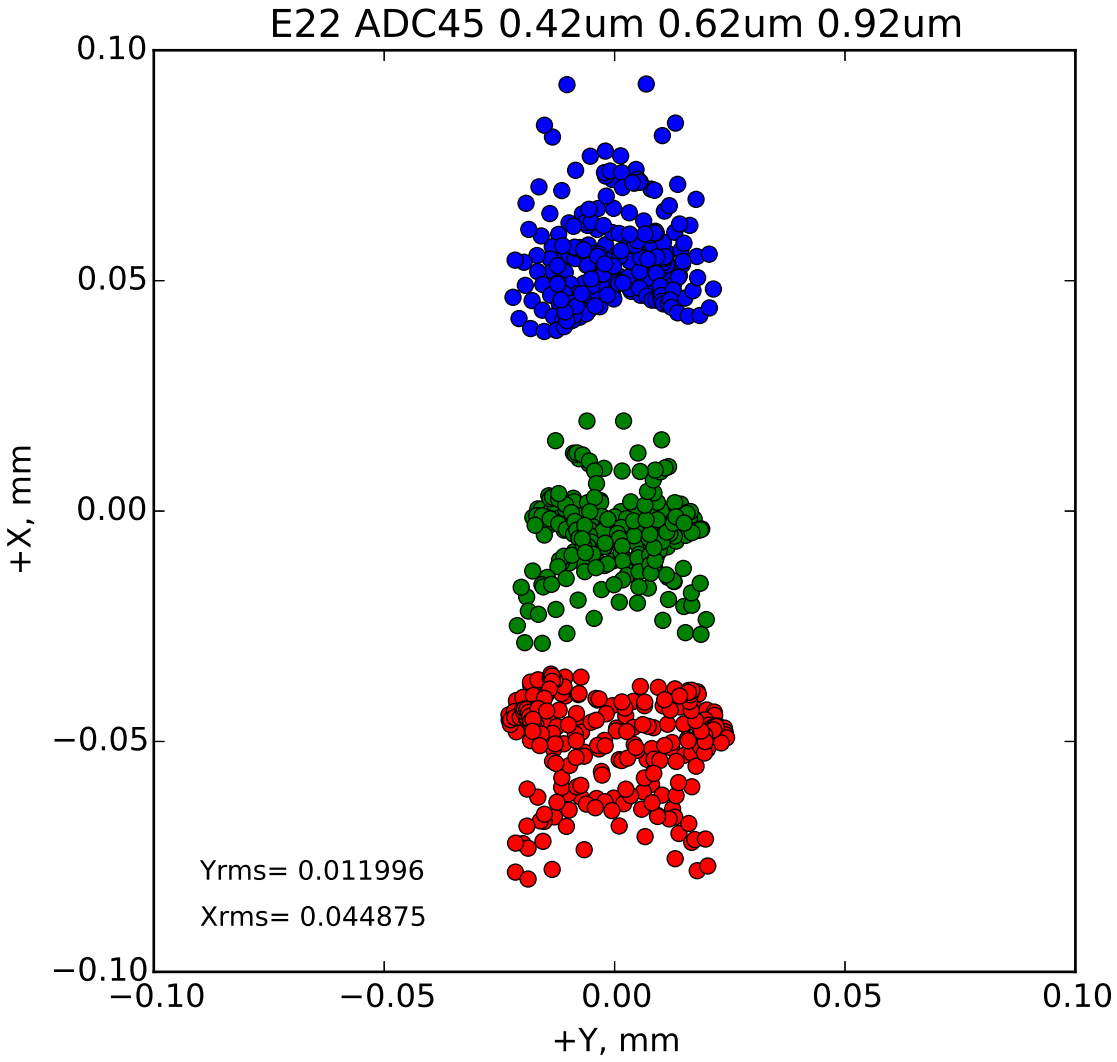

Figure 6: Illustration of ADC action with trichromatic illumination. No atmospheric refraction, but ADC prisms set to 45 degrees. The focal plane +X coordinate is upward and is expressed in mm. This spot appears at the bottom, or -X end, of the focal field. Box size is 200 μm.

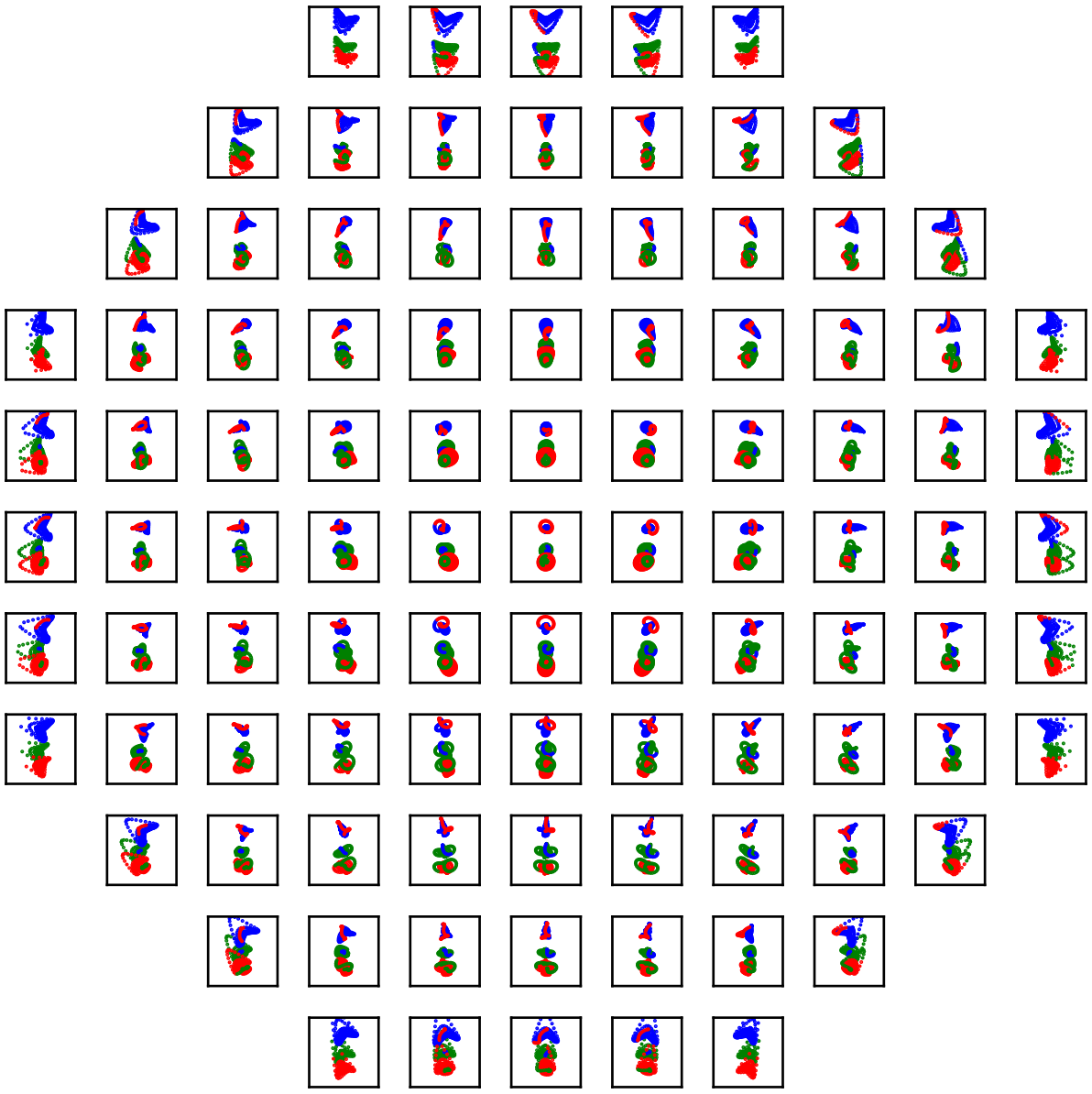

Figure 7: Multiple spot diagram spanning the DESI field of view. Step size in U and V is 5 milliradians. Wavelengths are 0.42, 0.62, and 0.92 um. No atmospheric refraction, but ADC prisms set to 45 degrees. The focal plane +X coordinate is upward. The spot shown in Figure 4 is near the bottom center of this figure. Each box is again 200 μm.

8.1 Creating Rotation Matrices

Mathematically, the combined effects of this sequence can be written as

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} M11 & M12 & M13 \\ M21 & M22 & M23 \\ M31 & M32 & M33 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

where [x,y,z] is any vector expressed in local coordinates and [X,Y,Z] is the same vector expressed in lab coordinates. The rotation matrix M is unitary: the sum of squares of any column = 1 and every column is orthogonal to the others; the sum of squares of any row = 1, and every row is orthogonal to the others. Its determinant = 1. M has six constraints on its nine members, and three degrees of freedom, which we fix by declaring the angles for tilt, pitch, and roll. Its inverse is its transpose.

Because the successive surfaces have various orientations, each surface has its own rotation matrix. Each is calculated after the optics table has been parsed, but before any rays get run. The routine $setEuler(j)$ does this setup for each surface $j = 1...Nsurfs$:

```
def setEuler(j):
    ct = np.cos(np.radians(Oarray[j, OTILT]))
    st = np.sin(np.radians(Oarray[j, OTILT]))
    cp = np.cos(np.radians(Oarray[j, OPITCH]))
    sp = np.sin(np.radians(Oarray[j, OPITCH]))
    cr = np.cos(np.radians(Oarray[j, OROLL]))
    sr = np.sin(np.radians(Oarray[j, OROLL]))
    Oarray[j,OE11] = cr*cp;               # X <- x; M11
    Oarray[j,OE12] = -sr*cp;              # X <- y; M12
    Oarray[j,OE13] = sp;                  # X <- z; M13
    Oarray[j,OE21] = cr*sp*st + sr*ct;    # Y <- x; M21
    Oarray[j,OE22] = cr*ct - sr*sp*st;    # Y <- y; M22
    Oarray[j,OE23] = -cp*st;              # Y <- z; M23
    Oarray[j,OE31] = -cr*sp*ct + sr*st;   # Z <- x; M31
    Oarray[j,OE32] = sr*sp*ct + cr*st;    # Z <- y; M32
    Oarray[j,OE33] = cp*ct;               # Z <- z; M33
```

8.2 Using Rotation Matrices

Python implementations performing rotation and translation is listed here for the lab-to-vertex task and the vertex-to-labframe task. Each returns the status flag OK because nothing can go wrong.

```
def labtovx(iray, jsurf):
    Xprev = Rarray[iray, jsurf-1, RX]
    Yprev = Rarray[iray, jsurf-1, RY]
    Zprev = Rarray[iray, jsurf-1, RZ]
    Uprev = Rarray[iray, jsurf-1, RU]
    Vprev = Rarray[iray, jsurf-1, RV]
    Wprev = Rarray[iray, jsurf-1, RW]

    dX = Xprev - Oarray[jsurf, OX]
    dY = Yprev - Oarray[jsurf, OY]
    dZ = Zprev - Oarray[jsurf, OZ]

    Rarray[iray,jsurf,Rx] = dX*Oarray[jsurf,OE11] + dY*Oarray[jsurf,OE21] + dZ*Oarray[jsurf,OE31]
    Rarray[iray,jsurf,Ry] = dX*Oarray[jsurf,OE12] + dY*Oarray[jsurf,OE22] + dZ*Oarray[jsurf,OE32]
    Rarray[iray,jsurf,Rz] = dX*Oarray[jsurf,OE13] + dY*Oarray[jsurf,OE23] + dZ*Oarray[jsurf,OE33]

    Rarray[iray,jsurf,Ru] = Uprev*Oarray[jsurf,OE11] + Vprev*Oarray[jsurf,OE21] + Wprev*Oarray[jsurf,OE31]
    Rarray[iray,jsurf,Rv] = Uprev*Oarray[jsurf,OE12] + Vprev*Oarray[jsurf,OE22] + Wprev*Oarray[jsurf,OE32]
    Rarray[iray,jsurf,Rw] = Uprev*Oarray[jsurf,OE13] + Vprev*Oarray[jsurf,OE23] + Wprev*Oarray[jsurf,OE33]
    return OK


def vxtolab(iray, jsurf):
    x = Rarray[iray, jsurf, Rx]
    y = Rarray[iray, jsurf, Ry]
    z = Rarray[iray, jsurf, Rz]
    u = Rarray[iray, jsurf, Ru]
    v = Rarray[iray, jsurf, Rv]
    w = Rarray[iray, jsurf, Rw]

    Rarray[iray, jsurf, RU] = u*Oarray[jsurf,OE11] + v*Oarray[jsurf,OE12] + w*Oarray[jsurf,OE13]
    Rarray[iray, jsurf, RV] = u*Oarray[jsurf,OE21] + v*Oarray[jsurf,OE22] + w*Oarray[jsurf,OE23]
    Rarray[iray, jsurf, RW] = u*Oarray[jsurf,OE31] + v*Oarray[jsurf,OE32] + w*Oarray[jsurf,OE33]

    Rarray[iray, jsurf, RX] = x*Oarray[jsurf,OE11] + y*Oarray[jsurf,OE12] + z*Oarray[jsurf,OE13]
    Rarray[iray, jsurf, RY] = x*Oarray[jsurf,OE21] + y*Oarray[jsurf,OE22] + z*Oarray[jsurf,OE23]
    Rarray[iray, jsurf, RZ] = x*Oarray[jsurf,OE31] + y*Oarray[jsurf,OE32] + z*Oarray[jsurf,OE33]

    Rarray[iray, jsurf, RX] = Rarray[iray, jsurf, RX] + Oarray[jsurf, OX]
    Rarray[iray, jsurf, RY] = Rarray[iray, jsurf, RY] + Oarray[jsurf, OY]
    Rarray[iray, jsurf, RZ] = Rarray[iray, jsurf, RZ] + Oarray[jsurf, OZ]
    return OK
```

9 Zernike Polynomials, from Beam Four Users Guide

$z0 = 1;$	Piston or Bias
$z1 = \rho \cos[\theta];$	Tilt x
$z2 = \rho \sin[\theta];$	Tilt y
$z3 = -1 + 2\rho^2;$	Power
$z4 = \rho^2 \cos[2\theta];$	Astig x
$z5 = \rho^2 \sin[2\theta];$	Astig y
$z6 = \rho (-2 + 3\rho^2) \cos[\theta];$	Coma x
$z7 = \rho (-2 + 3\rho^2) \sin[\theta];$	Coma y
$z8 = 1 - 6\rho^2 + 6\rho^4;$	Primary Spherical
$z9 = \rho^3 \cos[3\theta];$	Trefoil x
$z10 = \rho^3 \sin[3\theta];$	Trefoil y
$z11 = \rho^2 (-3 + 4\rho^2) \cos[2\theta];$	Secondary Astigmatism x
$z12 = \rho^2 (-3 + 4\rho^2) \sin[2\theta];$	Secondary Astigmatism y
$z13 = \rho (3 - 12\rho^2 + 10\rho^4) \cos[\theta];$	Secondary Coma x
$z14 = \rho (3 - 12\rho^2 + 10\rho^4) \sin[\theta];$	Secondary Coma y
$z15 = -1 + 12\rho^2 - 30\rho^4 + 20\rho^6;$	Secondary Spherical
$z16 = \rho^4 \cos[4\theta];$	Tetrafoil x
$z17 = \rho^4 \sin[4\theta];$	Tetrafoil y
$z18 = \rho^3 (-4 + 5\rho^2) \cos[3\theta];$	Secondary Trefoil x
$z19 = \rho^3 (-4 + 5\rho^2) \sin[3\theta];$	Secondary Trefoil y
$z20 = \rho^2 (6 - 20\rho^2 + 15\rho^4) \cos[2\theta];$	Tertiary Astigmatism x
$z21 = \rho^2 (6 - 20\rho^2 + 15\rho^4) \sin[2\theta];$	Tertiary Astigmatism y
$z22 = \rho (-4 + 30\rho^2 - 60\rho^4 + 35\rho^6) \cos[\theta];$	Tertiary Coma x
$z23 = \rho (-4 + 30\rho^2 - 60\rho^4 + 35\rho^6) \sin[\theta];$	Tertiary Coma y
$z24 = 1 - 20\rho^2 + 90\rho^4 - 140\rho^6 + 70\rho^8;$	Tertiary Spherical
$z25 = \rho^5 \cos[5\theta];$	Pentafoil x
$z26 = \rho^5 \sin[5\theta];$	Pentafoil y
$z27 = \rho^4 (-5 + 6\rho^2) \cos[4\theta];$	Secondary Tetrafoil x
$z28 = \rho^4 (-5 + 6\rho^2) \sin[4\theta];$	Secondary Tetrafoil y
$z29 = \rho^3 (10 - 30\rho^2 + 21\rho^4) \cos[3\theta];$	Tertiary Trefoil x
$z30 = \rho^3 (10 - 30\rho^2 + 21\rho^4) \sin[3\theta];$	Tertiary Trefoil y
$z31 = \rho^2 (-10 + 60\rho^2 - 105\rho^4 + 56\rho^6) \cos[2\theta];$	Quatenary Astigmatism x
$z32 = \rho^2 (-10 + 60\rho^2 - 105\rho^4 + 56\rho^6) \sin[2\theta];$	Quatenary Astigmatism y
$z33 = \rho (5 - 60\rho^2 + 210\rho^4 - 280\rho^6 + 126\rho^8) \cos[\theta];$	Quatenary Coma x
$z34 = \rho (5 - 60\rho^2 + 210\rho^4 - 280\rho^6 + 126\rho^8) \sin[\theta];$	Quatenary Coma y
$z35 = -1 + 30\rho^2 - 210\rho^4 + 560\rho^6 - 630\rho^8 + 252\rho^{10};$	Quaternary Spherical

Figure 8: Zernike polynomials. Numbering from J.C.Wyant, and from R.Wilson 2000, but is not well standardized. An alternative scheme is that of Born and Wolf "Principles of Optics," 7th edition pp. 523-526 (1999).

10 Surface Normals and Sag Map Interpolation

Two of the redirection tasks listed in section 4 require the local surface normal vector. This is trivially (0., 0., 1.0) for the plano case, and is easily derived as an analytic expression for the conic case, for the polynomial-of-revolution case, and for the Zernike case. For these cases the accompanying Python code implements the analytic derivatives to obtain the local surface normal direction.

Sag tables list surface deviations at a discrete set of points, and therefore require interpolation of height and slope for arbitrary (x,y) locations. A simple linear slope interpolation between neighboring tabulation points would fail: it would give a faceted, not smooth, surface. Here I employ the Catmull-Rom bicubic spline, based on Hermite polynomials, which offers continuity in slope and value across grid boundaries.

10.1 Theory of Hermite cubics

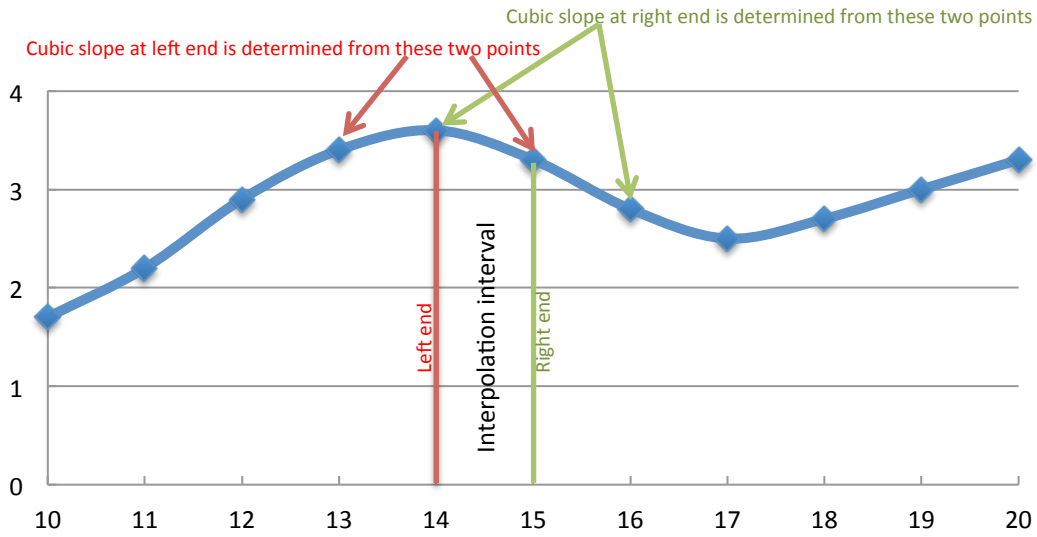

Figure 9: Given a regular 1D grid of data points, we need to interpolate the function's value and first derivative in any grid interval. Hermite's cubic is that unique cubic polynomial that satisfies four constraints: it matches the grid value at the left and right ends of its unit interval, and also matches the slopes there as determined by the finite differences shown. This guarantees that its neighboring interpolation intervals will be continuous in value and slope.

The Hermite cubic gives a smooth continuous behavior across grid points without kinks. This is an important feature for describing a smooth optical surface:

$$\begin{aligned} f(x) = &\left(-0.5x^3 + x^2 - 0.5x\right) p_{n-1} \\ &+ \left(1.5x^3 - 2.5x^2 + 1\right) p_n \\ &+ \left(-1.5x^3 + 2x^2 + 0.5x\right) p_{n+1} \\ &+ \left(0.5x^3 - 0.5x^2\right) p_{n+2} \end{aligned}$$

10.2 2D Implementation

The extension to two dimensions is straightforward: a 4x4 group is analyzed both in x, then in y. My Python implementation is listed below. Here, x and y are locations within the unit cell: 0<x<1, 0<y<1. The interpolation grid **pinterp[ ]** is the 4x4 square block of adjoining data points (extracted from the sag map) that surround any unit interpolation cell. The constant matrix **HERMITE** gives the polynomial coefficients for four successive data points.

```
HERMITE = np.array([[-0.5, +1.5, -1.5, +0.5],    # x**3 coefficient
                    [+1.0, -2.5, +2.0, -0.5],    # x**2 coefficient
                    [-0.5,  0.0, +0.5,  0.0],    # x coefficient
                    [ 0.0, +1.0,  0.0,  0.0]])   # 1 coefficient

def cubicFunc1D(pinterp, x):
    fpow = np.array([x*x*x, x*x, x, 1.])
    return fpow.dot(HERMITE.dot(pinterp))

def cubicFunc2D(pinterp, x, y):
    f = np.zeros([4])
    f[0] = cubicFunc1D(pinterp[0,:], x)  # row 0 at y=-1
    f[1] = cubicFunc1D(pinterp[1,:], x)  # row 1 at y=0
    f[2] = cubicFunc1D(pinterp[2,:], x)  # row 2 at y=1
    f[3] = cubicFunc1D(pinterp[3,:], x)  # row 3 at y=2
    return cubicFunc1D(f, y)

def cubicDeriv1D(pinterp, x):
    dpow = np.array([3*x*x, 2*x, 1., 0.])
    return dpow.dot(HERMITE.dot(pinterp))

def cubicDerivs2D(pinterp, x, y):
    dfdx = np.zeros(4)
    dfdx[0] = cubicDeriv1D(pinterp[0,:], x)  # row 0 at y=-1
    dfdx[1] = cubicDeriv1D(pinterp[1,:], x)  # row 1 at y=0
    dfdx[2] = cubicDeriv1D(pinterp[2,:], x)  # row 2 at y=1
    dfdx[3] = cubicDeriv1D(pinterp[3,:], x)  # row 3 at y=2
    xderiv = cubicFunc1D(dfdx, y)
    dfdy = np.zeros(4)
    dfdy[0] = cubicDeriv1D(pinterp[:,0], y)  # column 0 at x=-1
    dfdy[1] = cubicDeriv1D(pinterp[:,1], y)  # column 1 at x=0
    dfdy[2] = cubicDeriv1D(pinterp[:,2], y)  # column 2 at x=1
    dfdy[3] = cubicDeriv1D(pinterp[:,3], y)  # column 3 at x=2
    yderiv = cubicFunc1D(dfdy, x)
    return xderiv, yderiv
```

11 Refraction Tables and Sellmeier Interpolation

Manufacturers of optical glass materials provide lists of refractive indices for those materials. Wavelengths listed are the emission wavelengths of common line lamps. Usually what is wanted is refractive index at user specified wavelengths. For these data, the Sellmeier interpolation formula is traditionally used in the optical bands. The interpolation is controlled by six constants for each glass type. From the Schott 2016 guide:

Sellmeier dispersion formula

$$n^2(\lambda)-1 = B_1\lambda^2/(\lambda^2-C_1)+B_2\lambda^2/(\lambda^2-C_2)+B_3\lambda^2/(\lambda^2-C_3)$$

The most convenient way to apply this formula is using a spreadsheet. Indeed Schott's optical glass spreadsheet already contains columns that contain the B and C constants, for each of their $\sim$ 100 glasses. To interpolate into a set of user wavelengths, first eliminate all those columns except for the glass names (column A) and the six Sellmeier dispersion formula constants. Put your chosen wavelengths into cells H4, I4, ... The first working cell will be H5. Install the formula into it, and copy it as far down and rightward as needed:

```
= sqrt(1 + $B5*H$4^2/(H$4^2-$E5) + $C5*H$4^2/(H$4^2-$F5) + $D5*H$4^2/(H$4^2-$G5))
```

In the companion files Sellmeier.xlsx and Sellmeier.csv I have included fused silica, with dispersion constants taken from the Corning website. Another companion file is E22.MED.CSV listed here. The second field is the *nd* field from the manufacturers websites, and the adjacent field is the interpolated refractive index. As a check of the interpolation formulas, these should agree to about five digits accuracy.

```
   12  ,  glasses   E22.MED.CSV
Glass    ,nd @ 0.587562 ,0.587562 ,0.420    ,0.520    ,0.620    ,0.720    ,0.820    ,0.920    ,
FS       ,1.45844       ,1.458438 ,1.468068 ,1.461255 ,1.457374 ,1.454827 ,1.452956 ,1.451450 ,
F2       ,1.62004       ,1.620040 ,1.646092 ,1.627180 ,1.617461 ,1.611618 ,1.607710 ,1.604872 ,
F2HT     ,1.62004       ,1.620040 ,1.646092 ,1.627180 ,1.617461 ,1.611618 ,1.607710 ,1.604872 ,
F5       ,1.60342       ,1.603420 ,1.627567 ,1.610064 ,1.601015 ,1.595549 ,1.591876 ,1.589193 ,
K10      ,1.50137       ,1.501371 ,1.514365 ,1.505084 ,1.499993 ,1.496768 ,1.494492 ,1.492738 ,
K7       ,1.51112       ,1.511121 ,1.523409 ,1.514655 ,1.509805 ,1.506716 ,1.504526 ,1.502834 ,
N-BASF64 ,1.70400       ,1.704000 ,1.731117 ,1.711481 ,1.701284 ,1.695090 ,1.690900 ,1.687815 ,
N-BK10   ,1.49782       ,1.497821 ,1.508452 ,1.500920 ,1.496653 ,1.493865 ,1.491829 ,1.490199 ,
N-BK7    ,1.51680       ,1.516800 ,1.528385 ,1.520160 ,1.515539 ,1.512549 ,1.510389 ,1.508680 ,
N-F2     ,1.62005       ,1.620053 ,1.646226 ,1.627174 ,1.617482 ,1.611640 ,1.607695 ,1.604787 ,
N-FK5    ,1.48749       ,1.487490 ,1.497372 ,1.490376 ,1.486401 ,1.483799 ,1.481893 ,1.480364 ,
N-FK51A  ,1.48656       ,1.486561 ,1.494860 ,1.488967 ,1.485662 ,1.483547 ,1.482049 ,1.480893 ,
```

12 Preview Showing C3Front Sag Map Effects

As of this writing we do not yet have detailed maps of our corrector lens surfaces, except for the preliminary C3Front map provided to us by AOS. Nonetheless we can examine the effects of this single sag map on our image quality. An earlier study, DESI-DOC 2642, used a three-Gaussian model of the C3Front dimple, and concluded that its effects did not significantly worsen our net PSF since its ray deviations were smaller than the optical system's combined aberrations.

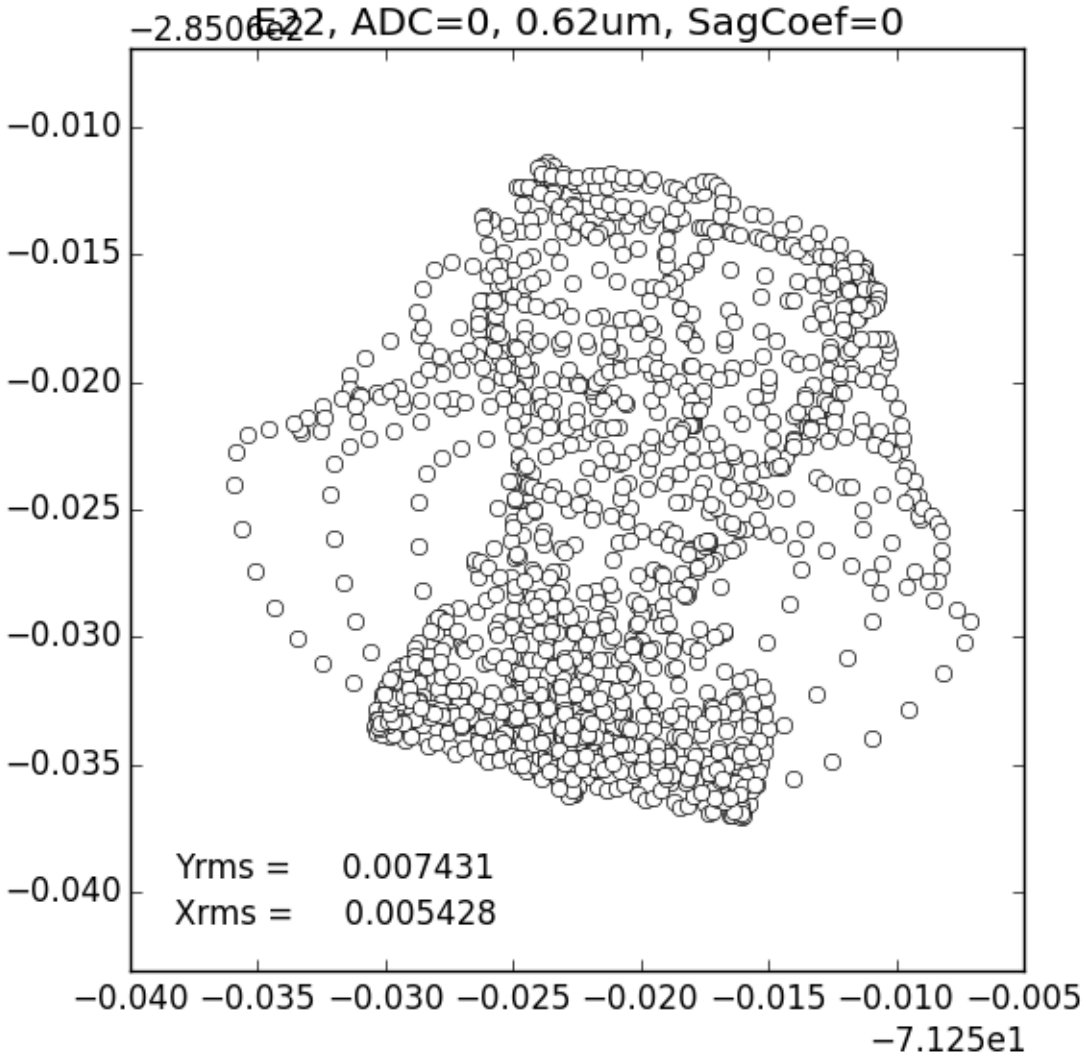

Figure 10: Spot diagram for a target whose annulus of light at C3Front inciudes the dimple. Here, the sag map multiplier is set to zero, so we see the idealized image that perfect optics could deliver.

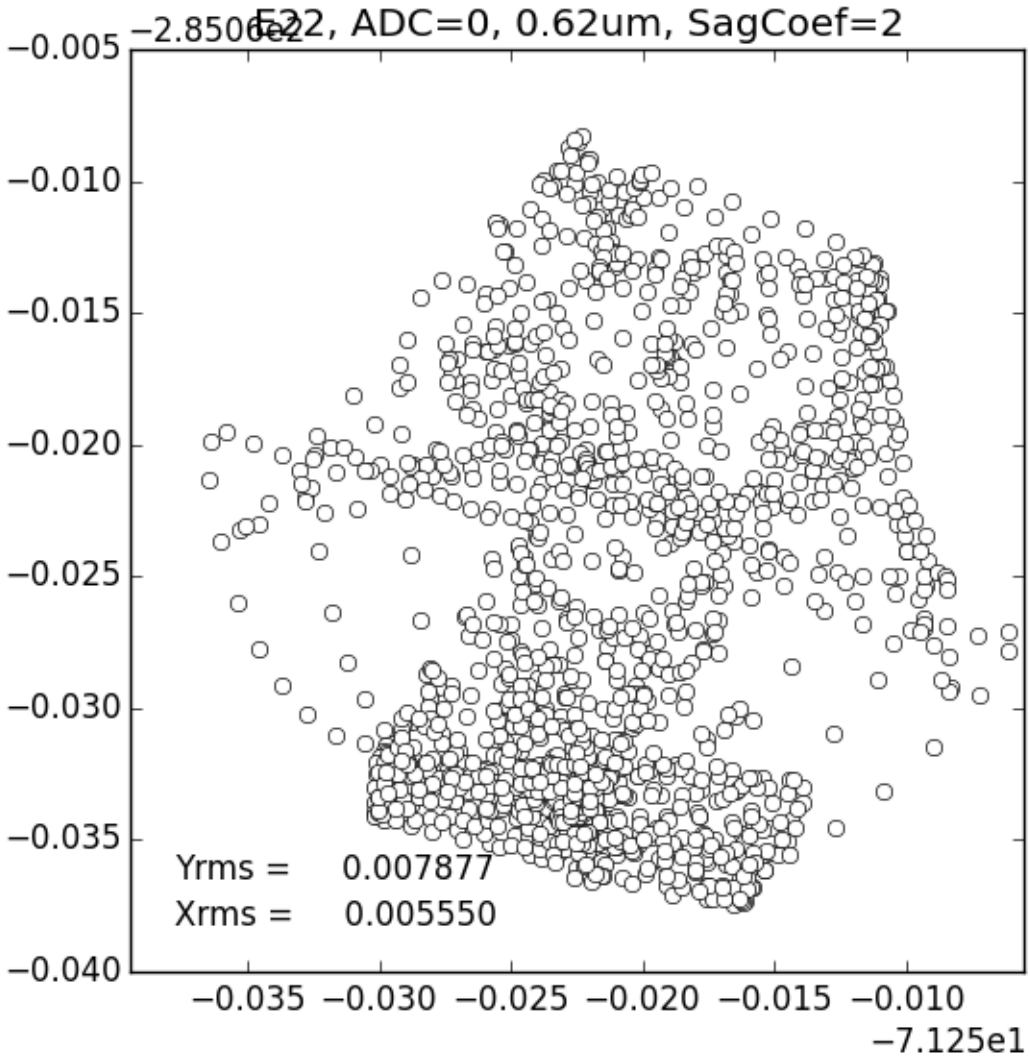

Figure 11: As above but with the sag multiplier = 2.0; the aberrations are barely changed.

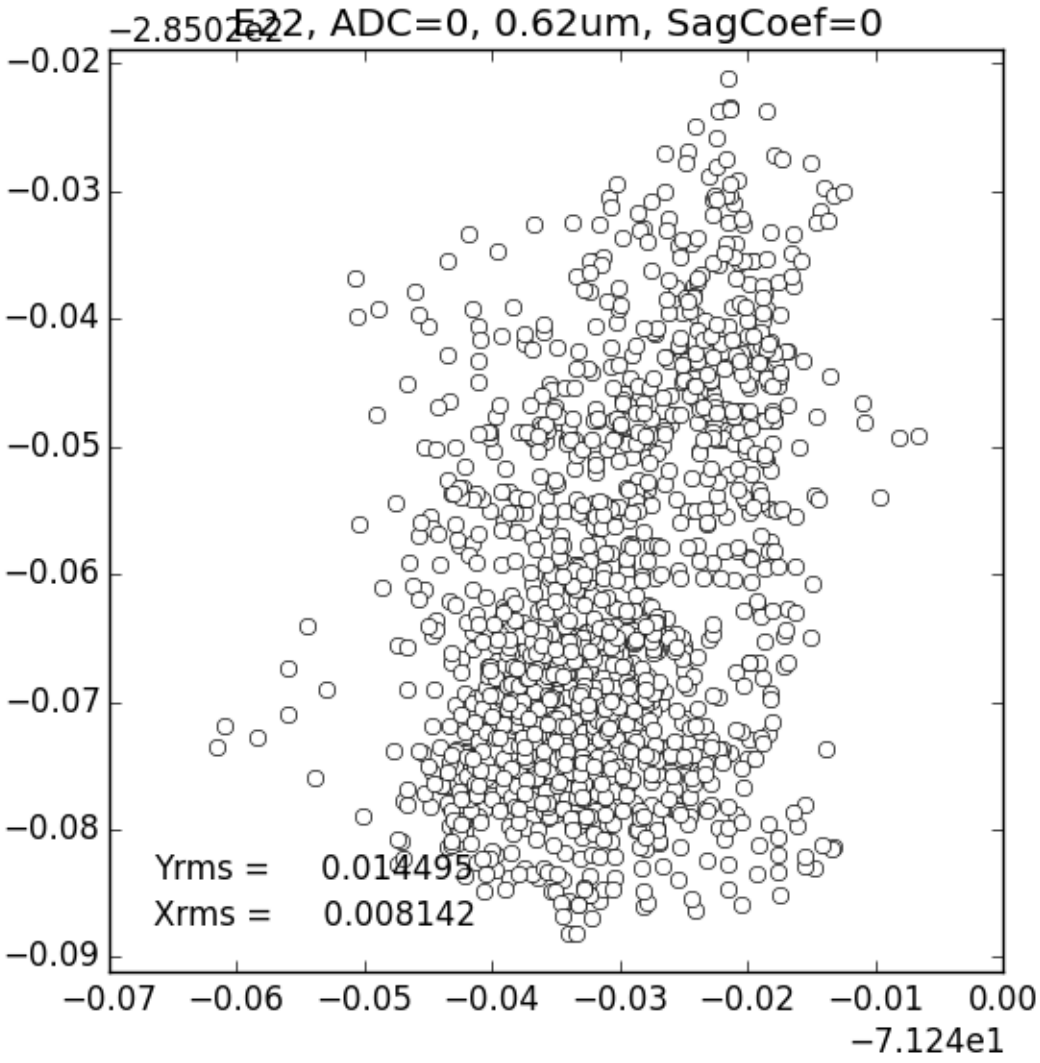

Figure 12: As above but with the sag multiplier = 10.0; now the surface errors have a strong influence on image quality.