QSOFT Documentation

Release 9.0

Richard Willingale

Dec 05, 2018

CONTENTS

QSOFT is a collection of data analysis and modelling applications for use in X-ray astronomy and related disciplines.

The applications are run as commands/functions within Python, R or IDL.

The core code is written in Fortran and C, compiled to produce shareable object libraries and imported as modules in Python or loaded by R or IDL.

The commands/functions are defined in Python modules, R or IDL scripts.

The QSOFT collection should be built using the gcc and gfortran compilers. The Python module f2py and/or R must be available to create the shareable objects.

CHAPTER

ONE

BUILD AND INSTALLATION

The following software items are required for the build:

- Fortran compiler - GNU gfortran used in development
- C compiler - GNU gcc used in development
- Python module f2py - to build the modules for Python
- R - to build the shareable libraries for R and IDL
- Python module sphinx - to build the documentation

1. Download from GitHub

   $ git clone git://github.com/dickwillingale/qsoft.git

   This will create a directory qsoft/

2) Move into /your/files/top/qsoft/src

   $ cd /your/files/top/qsoft/src

   Edit the compile.config so that the compilers CC and F77 and R, F2PY and IDL point to the correct executables on your system. If you don't have R or F2PY leave them blank. If your target is IDL you will need R to compile the shareable library.

3. Move into /your/files/top/qsoft and execute build

   $ cd /your/files/top/qsoft

   $ ./build

   This will check you have gcc gfortran and R and/or Python with f2py. It will then create the src/compiler.config file for make and compile the shareable objects.

4. Put the following line into your /home/.profile

   . /your/files/top/qsoft/setup_q

   Qsoft will be available when you launch a login terminal.

5. That's it.

   When you start R (or Rscript) it will automatically load the QSOFT applications using the /home/.Rprofile file.

   The environment variable PYTHONPATH will point to the qsoft/python_modules directory so you can load the modules into Python.

CHAPTER

TWO

PYTHON, R AND IDL

All the Fortran functions can be called from Python, R or IDL. Because of peculiarities in the syntax and structure of the scripting languages there are minor differences in the way the functions are accessed.

The documentation of all the functions uses the Python implementation. Where there are significant differences in the R or IDL versions these are mentioned in the text.

2.1 Python

The directory $QSOFT/python_modules is included in the PYTHONPATH at set up so that the python modules can be imported in the usual way. Here is a snippet of a Python script using the astro.cosmo() function

```
#!/usr/bin/env python
# Test of Cosmological parameter calculations etc.
import numpy as np
import astro
import matplotlib.pylab as plt
#
zmax=5
# Einstein de Sitter
c1=astro.cosmo(70,1,0,zmax)
# Low density
c2=astro.cosmo(70,0.05,0,zmax)
# High Lambda
c3=astro.cosmo(70,0.2,0.8,zmax)
...
```

2.2 RScript

The file .Rprofile in the users home directory is executed by Rscript at start up to dynamically load the shareable libraries. The QSOFT R function names are prefixed according to the module library/subject as follows

- utilities: qr_
- qfits: qr_fits
- images: qri_
- astro: qra_
- xscat: qrx_
- xsrt: qrt_

Here is a snippet of a Rscript using the astro.cosmo() function

```
#!/usr/bin/env Rscript
# Test of Cosmological parameter distance calculations
    zmax<-5
# Einstein de Sitter
    c1<-qra_cosmo(70,1,0,zmax)
# Low density
    c2<-qra_cosmo(70,0.05,0,zmax)
# High Lambda
    c3<-qra_cosmo(70,0.2,0.8,zmax)
...
```

2.3 IDL

CHAPTER

THREE

QFITS - USING FITS FILES

The qfits interface has not yet been implemented in IDL.

Reading and writing of FITS files is available in Python and R.

3.1 Reading FITS Files

The entire contents of a FITS file can be read into a Python or R object using a single call as illustrated below.

```
#!/usr/bin/env python
import qfits
filename="test_fitsnew.fits"
fitsin=qfits.fitsfile(filename)
# Print summary
fitsin.display()
# Access the number of HDUs in FITS file
print("number of hdu",fitsin.nhdu)
# The primary array
print(fitsin.hdu[0].data_array)
# All the keywords on extension 1
print("HDU 1 keywords",fitsin.hdu[1].kw)
# A Particular keyword on extension 1
print("HDU 1 keyword NAXIS1",fitsin.hdu[1].kw["NAXIS1"])
# The complete table on extension 2
print("HDU 2 table",fitsin.hdu[2].table)
# A particular column from table on extension 2
print("HDU 2 table column s",fitsin.hdu[2].table["s"])
```

```
#!/usr/bin/env Rscript
filename<-"test_fitsnew.fits"
fitsin<- qr_fitsread(filename)
# Print a summary
qr_fitsprint(fitsin)
# Get the number of HDU (primary + extensions)
cat("number of HDU",fitsin$NHDU,"\n")
# Print the primary data arra
print(fitsin$primary$DATA_ARRAY)
# Access a particular keyword in extension 2
cat("HDU extension 2 keyword TESTD",fitsin$extension[[2]]$TESTD,"\n")
# All the keywords on extension 2
print(fitsin$extension[[2]])
# The complete table on extension 2
print(fitsin$extension[[2]]$table)
```

(continues on next page)

(continued from previous page)

```
# A particular column from table on extension 2
print(fitsin$extension[[2]]$table[,3])
```

Because the indexing of lists and arrays starts at zero in Python and 1 in R the internal structure of the object returned is different.

In Python the primary HDU is hdu[0] and extensions are hdu[1], hdu[2] etc..

In R the primary is called primary and the extensions are a list accessed as extension[[1]], extension[[2]] etc..

In Python the keywords are stored in a dictionary and particular keywords are accessed using a name index.

In R the keywords are stored as named variables within the extension list.

The summary listings produced by fitsfile.display() in Python and qr_fitsprint() in R can be used to reveal the way in which the contents of the FITS file are stored in memory.

Details on how to access all the elements of FITS files can be found in the source for the Python class fitsfile and R function qr_fitsread() which are defined in $QSOFT/src/qfits/qfits.py and $QSOFT/src/qfits/qfits.R.

3.2 Writing FITS Files

FITS files can be created using a Python or R script as illustrated below. Python uses methods in class fitsfile and class fitshdu. R uses functions qr_fits*().

```
#!/usr/bin/env python
import numpy as np
import qfits
#
a=qfits.fitsfile("new")
# Primary array
hdu=qfits.fitshdu(0,0)
arr=np.arange(100)+101.1
arr.shape=[20,5]
hdu.data_array=arr
a.hdu.append(hdu)
# Extension Arrays
hdu=qfits.fitshdu(1,0)
iarr=np.arange(100)+1
iarr=iarr[::-1]
iarr.shape=[20,5]
hdu.data_array=iarr
# Add some comments
hdu.cr.append("This is a test comment")
hdu.cr.append("This is some history")
# keywords
hdu.kw["TESTD"]=55.5
hdu.kc["TESTD"]="a test double value"
hdu.kw["TESTJ"]=28
hdu.kc["TESTJ"]="a test integer value"
hdu.kw["TESTL"]=True
hdu.kc["TESTL"]="a true logical value"
hdu.kw["TESTF"]=False
hdu.kc["TESTF"]="a false logical value"
hdu.kw["TESTS"]="a string"
hdu.kc["TESTS"]="a test string value"
```

(continues on next page)

(continued from previous page)

```
a.hdu.append(hdu)
# Tables
hdu=qfits.fitshdu(2,2)
ix=np.arange(4)+1
x=np.array(ix*10+ix*.1,float)
ix=np.array(ix*10,int)
y=["aaaaaaaaaaa","bbbbbbbbbbb","ccccccccccc","ddddddddddd"]
z=np.array([True,True,False,True])
q=np.array([9,8,7,6],np.int8)
s=np.array([55+22j,66+23j,77+24j,88+25j])
xx=np.array(np.arange(12)+10,float)
xx.shape=[4,3]
b_bits=["1000000111","0111001101","0010000010","0001101100"]
rnames=["r1","r2","r3","r4"]
short=np.array([1.1,2.2,3.3])
hdu.table["ix"]=ix
hdu.units["ix"]="splogs"
hdu.table["x"]=x
hdu.table["xx"]=xx
hdu.table["y"]=y
hdu.table["z"]=z
hdu.table["q"]=q
hdu.table["s"]=s
hdu.table["b_bits"]=b_bits
hdu.table["short"]=short
hdu.table["rnames"]=rnames
a.hdu.append(hdu)
#
a.nhdu=3
# Print a summary and save
a.display()
a.save("test_fitsnew.fits")
```

```
#!/usr/bin/env Rscript
qr_fitsnew("fitswrite_test.fits")
# Primary array
arr<-seq(from=10.1,to=20,length.out=100)
dim(arr)<-c(5,20)
qr_fitsparrd(arr)
# Extension Arrays
iarr<-100:1
dim(iarr)<-c(5,20)
qr_fitsparrj(iarr)
# Comments and history cards
qr_fitspcom("This is a test comment")
qr_fitsphis("This is some history")
qr_fitsempty()
# keywords
qr_fitspkeyd("TESTD",55.5,"a test double value")
qr_fitspkeyj("TESTJ",28,"a test integer value")
qr_fitspkeyl("TESTL",T,"a true logical value")
qr_fitspkeyl("TESTF",F,"a false logical value")
qr_fitspkeys("TESTS","a string","a test string value")
# Tables
x<- c(10.10,20.20,30.30,40.40)
ix<- as.integer(c(10,20,30,40))
```

(continues on next page)

(continued from previous page)

```
y<- c("aaaaaaaaaaa","bbbbbbbbbbb","ccccccccccc","ddddddddddd")
z<- c(T,T,F,T)
q<-as.raw(c(9,8,7,6))
s<- complex(real=c(55,66,77,88),imaginary=c(22,23,24,25))
rnames<- c("r1","r2","r3","r4")
tt<- data.frame(x,ix,y,z,q,s,stringsAsFactors=F,row.names=rnames)
qr_fitspobj(tt,"dframe")
qr_fitsclose()
# Print a summary
fitsin<- qr_fitsread("fitswrite_test.fits")
qr_fitsprint(fitsin)
```

3.3 qfits.functions

qfits.**fitscolnam**(*ic*, *rr*, *nrows*)

Get FITS table column name - internal routine

Parameters

- **ic** – column index
- **rr** – 0 if variable column width
- **nrows** – number of rows

Returns

list of following

colnam column name
iname number of characters in colnam
vrep variable repeat count

qfits.**fitsgcol**(*ic*, *typ*, *nrows*, *rp*, *vr*)

Get column from table on FITS file - internal function

Parameters

- **ic** – column number
- **typ** – column data type
- **nrows** – number of column rows
- **rp** – repeat count
- **vr** – If rp zero then vr[nrows] is variable count for each row

The following variables are used internally:

nc: the number of calls required to get complete column
ne: length returned per call
qt: 1 complete column returned in 1 call
qt: 2 1 call per row fixed width returned as a list
qt: 3 variable width column 1 call per row returned as a list

qfits.**fitsgcolv**(*typ*, *ic*, *ii*, *ne*)
Get columm values from FITS file - internal function

Parameters

- **typ** – column data type returned
- **ic** – column number
- **ii** – first row number
- **ne** – number of elements

typ values:

0 none
1 integer
2 double
3 logical
4 bit as string 1 or 0
5 real complex
6 double complex
7 byte integer 8
8 bit row as a character string

Returns values from column

qfits.**fitsgetkey**(*ikey*)
Get FITS keyword - internal function

Parameters **ikey** – keyword index

Returns

key keyword string
ki number of characters in keyword string
sval string value
si number of characters in string value
jval integer value
dval double value
lval logical value
ktype type of value returned
1 integer in jval
2 real in dval
3 logical in jval
4 string in sval

qfits.**fitshduinfo**(*ihdu*)
Get FITS header dimension info - internal function

Parameters **ihdu** – HDU index

Returns

list of following
hdutype type of HDU
naxis number of dimensions
naxes size of dimensions (nrows,ncols)
nkeys number of keywords

qfits.**fitsparr**(*arr*)
Put array onto fits file - internal function

Parameters **arr** – arr of data values

qfits.**fitsptab**(*hdu*)
Create binary table HDU on FITS object

Parameters **hdu** – FITS HDU object

qfits.**fitstypes**(*hdutype*, *ncols*)
Get FITS header data types - internal function

Parameters

- **hdutype** – type of HDU
- **ncols** – number of columns (if hdutype=0)

Returns

ctype column type
rp column repeat count

qfits.**fitsupdate**(*filename*)
Open FITS file for read/write

Parameters **filename** – FITS file name

Returns number of HDU in file

qfits.**init**()
Initialize common blocks on import

class qfits.**fitsfile**(*filename*)

__init__(*filename*)
Read object from FITS file or create new FITS object

Parameters **filename** – FITS filename ("new" to create new object)

display()
List FITS object

save(*filename*)
Save FITS object to file

Parameters **filename** – FITS file name

class qfits.**fitshdu**(*ihdu*, *hdutype*)

__init__(*ihdu*, *hdutype*)
FITS HDU object

Parameters

- **ihdu** – HDU index
- **hdutype** – HDU type

display()
List FITS HDU object

CHAPTER

FOUR

IMAGES - IMAGE PROCESSING

Positions in images. The coordinates of an image field are set up using the function setfield(). The *current position* can be set using the function setpos(). This position is then used by functions like beam(). The position is given in pixel coordinates (a real scale running 0-NCOLS in X and 0-NROWS in Y) and local X,Y (usually taken as mm). If, in addition to setfield(), sky coordinates are set using the function setsky() the *current position* is also specified in local azimuth and elevation (degrees), Celestial RA,DEC (degrees J2000), Ecliptic EA,EL (degrees) and Galactic LII, BII (degrees). The sky coordinates can be set using different projections, Plate Carre, Aitoff (Hammer) or Lambert (equatorial aspect of Azimuthal equal-area).

- setfield() Set local coordinates for image field
- setsky() Set up sky coordinates for image field
- setpos() Set current position in image field
- getpos() Get current position in image field
- toxy() Convert position to local xy coordinates
- plt_show_locator() Get local coordinate positions using the cursor

Analysis of a beam containing a source or PSF. The beam is centred at the *current positon* (see above).

- beam() Analysis of source above background within a circular beam
- sqbeam() Analysis of source above background within a square beam
- lecbeam() Analysis of source above background in a lobster eye cross-beam

Creation of images from event lists or 2-d functions. In Python these function return a 2-d array. In R they return an image object which contains the array and ancillary information.

- binxy() x-y event binning to form an image array
- lebin() Create an image from an event list binning using lobster eye psf
- lecimage() Create an image array of the lobster eye cross-beam
- lepsf() Create an image of the lobster eye PSF

Drawing over images. The function hamgrid() and lamgrid() will only work if sky coordinates have been set up using setsky().

- rectangles() Draw rectangles
- hamgrid() Plot a Hammer projection grid on figure
- lamgrid() Plot a Lambert projection grid on figure

1-d profiles

- gaussian() Gaussian profile

- king_profile() King function (modified Lorentzian) profile
- lorentzian() Lorentzian profile

Model function fitting using a statistic. Can be used for fitting of PSF profiles to image data or more generally for fitting data with a model function.

- srchmin() Search for minimum statistic and return best fit parameters and confidence limits of the parameters
- peakchisq() Chi-squared for image peak fitting
- quaderr() Quadratic estimator for confidence limit

4.1 images.functions

images.**beam**(*arr*, *rbeam*, *blev*, *bvar*)

Analysis of source above background within a circular beam

Parameters

- **arr** – image array
- **rbeam** – radius of beam in pixels
- **blev** – average background level per pixel (to be subtracted)
- **bvar** – variance on blev (-ve for counting statistics)

Returns

list with the following
nsam: number of pixels in beam
bflux: background in beam (e.g. counts)
bsigma: standard deviation of background
flux: source flux above background in beam (e.g. counts)
fsigma: standard deviation of source flux
peak: source x,y peak position
cen: source x,y centroid position
tha: angle (degrees) of major axis wrt x (x to y +ve)
rmsa: source max rms width (major axis) (pixels)
rmsb: source min rms width (minor axis) (pixels)
fwhm: full width half maximum (pixels) about beam centre
hew: half energy width (pixels) about beam centre
w90: W90 (90% width) (pixels) about beam centre
fwhmp: full width half maximum (pixels) about peak
hewp: half energy width (pixels) about peak
w90p: W90 (90% width) (pixels) about peak
fwhmc: full width half maximum (pixels) about centroid
hewc: half energy width (pixels) about centroid
w90c: W90 (90% width) (pixels) about centroid
fit: parameters from peak fit using peakchisq()

Fit performed if bvar!=0.
Parameters are saved in the list fit (see function srchmin())
0: peak value (no error range calculated)
1: peak X pixel position with 90% error range
2: peak Y pixel position with 90% error range
3: Lorentzian width including 90% upper and lower bounds

The position of the beam is the current position within the image.
Use function setpos() to set the current position.

images.**binxy**(*x*, *y*, *iq*, *w*, *xleft*, *xright*, *ybot*, *ytop*, *nx*, *ny*)
x-y event binning to form an image

Parameters

- **x** – array of x positions
- **Y** – array of y positions
- **iq** – array of quality values (0 for OK)
- **w** – array of weights
- **xleft** – minimum x value (left edge) for image array
- **xright** – maximum x value (rigth edge) for image array
- **ybot** – minimum y value (bottom edge) for image array
- **ytop** – maximum y value (top edge) for image array
- **nx** – size of 1st dimension of image array (number of rows)
- **ny** – size of 2nd dimension of image array (number of columns)

Returns

image array
R version returns an image object - image array object$data_array

images.**gaussian**(*x*, *p*)
Gaussian profile

Parameters

- **x** – array of x values
- **p** – array of fitting parameters

par 0: normalisation (value at peak)
par 1: x-centre (pixel position)
par 2: Gaussian full width half maximum (pixels)

Returns array of function values evaluated at x

images.**getpos**()
Get current position in image field

Returns list with following

pix: pixel position
xyl: local position
aes: local azimuth,elevation degrees
equ: Celestial RA,DEC degrees J2000
ecl: Ecliptic EA,EL degrees
gal: Galactic LII,BII degrees

images.**hamgrid**(*pic*)
Plot a Hammer projection grid on figure

Parameters **pic** – figure object

images.**init**()
Initialise common blocks on import

images.**king_profile**(*x*, *p*)
King function (modified Lorentzian) profile

Parameters

- **x** – array of x values
- **p** – array of fitting parameters

par 0: normalisation (value at peak)
par 1: x-centre (pixel position)
par 2: full width half maximum (pixels)
par 3: power index (1 for Lorentzian)

Returns array of function values evaluated at x

images.**lamgrid**(*pic*)
Plot a Lambert projection grid on figure

Parameters **pic** – figure object

images.**lebin**(*xe*, *ye*, *s*, *h*, *g*, *eta*, *nx*, *ny*)
Create an image from an event list binning using lobster eye psf

The image is effectively a cross-correlation of the event list with the lobster eye cross-beam.

Parameters

- **xe** – x event positions, pixels
- **ye** – y event positions, pixels
- **s** – size of cross-beam square area in pixels
- **h** – height of cross-arm triangle in pixels (=2d/L)
- **g** – width of Lorentzian central spot in pixels

- **eta** – cross-arm to peak ratio at centre
- **nx** – first dimension of array
- **ny** – second dimension of array

Returns image array

Event positions assumed to run:
x: 0 left edge to nx right edge
y: 0 bottom edge to ny top edge
Therefore centre of left bottom pixel is 0.5,0.5

images.**lecbeam**(*arr*, *s*, *h*, *blev*, *bvar*, *nt*)

Analysis of source above background in a lobster eye cross-beam

Parameters

- **arr** – image array
- **s** – size of cross-beam square area in pixels
- **h** – height of cross-arm quadrant in pixels (=2d/L)
- **blev** – average background level per pixel (to be subtracted)
- **bvar** – variance on blev (-ve for counting statistics)
- **nt** – dimension of output quadrant flux distribution in pixels

Returns list of the following

qua: quadrant surface brightness distribution array nt by nt
quan: quadrant pixel occupancy array nt by nt
nsam: number of pixels in beam
bflux: background in beam (e.g. counts)
bsigma: standard deviation of background
flux: source flux above background in beam (e.g. counts)
fsigma: standard deviation of source flux
peak: source x,y peak position
cen: source x,y centroid position
hew: half energy width (pixels)
w90: W90 (90% width) (pixels)
ahe: half energy area (sq pixels)
aw9: W90 (90% width) area (sq pixels)
fpeak: flux in peak pixel

The position of the beam is the current position within the image.
Use function setpos() to set the current position.

images.**lecimage**(*s*, *h*, *b*, *xcen*, *ycen*, *nx*, *ny*)

Create an image array of the lobster eye cross-beam

Parameters

- **s** – size of cross-beam square area in pixels
- **h** – height of cross-arm triangle in pixels (=2d/L)
- **b** – jwidth of cross-arm triangle in pixels
- **xcen** – centre pixel (see coords below)
- **ycen** – centre pixel (see coords below)
- **nx** – first dimension of array
- **ny** – second dimension of array

Returns image array

Coordinate system for xcen,ycen is:

x runs from 0.0 on left to nx on right
Y runs from 0.0 on bottom to ny on top
Centre of bottom left pixel is therefore 0.5,0.5
Centre of top right pixel is nx-0.5,ny-0.5

images.**lepsf**(*s*, *h*, *g*, *eta*, *xcen*, *ycen*, *nx*, *ny*)

Create an image of the lobster eye PSF

Parameters

- **s** – size of cross-beam square area in pixels
- **h** – height of cross-arm triangle in pixels (=2d/L)
- **g** – width of Lorentzian central spot in pixels
- **eta** – cross-arm to peak ratio at centre
- **xcen** – centre of PSF (see below for coords. system)
- **ycen** – centre of PSF (see below for coords. system)
- **nx** – first dimension of array
- **ny** – second dimension of array

Returns image array

Coordinate system for xcen,ycen is:

x runs from 0.0 on left to nx on right
Y runs from 0.0 on bottom to ny on top
Centre of bottom left pixel is therefore 0.5,0.5
Centre of top right pixel is nx-0.5,ny-0.5

images.**lorentzian**(*x*, *p*)

Lorentzian profile

Parameters

- **x** – array of x values
- **p** – array of fitting parameters

par 0: normalisation (value at peak)
par 1: x-centre (pixel position)
par 2: Lorentian width (pixels)

Returns array of function values evaluated at x

images.**peakchisq**(*fpars*)
Chi-squared for image peak fitting

Parameters **fpars** – array of fitting parameters

par 0: normalisation (value at peak)
par 1: x-centre (pixel position)
par 2: y-centre (pixel position)
par 3: Lorentian width (pixels) 24th Sept. 2017 RW

Returns Chi-squared value

Used by function beam() which loads Fortran common block BEAMFIT with image data.

images.**plt_show_locator**(*fg*, *npos*)
Get local coordinate positions using the cursor

Parameters

- **fg** – figure id returned by plt.figure()
- **npos** – number of positions to be retured (clicked)

Returns xx,yy, 2 arrays containing npos local coordinates

This routine provides a basic level of interaction with a displayed figure. It is used in place of a simple plt.show() call so that npos local coordinate positions can be selected interactively using the cursor from a displayed image and returned in arrays to the user.

images.**quaderr**(*x0*, *y0*, *x1*, *y1*, *y*)
Quadratic estimator for confidence limit

Parameters

- **x0** – parameter at minimum
- **y0** – statistic at minimum
- **x1** – parameter near minimum
- **y1** – statistic near minimum
- **y** – required statistic value

Returns estimate of parameter corresponding to y

images.**rectangles**(*x*, *y*, *w*, *h*, *t*)
Draw rectangles

Parameters

- **x** – centres in local x
- **y** – centres in local y

- **w** – widths
- **h** – heights
- **t** – rotation angles radians

images.**reset**()
Reset common blocks to initial condition

images.**setfield**(*nx*, *xleft*, *xright*, *ny*, *ybot*, *ytop*)
Set local coordinates for image field

Parameters

- **nx** – number of columns
- **xleft** – local x left edge
- **xright** – local x right edge
- **ny** – number of rows
- **ybot** – local y bottom edge
- **ytop** – local y top edge

images.**setpos**(*ipos*, *p*)
Set current position in image field

Parameters

- **ipos** – coordinate index
- **p** – position coordinate pair

ipos values:

1: pixel 0-NCOLS, 0-NROWS
2: local X,Y
3: local azimuth,elevation degrees
4: Celestial RA,DEC degrees J2000
5: Ecliptic EA,EL degrees
6: Galactic LII,BII degrees

Returns current position using getpos()

images.**setsky**(*xtodeg*, *ytodeg*, *ipr*, *mjd*, *ra*, *dec*, *roll*)
Set up sky coordinates for image field

Parameters

- **xtodeg** – scale from local x to degrees (usually -ve)
- **ytodeg** – scale from local y to degrees
- **ipr** – projection between local XY and local spherical
- **mjd** – Modified Julian date
- **ra** – Right Ascension (degrees J2000 at local origin)
- **dec** – Declination (degrees J2000 at local origin)
- **roll** – Roll angle (degrees from North to +ve elev. +ve clockwise)

ipr values:

0: Plate Carre
1: Aitoff (Hammer)
2: Lambert (equatorial aspect of Azimuthal equal-area projection)

images.**sqbeam**(*arr*, *hbeam*, *blev*, *bvar*)

Analysis of source above background within a square beam

Parameters

- **arr** – image array
- **hbeam** – half width of square beam in pixels
- **blev** – average background level per pixel (to be subtracted)
- **bvar** – variance on blev (-ve for counting statistics)

Returns

list with the following
nx,ny: dimension of beam pixels (truncated if falls off edge)
xpi,xpr: x output arrays, pixel position and flux
ypi,ypr: y output arrays, pixel position and flux
bflux: background in beam (e.g. counts)
bsigma: standard deviation of background
flux: source flux above background in beam (e.g. counts)
fsigma: standard deviation of source flux
peak: source x,y peak position
cen: source x,y centroid position
rmsx: rms width in x (pixels) about centroid
rmsy: rms width in y (pixels) about centroid
pi5: 5% x,y position
pi25: 25% x,y position
med: median (50%) x,y position
pi75: 75% x,y position
pi95: 95% x,y position
hewx: HEW (half energy width) x (pixels)
hewy: HEW (half energy width) y (pixels)
w90x: W90 (90% width) x (pixels)
w90y: W90 (90% width) y (pixels)
fitx: parameters from x profile fit using king_profile()
fity: parameters from y profile fit using king_profile()

Fits performed if bvar!=0.
Parameters are saved in the lists fitx and fity
0: peak value (no error range calculated)
1: peak X pixel position (no error range calculated)
2: Lorentzian width including 90% upper and lower bounds

The position of the sqbeam is the current position within the image.
Use function setpos() to set the current position.

images.**srchmin**(*pars*, *pl*, *ph*, *stat*, *delstat*, *derr*)

Search for minimum statistic and return best fit parameters and confidence limits of the parameters

Parameters

- **pars** – initial parameter values
- **pl** – hard lower limit of parameter values
- **ph** – hard upper limit of parameter values
- **stat** – the statistic function to be minimised
- **delstat** – the change in statistic for confidence limits
- **derr** – initial error estimates for parameters, 0 fixed, >0 to estimate confidence range

Returns list from optim() plus confidence limits parlo and parhi

The statistic function call must return the value of a statistic with call of form stat(pars) (e.g. images.peakchisq(pars))

images.**toxy**(*ipos*, *p*)

Convert position to local xy coordinates

Parameters **ipos** – coordinate index

1: pixel 0-NCOLS, 0-NROWS
2: local X,Y
3: local azimuth,elevation degrees
4: Celestial RA,DEC degrees J2000
5: Ecliptic EA,EL degrees
6: Galactic LII,BII degrees

Returns position as local xy

CHAPTER

FIVE

ASTRO - ASTRONOMY APPLICATIONS

X-ray spectra. X-ray spectral data are usually presented as histograms of X-ray counts vs. energy (keV). The functions btoc() and ctob() provide conversion between energies at n bin centres and n+1 bin boundaries.

- brems() Bremsstrahlung spectrum
- habs() X-ray absorption by a Hydrogen column density
- setabnd() Set abundances for XSPEC routines used in absorption and optical depth calculations
- btoc() Convert n+1 bin boundaries to n bin centres
- ctob() Convert n bin centres to n+1 bin boundaries

X-ray optical depth. The local ISM and cosmic IGM gas is modelled as a hydrogen column including heavier elements at specified abundances and ionisation state.

- igmtau() X-ray optical depth of IGM gas
- iigmtau() X-ray optical depth of ionized IGM gas
- iigmtauvz() X-ray optical depth of ionized IGM gas vs. redshift
- ismtau() X-ray optical depth of cold ISM gas
- iismtau() X-ray optical depth of ionized ISM gas
- lyftau() X-ray optical depth of Lyman Forest
- lyftauvz() X-ray optical depth of Lyman Forest vs. redshift

Cosmology and redshift.

- cosmo() Calculation of cosmological quantities (luminosity distance etc.)
- kcorrb() K-correction using numerical integration of the Band function

5.1 astro.functions

astro.**brems**(*ekev*, *t*)

Bremsstrahlung spectrum

Parameters

- **ekev** – array of photon energies keV
- **t** – temperature keV

Returns Bremsstrahlung continuum photons/keV at energies ekev

astro.**btoc**(*x*)
Convert n+1 bin boundaries to n bin centres

Parameters **x** – array of bin boundaries length n+1

Returns array of bin centres length n

astro.**cosmo**(*h0*, *omegam*, *omegal*, *zmax*)
Calculation of cosmological quantities

Using equations from David Hogg astro-ph/9905116

Parameters

- **h0** – cosmological H0 km s-1 Mpc-1
- **omegam** – cosmological omegam
- **omegal** – cosmological omegal
- **zmax** – maximum redshift

Returns list with following sampled at dz=0.01

z: redshift values
ez: scaling function
dc: comoving line-of-sight distance
dm: transverse comoving distance
da: angular diameter distance
dl: luminosity distance
distm: distance modulus
dvc: comoving volume element
tlbak: look back time
vc: integrated comoving volume over whole sky
thsec: Hubble time in seconds
thgyr: Hubble time in Giga years
dhmpc: Hubble length in Mpc

astro.**ctob**(*x*)
Convert n bin centres to n+1 bin boundaries

Parameters **x** – array of bin centres length n

Returns array of bin boundaries length n+1

astro.**habs**(*cd*, *ekev*)
X-ray absorption by a Hydrogen column density

Parameters

- **cd** – hydrogen column 10**21 cm-2
- **ekev** – array of photon energies keV

Returns array of absorption factors

astro.**igmtau**(*n0*, *z*, *h0*, *omegam*, *omegal*, *ekev*)
X-ray optical depth of IGM gas

Parameters

- **n0** – number density cm-3 at z=0
- **z** – redshift of source
- **h0** – cosmological H0 km s-1 Mpc-1
- **omegam** – cosmological omegam
- **omegal** – cosmological omegal
- **ekev** – array of photon energies keV

Returns optical depth for each energy

astro.**iigmtau**(*n0*, *pl*, *tk*, *ist*, *z*, *h0*, *omegam*, *omegal*, *ekev*)
X-ray optical depth of ionized IGM gas

Parameters

- **n0** – number density cm-3 at z=0
- **pl** – powerlaw index of continuum spectrum
- **tk** – temperature Kelvin
- **ist** – ionization state L/nR^2
- **z** – redshift of source
- **h0** – cosmological H0 km s-1 Mpc-1
- **omegam** – cosmological omegam
- **omegal** – cosmological omegal
- **ekev** – array of photon energies keV

Returns optical depth for each energy

astro.**iigmtauvz**(*n0*, *dind*, *m0*, *mind*, *pl*, *tk*, *ist*, *h0*, *omegam*, *omegal*, *ekev*, *z*)
X-ray optical depth of ionized IGM gas vs. redshift

Parameters

- **n0** – number density cm-3 at z=0
- **dind** – density index wrt z+1
- **m0** – metalicity log[X/H] at z=0
- **mind** – metalicity log[X/H] index wrt z
- **pl** – powerlaw index of continuum spectrum
- **tk** – temperature Kelvin
- **ist** – ionization state = L/nR^2
- **h0** – cosmological H0 km s-1 Mpc-1
- **omegam** – cosmological omegam
- **omegal** – cosmological omegal
- **ekev** – energy keV

- **z** – array of redshift values

Returns optical depth at ekev for each redshift

astro.**iismtau**(*nh*, *z*, *tk*, *pl*, *ist*, *ekev*)
X-ray optical depth of ionized ISM gas

Parameters

- **nh** – column density 10**21 cm-3 at redshift z
- **z** – redshift of source
- **tk** – temperature Kelvin
- **pl** – powerlaw index of continuum spectrum
- **ist** – ionization state = L/nR^2
- **ekev** – array of photon energies keV

Returns: optical depth for each energy

astro.**init**()
Initialise common blocks on import

astro.**ismtau**(*nh*, *z*, *ekev*)
X-ray optical depth of cold ISM gas

Parameters

- **nh** – column density 10**21 cm-2 at redshift z
- **z** – redshift of source
- **ekev** – array of photon energies keV

Returns optical depth for each energy

astro.**kcorrb**(*e1src*, *e2src*, *e1obs*, *e2obs*, *z*, *gamma1*, *gamma2*, *ecobs*)
K-correction using numerical integration of Band function

Parameters

- **e1src** – lower source frame energy
- **e2src** – upper source frame energy
- **e1obs** – array of lower observed energies
- **e2obs** – array of upper observed energies
- **z** – array of redshifts
- **gamma1** – array of observed photon indices below Ec
- **gamma2** – array of observed photon indices above Ec
- **ecobs** – array of observed Ec energies

Returns list with the following for each object

kcorr: K-correction factor
obint: integral over observed band
boint: integral over Eiso band in observed frame

astro.**lyftau**(*n0*, *dind*, *m0*, *mind*, *z*, *h0*, *omegam*, *omegal*, *ekev*)
X-ray optical depth of Lyman Forest

Parameters

- **n0** – number density cm-3 at z=0
- **dind** – density index wrt z+1
- **m0** – metalicity log[X/H] at z=0
- **mind** – metalicity log[X/H] index wrt z
- **z** – redshift of source
- **h0** – cosmological H0 km s-1 Mpc-1
- **omegam** – cosmological omegam
- **omegal** – cosmological omegal
- **ekev** – array of photon energies keV

Returns optical depth for each energy

astro.**lyftauvz**(*n0*, *dind*, *m0*, *mind*, *h0*, *omegam*, *omegal*, *ekev*, *z*)
X-ray optical depth of Lyman Forest vs. z

Parameters

- **n0** – number density cm-3 at z=0
- **dind** – density index wrt z+1
- **m0** – metalicity log[X/H] at z=0
- **mind** – metalicity log[X/H] index wrt z
- **h0** – cosmological H0 km s-1 Mpc-1
- **omegam** – cosmological omegam
- **omegal** – cosmological omegal
- **ekev** – photon energy keV
- **z** – array of redshifts

Returns optical depth at ekev for each redshift

astro.**reset**()
Reset common blocks to initial condition

astro.**setabund**(*abun*, *amet*)
Set abundances for XSPEC routines

Parameters

- **abun** – XSPEC abundance table
- **amet** – metalicity

CHAPTER

SIX

XSCAT - X-RAY PHYSICS

X-ray optical constants.

- xopt() X-ray optical properties of a material
- xfresnel() Calculate X-ray reflectivity using Fresnel's equations

X-ray scattering by dust.

- dustrings() Model fitting to dust X-ray scattering halo rings
- duststat() Statistic of fit of model to dust rings
- dustthetascat() Scattering angle radians
- miev0() Wiscombe subroutine for Mie scattering calculations

6.1 xscat.functions

xscat.**dustrings**(*data*, *derr*, *dsou*, *ekv*, *srate*, *dts*, *td*, *zd*, *amin*, *amax*, *qa*, *sig1*)

Model fitting to dust X-ray scattering halo rings

Parameters

- **data** – data array of rings surface brightness cts/s/str
- **derr** – array of errors on data
- **dsou** – distance to source PC
- **ekv** – array of energies keV (equally spaced across observed band)
- **srate** – source spectrum cts/s/keV
- **dts** – source burst duration
- **td** – delay time of observations secs
- **zd** – fraction of source distance to rings
- **amin, amax** – grain size radius range microns
- **qa** – grain size distribution index N(a)=A.a^-qa
- **sig1** – differential cross-section of 1 grain cm2, 1 keV, 0.1 microns

Returns the following

angs: array of angles

ndust: N dust columns cm-2
edust: errors on N dust columns cm-2
model: model cts/s/str in rings
chisq: Chi-Squared between data and model
ndof: ndof

class xscat.**duststat**(*data*, *derr*, *dsou*, *ekv*, *srate*, *dts*, *td*, *zd*, *sig1*)
Statistic of fit of model to dust rings

xscat.**dustthetascat**(*td*, *ds*, *zd*)
Scattering angle radians

Parameters

- **td** – delay time seconds after source flare (assumed delta function)
- **ds** – distance to source parsecs (convert to m - 3.086e16/parsec)
- **zd** – fractional distance to dust

Returns angle in radians

xscat.**miev0**(*xx*, *crefin*, *pfct*, *mimc*, *anya*, *xmu*, *nmom*, *ipolzn*, *momd*, *prt*)
Wiscombe subroutine

Computes Mie scattering and extinction efficiencies; asymmetry factor; forward- and backscatter amplitude; scattering amplitudes vs. scattering angle for incident polarization parallel and perpendicular to the plane of scattering; coefficients in the Legendre polynomial expansions of either the unpolarized phase function or the polarized phase matrix; some quantities needed in polarized radiative transfer; and information about whether or not a resonance has been hit.

Input and output variables are described in file MIEV.doc. Many statements are accompanied by comments referring to references in MIEV.doc, notably the NCAR Mie report which is now available electronically and which is referred to using the shorthand (Rn), meaning Eq. (n) of the report.

Parameters

- **xx** – Mie size parameter (2 * pi * radius / wavelength)
- **crefin** – Complex refractive index (imag part can be + or -, but internally a negative imaginary index is assumed). If imag part is - , scattering amplitudes as in Van de Hulst are returned; if imag part is + , complex conjugates of those scattering amplitudes are returned (the latter is the convention in physics). ** NOTE ** In the 'PERFECT' case, scattering amplitudes in the Van de Hulst (Ref. 6 above) convention will automatically be returned unless Im(CREFIN) is positive; otherwise, CREFIN plays no role.
- **pfct** – TRUE, assume refractive index is infinite and use special case formulas for Mie coefficients 'a' and 'b' (see Kerker, M., The Scattering of Light and Other Electromagnetic Radiation, p. 90). This is sometimes called the 'totally reflecting', sometimes the 'perfectly conducting' case. (see CREFIN for additional information)
- **mimc** – (positive) value below which imaginary refractive index is regarded as zero (computation proceeds faster for zero imaginary index)
- **anya** – TRUE, any angles whatsoever may be input through XMU. FALSE, the angles are monotone increasing and mirror symmetric about 90 degrees (this option is advantageous because the scattering amplitudes S1,S2 for the angles between 90 and 180 degrees are evaluable from symmetry relations, and hence are obtained with little added computational cost.)

- **numang** – No. of angles at which scattering amplitudes S1,S2 are to be evaluated (set = 0 to skip calculation of S1,S2). Make sure NUMANG does not exceed the parameter MAXANG in the program.
- **xmu** (*N*) – Cosines of angles (N = 1 TO NUMANG) at which S1,S2 are to be evaluated. If ANYANG = FALSE, then (a) the angles must be monotone increasing and mirror symmetric about 90 degrees (if 90-A is an angle, then 90+A must be also) (b) if NUMANG is odd, 90 degrees must be among the angles
- **nmom** – Highest Legendre moment PMOM to calculate, numbering from zero (NMOM = 0 prevents calculation of PMOM)
- **ipolzn** – POSITIVE, Compute Legendre moments PMOM for the Mueller matrix elements determined by the digits of IPOLZN, with 1 referring to M1, 2 to M2, 3 to S21, and 4 to D21 (Ref. 3). E.g., if IPOLZN = 14 then only moments for M1 and D21 will be returned. 0, Compute Legendre moments PMOM for the npolarized unnormalized phase function. NEGATIVE, Compute Legendre moments PMOM for the Sekera phase quantities determined by the digits of ABS(IPOLZN), with 1 referring to R1, 2 to R2, 3 to R3, and 4 to R4 (REF. 4). E.g., if IPOLZN = -14 then only moments for R1 and R4 will be returned. (NOT USED IF NMOM = 0)
- **momd** – Determines first dimension of PMOM, which is dimensioned internally as PMOM(0:MOMDIM, *) (second dimension must be the larger of unity and the highest digit in IPOLZN; if not, serious errors will occur). Must be given a value, even if NMOM = 0. Minimum: 1.
- **prt** (*L*) – Print flags (LOGICAL). L = 1 prints S1,S2, their squared absolute values, and degree of polarization, provided NUMANG is non-zero. L = 2 prints all output variables other than S1,S2.

Returns list containing the following

qext: (REAL) extinction efficiency factor (Ref. 2, Eq. 1A)
qsca: (REAL) scattering efficiency factor (Ref. 2, Eq. 1B)
gqsc: (REAL) asymmetry factor times scattering efficiency
(Ref. 2, Eq. 1C) (allows calculation of radiation
pressure efficiency factor QPR = QEXT - GQSC)
NOTE – S1, S2, SFORW, SBACK, TFORW, AND TBACK are calculated
internally for negative imaginary refractive index;
for positive imaginary index, their complex conjugates
are taken before they are returned, to correspond to
customary usage in some parts of physics (in particular,
in papers on CAM approximations to Mie theory).
pmom(M,NP): (REAL) moments M = 0 to NMOM of unnormalized NP-th
phase quantity PQ (moments with M .GT. 2*NTRM are
zero, where NTRM = no. terms in Mie series =
XX + 4*XX**1/3 + 1)
PQ(MU, NP): = sum(M=0 to infinity) ((2M+1)
* PMOM(M,NP) * P-sub-M(MU))
WHERE MU = COS(scattering angle)
P-sub-M = M-th Legendre polynomial
and the definition of 'PQ' is as follows:
IPOLZN.GT.0: PQ(MU,1) = CABS(S1(MU))**2

PQ(MU,2) = CABS(S2(MU))**2
PQ(MU,3) = RE(S1(MU)*CONJG(S2(MU)))
PQ(MU,4) = - IM(S1(MU)*CONJG(S2(MU)))
(called M1, M2, S21, D21 in literature)
IPOLZN=0: PQ(MU,1) = (CABS(S1)**2 + CABS(S2)**2) / 2
(the unnormalized phase function)
IPOLZN.LT.0: PQ(MU,1) = CABS(T1(MU))**2
PQ(MU,2) = CABS(T2(MU))**2
PQ(MU,3) = RE(T1(MU)*CONJG(T2(MU)))
PQ(MU,4) = - IM(T1(MU)*CONJG(T2(MU)))
(called R1, R2, R3, R4 in literature)
The sign of the 4th phase quantity is a source of
confusion. It flips if the complex conjugates of
S1,S2 or T1,T2 are used, as occurs when a
refractive index with positive imaginary part is
used (see discussion below). The definition above
is consistent with a negative imaginary part.
See Ref. 5 for correct formulae for PMOM (Eqs. 2-5
of Ref. 3 contain typographical errors). Ref. 5 also
contains numerous improvements to the Ref. 3 formulas.
NOTE THAT OUR DEFINITION OF MOMENTS DIFFERS FROM REF. 3
in that we divide out the factor (2M+1) and number the
moments from zero instead of one.
** WARNING ** Make sure the second dimension of PMOM
in the calling program is at least as large as the
absolute value of IPOLZN.
For small enough values of XX, or large enough values
of M, PMOM will tend to underflow. Thus, it is
unwise to assume the values returned are non-zero and,
for example, to divide some quantity by them.

s1(N),s2(N): (COMPLEX) Mie scattering amplitudes at angles specified
by XMU(N) (N=1 to NUMANG) (Ref. 2, Eqs. 1d-e).

sforw: (COMPLEX) forward-scattering amplitude S1 at
0 degrees. (S2(0 deg) = S1(0 deg))

sback: (COMPLEX) backscattering amplitude S1 at
180 degrees. (S2(180 deg) = - S1(180 deg))

tforw(I): (COMPLEX) values of
I=1: T1 = (S2 - (MU)*S1) / (1 - MU**2)
I=2: T2 = (S1 - (MU)*S2) / (1 - MU**2)
At angle theta = 0 (MU = COS(theta) = 1), where the
expressions on the right-hand side are indeterminate.
(these quantities are required for doing polarized
radiative transfer (Ref. 4, Appendix).)

tback(I): (COMPLEX) values of T1 (for I=1) or T2 (for I=2) at
angle theta = 180 degrees (MU = COS(theta) = - 1).

spike: (REAL) magnitude of the smallest denominator of
either Mie coefficient (a-sub-n or b-sub-n),

taken over all terms in the Mie series past
N = size parameter XX. Values of SPIKE below
about 0.3 signify a ripple spike, since these
spikes are produced by abnormally small denominators
in the Mie coefficients (normal denominators are of
order unity or higher). Defaults to 1.0 when not
on a spike. Does not identify all resonances
(we are still working on that).

class xscat.**mievs**

Mie scattering and Rayleigh-Gans approximation for scattering by dust

list()

List object

xscat.**xfresnel**(*alpha*, *gamma*, *angs*)

Calculate X-ray reflectivity using Fresnel's equations

Parameters

- **alpha** – real incremental part dielectric constant
- **gamma** – imaginary part of dielectric constant
- **angs** – incidence angles (degrees)

Returns

Return type list of following

rs: sigma reflectivity
rp: pi reflectivity
runp: unpolarized reflectivity

If angs(I) out of range 0-90 degrees returns zero reflectivity

xscat.**xopt**(*mspec*, *rho*, *ekev*, *itype*)

X-ray optical properties of a material

Parameters

- **mspec** – specification of composition
- **rho** – density gm/cm**3
- **ekev** – array of photon energies in keV
- **itype** – data source (0=Cromer, 1=Henke)

Returns list with the following

alpha: array of real parts dielectric constant
gamm: array of imaginary parts dielectric constant
absl: array of absortion lengths cm-1
f1: array of real parts scattering factor

f2: array of imaginary parts scattering factor

CHAPTER

SEVEN

XSRT - SEQUENTIAL RAY TRACING

The code was written for modelling grazing incidence X-ray telescopes but it works at normal incidence and includes many common optical elements.

The code is *sequential* in the sense that rays encounter the optical elements in the order that they are specified. However, by setting flags associated with each element the sequence can be controlled dynamically to handle multiple interactions, between different optical elements, in any sequence.

7.1 Example Scripts

The following Python and R scripts illustrate how the ray tracing is done.

The sequence of elements is:

source –> mirrors/stops/lenses/gratings/etc. –> detector

```
#!/usr/bin/env python
# Use Swift XRT geometry as an example
from __future__ import print_function
import sys
import numpy as np
import images
import xsrt
# Useful vectors
sn=np.array([1,0,0])
nn=np.array([0,0,0])
rx=np.array([0,1,0])
# Set look-up table reflectivity to 1.0
angs=np.array([0,90])
refs=np.array([1,1])
# Support spiders
spi=np.array([3838.8,0,0])
tp=np.array([3800,0,0])
conea=10.05
nsec=12
cwid= 0.0
awid= 3.0
edf2=np.array([3200,0,0])
edf1=np.array([3161.2,0,0])
# Wolter I shell parameters
fl= 3500
ph= 3800
hl= 3200
ra= 1.0
```

(continues on next page)

(continued from previous page)

```
ns= 13
rj=np.array([146.880,140.980,135.320,129.890,124.670,119.660,114.850,
110.240,105.810,101.560,97.490,93.560,90.833])
tt=np.array([1.25,1.20,1.15,1.10,1.05,1.00,0.95,0.90,0.85,0.80,0.75,0.70,0.70])
# Source
di=np.array([-1,0,0])
rlim=np.array([92.0,151.0,0,0,0,0])
nray= 10000
# Detector
rdet= 30
dlim=np.array([0,rdet,0,0])
dpos=np.array([0,0,0])
# image paramters
nx= 100
ny= 100
hwid= 5.0
# Ray tracing calls
xsrt.reset()
xsrt.source(1,di,nn,spi,sn,rx,rlim,0.0,nray,0)
xsrt.surface(1,2,0.0,0.0,0.0,0.0,0.0,0.0,angs,refs,0,0,0)
xsrt.spider(-conea,spi,sn,rx,nsec,cwid,awid)
xsrt.spider(0.0,tp,sn,rx,nsec,cwid,awid)
xsrt.w1nest(fl,rj,ra,fl,ph,hl,fl,tt,tt,tt,sn,rx,nn,0,1,0)
xsrt.spider(0.0,edf2,sn,rx,nsec,cwid,awid)
xsrt.spider(conea,edf1,sn,rx,nsec,cwid,awid)
xsrt.detector(1,dpos,sn,rx,dlim,0.0)
results=xsrt.trace(0,rdet,-2)
# Create an image of the detected area
XD,YD,ZD,XC,YC,ZC,XR,YR,ZR,YDET,ZDET,AREA,IREF=np.loadtxt("detected.dat",skiprows=1,
 ↪unpack=True)
arr=images.binxy(YDET,ZDET,0,AREA,-hwid,hwid,-hwid,hwid,nx,ny)
# Analyse beam to get total collecting area
images.setfield(nx,-hwid,hwid,ny,-hwid,hwid)
images.setpos(2,[0,0])
bb=images.beam(a,hwid,0,0)
area=bb.flux/100.
print("area cm^2",area)
```

```
#!/usr/bin/env Rscript
# Use Swift XRT geometry as an example
# Useful vectors
sn<-c(1,0,0)
nn<-c(0,0,0)
rx<-c(0,1,0)
# Set look-up table reflectivity to 1.0
angs<- c(0,90)
refs<- c(1,1)
# Support spiders
spi<- c(3838.8,0,0)
tp<- c(3800,0,0)
conea<- 10.05
nsec<- 12
cwid<- 0.0
awid<- 3.0
edf2<- c(3200,0,0)
edf1<- c(3161.2,0,0)
```

(continues on next page)

(continued from previous page)

```
# Wolter I shell parameters
fl<- 3500
ph<- 3800
hl<- 3200
ra<- 1.0
ns<- 13
rj<-   c(146.880,140.980,135.320,129.890,124.670,119.660,114.850,110.240,
105.810,101.560,97.490,93.560,90.833)
tt<- c(1.25,1.20,1.15,1.10,1.05,1.00,0.95,0.90,0.85,0.80,0.75,0.70,0.70)
# Source
di<- c(-1,0,0)
rlim<- c(92.0,151.0)
nray<- 10000
# Detector
rdet<- 30
dlim=c(0,rdet,0,0)
dpos<- c(0,0,0)
# image paramters
nx<- 100
ny<- 100
hwid<- 5.0
# Ray tracing calls
qrt_reset()
qrt_source(1,di,nn,spi,sn,rx,rlim,0.0,nray,0)
qrt_surface(1,2,0.0,0.0,0.0,0.0,0.0,0.0,angs,refs,0,0,0)
qrt_spider(-conea,spi,sn,rx,nsec,cwid,awid)
qrt_spider(0.0,tp,sn,rx,nsec,cwid,awid)
qrt_w1nest(fl,rj,ra,fl,ph,hl,fl,tt,tt,tt,sn,rx,nn,0,1,0)
qrt_spider(0.0,edf2,sn,rx,nsec,cwid,awid)
qrt_spider(conea,edf1,sn,rx,nsec,cwid,awid)
qrt_detector(1,dpos,sn,rx,dlim,0.0)
results<- qrt_trace(0,rdet,-2)
# Create an image of the detected area
detpos<-read.table("detected.dat",header=TRUE)
a<-qri_binxy(detpos$YDET,detpos$ZDET,0,detpos$AREA,-hwid,hwid,nx,-hwid,hwid,ny)
# Analyse beam to get total collecting area
bb<-qri_beam(a$data_array,hwid,0,0)
area<-bb$flux/100.
cat("area cm^2",area,"\n")
```

7.2 Optical Elements and Coordinates

Each of the elements, including the source and detector, are specified by:

- 3 surface reference vectors - origin position, surface normal at origin and reference tangent at origin
- The surface figure - planar, spherical or conic section - parameters to define the curvature etc. of the figure
- The surface boundary - circles or rectangles in local surface coordinates
- A surface quality - source of rays, detection, reflection, diffraction, scattering, refraction, absorption
- The surface deformation - a grid of displacements defined in local surface coordinates

A full list of all the currently defined elements is produced by the function srtlist().

Individual elements referenced by the surface element index can be shifted and rotated using shift() and rotate().

The data base of elements can be cleared to the initial condition (no elements defined) using the function reset(). If elements are repeatedly defined within a procedure (for instance within a loop) the safe and prefered option is to reset() everything and redefine all elements each time they are required.

Coordinates

There is no fixed coordinate system and elements can be set/defined at any orientation. However it is conventional to use the X-axis as the optical axis (which defines the direction of paraxial rays) and the Y-axis as the nominal tangent reference axis. Of course given elements may not be aligned exactly with the X-axis and Y-axis. In most cases the local coordinates in the detector plane are nominally aligned with the Y-axis and Z-axis. Rays are usually traced from right to left travelling in the -X direction but this is not necessary and it is possible for rays to bounce back and forth as in, for example, a cassegrain system.

The source is always the first element in the sequence. All other elements are placed in sequence as they are defined. If the source() function is used repeatedly the source specification will be overwritten each time. If the detector command is used repeatedly a new detector will be added to the sequence each time and all detectors defined will be active.

Local surface coordinates are specified using the tangent plane to the surface at the point defined as the surface origin. For a sphere points on this tangent plane are projected onto the surface along the normal to the surface at the surface origin (Lambert's projection). The local x-axis is specified by a tangent vector at the surface origin. The local y-axis is the cross product of the normal and tangent vector at the surface origin.

The coordinates used for the limits of apertures and stops are given in the docstrings of the xsrt.aperture() function.

The local coordinates used for surfaces of revolution generated from conic sections (hyperbola, parabola, ellipse) depend on whether the surface is designated as being "normal" or "grazing" incidence. For normal incidence they are defined in a similar way to the planar or spherical surfaces as given above. For grazing incidence a cylindrical coordinate system is used where the axis is the normal to the surface at the surface origin and the azimuth is the rotation about this axis with zero at the surface reference axis at the surface origin. Local coordinates are given as axial position and azimuthal position (radians). The limits of such surfaces are specified by axial and/or radial limits corresponding to the bottom and top edges of the surface of revolution.

7.3 Source and Detector

Source of Rays

The source of rays consists of an annular or rectangular aperture on a planar surface. The origin of each ray is a random point within the aperture. The direction of the rays is specified either by a source at infinity, a source at a finite distance or diffuse. For a source at infinity all rays are parallel with the direction set by direction cosines. A source at a finite distance is specified by a position vector somewhere behind the aperture. Diffuse rays are generated so as to give a uniform random distribution over a hemisphere. The total number of rays generated is either set explicitly or by using an aperture area per ray.

Only one source can be specified. If the source command is used in a loop then the source will change on each pass through the loop.

The deformation index is used to specify a pixel array which spreads out a point source into an angular distribution. The deformation data are set using two functions xsrt.deform() and xsrt.defmat(). If the source is at infinity the x and y sample arrays must be in radians measured from the direction **sd** along the reference axis **ar** and the other axis. (**an** cross **ar**). If the source is at a finite distance then x and y are displacements in mm (or whatever distance unit is used) of the position **sp** along the reference axis **ar** and the other axis (**an** cross **ar**).

Detector

The detector consists of an annular or rectangular aperture on a planar or spherical surface. More than one detector can be specified for an instrument. Each detector defined will occupy a given position within the sequence of optical elements specified.

7.4 Monte Carlo and Random Numbers

The starting positions of rays, X-ray scattering angles from surface roughness and some surface figure errors/deformations are generated using random numbers. The sequence of random numbers used will be different each run unless the random number seed is set using function rseed(). If the same seed value is set before calling the function trace() exactly the same random sequence will be generated for the ray tracing and the results will be identical.

7.5 Deformations

Deformations of surfaces can be specified using matricies which either span a grid of points in the local coordinate system of the surface or are indexed using integer labels for sectors or areas on a surface. A set of deformations pertaining to a single surface or group of related surfaces are given a deformation index (integer 1,2,3...). The positions of the deformation grid points in local coordinates are specified by two 1-dimensional arrays.

Alternatively deformations can be specified using functions with parameters set separately for local x and y coordinates.

A surface deformation is applied along the normal to the surface. The deformation value is interpolated from the 2-d grid of points.

Radial deformations for annula apertures are specified by a vector, sampling in 1-d in azimuth, and the deformation is applied as a perturbation in the radial direction.

The function xsrt.deform() is used to set up the type and dimensionality of a particular deformation and must be the first call. The component matricies or function parameters are then set using calls to xsrt.defmat() or xsrt.defparxy().

A deformation applied to the source() spreads the point source into a pixel array. See **Source of Rays**.

7.6 Surface Quality, Reflectivity and Scattering

Several surface qualities can be set up for the simulation of a given instrument. Each is referenced using a surface quality index (integer 1,2,3...). The type of surface can be reflecting (with reflectivity specified using Fresnel's equations or a lookup table), refracting or diffracting. The roughness of the surface can also be specified using a power law distribution.

The X-ray optical constants **alpha** and **gamma** can be calculated for a material of specified composition using the function xscat.xopt(). Within the ray tracing the reflectivity is calculated using these constants using the same code as in function xscat.xfresnel().

The reflectivity as a function of incidence angle in other energy bands can be calculated from the real and imaginary part of the refractive index using the function fresnel().

Stops which are intended to block radiation have a surface quality index set to 0. When rays hit such surfaces they are terminated (absorbed). Detectors have surface quality index -1. If a ray hits such a surface it is terminated (detected). The source aperture surface has quality index -2. The quality indices of the source, stops and detectors are set automatically. As ray tracing proceeds rays are stored for further analysis. Each position along a ray where an intersection with a surface element occured is labelled with the quality index of the surface.

For a grating the surface type is it=4. In this case the ruling direction is specified by the surface element axis and dhub controls the geometry. dhub < 1 in-plane in which the dhub specifies the d-spacing gradient across the ruling and dhub > 1 off-plane where the d-spacing gradient along the ruling is determined from the distance to the hub.

7.7 X-ray Telescopes, Lens and Prism

Wolter Telescopes

A nest of Wolter I shells is set up using the function w1nest() and a conical approximation to the same by c1nest(). A Wolter II telescope is set up using the function wolter2(). A Wolter I telescope manufactured as an array of Silicon Pore Optics (like Athena) is set up using the function spoarr().

Lobster Eye and Kirkpatrick-Baez Telescopes

A lobster eye telscope is set up using the function sqmpoarr().

A silicon pore Kirkpatrick-Baez stack is defined using the function kbs().

A Schmidt configuration lobster eye telescope is set up using the function sle().

Apertures, stops, baffles and support structure

The function aperture() sets up stops with various geometries, single annulus, nested annuli, rectangular holes/blocks, rectangular grid, polar sectors, paralleogram. Cylindrical baffles in front of behind circular apertures are set up using the function baffle(). Spider support structures commomly used in Wolter systems are set up using the function spider().

Lens and Prism

Refracting lens and prism are defined using functions lens() and prism().

7.8 Ray Tracing and Saving Rays

Once the source, detector and other elements have been defined rays can be traced through the instrument using the function trace(). The form of the output is controlled by the parameter **iopt**.

- -2 save traced.dat and detected.dat files
- -1 save detected.dat
- 0 don't save files or adjust focus
- 1 adjust focus and save detected.dat
- 2 adjust focus and save detected.dat and traced.dat
- Only rays with **iopt** reflections are used in adjustment

The files traced.dat and detected.dat are ASCII tabulations.

When **iopt** is +ve then the detector position which gives the best focus is determined. Only rays which have **iopt** reflections and impact the detector within a radius **riris** of the centre of the detector are included in the analysis. The detector is shifted along the normal direction to find the axial position of minimum rms radial spread. The results of this analysis are returned as:

- **area** detected area within RIRIS
- **dshft** axial shift to optimum focus (0.0 if IOPT<=0)
- **ybar** y centroid of detected distribution
- **zbar** z centroid of detected distribution
- **rms** rms radius of detected distribution

The file traced.dat contains the paths of all the rays. It can be very large so should not be saved unless required for detailed analysis.

- **RXP,RYP,RZP** positions of points along each ray
- **AREA** aperture area associated with ray
- **IQU** quality index -2 at source, 1 reflected, 0 absorbed, -1 detected

Note: in the tabulation the beginning of each ray is identified using **IQU=-2** and the end using **IQU=0** absorbed or **IQU=-1** detected. Using these data you can plot the paths of all the rays.

The file detected.dat contains information about the detected rays.

- **XD,YD,ZD** the detected position for each ray
- **XC,YC,ZC** the direction cosines for each ray
- **XR,YR,ZR** the position of the last interaction before detection
- **YDET,ZDET** the local detected position on detector
- **AREA** the aperture area associated with the ray
- **IREF** the number of reflections suffered by the ray

The position **XR,YR,ZR** is used to indicate where the ray came from.

The following snippets of code show how an image of the detected rays can be generated in Python or R.

```
import numpy as np
import images
import xsrt
...
...
# half width of image mm
hwid=5.0
# trace all the rays
results=xsrt.trace(0,rdet,-2)
# Create an image of the detected area
XD,YD,ZD,XC,YC,ZC,XR,YR,ZR,YDET,ZDET,AREA,IREF=np.loadtxt("detected.dat",
    skiprows=1,unpack=True)
arr=images.binxy(YDET,ZDET,0,AREA,-hwid,hwid,-hwid,hwid,nx,ny)
```

```
# half width of image mm
hwid<- 5.0
# trace all the rays
results<- qrt_trace(0,rdet,-2)
# Create an image of the detected area
detpos<-read.table("detected.dat",header=TRUE)
aim<-qri_binxy(detpos$YDET,detpos$ZDET,0,detpos$AREA,-hwid,hwid,nx,-hwid,hwid,ny)
```

In Python **arr** is an image array. In R **aim** is an image object which contains the image array **aim$data_array**. In both cases the function images.binxy() is used to bin up the aperture area associated with each ray into an image (2-d histogram). The effective area is found by summing up areas of the image.

7.9 xsrt.functions

Utility functions

- rotate() Rotate surface element
- shift() Shift position of surface element

- rseed() Set random number seed
- srtlist() List all current xsrt parameters
- setreset() Reset Fortran common blocks to inital condition

Apertures, baffles and support structure

- aperture() Set up an aperture stop
- baffle() Set up a cylindrical baffle
- spider() Set up a support spider

Wolter systems

- w1nest() Set up a Wolter Type I nest
- c1nest() Set up conical approximation to a Wolter type I nest
- sipore() Set up Silicon Pore Optics
- spoarr() Set up Silicon Pore Optics array
- wolter2() Set up Wolter Type II surfaces

Square pore and Kirkpatrick-Baez systems

- sqpore() Set up slumped square pore MPOs
- sqmpoarr() Set up an array of square pore MPOs
- kbs() Set up a Silicon Kirkpatrick-Baez stack array
- sle() Set up a Schmidt lobster eye stack

Common optical elements

- lens() Set up a lens
- prism() Set up a prism
- mirror() Set up a plane mirror
- elips() Set up elliptical grazing indidence mirror
- moa() Set up a cylindrical Micro Optic Array
- opgrat() Set up a single off-plane grating

Surface quality and deformations

- surface() Set surface quality parameters
- fresnel() Calculate reflectivity using Fresnel's equations
- deform() Set up surface deformation dimensions
- defmat() Load deformation matrix
- defparxy() Set up deformation defined by parameters in x and y axes

Source of rays, tracing rays, detecting rays

- source() Set up source of rays
- trace() Perform ray tracing
- detector() Set up detector

Tracing charged particles through magnetic fields

- bfield() Calculation of magnetic field for array of dipoles
- eltmxt() Trace electrons through SVOM MXT telescope with magnetic diverter
- prtathena() Trace proton through Athena telescope with magnetic diverter

xsrt.**aperture**(*idd*, *idf*, *ap*, *an*, *ar*, *alim*, *nsurf*)
Set up an aperture stop

Parameters

- **id** – aperture type
- **idf** – deformation index
- **ap** – position of aperture
- **an** – normal to aperture plane
- **ar** – reference axis in aperture plane
- **alim** – limit values (depend on id see above)
- **nsurf** – number of subsequent surfaces per aperture (id=2)

id values:

1 Single annulus, radial limits (aref,rmin,rmax,0,0,0)
2 Nested annuli, radial limits (aref,rmin1,rmax1,rmin2,0,0)
aref is an axial reference position used for deformation
3 Hole Cartesian limits (xmin,ymin,xmax,ymax,0,0)
4 Block Cartesian limits (xmin,ymin,xmax,ymax,0,0)
5 Cartesian grid limits (pitchx pitchy ribx riby,0,0)
6 Radial/azimuthal sector limits (rmin,rmax,amin,amax,0,0)
amin and amax in radians range 0-2pi
7 Parallelogram limits (xmin,ymin,xmax,ymax,dx,0)
where dx is the shear in X over the distance ymax-ymin
8 Aperture for MCO test station limits (hsize dols,0,0,0,0)

The parameter **nsurf** is used for radially nested apertures so that the code knows how many surface elements to skip if a ray penitrates a particular annulus.

If the limits are radial the deformation is defined as a vector (1-d sampless in azimuth) and is applied in the radial direction.

If the limits are cartesian, radial/azimuthal or parallelogram the deformation is defined over a matrix (2-d) and is applied in the direction of the normal.

No deformation is used for **id=8**.

xsrt.**baffle**(*xmin*, *xmax*, *rad*, *ax*, *ar*, *rp*, *iq*)
Set up a cylindrical baffle

Parameters

- **xmin** – axial position of base
- **xmax** – axial position of top
- **rad** – radius of cylinder
- **ax** – axis direction

- **ar** – reference direction perpendicular to axis
- **rp** – position of vertex
- **iq** – surface quality index

The axial positions of the base and top, **xmin** and **xmax** are local coordinates wrt **ap** along the **ax** direction.

xsrt.**bfield**(*dm*, *pdx*, *pdy*, *pdz*, *ddx*, *ddy*, *ddz*, *px*, *py*, *pz*)

Calculation of magnetic field for array of dipoles

Parameters

- **dm** – dipole moments (Gauss cm3)
- **pdx** – x positions of dipoles (cm)
- **pdy** – y positions of dipoles (cm)
- **pdz** – z positions of dipoles (cm)
- **ddx** – x direction cosines of dipole moments
- **ddy** – y direction cosines of dipole moments
- **ddz** – z direction cosines of dipole moments
- **px** – x positions for calculation
- **py** – y positions for calculation
- **pz** – z positions for calculation

Returns

list of following
bf magnitude of B-field Gauss
dx direction cosines of B-field in x direction
dy direction cosines of B-field in y direction
dz direction cosines of B-field in z direction
rmin minimum distance from dipoles

xsrt.**clnest**(*ax*, *ar*, *ff*, *iq*, *ib*, *idf*, *pl*, *ph*, *hl*, *hh*, *rpl*, *rph*, *rhl*, *rhh*, *tin*, *tj*, *tout*)

Set up conical approximation to a Wolter type I nest

Parameters

- **ax** – optical axis
- **ar** – reference axis
- **ff** – position of focus
- **pl** – array of low axial position of parabola
- **ph** – array of high axial position of parabola
- **hl** – array of low axial position of hyperbola
- **hh** – array of high axial position of hyperbola
- **rp** – array of radii parabola near join
- **rp** – array of radii parabola at input aperture

- **rhl** – array of radii hyperbola at exit aperture
- **rjh** – array of radii hyperbola near join
- **tin** – array of thicknesses of shells at input aperture
- **tj** – array of thicknesses of shells at join plane (or near join)
- **tout** – array of thicknesses of shells at the output aperture

The innermost shell is a dummy and provides an inner stop for the nest

Deformation: A deformation sub-matrix is required for each shell (not including the innermost shell) Each matrix covers a grid of points along the axis in mm and azimuth in radians. The axial range covers both the 1st and 2nd reflection surfaces. The values in the matrices are radial displacments mm at the grid points.

xsrt.**defmat**(*idd*, *im*, *xsam*, *ysam*, *zdef*)
Load deformation matrix

Parameters

- **id** – deformation index
- **im** – sub-matrix index (runs from 1-N)
- **xsam** – x values
- **ysam** – y values
- **zdef** – deformation matrix

Note for deformation of a surface of revolution X is axial direction and Y is azimuth.

Used when **it=1** in deform().

xsrt.**deform**(*idd*, *it*, *nm*, *nx*, *ny*)
Set up surface deformation dimensions

Parameters

- **id** – deformation index
- **it** – deformation type (1 matrix, 2 sin() in x and/or y)
- **nm** – number of sub-matrices
- **nx** – number of x samples
- **ny** – number of y samples

Note when **it=2** the sin() function requires 3 parameters, amplitude (**amp**), wavelength (**lam**) and phase (**phi**) all in mm (or the same units) **defx=amp.sin(2.pi.(x-phi)/lam)**

If **it** -ve then uses **abs(it)** but expects a 2nd deformation to be defined which will be added to the current deformation.

xsrt.**defparxy**(*idd*, *im*, *xpar*, *ypar*)
Sets parameters of deformation defined by parameters in x and y axes.

Parameters

- **idd** – deformation index
- **im** – sub-surface index
- **xpar** – parameters of deformations in x axis
- **ypar** – parameters of deformations in y axis

Empty field [] can be used as xpar or ypar if there are no deformations in the corresponding axis.

Use this routine to set parameteric deformations, e.g. when **it=2** in deform() the sin() function requires 3 parameters, amplitude (**amp**), wavelength (**lam**) and phase (**phi**) all in mm (or the same units) **defx=amp.sin(2.pi.(x-phi)/lam)**

xsrt.**detector**(*idd*, *dpos*, *dnml*, *drfx*, *dlim*, *radet*)
Set up detector

Parameters

- **id** – detector type
- **dpos** – detector position
- **dnml** – detector normal
- **drfx** – detector reference axis
- **dlim** – detector limits
- **radet** – radius of curvature of spherical detector

id values:

1 planar detector, radial limits
2 planar detector, cartesian limits
3 spherical detector, radial limits
4 spherical detector, cartesian limits

xsrt.**elips**(*org*, *axs*, *cen*, *xmin*, *xmax*, *amin*, *amax*, *smb*, *rab*, *ide*, *iq*)
Set up elliptical grazing indidence mirror

Parameters

- **org** – local origin on surface of ellipse
- **axs** – reference axis in aperture
- **cen** – focus of ellipse
- **xmin** – minimum axial position
- **xmax** – maximum axial position
- **amin** – minimum azimuth radians
- **amax** – maximum azimuth radians
- **smb** – conic coefficient
- **rab** – conic coefficient conic equation of form r**2=rab**2.x**2+smb**2
- **ide** – deformation index
- **iq** – reflecting surface quality

xsrt.**eltmxt**(*dm*, *pdx*, *pdy*, *pdz*, *ddx*, *ddy*, *ddz*, *ekv*, *tsig*, *xaper*, *xdiv*, *apy*, *apz*, *apsiz*, *wdiv*, *ddiv*, *sdet*, *maxst*)
Trace electrons through SVOM MXT telescope with magnetic diverter

Parameters

- **dm** – dipole moments (Gauss cm3)
- **pdx** – x positions of dipoles (cm)

- **pdy** – y positions of dipoles (cm)
- **pdz** – z positions of dipoles (cm)
- **ddx** – x direction cosines of dipole moments
- **ddy** – y direction cosines of dipole moments
- **ddz** – z direction cosines of dipole moments
- **ekv** – proton energy keV
- **tsig** – rms width of beam degrees
- **xaper** – x position of mirror aperture cm (focal length)
- **xdiv** – x position of diverter input aperture cm
- **apy** – y centre of apertures cm
- **apz** – z centre of apertures cm
- **apsiz** – aperture size (3.8) cm
- **wdiv** – extra width of diverter apertures (0.2) cm
- **ddiv** – axial depth of diverter cm
- **sdet** – size of detector cm (axial position XDET=0.0)
- **maxst** – maximum number of steps along path

Returns

list of the following
npath number of points along path
iqual type of path
0 hits active detector
1 too close to dipole
2 hits telescope tube
3 hits mirror aperture
4 hits diverter
5 hits focal plane beyond detector
6 hit maximum number of steps
xp x positions along path cm
yp y positions along path cm
zp z positions along path cm

xsrt.**fresnel**(*nreal*, *kimag*, *angs*)
Calculate reflectivity using Fresnel's equations

Parameters

- **nreal** – real part of refractive index
- **kimag** – imaginary part of refractive index
- **angs** – array of incidence angles (degrees range 0-90)

Returns

list of following
rs sigma reflectivity
rp pi reflectivity
runp unpolarized reflectivity

Reference "Handbook of Optical Constants of Solids" Ed. Edward D.Palik
Academic Press 1985, page 70
If angs(I) out of range 0-90 degrees returns zero reflectivities

xsrt.**init**()
Initialisation of Fortran common blocks

xsrt.**kbs**(*pcen*, *pnor*, *raxi*, *ipack*, *rmin*, *rmax*, *flen*, *csize*, *pitch*, *wall*, *plmin*, *plmax*, *idf*, *iq*)
Set up a Silicon Kirkpatrick-Baez stack array

Parameters

- **pcen** – centre of telescope aperture
- **pnor** – normal to aperture
- **razi** – reference axis at centre of aperture
- **ipack** – packing 0 single module, 1 sunflower, 2 cartesian, 3 wide field cartesian
- **rmin** – minimum radius for aperture of constellation
- **rmax** – maximum radius for aperture of constellation
- **flen** – focal length (-ve for 2nd stack)
- **csize** – size of each module in constellation
- **pitch** – pitch of K-B slots
- **wall** – wall thickness of K-B slots
- **plmin** – minimum axial length of K-B slots
- **plmax** – maximum axial length of K-B slots
- **idf** – deformation index
- **iq** – surface quality index

Returns

rc radius of each module
pc azimuth of each module
tc rotation of each module
ac axial length of each module
nset number of module coordinates returned

Deformation:

matrix is (5,nmod) specifying 5 deformations per module.
1 displacement of module vertex in **razi** mm
2 displacement of module vertex in **pnor X razi** mm

3 displaement of module vertex in **pnor** mm
4 rms Gaussian in-plane figure errors radians
5 rms Gaussian out-of-plane figure errors radians

xsrt.**lens**(*idd*, *idf*, *iq*, *anml*, *arfx*, *apos*, *rap*, *r1*, *r2*, *refind*, *thickq*)
Set up a lens

Parameters

- **id** – lens type 1 spherical, 2 cylindrical
- **idf** – deformation index
- **iq** – surface quality index
- **anml** – surface normal
- **arfx** – surface reference axis
- **apos** – surface reference position
- **rap** – radius of aperture
- **r1,r2** – radii of curvature of lens surfaces
- **refind** – refractive index of lens material (or n2/n1)
- **thick** – lens thickness

Surface quality: This function sets up 2 surface qualities **iq** and **iq+1** to represent the entrance and exit surfaces.

xsrt.**mirror**(*idd*, *idf*, *iq*, *anml*, *arfx*, *apos*, *alim*, *nsurf*)
Set up a plane mirror

Parameters

- **idd** – aperture type
- **idf** – deformation index
- **iq** – surface quality index
- **anml** – surface normal
- **arfx** – surface reference axis
- **apos** – surface reference position
- **alim** – limits array (see aperture())
- **nsurf** – number of subsequent surfaces ID=2

The aperture limits of the mirror are specified in the same way as for a top. See function aperture().

xsrt.**moa**(*pcen*, *pno*, *rax*, *rcur*, *xyap*, *pitch*, *wall*, *plen*, *idf*, *iq*)
Set up a cylindrical Micro Optic Array

Parameters

- **pcen** – centre of MOA
- **pno** – normal at centre of MOA
- **rax** – cylindrical axis of MOA
- **rcur** – radius of curvature of cylinder
- **xyap** – half width of MOA aperture

- **pitchnput** – pitch of slots
- **wall** – wall thickness between slots
- **plen** – depth of slots (thickness MOA)
- **idf** – deformation index
- **iq** – surface quality index

xsrt.**opgrat**(*idd*, *defi*, *iq*, *al*, *fpos*, *zpos*, *gpos*, *alim*)
Set up a single off-plane grating

Parameters

- **id** – 1 radial, 2 nested rad., 4 cart., 6 slats
- **defi** – deformation index
- **iq** – surface quality index
- **al** – azimuthal exit angle of zeroth order on cone degrees
- **fpos** – position of primary focus (not reflected)
- **zpos** – position of zeroth order focus (reflected)
- **gpos** – position of centre of grating
- **alim** – limits on surface

Returns

adir grating normal
rdir grating ruling
dpos grating hub
graz grating grazing angle radians
gam grating cone angle radians
dhub grating hub distance

The grating parameters **gpitch**, **dhub** and **order** and the reflectivity as a function of incidence angle are set by the surface quality (index **iq**).

xsrt.**prism**(*idd*, *idf*, *iq*, *anml*, *arfx*, *apos*, *rap*, *d1*, *d2*, *refind*, *thick*)
Set up a prism

Parameters

- **id** – 1 small angle, 2 right-angle
- **idf** – deformation index
- **iq** – quality index
- **anml** – entrance surface normal
- **arfx** – reference axis
- **apos** – reference position
- **rap** – aperture radius
- **d1** – small angle radians on entry side (if ID=1)

- **d2** – small angle radians on exit side (if ID=1)
- **refind** – refractive index of material or n2/n1
- **thick** – thickness

Surface quality: This function sets up 2 surface qualities **iq** and **iq+1** to represent the entrance and exit surfaces.

xsrt.**prtathena**(*dm*, *pdx*, *pdy*, *pdz*, *ddx*, *ddy*, *ddz*, *ekv*, *tsig*, *xaper*, *xdiv*, *rrings*, *trings*, *drings*, *rdet*, *maxst*)
Trace proton through Athena telescope with magnetic diverter

Parameters

- **dm** – dipole moments (Gauss cm3)
- **pdx** – x positions of dipoles (cm)
- **pdy** – y positions of dipoles (cm)
- **pdz** – z positions of dipoles (cm)
- **ddx** – x direction cosines of dipole moments
- **ddy** – y direction cosines of dipole moments
- **ddz** – z direction cosines of dipole moments
- **ekv** – proton energy keV
- **tsig** – rms width of beam degrees
- **xaper** – x position of mirror aperture cm (focal length)
- **xdiv** – x position of diverter input aperture cm
- **nrings** – number of rings in diverter
- **rrings** – radius of rings in diverter cm
- **trings** – radial thickness of rings in diverter cm
- **drings** – axial depth of rings in diverter cm
- **rdet** – radius of detector cm (axial position XDET=0.0)
- **maxst** – maximum number of steps along path

Returns

list of following
npath number of points along path
iqual type of path
0 hits active detector
1 too close to dipole
2 hits telescope tube
3 hits mirror aperture
4 hits diverter
5 hits focal plane beyond detector
6 hit maximum number of steps
xp x positions along path cm
yp y positions along path cm
zp z positions along path cm

xsrt.**reset**()
Reseting of Fortran common blocks to initial condition

xsrt.**rotate**(*iss*, *pl*, *ax*, *angle*)
Rotate surface element

Parameters

- **iss** – surface element index
- **pl** – position of rotation centre (x,y,z)
- **ax** – rotation axis (ax,ay,az)
- **angle** – rotation angle degrees

xsrt.**rseed**(*iseed*)
Set random number seed

Parameters **iseed** – integer seed

xsrt.**shift**(*iss*, *pl*)
Shift position of surface element

Parameters

- **iss** – surface element index
- **pl** – 3 vector shift (dx,dx,dz)

xsrt.**sipore**(*pcen*, *pnorm*, *raxis*, *flen*, *rpitch*, *apitch*, *wall*, *rm*, *pm*, *tm*, *wm*, *hm*, *am*, *cm*, *gm*, *wfr*, *a2j*, *idf*, *iq*)
Set up Silicon Pore Optics

Parameters

- **pcen** – position of centre of aperture (above join plane)
- **rnorm** – normal at centre of aperture
- **raxis** – reference axis at centre of aperture
- **flen** – focal length
- **rpitch** – pore radial pitch
- **apitch** – pore azimuthal pitch
- **wall** – wall thickness
- **rm** – array of module radii
- **pm** – array of module azimuths (radians)
- **tm** – array of module rotations (radians normally 0)
- **wm** – array of module widths (radial mm)
- **hm** – array of module heights (azimuthal mm)
- **am** – array of module lengths (axial pore length mm)
- **cm** – array of module curvature signatures 0 conical, 1 Wolter, 2 curve-plane, 3 constant…
- **gm** – array of module grazing angle ratios
- **wrf** – module frame width (surrounding module aperture)
- **a2j** – aperture to join plane axial distance

- **idf** – deformation index
- **iq** – reflecting surface quality

xsrt.**sle**(*pcen*, *pnor*, *raxi*, *flen*, *cwidth*, *cheight*, *pitch*, *wall*, *pl*, *nmir*, *idf*, *iq*)

Defines Schmidt configuration lobster eye stack

Parameters

- **pcen** – centre of telescope aperture
- **pnor** – normal to aperture
- **raxi** – reference axis at centre of aperture
- **flen** – focal length (-ve for second stack) measured to front aperture
- **cwidth** – cell width = mirror width (its active part)
- **cheight** – cell height
- **pitch** – pitch including mirrors thickness = pdd+wall
- **wall** – mirror thickness
- **pl** – mirror axial length (depth)
- **idf** – deformation index
- **iq** – surface quality index

Returns

rc radius of each module
pc azimuth of each module
tc rotation of each module
ac axial length of each module
nset number of module coordinates returned

xsrt.**source**(*it*, *sd*, *sp*, *ap*, *an*, *ar*, *al*, *apry*, *nr*, *idef*)

Set up source

Parameters

- **it** – source type
- **sd** – source direction cosines
- **sp** – source position
- **ap** – aperture position
- **an** – aperture normal
- **ar** – aperture reference axis
- **al** – aperture limits
- **apry** – area per ray - if 0 then use NR
- **nr** – number of rays
- **idef** – deformation index

it values:

1 point source at infinity, radial limits

2 point source at infinity, cartesian limits

3 point source at finite distance, radial limits

4 point source at finite distance, cartesian limits

Deformation: The deformation index is used to specify a pixel array which spreads out a point source into an angular distribution. The deformation data are set using two functions xsrt.deform() and xsrt.defmat(). If the source is at infinity the x and y sample arrays must be in radians measured from the direction **sd** along the reference axis **ar** and the other axis. (**an** cross **ar**). If the source is at a finite distance then x and y are displacements in mm (or whatever distance unit is used) of the position **sp** along the reference axis **ar** and the other axis (**an** cross **ar**).

xsrt.**spider**(*cone*, *apos*, *anml*, *arfx*, *nsec*, *cwid*, *awid*)

Set up a support spider

Parameters

- **cone** – 90-half cone angle degrees (0.0 for plane)
- **apos** – axial position of vertex (centre)
- **anml** – direction of normal to aperture (optic axis)
- **afrx** – reference axis in aperture
- **nsec** – number of sectors (number of arms)
- **cwid** – constant arm width
- **awid** – angular arm width degrees

The surface figure of the spider aperture is a cone. The width of the arms is given by **cwid+aw*R** where **R** is the radial distance from the axis of the cone and **aw** is **awid** degrees converted into radians.

xsrt.**spoarr**(*pcen*, *pnorm*, *raxis*, *flen*, *a2j*, *rm*, *pm*, *tm*, *wm*, *hm*, *am*, *cm*, *gm*, *rpitch*, *wall*, *apitch*, *wfr*, *siq*, *idf*)

Set up Silicon Pore Optics Wolter I array

Parameters

- **pcen** – position of centre of aperture (above join plane)
- **rnorm** – normal at centre of aperture
- **raxis** – reference axis at centre of aperture
- **flen** – focal length
- **a2j** – aperture to join plane axial distance
- **rm** – array of module radii
- **pm** – array of module azimuths (radians)
- **tm** – array of module rotations (radians normally 0)
- **wm** – array of module widths (radial mm)
- **hm** – array of module heights (azimuthal mm)
- **am** – array of module lengths (axial pore length mm)
- **cm** – array of module curvature signatures 0 conical, 1 Wolter, 2 curve-plane, 3 constant…
- **gm** – array of module grazing angle ratios

- **rpitch** – array of module pore radial pitch
- **wall** – array of module wall thickness
- **apitch** – array of module pore azimuthal pitch
- **rwi** – array of module pore rib thickness
- **wrf** – array of module frame widths (surrounding module aperture)
- **siq** – array of surface quality indices
- **idf** – deformation index

Deformation:

matrix is (6,nmod) specifying 6 deformations per module.
1 displacement of module in direction **raxis** mm
2 displacement of module in direction of **rnorm X raxis** mm
3 error in focal length of module
4 Gaussian rms in-plane figure error radians
5 Gaussian rms out-of-plane figure error radians
6 width mm along axial edges of module in which figure degrades

xsrt.**sqmpoarr**(*pcen*, *pnorm*, *raxis*, *rcur*, *hwid*, *idf*, *ar*)
Set up an array of square pore MPOs

Parameters

- **pcen** – position of centre of array
- **rnorm** – normal at centre of array
- **raxis** – reference axi at centre of array
- **rcur** – radius of curvature of array
- **hwid** – half width of array aperture
- **idf** – deformation index for array
- **ar** – array of additional parameters for each MPO in array

Additional parameter array:

there are 15 parameters for each MPO
XP x position of each MPO
YP y position of each MPO
TC rotation angle of each MPO
WC width of each MPO delx
HC height of each MPO dely
AC axial length of each MPO (thickness)
CU radius of curvature of each MPO
MF size of multifibres in MPO
PP pitch of pores (pore size + wall thickness)
WA wall thickness between pores
SQ reflecting surface quality index for pores
BU bias angle x radians
BV bias angle y radians

BZ spare parameter!
SP spare parameter!

Deformation:

matrix (5,nmpo) specifying 5 deformations for each MPO
1 scaling of intrinsic slumping distortion - 1.0 for model
2 +ve amplitude thermoelastic pore axial pointing errors radians
-ve amplitude fixed pattern tilt errors radians
3 Gaussian rms pore rotation errors (about pore axis) radians
4 pore shear error amplitude mm within multifibre structure
5 +ve Gaussian rms pore tilt/figure error radians (x2.37=FWHM)
-ve Cauchy (Lorentzian) pore tilt/figure errors radians (x2=FWHM)
both applied independently in 2 tilt axes

xsrt.**sqpore**(*pcen*, *pnorm*, *raxis*, *rcur*, *ipack*, *rap*, *pitch*, *wall*, *plen*, *idf*, *iq*, *plmin*, *plmax*, *fibre*, *ar*)
Set up slumped square pore MPO

Parameters

- **pcen** – position of centre of plate
- **rnorm** – normal at centre of plate
- **raxis** – reference axi at centre of plate
- **rcur** – radius of curvature
- **ipack** – pore packing 1 cart, 2 rad, 3 waff, 4 octag, 5 rand, 6 MIXS, 7 NFL
- **rap** – half width of plate aperture
- **pitch** – pitch of pores on a cartesian grid
- **wall** – pore wall thickness
- **plen** – length of pores
- **idf** – deformation index
- **iq** – surface quality index
- **plmin** – minimum pore length
- **plmax** – maximum pore length
- **fibre** – size of fibre bundle in packing
- **ar** – array of additional parameters specifying plate apertures

Additional parameter array:

used for ipack 6 or 7
RC,PC radius and azimuth of each plate
TC rotation angle of plate
WC width of each plate x
HC height of each plate y
AC axial length of each plate
CC spare
GC spare

xsrt.**srtlist**()
List all current xsrt parameters

xsrt.**srtreset**()
Reset Fortran common blocks to intial condition

xsrt.**surface**(*iss*, *it*, *ekev*, *srgh*, *fmin*, *pind*, *alpha*, *gamma*, *angs*, *refs*, *gpitch*, *dhub*, *order*)
Set surface quality parameters

Parameters

- **is** – surface quality index
- **it** – surface type 1 refl. (Fresnel), 2 refl. (look-up), 3 refract, 4 diffract
- **elev** – photon energy keV
- **srgh** – Specific roughness (A**2 mm) if -ve then rms figure gradient error radians
- **fmin** – Minimum surface spatial frequency
- **pind** – Surface roughness power spectrum index
- **alpha** – real part of diel. constant or refractive index ratio N1/N2
- **gamma** – imaginary part of dielectric constant
- **angs** – incidence angles (degrees increasing) (QTYPE 2 and 4)
- **refs** – reflectivity values (QTYPE 2 and 4)
- **gpitch** – d-spacing for grating mm (QTYPE 4)
- **dhub** – distance from ruling hub to surface reference point (QTYPE 4) if <1.0 mm then d-spacing gradient across ruling (QTYPE 4)
- **order** – diffraction order (QTYPE 4)

The X-ray optical constants **alpha** and **gamma** can be calculated for a material of specified composition using the function xscat.xopt(). It **it=1** the X-ray reflectivity is calculated using the same code as in the function xscat.

xsrt.**trace**(*ideb*, *riris*, *iopt*)
Perform ray tracing

Parameters

- **ideb** – debugging level (0 none)
- **riris** – radius about centre of detector for analysis if 0.0 then no analysis of detected distribution
- **iopt** – controls form of output

iopt:

-2 save traced.dat and detected.dat files
-1 save detected.dat
0 don't save files or adjust focus
1 adjust focus and save detected.dat
2 adjust focus and save detected.dat and traced
Only rays with **iopt** reflections are used in adjustment

Returns

list of the following

area detected area within RIRIS

dshft axial shift to optimum focus (0.0 if IOPT<=0)

ybar y centroid of detected distribution

zbar z centroid of detected distribution

rms rms radius of detected distribution

xsrt.**w1nest**(*xj*, *rj*, *ra*, *pl*, *ph*, *hl*, *hh*, *tin*, *tj*, *tout*, *ax*, *ar*, *ff*, *defi*, *iq*, *ib*)

Set up a Wolter Type I nest

Parameters

- **xj** – axial position of join plane
- **rj** – array of radii of shells at join
- **ra** – ratio of grazing angles
- **pl** – low axial position of parabola
- **ph** – high axial position of parabola
- **hl** – low axial position of hyperbola
- **hh** – high axial position of hyperbola
- **ti** – array of thicknesses of shells at input aperture
- **tj** – array of thicknesses of shells at join plane
- **tout** – array of thicknesses of shells at the output aperture
- **ax** – direction of axis
- **ar** – reference axis in aperture
- **ff** – position of focus
- **defi** – deformation index
- **iq** – reflecting surface quality index
- **ib** – back of shells surface quality index

The innermost shell is a dummy and provides an inner stop for the nest

Deformation: A deformation sub-matrix is required for each shell (not including the innermost shell) Each matrix covers a grid of points along the axis in mm and azimuth in radians. The axial range covers both the 1st and 2nd reflection surfaces. The values in the matrices are radial displacments mm at the grid points.

xsrt.**wolter2**(*rp*, *gp*, *rh*, *gh*, *rm*, *fovr*, *ax*, *ar*, *ff*, *idf*, *iq*)

Set up Wolter Type II surfaces

Parameters

- **rp** – maximum radius of parabola
- **gp** – grazing angle (degrees) at maximum radius on parabola
- **rh** – maximum radius of hyperbola
- **gh** – grazing angle (degrees) at maximum radius on hyperbola
- **rm** – minimum radius of parabola
- **fovr** – radius of field of view degrees

- **ax** – direction of axis of telescope
- **ar** – reference direction perpendicular to axis
- **ff** – position of focus of telescope
- **id** – deformation index
- **iq** – surface quality index

Deformation: A single deformation matrix specifies radial displacement errors mm over a grid of points in axial mm and azimuthal radians. The grid covers the axial positions along both the 1st and 2nd reflection surfaces.

CHAPTER

EIGHT

ADDING NEW FUNCTIONS

The top-level subroutines are written in Fortran 77.

Subroutines names are prefixed with:

- QRA_ astronomy/astrophysics
- QRX_ X-ray astronomy/physics
- QRI_ image processing
- QR_FITS for the FITS file interface
- QRT_ sequential rays tracing interface
- QR_ general utilities

Internal Fortran routines also have prefixes:

- SRT_ internal sequential ray tracing
- AX_ coordinate transformations
- XX_ X-ray physics
- SYS_ system utilities

Two general Fortran routines are also suppied, SCAN and LEN_TRIM.

A few internal routines are written in C or C++. These are best left alone!

All the source code is held in directory:

$QSOFT/src/

8.1 Module Source Code

The modules are independent and have no extenal dependencies. If a routine is required by more than one module then a copy of the source code is included in each of the relevant source directories.

The source directories are:

- $QSOFT/src/qfits
- $QSOFT/src/astro
- $QSOFT/src/images
- $QSOFT/src/xscat
- $QSOFT/src/xsrt

Each directory contains the Fortran interface routines, prefixed by QR, that are called by Python, R or IDL and any internal Fortran subroutines required.

The source code for common blocks is held in files with uppercase names without a Fortran extension *.f, e.g. SRT_COM, SPX_COM. These are dragged into the subroutine source using the Fortran INCLUDE statement.

In addition the directories contain the module definition scripts, e.g. images.py, images.R. The IDL function interface is provided by individual *.pro files for each command/function and these are held in a subdirectory qIDL/ e.g. qIDL/qri_getpos.pro, qIDL/qri_beam.pro.

Each module directory contains a Makefile used to compile all the routines and create shareable libraries for Python, R and IDL. Note that IDL uses the library image created by R so R is required to produce the IDL shareable. Once compiled you can install the library and definition scripts using

$ make install

This copies the shareable libraries and definition scripts to the directories $QSOFT/python_modules, $QSOFT/R_libraries and $QSOFT/qIDL.

8.2 The Ray Tracing Routines

The driving routine, which is called by the interface routine QRT_TRACE is SRT_TRACE. Within this routine there are two loops. The outer loop is definite and works through the NRAYS which are to be generated by the source. The inner loop is indefinite and traces each ray through the surface elements. The next surface in the sequence is determined by the outcome of encounters with elements and multiple reflections between surfaces are allowed. This inner loop is terminated when the ray is absorbed, detected or finally misses the last element in the sequence.

Each surface element type is specified by an index as listed in this table:

#	surface	form	deform	limits	single/nest	code	ikon
1	plane	open	normal	cartesian	single	plna	1
2	plane	open	normal	radial	single	plna	2
3	plane	open	radial	radial	single	plna	3
4	plane	open	radial	radial	nested	plna	-n
5	plane	closed	normal	cartesian	single	plne	1
6	plane	closed	normal	radial	single	plne	2
7	plane	closed	radial	radial	single	plne	3
8	plane	closed	radial	radial	nested	plne	-n
9	conic	grazing	radial	axial	single	cnic	1
10	conic	normal	axial	radial	single	cnin	1
11	conic	normal	axial	cartesian	single	cnin	2
12	sphere	grazing	radial	cartesian	single	sphr	1
13	sphere	grazing	radial	radial	single	sphr	2
14	sphere	normal	radial	cartesian	single	sphr	3
15	sphere	normal	radial	radial	single	sphr	4
16	conic	normal	azimuthal	cartesian	single	cnin	3
17	plane	open	normal	azimuthal	single	plna	4
18	plane	open	normal	cartesian grid	single	plna	6
19	conic	grazing	radial	axial+azimuthal	single	cnic	2
20	plane	closed	normal	cartesian slat	single	plna	7
21	sphere	pore	normal	pore	multiple	sphr+pore	5
22	plane	open	normal	sector	single	plna	8
23	conic	moa	radial	axial+azimuthal	multiple	cnic	3

Continued on next page

Table 1 – continued from previous page

#	surface	form	deform	limits	single/nest	code	ikon
24	plane	Sipores	normal	pore	multiple	plna	9
25	plane	KBstack	normal	slots	multiple	plna	9
26	conic	normal	normal	axial+azimuthal	single	cnic	3
27	plane	open	normal	parallelogram	single	plna	10
28	sphere	MPOarr	normal	pore	multiple	sphr+arr	5
29	plane	MPOtest	normal	pore	single	plna	11
30	plane	SPOarr	normal	pore	multiple	plna	9
31	sphere	Schmidt	normal	slot	multiple	sle	5
32	plane	Schmidt	normal	slot	multiple	plns	1

Each of these surface types has a subroutine SRT_SUnn associated with it where nn is the type index. These routines call 1 of 13 routines which find the intersection of rays with a plane, sphere, conic, square pores, MOA slots, Si pores, Si pore array, K-B stack slots or Schmidt lobster stacks. The pore and slot routines set up the parameters for 4 plane surfaces which are then serviced by the SRT_PLNE routine or SRT_PLNS. The Si pore routine sets up the parameters for 8 plane surfaces to define the 1st and 2nd section of the Si pore.

SRT_PLNA	aperture on plane surface
SRT_PLNE	plane surface
SRT_CNIC	conic at grazing incidence
SRT_CNIN	conic at normal incidence
SRT_SPHR	spherical surface
SRT_PORE	square pore
SRT_MOA	moa slots
SRT_SIPORE	Si pores
SRT_KBS	Kirpatrick-Baez stack
SRT_PLNS	Schmidt version of PLNE
SRT_SLE	Schmidt lobster stack
SRT_SPOARR	Si pores module array
SRT_SQMPOARR	Square pore MPO array

The combination of form, deformation, limits and single/nested used in each of these routines is set by the integer argument IKON (see table above). Full details about these configurations and the parameters which they use are given in the comments at the start of the code for each routine.

All the surface element parameters etc. are held in a common area. Individual surface elements pick up their parameters using an index into this area. The parameters are set by the following routines:

routine	parameters	called by
SRT_SETF	surface	QRT_MIRROR, QRT_BAFFLE etc.
SRT_SETT	surface quality	QRT_SURFACE, QRT_LENS, QRT_PRISM
SRT_SETD	deformation	QRT_DEFS
SRT_SETS	source	QRT_SOURCE
SRT_SETCONCOM	SPO constellation	QRT_SIPORE
SRT_SETKBSCOM	KBS constellation	QRT_KBS
SRT_SETMPOARR	MPO array	QRT_SQMPOARR
SRT_SETSLECOM	Schmidt lobster eye	QRT_SLE
SRT_SETSPOCOM	Si Pore Optics	QRT_SPOARR

These routines put the parameters into common blocks. The order of the parameters is important and must match the order expected by the target routine. The sequence of parameters expected is specified in the comment lines at the top of the surface element routines.

8.3 New Surface Elements

In order to introduce a new type of ray tracing optical element you should check the list of available surface types. For example a spherical mirror at normal incidence using radial limits would be implemented using TYPE=15. If the required surface type exists then you only need to write a QRT_new routine which provides the QSOFT xsrt interface. This is easily done by copying and editing an existing routine, e.g. QRT_SPIDER.

If you want to add a new element which contains multiple surface elements then you should start with a routine like QRT_SPOARR. In this case you will also require new surface element routines e.g. SRT_SU30 and SRT_SPOARR are required by QRT_SPOARR.

The steps required to compose, compile and link are as follows:

1. Go to the source directory $QSOFT/src/xsrt
2. Write the new QRT_new routine as file qrt_new.f.

   Note down the parameters names required by this routine.
3. Edit the Makefile to include qrt_new.f in the source file list.
4. Use make to compile the new routine and link the shareble library

   $ make
5. Edit the xsrt.R and xsrt.py script files to include the new function.

   If you are going to use IDL move into the qIDL directory and create a qrt_new.pro file to define the function for IDL.

   You can use the definitions already present in these files to see how the interface works.

If the new optical element is not supported by any existing surface type then a new type must be invented. The programmer must write a new SRT_SUnn routine and modify and existing or produce a new SRT_type routine. A call to the new SRT_SUnn must also be included in the inner loop of the srt_trc.f file. The new SRT_SUnn and SRT_type routines must be edited into the makefile. Otherwise the process is the same as indicated above.

It is important that the parameters gathered by QRT_new are packed into common in the right order so that the relevant surface routine (SRT_PLNE etc.) access the parameters correctly. The programmer must check this by reading the comment lines at the start of the relevant surface routine.

The routine SRT_SETF is used to push the parameters into common. This has the following interface:

```
*+SRT_SETF      Set surface form and limits parameters
    SUBROUTINE SRT_SETF(NS,IT,NP,P,IDEF,IQ,IH,IM,ISTAT)
    IMPLICIT NONE
    INTEGER NS,IT,NP,IDEF(2),IQ,IH,IM,ISTAT
    DOUBLE PRECISION P(NP)
*NS     input   surface number (0 for new entry)
*IT     input   surface type
*NP     input   number of parameters
*P      input   array of parameters
*IDEF   input   deformation
*IQ     input   surface quality
*IH     input   hit index (-ve for next in sequence)
*IM     input   miss index (-ve for next in sequence)
*ISTAT  in/out  returned status
*-Author Dick Willingale 1996-Dec-6
```

NS=0 if you want the surface to be allocated the next free index in the sequence. IT is the surface index and determines which SRT_SUnn routine is going to be called in the ray tracing loop. Note that the parameters are held in a double precision array. IDEF and IQ are deformation and surface quality indices that have already be set by DEFORM and

SURFACE commands. If IDEF=0 no deformation will be used. If IQ=0 then the surface will act as a stop. IH and IM are used to steer the sequence in the ray tracing. They specify which surface in the sequence should be next depending on whether or not the present surface is hit or missed. In most cases IH=-1 and IM=-1. Examples of cases where a more complicated behaviour is required are SRT_PORE and QRT_SQPORE.

6. Use make to install the new libraries and scripts.

```
$ make install
```

PYTHON MODULE INDEX

INDEX

Symbols

A

B

C

D

E

F

G

H

I

K

L

M

O

P

Q

R

S

T

W

X