
specter Documentation

Release 0.10.1

DESI

Jan 13, 2023

Contents

| | | |
|----------|----------------------------|-----------|
| 1 | Introduction | 1 |
| 2 | Contents | 3 |
| 3 | Indices and tables | 33 |
| | Python Module Index | 35 |
| | Index | 37 |

CHAPTER 1

Introduction

This is the documentation for specter.

2.1 specter API

2.1.1 specter

A toolkit for simulating multi-object spectrographs.

The specter toolkit is centered around a generic PSF object, which describes how a given fiber (spectrum) at a particular wavelength is projected onto the detector CCD. The base class `specter.psf.psf.PSF` defines the interface for any PSF object. Specific instruments (*e.g.* BOSS or DESI) implement a subclass which provides:

```
xslice, yslice, pixels[ny,nx] = PSF.xypix(ispec, wavelength)
```

to describe how a given spectrum `ispec` and `wavelength` project onto the CCD.

2.1.2 specter.extract

Tools for extracting spectra from 2D images.

Notes

Initial work for adding extractions. Starting with row-by-row. This may be renamed and/or refactored into classes:

```
from specter.psf import load_psf
from specter.extract.exld import extractld

psf = load_psf('data/boos/pixpsf-r1-00140299.fits')
img = fitsio.read('data/boos/img-r1-00140299.fits', 0)
ivar = fitsio.read('data/boos/img-r1-00140299.fits', 1)

specflux, specivar = extractld(img, ivar, psf, specrange=(20,60), yrange=(1000,1100))
```

(continues on next page)

(continued from previous page)

```
%time specflux, specivar = extract1d(img, ivar, psf, yrange=(1000,1100),
↪specrange=(20,40) )
```

| rows | time |
|------|------|
| 10 | 0.32 |
| 20 | 0.59 |
| 50 | 1.75 |
| 100 | 4.94 |
| 200 | 30.7 |

0.59 * 25 * (4000/100) / 60 = 10 minutes. Not bad. Up to 15 minutes now after refactoring xsigma.

```
from specter.psf import load_psf
psf = load_psf('data/boss/pixpsf-r1-00140299.fits')
psf.xsigma(0, 7000)

### yy = N.linspace(10,4000)
yy = N.arange(10,4000,5)
ww = psf.wavelength(0, y=yy)
xsig = [psf.xsigma(0, wavelength=w) for w in ww]

plot(yy, xsig)

from specter.util import sincshift
from scipy.ndimage import center_of_mass
def xsigma(spot):
    yc, xc = center_of_mass(spot)
    xx = N.arange(spot.shape[1])
    xspot = spot.sum(axis=0)
    return N.sqrt(N.sum(xspot*(xx-xc)**2) / N.sum(xspot))
```

```
xr, yr, spot = psf.xypix(0, psf.wavelength(0, y=1000))

xx = N.arange(xr.start, xr.stop)
yy = N.arange(yr.start, yr.stop)
xspot = spot.sum(axis=0)
yspot = spot.sum(axis=1)

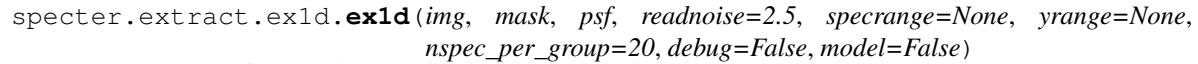
N.sum(xx*xspot) / N.sum(xspot)
N.sum(yy*yspot) / N.sum(yspot)

#- Variance
from scipy.ndimage import center_of_mass
yc, xc = center_of_mass(spot)
xx = N.arange(spot.shape[1])
yy = N.arange(spot.shape[0])
N.sum(xspot*(xx-xc)**2) / N.sum(xspot)
N.sum(yspot*(yy-yc)**2) / N.sum(yspot)
```

2.1.3 specter.extract.ex1d

1D Extraction like Horne 1986

Stephen Bailey, LBL Spring 2013

`specter.extract.ex1d.ex1d (img, mask, psf, readnoise=2.5, specrange=None, yrange=None, nspec_per_group=20, debug=False, model=False)`

Extract spectra from an image using row-by-row weighted extraction.

Parameters

- **img** (*array-like*) – CCD image.
- **mask** (*array-like*) – Image mask, 0=good, non-zero=bad
- **psf** (*object*) – PSF object.
- **readnoise** (*float, optional*) – CCD readnoise
- **specrange** (*list, optional*) – (specmin, specmax) Spectral range to extract (default all)
- **yrange** (*list, optional*) – (ymin, ymax) CCD y (row) range to extract (default all rows) ranges are python-like, *i.e.* `yrange=(0,100)` extracts 100 rows from 0 to 99 inclusive but not row 100.
- **nspec_per_group** (*int, optional*) – Extract spectra in groups of N spectra (faster if spectra are physically separated into non-overlapping groups).
- **debug** (*bool, optional*) – If True, stop with prompt after each row
- **model** (*bool, optional*) – Unknown parameter

Returns Tuple containing `spectra[nspec, ny]` the extracted spectra, `specivar[nspec, ny]` the inverse variance of spectra.

Return type `tuple`

2.1.4 specter.extract.ex2d

2D Spectroperfectionism extractions

`specter.extract.ex2d.eigen_compose (w, v, invert=False, sqr=False)`

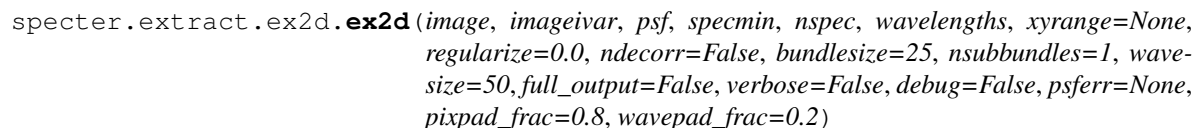
Create a matrix from its eigenvectors and eigenvalues.

Given the eigendecomposition of a matrix, recompose this into a real symmetric matrix. Optionally take the square root of the eigenvalues and / or invert the eigenvalues. The eigenvalues are regularized such that the condition number remains within machine precision for 64bit floating point values.

Parameters

- **w** (*array*) – 1D array of eigenvalues
- **v** (*array*) – 2D array of eigenvectors.
- **invert** (*bool*) – Should the eigenvalues be inverted? (False)
- **sqr** (*bool*) – Should the square root eigenvalues be used? (False)

Returns A 2D numpy array which is the recomposed matrix.

`specter.extract.ex2d.ex2d (image, imageivar, psf, specmin, nspec, wavelengths, xyrange=None, regularize=0.0, ndecorr=False, bundleize=25, nsubbundles=1, wave-size=50, full_output=False, verbose=False, debug=False, psferr=None, pixpad_frac=0.8, wavepad_frac=0.2)`

2D PSF extraction of flux from image patch given pixel inverse variance.

Parameters

- **image** (*array-like*) – 2D array of pixels
- **imageivar** (*array-like*) – 2D array of inverse variance for the image
- **psf** (*object*) – PSF object
- **specmin** (*int*) – index of first spectrum to extract
- **nspec** (*int*) – number of spectra to extract
- **wavelengths** (*array-like*) – 1D array of wavelengths to extract
- **xyrange** (*list, optional*) – (xmin, xmax, ymin, ymax): treat image as a subimage cutout of this region from the full image
- **regularize** (*float, optional*) – experimental regularization factor to minimize ringing
- **ndecorr** (*bool, optional*) – if True, decorrelate the noise between fibers, at the cost of residual signal correlations between fibers.
- **bundle_size** (*int, optional*) – extract in groups of fibers of this size, assuming no correlation with fibers outside of this bundle
- **nsubbundles** (*int, optional*) – number of overlapping subbundles to use per bundle
- **wavesize** (*int, optional*) – number of wavelength steps to include per sub-extraction
- **full_output** (*bool, optional*) – Include additional outputs based upon chi2 of model projected into pixels
- **verbose** (*bool, optional*) – print more stuff
- **debug** (*bool, optional*) – if True, enter interactive ipython session before returning
- **psferr** (*float, optional*) – fractional error on the psf model. if not None, use this fractional error on the psf model instead of the value saved in the psf fits file. This is used only to compute the chi2, not to weight pixels in fit
- **pixpad_frac** (*float, optional*) – fraction of a PSF spotsizes to pad in pixels when extracting
- **wavepad_frac** (*float, optional*) – fraction of a PSF spotsizes to pad in wavelengths when extracting

Returns

A tuple of (flux, ivar, Rdata):

- flux[nspec, nwave] = extracted resolution convolved flux
- ivar[nspec, nwave] = inverse variance of flux
- Rdata[nspec, 2*ndiag+1, nwave] = sparse Resolution matrix data

Return type `tuple`

Notes

- TODO: document output if full_output=True

- `ex2d` uses divide-and-conquer to extract many overlapping subregions and then stitches them back together. Params `wavesize` and `bundlesize` control the size of the subregions that are extracted; the necessary amount of overlap is auto-calculated based on PSF extent.

```
specter.extract.ex2d.ex2d_patch(image, ivar, psf, specmin, nspec, wavelengths, xyrange=None,
                                full_output=False,      regularize=0.0,      ndecorr=False,
                                use_cache=None)
```

2D PSF extraction of flux from image patch given pixel inverse variance.

Parameters

- **image** – 2D array of pixels
- **ivar** – 2D array of inverse variance for the image
- **psf** – PSF object
- **specmin** – index of first spectrum to extract
- **nspec** – number of spectra to extract
- **wavelengths** – 1D array of wavelengths to extract
- **xyrange** – (xmin, xmax, ymin, ymax): treat image as a subimage cutout of this region from the full image
- **full_output** – if True, return a dictionary of outputs including intermediate outputs such as the projection matrix.
- **ndecorr** – if True, decorrelate the noise between fibers, at the cost of residual signal correlations between fibers.
- **use_cache** – default behavior, can be turned off for testing purposes

Returns (flux, ivar, R): `flux[nspec, nwave]` = extracted resolution convolved flux `ivar[nspec, nwave]` = inverse variance of flux `R` : 2D resolution matrix to convert

```
specter.extract.ex2d.psfabsbias(p1, p2, wave, phot, ispec=0, readnoise=3.0)
```

Return absolute bias from extracting with PSF `p2` if the real PSF is `p1`.

Inputs: `p1, p2` : PSF objects `wave[]` : wavelengths in Angstroms `phot[]` : spectrum in photons

Optional Inputs: `ispec` : spectrum number `readnoise` : CCD read out noise (optional)

Returns bias, R `bias` array same length as `wave` `R` resolution matrix for PSF `p1`

See `psfbias()` for relative bias

```
specter.extract.ex2d.psfbias(p1, p2, wave, phot, ispec=0, readnoise=3.0)
```

Return bias from extracting with PSF `p2` if the real PSF is `p1`

Inputs: `p1, p2` : PSF objects `wave[]` : wavelengths in Angstroms `phot[]` : spectrum in photons

Optional Inputs: `ispec` : spectrum number `readnoise` : CCD read out noise (optional)

Returns `bias` array same length as `wave`

```
specter.extract.ex2d.resolution_from_icov(icov, decorr=None)
```

Function to generate the ‘resolution matrix’ in the simplest (no unrelated crosstalk) Bolton & Schlegel 2010 sense. Works on dense matrices. May not be suited for production-scale determination in a spectro extraction pipeline.

Parameters

- **icov** (*array*) – real, symmetric, 2D array containing inverse covariance.
- **decorr** (*list*) – produce a resolution matrix which decorrelates signal between fibers, at the cost of correlated noise between fibers (default). This list should contain the number of elements in each spectrum, which is used to define the size of the blocks.

Returns (R, ivar): R : resolution matrix ivar : $R C R.T$ – decorrelated resolution convolved inverse variance

`specter.extract.ex2d.split_bundle (bundlesize, n)`

Partitions a bundle into subbundles for extraction

Parameters

- **bundlesize** – (int) number of fibers in the bundle
- **n** – (int) number of subbundles to generate

Returns

(subbundles, extract_subbundles) where subbundles = list of arrays of indices belonging to each subbundle; extract_subbundles = list of arrays of indices to extract for each subbundle, including edge overlaps except for first and last fiber

NOTE: resulting partition is such that the lengths of the extract_subbundles differ by at most 1.

Example:

```
>>> split_bundle(10, 3)
([array([0, 1, 2]), array([3, 4, 5]), array([6, 7, 8, 9])],
 [array([0, 1, 2, 3]), array([2, 3, 4, 5, 6]), array([5, 6, 7, 8, 9])])
```

2.1.5 specter.io

Routines to standardize specter I/O. This may be refactored into a separate directory as this grows.

Stephen Bailey, LBL January 2013

`specter.io.read_image (filename)`

Read (image, ivar, header) from input filename

`specter.io.read_simspec (filename)`

Read an input simulated spectrum file, parse the various format options, and return a standardized spectrum dictionary for use.

Returns a dictionary with keys flux, wavelength, units, objtype

`specter.io.read_simspec_image (filename)`

Read an input simulated spectrum file formatted in multi-HDU FITS images.

Returns a dictionary with keys flux, wavelength, units, objtype

`specter.io.read_simspec_table (filename)`

Read an input simulated spectrum file formatted as a FITS binary table.

Returns a dictionary with keys flux, wavelength, units, objtype

`specter.io.write_spectra (outfile, wave, flux, ivar, resolution, header)`

Write spectra to outfile

Parameters

- **wave** – 1D[nwave] array of wavelengths

- **flux** – 2D[nspec, nwave] flux
- **ivar** – 2D[nspec, nwave] inverse variance of flux
- **resolution** – 3D[nspec, ndiag, nwave] sparse resolution matrix data
- **header** – fits header to include in output

Writes data to outfile in 4 HDUs with EXTNAME FLUX, IVAR, WAVE, RESOLUTION

2.1.6 specter.psf

`specter.psf.load_psf(filename, psftype=None)`

Load a PSF fits file, using the “PSFTYPE” header keyword to determine which specific subclass to use.

psftype : override fits header keyword PSFTYPE

2.1.7 specter.psf.gausshermite

GaussHermitePSF - PSF modeled with 2D Gauss-Hermite polynomials as generated by the specex package at <https://github.com/julienguy/specex>

Stephen Bailey December 2013

class `specter.psf.gausshermite.GaussHermitePSF(filename)`

Model PSF with two central Gauss-Hermite cores with different sigmas plus power law wings.

`_gh(x, m=0, xc=0.0, sigma=1.0)`

return Gauss-Hermite function value, NOT integrated, for display of PSF. :param x: coordinates baseline array :param m: order of Hermite polynomial multiplying Gaussian core :param xc: sub-pixel position of Gaussian centroid relative to x baseline :param sigma: sigma parameter of Gaussian core in units of pixels

`_value(x, y, ispec, wavelength)`

return PSF value (same shape as x and y), NOT integrated, for display of PSF.

Parameters

- **x** – x-coordinates baseline array
- **y** – y-coordinates baseline array (same shape as x)
- **ispec** – fiber
- **wavelength** – wavelength

`_xypix(ispec, wavelength, ispec_cache=None, iwave_cache=None)`

Two branches of this function which does legendre series fitting First branch = no caching, will recompute legval every time eval is used (slow) Second branch = yes caching, will look up precomputed values of legval (faster)

Parameters

- **ispec** – the index of each spectrum
- **wavelength** – the wavelength at which to evaluate the legendre series

Optional arguments: `ispec_cache`: the index of each spectrum which starts again at 0 for each patch
`iwave_cache`: the index of each wavelength which starts again at 0 for each patch

`cache_params(specrange, wavelengths)`

this is implemented in `specter.psf.gausshermite`, everywhere else just an empty function

xsigma (*ispec, wavelength*)

Return Gaussian sigma of PSF spot in cross-dispersion direction in CCD pixel units.

ispec : spectrum index wavelength : scalar or vector wavelength(s) to evaluate spot sigmas

ysigma (*ispec, wavelength*)

Return Gaussian sigma of PSF spot in dispersion direction in CCD pixel units.

ispec : spectrum index wavelength : scalar or vector wavelength(s) to evaluate spot sigmas

`specter.psf.gausshermite.generate_core` (*degx1, degy1, npfx, npfy, spot1, core1, c1_array*)

Funtion to speed up some of the expensive parts of `_xypix` by using numba to JIT-compile.

Parameters

- **degx1** – self._polyparams[‘GHDEGX’], gauss-hermite x degree
- **degy1** – self._polyparams[‘GHDEGY’], gauss-hermite y degree
- **npfx** – xfunc1 converted to numpy array (numba doesn’t like lists of numpy arrays)
- **npfy** – yfunc1 converted to numpy array (numba doesn’t like lists of numpy arrays)
- **spot1** – a preallocated empty 2d array that is the same size as core1
- **core1** – a 2d array that is modified in place and then returned as the final function output
- **c1_array** – the legval values for degx1, degy1, ispec_cache, and iwave_cache which cannot be fed in directly because numba will not handle dictionaries

`specter.psf.gausshermite.pgh` (*x, m=0, xc=0.0, sigma=1.0*)

Pixel-integrated (probabilist) Gauss-Hermite function. :param x: pixel-center baseline array :param m: order of Hermite polynomial multiplying Gaussian core :param xc: sub-pixel position of Gaussian centroid relative to x baseline :param sigma: sigma parameter of Gaussian core in units of pixels

Uses the relationship $\int H_k(x) \exp(-0.5 x^2) dx = -H_{k-1}(x) \exp(-0.5 x^2) + \text{const}$ Written: Adam S. Bolton, U. of Utah, fall 2010 Adapted for efficiency by S. Bailey while dropping generality modified from the orig `_pgh` to enable jit-compiling → no longer passing in self, calling custom numba functions in util

2.1.8 specter.psf.gausshermite2

GaussHermitePSF - PSF modeled with 2D Gauss-Hermite polynomials as generated by the specex package at <https://github.com/julienguy/specex>

Stephen Bailey December 2013

TODO: GaussHermitePSF (no 2) was copied and pasted from this and then modified. Can they be refactored to share code?

class `specter.psf.gausshermite2.GaussHermite2PSF` (*filename*)

Model PSF with two central Gauss-Hermite cores with different sigmas plus power law wings.

`_pgh` (*x, m=0, xc=0.0, sigma=1.0*)

Pixel-integrated (probabilist) Gauss-Hermite function.

Parameters

- **x** – pixel-center baseline array
- **m** – order of Hermite polynomial multiplying Gaussian core
- **xc** – sub-pixel position of Gaussian centroid relative to x baseline
- **sigma** – sigma parameter of Gaussian core in units of pixels

Uses the relationship $\text{Integral}\{H_k(x) \exp(-0.5 x^2) dx\} = -H_{k-1}(x) \exp(-0.5 x^2) + \text{const}$

Written: Adam S. Bolton, U. of Utah, fall 2010 Adapted for efficiency by S. Bailey while dropping generality

2.1.9 specter.psf.monospot

MonoSpotPSF - ...

class `specter.psf.monospot.MonoSpotPSF` (*filename, spot=None, scale=1.0*)

__xypix (*ispec, wavelength, ispec_cache=None, iwave_cache=None*)
Return xslice, yslice, pix for PSF at spectrum ispec, wavelength

2.1.10 specter.psf.pixpsf

Pixelated 2D PSF

David Schlegel & Stephen Bailey, Summer 2011

class `specter.psf.pixpsf.PixPSF` (*filename*)
Pixelated PSF[ny, nx] = Ic[ny, nx] + x*Ix[ny, nx] + y*Iy[ny, nx] + ...
__xypix (*ispec, wavelength, ispec_cache=None, iwave_cache=None*)
Evaluate PSF for a given spectrum and wavelength
returns xslice, yslice, pixels[yslice, xslice]

2.1.11 specter.psf.psf

Base class for 2D PSFs

Provides PSF base class which defines the interface for other code using PSFs. Subclasses implement specific models of the PSF and override/extend the `__init__` and `xypix(ispec, wavelength)` methods, while allowing interchangeable use of different PSF models through the interface defined in this base class.

Stephen Bailey, Fall 2012

class `specter.psf.psf.PSF` (*filename*)
Base class for 2D PSFs
Subclasses need to extend `__init__` to load format-specific items from the input fits file and implement `_xypix(ispec, wavelength)` to return xslice, yslice, pixels[y,x] for the PSF evaluated at spectrum ispec at the given wavelength. All interactions with PSF classes should be via the methods defined here, allowing interchangeable use of different PSF models.
_fit_spot_sigma (*ispec, axis=0, npoly=5*)
Fit the cross-sectional Gaussian sigma of PSF spots vs. wavelength. Return callable Legendre object.

Parameters

- **ispec** – spectrum number
- **axis** – 0 or ‘x’ for cross dispersion sigma; 1 or ‘y’ or ‘w’ for wavelength dispersion
- **npoly** – order of Legendre poly to fit to sigma vs. wavelength

Returns legfit such that `legfit(w)` returns fit at wavelengths w

_value (*x, y, ispec, wavelength*)
 this is implemented in specter.psf.gaussshermite and specter.psf.spotgrid, everywhere else just an empty function

_xypix (*ispec, wavelength, ispec_cache=None, iwave_cache=None*)
 Subclasses of PSF should implement this to return xslice, yslice, pixels[iy,ix] for their particular models. Don't worry about edge effects – PSF.xypix will take care of that.

angstroms_per_pixel (*ispec, wavelength*)
 Return CCD pixel width in Angstroms for spectrum ispec at given wavelength(s). Wavelength may be scalar or array.

cache_params (*spec_range, wavelengths*)
 this is implemented in specter.psf.gaussshermite, everywhere else just an empty function

pix (*ispec, wavelength*)
 Evaluate PSF for spectrum[ispec] at given wavelength
 returns 2D array pixels[iy,ix]
 also see xypix(ispec, wavelength)

project (*wavelength, phot, specmin=0, xyrange=None, verbose=False*)
 Returns 2D image or 3D images of spectra projected onto the CCD

Required inputs:
phot[nwave] or phot[nspec, nwave] or phot[nimage, nspec, nwave] as photons on CCD per bin
wavelength[nwave] or wavelength[nspec, nwave] in Angstroms if wavelength is 1D and spectra is 2D or 3D, then wavelength[] applies to all phot[i]

Optional inputs: specmin : starting spectrum number xyrange : (xmin, xmax, ymin, ymax) range of CCD pixels
 if phot is 1D or 2D, output is a single 2D[ny,nx] image if phot is 3D[nimage,nspec,nwave], output is 3D[nimage,ny,nx]

projection_matrix (*spec_range, wavelengths, xyrange, use_cache=None*)
 Returns sparse projection matrix from flux to pixels

Inputs: spec_range = (ispecmin, ispecmax) or scalar ispec wavelengths = array_like wavelengths xyrange = (xmin, xmax, ymin, ymax)

Optional inputs: use_cache= default True, legval values will be precomputed

Usage: xyrange = xmin, xmax, ymin, ymax A = psf.projection_matrix(spec_range, wavelengths, xyrange)
 nx = xmax-xmin ny = ymax-ymin img = A.dot(phot.ravel()).reshape((ny,nx))

shift_xy (*dx, dy*)
 Shift the x,y trace locations of this PSF while preserving wavelength grid: xnew = x + dx, ynew = y + dy

wavelength (*ispec=None, y=None*)
 Return wavelength of spectrum[ispec] evaluated at y.
 ispec can be None, scalar, or vector y can be None, scalar, or vector
 May return a view of the underlying array; do not modify unless specifying copy=True to get a copy of the data.

wdisp (*ispec, wavelength*)
 Return Gaussian sigma of PSF spot in wavelength-dispersion direction in units of Angstroms.
 Also see ysigma(...) which returns sigmas in units of pixels.

`ispec` : spectrum index `wavelength` : scalar or vector wavelength(s) to evaluate spot sigmas

See notes in `xsigma(...)` about caching of Legendre fit coefficients.

wmax

Maximum wavelength seen by any spectrum

wmax_all

Maximum wavelength seen by all spectra

wmin

Minimum wavelength seen by any spectrum

wmin_all

Minimum wavelength seen by all spectra

x (*ispec=None, wavelength=None*)

Return CCD X centroid of spectrum `ispec` at given wavelength(s).

`ispec` can be None, scalar, or vector `wavelength` can be None, scalar or a vector

`ispec wavelength` returns +-----+-----+----- None None array[`nspec`, `npix_y`] None scalar vector[`nspec`] None vector array[`nspec`, `nwave`] scalar None array[`npix_y`] scalar scalar scalar scalar vector vector[`nwave`] vector None array[`nspec`, `npix_y`] vector scalar vector[`nspec`] vector vector array[`nspec`, `nwave`]

xsigma (*ispec, wavelength*)

Return Gaussian sigma of PSF spot in cross-dispersion direction in CCD pixel units.

`ispec` : spectrum index `wavelength` : scalar or vector wavelength(s) to evaluate spot sigmas

The first time this is called for a spectrum, the PSF is sampled at 20 wavelengths and the variation is fit with a 5th order Legendre polynomial and the coefficients are cached. The actual value (and subsequent calls) use these cached Legendre fits to interpolate the sigma value. If this is not fast enough and/or accurate enough, PSF subtypes may override this function to provide a more accurate `xsigma` measurement.

xy (*ispec=None, wavelength=None*)

Utility function to return `self.x(...)` and `self.y(...)` in one call

xypix (*ispec, wavelength, xmin=0, xmax=None, ymin=0, ymax=None, ispec_cache=None, iwave_cache=None*)

Evaluate PSF for spectrum[`ispec`] at given wavelength

returns `xslice`, `yslice`, `pixels[iy,ix]` such that `image[yslice,xslice] += photons*pixels` adds the contribution from spectrum `ispec` at that wavelength.

if `xmin` or `ymin` are set, the slices are relative to those minima (useful for simulating subimages)

Optional inputs: `ispec_cache` = an index into the spectrum number that starts again at 0 for each patch

`iwave_cache` = an index into the wavelength number that starts again at 0 for each patch

xyrange (*spec_range, wavelengths*)

Return recommended range of pixels which cover these spectra/fluxes: (`xmin`, `xmax`, `ymin`, `ymax`)

spec_range = indices `specmin,specmax` (python style indexing), or scalar for single spectrum index

wavelengths = wavelength range `wavemin,wavemax` inclusive or sorted array of wavelengths

BUG: will fail if asking for a range where one of the spectra is completely off the CCD

y (*ispec=None, wavelength=None*)

Return CCD Y centroid of spectrum `ispec` at given wavelength(s).

`ispec` can be None, scalar, or vector `wavelength` can be scalar or a vector (but not None)

ispec wavelength returns +-----+-----+----- None scalar vector[nspec] None vector array[nspec,nwave] scalar scalar scalar scalar vector vector[nwave] vector scalar vector[nspec] vector vector array[nspec, nwave]

ysigma (*ispec, wavelength*)

Return Gaussian sigma of PSF spot in wavelength-dispersion direction in units of pixels.

Also see `wdisp(...)` which returns sigmas in units of Angstroms.

ispec : spectrum index *wavelength* : scalar or vector wavelength(s) to evaluate spot sigmas

See notes in `xsigma(...)` about caching of Legendre fit coefficients.

2.1.12 specter.psf.spotgrid

SpotGridPSF - Linear interpolate hi-res sampled spots to model PSF

Stephen Bailey Fall 2012

class `specter.psf.spotgrid.SpotGridPSF` (*filename*)

Model PSF with a linear interpolation of high resolution sampled spots

_value (*x, y, ispec, wavelength*)

return PSF value (same shape as x and y), NOT integrated, for display of PSF.

Parameters

- **x** – x-coordinates baseline array
- **y** – y-coordinates baseline array (same shape as x)
- **ispec** – fiber
- **wavelength** – wavelength

_xypix (*ispec, wavelength, ispec_cache=None, iwave_cache=None*)

Return xslice, yslice, pix for PSF at spectrum *ispec*, *wavelength*

_xypix_interp (*ispec, wavelength*)

Return xslice, yslice, pix for PSF at spectrum *ispec*, *wavelength*

`specter.psf.spotgrid.new_pixshift` (*xc, yc, pix_spot_values, rebin*)

Does a new pixel shift.

Parameters

- **yc** (*xc,*) – Center of the PSF in CCD coordinates.
- **pix_spot_values** (*array-like*) – 2D array of interpolated values.
- **rebin** (*array-like*) – Ratio of spot to CCD pixel size.

Returns The resampled and scaled 2D array of *pix_spot_values*.

Return type array-like

2.1.13 specter.throughput

A class for tracking throughput.

Hacks:

- Doesn't support spatial variation of input sources.

- Doesn't support per-fiber throughput.
- Do I really want to impose a clumsy ObjType ENUM?

How to handle fiber size and sky units?

`specter.throughput.load_throughput(filename)`

Create Throughput object from FITS file with EXTNAME=THROUGHPUT HDU

2.1.14 specter.util

2.1.15 specter.util.cachedict

Implement a dictionary that caches the last N entries and throws the rest away

class `specter.util.cachedict.CacheDict(n, d=None)`

A dictionary that keeps only the last n items added

2.1.16 specter.util.pixspline

Pixel-integrated spline utilities.

Written by A. Bolton, U. of Utah, 2010-2013.

exception `specter.util.pixspline.PixSplineError(value)`

class `specter.util.pixspline.PixelSpline(pixbound, flux)`

Pixel Spline object class.

Initialize as follows: PS = PixelSpline(pixbound, flux)

where

pixbound = array of pixel boundaries in baseline units (if same len as flux, treat as centers instead of edges)

and flux = array of specific flux values in baseline units.

Assumptions: 'pixbound' should have one more element than 'flux', and units of 'flux' are -per-unit-baseline, for the baseline units in which pixbound is expressed, averaged over the extent of each pixel.

find_extrema (*minima=False*)

Return array of extrema of underlying pixelspline

if minima is False, return only maxima

point_evaluate (*xnew, missing=0.0*)

Evaluate underlying pixel spline at array of points

Also see: PixelSpline.resample()

resample (*pb_new*)

Resample a pixelspline analytically onto a new set of pixel boundaries

`specter.util.pixspline.cen2bound(pixelcen)`

Convenience function to do the obvious thing to transform pixel centers to pixel boundaries.

2.1.17 specter.util.traceset

Handle sets of Legendre coefficients

`specter.util.traceset.fit_traces` (*x*, *yy*, *deg*=5, *domain*=None)
returns TraceSet object modeling *y*[*i*] vs. *x*

Parameters

- **x** – 1D array
- **y** – 2D array[nspec, nx]
- **deg** – optional Legendre degree

2.1.18 specter.util.util

Utility functions and classes for specter

Stephen Bailey Fall 2012

class `specter.util.util.LinearInterp2D` (*x*, *y*, *data*)
Linear interpolation on a 2D grid. Allows values to be interpolated to be multi-dimensional.

`specter.util.util._sincfunc` (*x*, *dx*, *dampfac*=3.25)
sinc helper function for sincshift()

`specter.util.util.custom_erf` (*y*)
Custom implementation of `scipy.special.erf()` to enable jit-compiling with Numba (which as of 10/2018 does not support scipy). This functionality is equivalent to:

`scipy.special.erf(y)`

with the exception that scalar values of *y* are not supported.

Parameters *y* (*array-like*) – Points at which the error function will be evaluated.

Returns The value of the error function at points in array *y*.

Return type array-like

Notes

This function has been translated from the original fortran function to Python. The original scipy erf function can be found at: <https://github.com/scipy/scipy/blob/8dba340293fe20e62e173bdf2c10ae208286692f/scipy/special/cdflib/erf.f> Note that this new function introduces a small amount of machine-precision numerical error as compared to the original scipy function.

`specter.util.util.custom_hermitenorm` (*n*, *u*)

Custom implementation of `scipy.special.hermitenorm` to enable jit-compiling with Numba (which as of 10/2018 does not support scipy). This functionality is equivalent to: `fn = scipy.special.hermitenorm(n)` return `fn(u)` with the exception that scalar values of *u* are not supported.

Inputs: *n*: the degree of the hermite polynomial *u*: (requires array) points at which the polynomial will be evaluated.

Outputs: *res*: the value of the hermite polynomial at array points(*u*)

`specter.util.util.gaussint` (*x*, *mean*=0.0, *sigma*=1.0)

Return integral from -inf to *x* of normalized Gaussian with mean and sigma

`specter.util.util.gausspix` (*x*, *mean=0.0*, *sigma=1.0*)
Return Gaussian(mean,sigma) integrated over unit width pixels centered at *x*[].

`specter.util.util.rebin_image` (*image*, *n*)
rebin 2D array pix into bins of size *n* x *n*

New binsize must be evenly divisible into original pix image

`specter.util.util.resample` (*x*, *xp*, *yp*, *xedges=False*, *xpedges=False*)
IN PROGRESS. Resample a spectrum to a new binning using PixelSpline

 $1 \leq x.ndim \leq xp.ndim \leq yp.ndim \leq 2$

`specter.util.util.sincshift` (*image*, *dx*, *dy*, *sincrad=10*, *dampfac=3.25*)
Return image shifted by *dx*, *dy* using sinc interpolation.

For speed, do each dimension independently which can introduce edge effects. Also see `sincshift2d()`.

`specter.util.util.sincshift2d` (*image*, *dx*, *dy*, *sincrad=10*, *dampfac=3.25*)
Return image shifted by *dx*, *dy* using full 2D sinc interpolation

`specter.util.util.trapz` (*edges*, *xp*, *yp*)
Perform trapezoidal integration between edges using sampled function *yp* vs. *xp*. Returns array of length `len(edges)-1`.

Input *xp* array must be sorted in ascending order.

See also `numpy.trapz`, which integrates a single array

`specter.util.util.weighted_solve` (*A*, *b*, *w*)
Solve $Ax = b$ with weights *w* on *b* Returns *x*, `inverseCovariance(x)`

2.2 specter change log

2.2.1 0.10.2 (unreleased)

- No changes yet.

2.2.2 0.10.1 (2023-01-12)

- Avoid at-edge-of-ccd extraction crash (PR #84).
- FITS writing use “overwrite” not “clobber” for astropy ≥ 5.1 (PR #86).

2.2.3 0.10.0 (2021-02-15)

- Replace biasing test >0 by $!=0$ in `psf.project`
- Tune extraction patch boundary parameters to limit edge effects (PR #82).

2.2.4 0.9.4 (2020-08-03)

- Update documentation configuration for ReadTheDocs (PR #77).
- Fix last-bin bug in pixellated Gauss-Hermite integration (PR #79).
- Astropy deprecation rename `clobber` -> `overwrite` (PR #81).

- Add reproducibility unit tests (PR #81).

2.2.5 0.9.3 (2020-04-16)

- Improve handling of heavily (or completely) masked inputs (PR #78).

2.2.6 0.9.2 (2020-04-07)

- Fix NaN flux coming from masked input pixels (PR #76).

2.2.7 0.9.1 (2018-11-07)

- Faster Pixelated Gauss-Hermite (pgh) for ~25-30% extraction speedup (PR #71 and #73).
- Faster xypix 10-50% extraction speedup (PR #74).
- Memory and operation order improvements for ~few percent speedup (PR #75).

2.2.8 0.9.0 (2018-09-26)

- Faster extractions by vectorizing and caching legval calls (PR #70).

2.2.9 0.8.7 (2018-07-26)

- Add custom *xsigma* and *ysigma* functions to GaussHermitePSF (PR #66).
- Don't use numba caching due to MPI race condition (PR #67).
- Small speed improvements (PR #68 and #69).

2.2.10 0.8.6 (2018-06-27)

- Added numba-ized legval for ~20% overall ex2d speedup (PR #61).
- Fixed tests (PR #62).
- Less regularization for ringing to lower bias (PR #63).

2.2.11 0.8.5 (2018-05-10)

- Allow user to override psferr in ex2d (PR #60)

2.2.12 0.8.4 (2018-03-29)

- np.outer replacement for 6% faster runtime (PR #58)

2.2.13 0.8.3 (2018-02-23)

- SpotGrid speedup (used by DESI pixsim); adds numba dependency (PR #56)

2.2.14 0.8.2 (2017-12-20)

- Don't require 2to3 during installations; fix license (PR #55)

2.2.15 0.8.1 (2017-10-25)

- Robust even if nsubbundles>bundlesize (PR #53)

2.2.16 0.8.0 (2017-09-29)

- Added subbundle divide-and-conquer extractions for ~2x speedup (PR #51)
- Added GaussHermite PSF format v3 (PR #52)

2.2.17 0.7.0 (2017-03-02)

- Update template Module file to reflect DESI+Anaconda infrastructure.
- Enable projecting photons onto multiple images simultaneously
- Fix GaussHermite PSF spot size and centering bugs
- New PSF function `._value` to evaluate non-pixel-integrated PSF values

2.2.18 0.6.0 (2016-08-16)

PR #40:

- Added `full_output` option to `ex2d` to get model image and metrics based upon goodness of fit
- PSFs can specify their model error with `PSFERR` header keyword; default 0.01

2.2.19 0.5.0 (2016-05-23)

- Move data files into Python package so pip can install the data files.
- Load test files in class methods to hopefully speed up tests.
- Improve Travis test support to latest standards.
- Added a documentation page for the specter API.

2.2.20 0.4.1 (2016-03-10)

- Bug fixes for small PSFs, and fixes of the fixes
- This is a release candidate for DESI Spectro Pipeline 2016a

2.2.21 0.4 (2016-03-03)

- refactored `bin/exspec` to move most functionality into `specter.extract.ex2d` API change to `ex2d()` to use `specmin,nspec` instead of `specrange=(specmin,specmax)`
- removed `desiutil` dependency

2.2.22 0.3 (2015-12-15)

- pip install support, among many changes.
- This version includes the desiutil infrastructure. This will probably be removed in the future, but for now this is needed for installation support.

2.2.23 0.2.5 (2015-04-14)

- Includes cachedict bug fix and traceset.fit_traces utility function.

2.2.24 0.2.4 (2015-02-13)

- “robot overlords”
- use scipy.linalg instead of numpy.linalg

2.2.25 0.2.3 (2015-02-05)

- more linalg stability attempts
- ivar renaming typo

2.2.26 0.2.2 (2015-02-03)

- trim by percent of median not percentile

2.2.27 0.2.1 (2015-02-02)

- Added better (?) linear algebra conditioning; dump a debug file if the linear algebra fails.

2.2.28 0.2 (2015-02-02)

- GaussHermite vs. GaussHermite2 from dev branch

2.2.29 0.1.3 (2015-01-24)

- More robust when pixels are masked
- Adds a linear algebra robustness check for when pixels are masked or when asking for wavelengths that are entirely off the CCD.

2.2.30 0.1.2 (2015-01-07)

- Fixes a bug when asking for xrange for wavelengths that are way off the CCD and the extrapolation has gone very bad.

2.2.31 0.1.1 (2015-01-06)

- Bug fix to xyrange when wavelengths are within a half a pixel of the CCD boundary.

2.2.32 0.1 (2014-12-29)

- Initial tag.

2.3 specter Data Files

2.3.1 Introduction

Several data files are bundled with the specter package. This document summarizes these files.

2.3.2 specter/data/

Sky Data Files

These files are used for building up input models and throughputs.

ZenExtinct-KPNO.fits HDU 1 has a binary table with columns `WAVELENGTH [A]` and `EXTINCTION [mag/airmass]` for KPNO. Below 5000 A the curve is from `kpnoextinct.dat`; above 5000 A is from Arjun Dey. The data were merged by Nick Mostek for `bbspecsim`.

sky-uves.fits HDU 1 has a binary table with columns `wavelength [Angstroms]` and `flux [1e-17 erg/s/cm^2/A/arcsec^2]`. This is a sky spectrum from 3400-10600 A (vacuum) stitched together from the individual spectra described and [available from ESO](#). The code that originally generated this file is in a separate product, `bbspecsim/pro/sky/uves_sky.pro`.

2.3.3 specter/test/t/

This directory contains sample data files used for unit tests.

| Filename | Format | Type | Loglam | wavedim | fluxdim |
|---------------|--------|-------|--------|---------|---------|
| spec-000.fits | image | STAR | None | 1 | 1 |
| spec-001.fits | table | STAR | None | 1 | 1 |
| spec-002.fits | image | STAR | False | 1 | 1 |
| spec-003.fits | table | STAR | False | 1 | 1 |
| spec-004.fits | image | STAR | True | 1 | 1 |
| spec-005.fits | table | STAR | True | 1 | 1 |
| spec-006.fits | image | STAR | None | 0 | 1 |
| spec-007.fits | image | STAR | False | 0 | 1 |
| spec-008.fits | image | STAR | True | 0 | 1 |
| spec-009.fits | image | STAR | None | 1 | 2 |
| spec-010.fits | image | STAR* | None | 1 | 2 |
| spec-011.fits | image | STAR | False | 1 | 2 |
| spec-012.fits | image | STAR* | False | 1 | 2 |
| spec-013.fits | image | STAR | True | 1 | 2 |

Continued on next page

Table 1 – continued from previous page

| Filename | Format | Type | Loglam | wavedim | fluxdim |
|---------------|--------|-------|--------|---------|---------|
| spec-014.fits | image | STAR* | True | 1 | 2 |
| spec-015.fits | image | STAR | None | 2 | 2 |
| spec-016.fits | table | STAR | None | 2 | 2 |
| spec-017.fits | image | STAR* | None | 2 | 2 |
| spec-018.fits | table | STAR* | None | 2 | 2 |
| spec-019.fits | image | STAR | False | 2 | 2 |
| spec-020.fits | table | STAR | False | 2 | 2 |
| spec-021.fits | image | STAR* | False | 2 | 2 |
| spec-022.fits | table | STAR* | False | 2 | 2 |
| spec-023.fits | image | STAR | True | 2 | 2 |
| spec-024.fits | table | STAR | True | 2 | 2 |
| spec-025.fits | image | STAR* | True | 2 | 2 |
| spec-026.fits | table | STAR* | True | 2 | 2 |
| spec-027.fits | image | STAR | None | 0 | 2 |
| spec-028.fits | image | STAR* | None | 0 | 2 |
| spec-029.fits | image | STAR | False | 0 | 2 |
| spec-030.fits | image | STAR* | False | 0 | 2 |
| spec-031.fits | image | STAR | True | 0 | 2 |
| spec-032.fits | image | STAR* | True | 0 | 2 |

2.4 Specter Data Model

2.4.1 Contents

Extracted Spectra Format

There are 4 HDUs with EXTNAME='FLUX', 'IVAR', 'WAVELENGTH', and 'RESOLUTION'

HDU EXTNAME='FLUX'

2D image with `flux[ispec, wavelength]` in photons per Angstrom. No fiber flat fielding or any calibrations have been applied.

Header keywords are propagated from the input image, with the following changes/additions:

```
NAXIS1 = Number of wavelength bins
NAXIS2 = Number of spectra
SPECMIN = First spectrum
SPECMAX = Last spectrum (inclusive)
NSPEC = NAXIS2 = Number of spectra
WAVEMIN = First wavelength [Angstroms]
WAVEMAX = Last wavelength [Angstroms]
SPECTER = Specter version
IN_PSF = Input spectral PSF
IN_IMG = Input image
```

HDU EXTNAME='IVAR'

Inverse variance of extracted spectra, same dimensions as FLUX. Units (photons/A)⁻².

HDU EXTNAME='WAVELENGTH'

1D array with the wavelength grid in Angstroms. NAXIS1 same as 'FLUX' and 'IVAR' HDUs.

HDU EXTNAME='RESOLUTION'

3D array storing the per-fiber band-diagonal elements of the spectroperfectionism "Resolution Matrix":

```
NAXIS1 = Number of wavelength bins
NAXIS2 = Number of diagonal bands
NAXIS3 = Number of spectra
```

The format is designed such that the following python code will create a sparse matrix representation of the resolution matrix R for spectrum i:

```
nwave = header['NAXIS1']
nband = header['NAXIS2']
offsets = range(nband//2, -nband//2, -1)
R = scipy.sparse.dia_matrix((data[i], offsets), (nwave, nwave))
```

Also see `ex2d_ResMatrix.pdf` for how this is created from the divide-and-conquer extractions.

Cross terms between fibers are not stored.

Input Spectra Format

Input spectra in FITS format are supported as either image or binary table HDUs. The wavelength grid, object type, and flux units are specified by additional columns, images, and keywords. Details below.

Input Spectra: Binary Table**HDU 0 : ignored****HDU 1 : Binary table of spectra**

Required:

- `flux[nwave]` or `flux[nspec, nwave]` column
- Wavelength grid: - header keywords `CRVAL1` and `CDEL11`, or - `wave`, `wavelength` or `loglam` column with dimensions matching `flux`

Optional:

- `objtype[nspec]` column, or header keyword `OBJTYPE`. Default 'STAR'. See below for details on `OBJTYPE`.
- `FLUXUNIT` header keyword (see below)

Other columns and HDUs may be present and will be ignored.

Input Spectra: Image HDU

HDU 0 image

- `flux[nspec, nflux]`
- `FLUXUNIT` header keyword (see below)

Wavelength grid defined by one of these:

- HDU 0 keywords `CRVAL1` and `CDEL11` with optional `LOGLAM` flag (see below)
- HDU `EXTNAME='WAVELENGTH'` image with wavelength grid in Angstroms - `wavelength[nflux]` or `wavelength[nspec, nflux]`
- HDU `EXTNAME='LOGLAM'` image with wavelength grid in \log_{10} (Angstroms) - `loglam[nflux]` or `loglam[nspec, nflux]`

Object type defined by one of these:

- HDU 0 keyword `OBJTYPE` (if all objects are the same)
- HDU `EXTNAME='TARGETINFO'` binary table - `OBJTYPE` column required - other columns are user-specific and will be ignored

Flux Units

Flux units are specified by one of the following:

- `TUNITnn` keyword for binary table `flux` column
- `FLUXUNIT` keyword in same HDU as image/table with `flux`

Options are:

- Treated as function values to be multiplied by bin width: - $\text{erg/s/cm}^2/\text{\AA}$ (default) - $\text{erg/s/cm}^2/\text{\AA}/\text{arcsec}^2$ - $\text{photon}/\text{\AA}$ - $\text{photon}/\text{\AA}/\text{arcsec}^2$
- Treated as delta functions at each given wavelength: - photons - erg/s/cm^2 - $\text{erg/s/cm}^2/\text{arcsec}^2$

For example, an astronomical object is typically in units “ $\text{erg/s/cm}^2/\text{\AA}$ ” and will be converted to photons using all throughput terms of the throughput model. A sky spectrum may be in “ $\text{erg/s/cm}^2/\text{\AA}/\text{arcsec}^2$ ” and will be multiplied by the area of the fiber instead of having a fiber input geometric loss applied.

Wavelength Grid

If the wavelength grid is not specified by a binary table column or an image HDU, it may be specified by header keywords `CRVAL1` and `CDEL11` and optionally `LOGLAM` (0/1 for linear/ \log_{10} , default=linear=0).

e.g. to specify wavelengths [3600, 3601, 3602, ...]:

| | | | | |
|---------------------|---|------|---|-----------------------------------|
| <code>CRVAL1</code> | = | 3600 | / | Reference wavelength in Angstroms |
| <code>CDEL11</code> | = | 1 | / | delta wavelength |

or to specify \log_{10} (wavelength) spacing [3.5500, 3.5501, 3.5502, ...]:

| | | | | |
|---------------------|---|--------|---|-----------------------------------|
| <code>CRVAL1</code> | = | 3.5500 | / | Reference \log_{10} (Angstroms) |
| <code>CDEL11</code> | = | 0.0001 | / | delta \log_{10} (Angstroms) |
| <code>LOGLAM</code> | = | 1 | / | \log_{10} spaced wavelengths |

The FITS standard also requires CRPIX1 and CTYPE1 but these are ignored.

Object Type

The default object type is “STAR” (astronomical point source), but other options may be given by either

- OBJTYPE header keyword, or
- objtype[nspec] column of binary table format

Options:

- SKY : Geometric losses of input fiber size are not applied
- CALIB : Calibration sources such as arc or flat lamps, or tuneable laser systems. Atmospheric extinction is not applied, nor are geometric losses of input fiber size.
- STAR or anything else: treated as an astronomical object with all sources of throughput loss applied.

Output Image Format

HDU 0 CCDIMAGE

Image of spectra projected onto the CCD with the PSF, with optional noise. The readout noise is always Gaussian, but the photon noise could be Poisson (default) or Gaussian (if `-gaussnoise` option was used.)

Header keywords:

- SIMDATA = True
- PREPROC = True
- GAIN = CCD gain in electrons/ADU
- RDNOISE = CCD amplifier readout noise in electrons
- SIMNOISE = “Gaussian”, “Poisson”, or “None”

HDU 1 IVAR

If noise was added to the CCD, this contains the pixel inverse variance if the noise was treated as Gaussian (even if the photon shot noise is Poisson).

HDU 2 PHOTONS

NOTE : The format of this HDU will likely change to match the format of input spectra (flux plus FLUXUNIT keyword instead of PHOTONS). This has not yet been implemented.

Optional HDU with binary table giving the photon spectra projected onto the CCD after all throughputs were applied. Extraction code should produce this before any calibrations. There is one row per spectrum, with columns:

- PHOTONS[nwave]
- WAVELENGTH[nwave]

HDU 3 XYWAVE

Optional HDU with x,y vs. wavelength of spectral traces on CCD. There is one row per spectrum, with columns:

- X[nwave]
- Y[nwave]
- WAVELENGTH[nwave]

PSF Formats

Base PSF

NOTE: we are in the process of switching the format of how x and y vs. wavelength are stored. This will create incompatibilities between old and new PSFs.

HDUs 0 and 1 contain the PSF centroid x and y location vs. wavelength stored as Legendre polynomial coefficients:

```
HDU 0 : XCOEFF[nspec, ncoeff]
HDU 1 : YCOEFF[nspec, ncoeff]
```

The header contains keywords WAVEMIN, WAVEMAX to define the wavelength range in Angstroms to map to domain [-1,1] for evaluating the Legendre polynomials. *e.g.* the x position for spectrum i at wavelength:

```
w = 2.0*(w-WAVEMIN)/(WAVEMAX-WAVEMIN) - 1.0  #- Map to [-1,1]
x[i] = Sum_j XCOEFF[i,j] L_j(w)                #- evaluate Legendre series
```

Or in Python:

```
import fitsio
from numpy.polynomial.legendre import legval
h = fitsio.read_header('psf.fits', 'XCOEFF')
xcoeff = fitsio.read('psf.fits', 'XCOEFF')
w = 2.0 * (wavelength-h['WAVEMIN']) / (h['WAVEMAX'] - h['WAVEMIN']) - 1.0
x = legval(w, xcoeff[i])
```

Additional HDUs contain data specific to various parameterizations of the PSF as listed below. Any additional extensions should be read by EXTNAME, not by number – the order and number of other extensions is arbitrary.

HDU 0 keywords:

- NPIX_X, NPIX_Y : CCD dimensions in pixels

(x,y) coordinates are *centered* on each pixel and start counting at 0. i.e. the lower left corner of the CCD is (-0.5, -0.5) and the center of the lower left pixel is (0,0).

Optional HDU EXTNAME=THROUGHPUT in the format as described in throughput.md . *i.e.* the throughput may be kept in a separate FITS file, or bundled with the instrument PSF for convenience.

If throughput isn't available, the PSF can still be used to project photons onto a CCD, but not flux in erg/s/cm²/Å .

Old format for x and y

The old base PSF format parameterized x and y vs. wavelength as:

```

HDU 0 : x[nspec, nwave]      EXTNAME="X"
HDU 1 : y[nspec, nwave]      EXTNAME="Y"
HDU 2 : wavelength[nspec, nwave] EXTNAME="WAVELENGTH", or
      loglam[nspec, nwave]   EXTNAME="LOGLAM"
HDU 3+ : specific to each subformat

```

We may revive this format.

Spot Grid PSF

PSFTYPE = "SPOTGRID"

This PSF type provides an x,y grid of spots to be interpolated. The first interpolation dimension must monotonically increase with spectrum number, *e.g.* the position of a fiber along a slit head. The second interpolation dimension is wavelength.

HDU 0-2 : Same as Base PSF: X, Y, wavelength or loglam of traces:

```

HDU SPOTS : spot[i, j, iy, ix]    #- 2D PSF spots
      NAXIS1 = i = number of spot samples in the spectrum number dimension
      NAXIS2 = j = number of spot samples in the wavelength direction
      NAXIS3 = iy = size of spot in the CCD y dimension
      NAXIS4 = ix = size of spot in the CCD x dimension

HDU SPOTX : spotx[NAXIS1, NAXIS2]  #- CCD X pixel loc of spot centroid
HDU SPOTY : spoty[NAXIS1, NAXIS2]  #- CCD Y pixel loc of spot centroid
HDU FIBERPOS : fiberpos[nspec]     #- Slit position of each fiber
HDU SPOTPOS : spotpos[NAXIS1]      #- Slit positions where spots are sampled
HDU SPOTWAVE : spotwave[NAXIS2]    #- Wavelengths where spots are sampled

```

spot[i, j] is a 2D PSF spot sampled at slit position spotpos[i] and wavelength spotwave[j]. Its center is located on the CCD at spotx[i, j], spoty[i, j].

Pixellated PSF PCA expansion

PSFTYPE = "PCA-PIX"

This format is a PCA-like pixelated model such that:

```
pix = ConstImage + x*ImageX + y*ImageY + x*y*ImageXY + ...
```

HDU 0-2 : Same as Base PSF: X, Y, wavelength or loglam of traces

HDU 3 : table with x and y exponents to use for each image model:

```

Columns IMODEL, XEXP, YEXP
One row per model image

```

HDU 4 : table with x and y scale factors:

```

Columns IGROUP, X0, XSCALE, Y0, YSCALE
One row per fiber

```

The model images are separately calculated for groups of fibers (could be bundles, but doesn't have to be). Within each group, the x and y dimensions are rescaled such that:

```
x = xscale * (xpix - x0)
y = yscale * (ypix - y0)
```

HDU 5 : 4-D array with the PSF images:

```
Array[IGROUP, IMODEL, IY, IX]
```

e.g. to find the PSF for ifiber iflux:

```
xpix = HDU0[ifiber, iflux]
ypix = HDU1[ifiber, iflux]
x0 = HDU4.X0[ifiber]
y0 = HDU4.Y0[ifiber]
xscale = HDU4.XSCALE[ifiber]
yscale = HDU4.YSCALE[ifiber]
x = xscale*(xpix - x0)
y = yscale*(ypix - y0)

igroup = HDU4.IGROUP[ifiber]
psf = [BlankImage]
for imodel in range( len(HDU3) ):
    xexp = HDU3.XEXP[imodel]
    yexp = HDU3.YEXP[imodel]
    psf += x^xexp * y^yexp * HDU5[igroup, imodel]
```

Gauss Hermite PSF

This PSF format is generated by the [specex PSF package](#). All parameters are stored in HDU 1 as Legendre coefficients describing how each parameter varies with wavelength for each fiber.

The PSF is modeled by two Gauss-Hermite cores with different sigmas, plus a semi-Lorentzian power-law tail.

Header Keywords for HDU 1

| Parameter | Description |
|-----------|--|
| PSFTYPE | Must be 'GAUSS-HERMITE2' |
| PSFVER | Must be '1 ' |
| NPIX_X | number of columns in input CCD image |
| NPIX_Y | number of rows in input CCD image |
| HSIZEX | Half size of PSF in fit, NX=2*HSIZEX+1 |
| HSIZEY | Half size of PSF in fit, NY=2*HSIZEY+1 |
| BUNDLMIN | first bundle of fibers (starting at 0) |
| BUNDLMAX | last bundle of fibers (included) |
| FIBERMIN | first fiber (starting at 0) |
| FIBERMAX | last fiber (included) |
| NPARAMS | number of PSF parameters |
| LEGDEG | degree of Legendre pol.(wave) for parameters |
| GHDEGX | degree of Hermite polynomial along CCD columns |
| GHDEGY | degree of Hermite polynomial along CCD rows |
| GHDEGX2 | degree of Hermite polynomial along CCD columns |
| GHDEGY2 | degree of Hermite polynomial along CCD rows |

Binary table in HDU 1

| Column | Description |
|---------|---|
| PARAM | Parameter name |
| WAVEMIN | Minimum wavelength |
| WAVEMAX | Maximum wavelength |
| COEFF | Legendre coefficients. Each row is size [NSPEC, LEGDEG+1] |

Each row contains the coefficients for the parameter name in the PARAM column. WAVEMIN and WAVEMAX define the wavelength extent in Angstroms that should be mapped to [-1,1] for evaluating the Legendre polynomials. The coefficients themselves are in the rows of the COEFF column. Each row of COEFF is a 2D array of shape [NSPEC, LEGDEG+1]

The parameters named in the PARAM column are:

| Parameter | Description |
|-----------|--|
| X | CCD column coordinate (as a function of fiber and wavelength) |
| Y | CCD row coordinate (as a function of fiber and wavelength) (X,Y)=(0,0) means that PSF is centered on center of first pixel |
| GH-SIGX | Sigma of first Gaussian along CCD columns for PSF core |
| GH-SIGY | Sigma of first Gaussian along CCD rows for PSF core |
| GHNSIG | NxSigma cutoff for first Gaussian-Hermite core term |
| GH-SIGX2 | Sigma of second Gaussian along CCD columns for PSF wings |
| GH-SIGY2 | Sigma of second Gaussian along CCD rows for PSF wings |
| GH-i-j | Hermite pol. coefficients, i along columns, j along rows, i is integer from 0 to GHDEGX, j is integer from 0 to GHDEGY, there are (GHDEGX+1)*(GHDEGY+1) such coefficients. |
| GH2-i-j | Hermite pol. coefficients for sec. GH psf, i=columns, j=rows |
| TAIL-AMP | Amplitude of PSF tail |
| TAIL-CORE | Size in pixels of PSF tail saturation in PSF core |
| TAILXS | Scaling apply to CCD coordinate along columns for PSF tail |
| TAILYS | Scaling apply to CCD coordinate along rows for PSF tail |
| TAILINDEX | Asymptotic power law index of PSF tail |
| CONT | Continuum flux in arc image (not part of PSF) |

These are evaluated as:

```
PSF_core1(X,Y) = SUM_ij (GH-i-j)*HERM(i,X/GHSIGX)*HERM(j,Y/GHSIGY)
                * GAUS(X,GHSIGX)*GAUS(Y,GHSIGY)
--> Evaluated only for pixels where (X/GHSIGX)^2 + (Y/GHSIGY)^2 < GHNSIG^2

PSF_core2(X,Y) = SUM_ij (GH2-i-j)*HERM(i,X/GHSIGX2)*HERM(j,Y/GHSIGY2)
```

(continues on next page)

(continued from previous page)

```

* GAUS (X,GHSIGX2)*GAUS (Y,GHSIGY2)

PSF_tail(X,Y) = TAILAMP*R^2/(TAILCORE^2+R^2)^(1+TAILINDE/2)
               with R^2=(X*TAILXSCA)^2+(Y*TAILYSCA)^2

```

PSF_core is *integrated* in each pixel.

PSF_tail is not integrated, it is *evaluated* at the center of each pixel.

For example, if PARAM[0] = 'X', this means that row 0 contains the Legendre coefficients for the X centroids of the PSFs on the CCD.:

```

# Map wavelength -> [-1,1] w = 2.0*(wavelength - WAVEMIN[0]) / (WAVEMAX[0]-WAVEMIN[0]) - 1.0

# Evaluate the CCD X centroid for Fiber ispec # x = Sum_i COEFF[ispec,i] * L_i(w) from
numpy.polynomial.legendre import legval xcoeff = COEFF[0] #- since PARAM[0] == 'X' x = legval(w,
xcoeff[ispec])

```

Gauss Hermite PSF (Deprecated)

NOTE : This format is implemented but somewhat cumbersome. We may abandon it for a simpler format.

This format models the PSF as 2D Gauss-Hermite polynomials. The coefficients of the polynomials are modeled as smoothly varying in (x,y) using 2D Legendre polynomials. Spectra grouped in bundles with a smoothly varying solution within the bundle but independent of other bundles.

HDU 0-2 : Same as Base PSF: X, Y, wavelength or loglam of traces:

```

PSFTYPE = "PCA-PIX"
PSFVERS = "2.0" or above

```

HDU 3+ : One HDU per bundle of spectra on the CCD

Header Keywords for HDUs 3+

| Keyword | Meaning |
|--------------------|---|
| GHSIGX, GHSIGY | Gauss-Hermite sigma in x and y directions [pixel units] |
| GHDEGX, GHDEGY | Gauss-Hermite degree in x and y directions |
| FIBERMIN, FIBERMAX | Fibers covered by this bundle min to max inclusive, 0-indexed |
| LXMIN, LXMAX | X-coordinate min/max to transform CCD x -> [-1,1] range for Legendre polynomial |
| LYMIN, LYMAX | Y-coordinate min/max to transform CCD y -> [-1,1] range for Legendre polynomial |
| LDEGX, LDEGY | Degree of Legendre polynomial in x and y directions |

Data in HDU 3+

A 4D image *coeff[GHDEGY+1, GHDEGX+1, LDEGY+1, LDEGX+1]* for that bundle.

Example

Find the PSF for fiber 5 at wavelength 6000 Angstroms:

- Map fiber 5, wavelength 6000 Å -> (x,y) on the CCD using HDUs 0-2:
 - `x = numpy.interp(6000, WAVELENGTH[5], X[5])`
 - `y = numpy.interp(6000, WAVELENGTH[5], Y[5])`
- Find which bundle fiber 5 is included in (probably bundle 0 in HDU 3)
 - Convert x,y -> to ranges [-1,1]
 - * `xx = 2*(x-LXMIN)/(LXMAX - LXMIN) - 1`
 - * `yy = 2*(y-LYMIN)/(LYMAX - LYMIN) - 1`
- The Gauss-Hermite coefficient $c_{ij} = \text{Sum_kl data}[i,j,k,l] L_k(yy) L_l(xx)$ where L_k is the kth order Legendre Polynomial
- $\text{PSF}(dx, dy) = \text{Sum_ij } c_{ij} H_i(dy/GHSIGY) H_j(dx/GHSIGX)$
- Then integrate $\text{PSF}(dx,dy)$ over the individual pixels
 - In practice it is better to directly integrate the functions

Notes

This is different from the original Gauss-Hermite format used by bbspec. These may be distinguished by the existence of `PSFVERS >= 2.0`

Future versions of this format may also include additional HDUs to model the wings of the PSF and the covariance of the coefficients.

Other PSF Formats

Specter grew out of “bbspec” which includes PSF formats for:

- 2D rotated asymmetric Gaussian
- An older format for Gauss-Hermite polynomials

These are structurally compatible with Specter PSFs but haven’t been ported yet.

Throughput Format

HDU EXTNAME=THROUGHPUT

binary table with columns:

- `wavelength[nwave]` Wavelength in Angstroms, or *loglam[nwave]* for $\log_{10}(\text{Angstroms})$
- `extinction[nwave]` Atmospheric extinction in mags/airmass; $\text{Atm throughput} = 10^{(0.4 * \text{extinction} * \text{airmass})}$
- `throughput[nwave]` Telescope, fibers, spectrographs, and CCDs
- `fiberinput[nwave]` Geometric loss at input of fiber for point source.

Required keywords in same HDU EXTNAME=THROUGHPUT (not HDU 0)

- `EXPTIME`: Standard exposure time in seconds
- `EFFAREA`: Effective area of mirror, including obscuration effects, in cm^2

- FIBERDIA: Fiber diameter in arcsec

Note: The throughput format currently does not support per-fiber throughput.

Note: `fiberinput` in general depends upon source size, seeing, and fiber position misalignments. The default is for perfectly centered point sources. The format currently does not support extended objects (the user can mimic this though the input spectra provided to Specter.)

The various forms of throughput affect different types of sources:

| | OBJECT | SKY | CALIB |
|------------|--------|-----|-------|
| EXTINCTION | yes | yes | no |
| FIBERINPUT | yes | no | no |
| THROUGHPUT | yes | yes | yes |

CALIB = calibration lamps or laser systems mounted with the telescope, *i.e.* not experiencing sky absorption.

CHAPTER 3

Indices and tables

- `genindex`
- `modindex`
- `search`

S

- `specter`, [3](#)
- `specter.extract`, [3](#)
- `specter.extract.ex1d`, [4](#)
- `specter.extract.ex2d`, [5](#)
- `specter.io`, [8](#)
- `specter.psf`, [9](#)
- `specter.psf.gausshermite`, [9](#)
- `specter.psf.gausshermite2`, [10](#)
- `specter.psf.monospot`, [11](#)
- `specter.psf.pixpsf`, [11](#)
- `specter.psf.psf`, [11](#)
- `specter.psf.spotgrid`, [14](#)
- `specter.throughput`, [14](#)
- `specter.util`, [15](#)
- `specter.util.cachedict`, [15](#)
- `specter.util.pixspline`, [15](#)
- `specter.util.traceset`, [15](#)
- `specter.util.util`, [16](#)

Symbols

`_fit_spot_sigma()` (*specter.psf.psf.PSF method*), 11

`_gh()` (*specter.psf.gausshermite.GaussHermitePSF method*), 9

`_pgh()` (*specter.psf.gausshermite2.GaussHermite2PSF method*), 10

`_sincfunc()` (*in module specter.util.util*), 16

`_value()` (*specter.psf.gausshermite.GaussHermitePSF method*), 9

`_value()` (*specter.psf.psf.PSF method*), 11

`_value()` (*specter.psf.spotgrid.SpotGridPSF method*), 14

`_xypix()` (*specter.psf.gausshermite.GaussHermitePSF method*), 9

`_xypix()` (*specter.psf.monospot.MonoSpotPSF method*), 11

`_xypix()` (*specter.psf.pixpsf.PixPSF method*), 11

`_xypix()` (*specter.psf.psf.PSF method*), 12

`_xypix()` (*specter.psf.spotgrid.SpotGridPSF method*), 14

`_xypix_interp()` (*specter.psf.spotgrid.SpotGridPSF method*), 14

A

`angstroms_per_pixel()` (*specter.psf.psf.PSF method*), 12

C

`cache_params()` (*specter.psf.gausshermite.GaussHermitePSF method*), 9

`cache_params()` (*specter.psf.psf.PSF method*), 12

`CacheDict` (*class in specter.util.cachedict*), 15

`cen2bound()` (*in module specter.util.pixspline*), 15

`custom_erf()` (*in module specter.util.util*), 16

`custom_hermitenorm()` (*in module specter.util.util*), 16

E

`eigen_compose()` (*in module specter.extract.ex2d*), 5

`ex1d()` (*in module specter.extract.ex1d*), 5

`ex2d()` (*in module specter.extract.ex2d*), 5

`ex2d_patch()` (*in module specter.extract.ex2d*), 7

F

`find_extrema()` (*specter.util.pixspline.PixelSpline method*), 15

`fit_traces()` (*in module specter.util.traceset*), 16

G

`GaussHermite2PSF` (*class in specter.psf.gausshermite2*), 10

`GaussHermitePSF` (*class in specter.psf.gausshermite*), 9

`gaussint()` (*in module specter.util.util*), 16

`gausspix()` (*in module specter.util.util*), 16

`generate_core()` (*in module specter.psf.gausshermite*), 10

L

`LinearInterp2D` (*class in specter.util.util*), 16

`load_psf()` (*in module specter.psf*), 9

`load_throughput()` (*in module specter.throughput*), 15

M

`MonoSpotPSF` (*class in specter.psf.monospot*), 11

N

`new_pixshift()` (*in module specter.psf.spotgrid*), 14

P

`pgh()` (*in module specter.psf.gausshermite*), 10

`pix()` (*specter.psf.psf.PSF method*), 12

`PixelSpline` (*class in specter.util.pixspline*), 15

`PixPSF` (*class in specter.psf.pixpsf*), 11

`PixSplineError`, 15

`point_evaluate()` (*specter.util.pixspline.PixelSpline method*), 15

`project()` (*specter.psf.psf.PSF method*), 12
`projection_matrix()` (*specter.psf.psf.PSF method*), 12
`PSF` (*class in specter.psf.psf*), 11
`psfabsbias()` (*in module specter.extract.ex2d*), 7
`psfbias()` (*in module specter.extract.ex2d*), 7

R

`read_image()` (*in module specter.io*), 8
`read_simspec()` (*in module specter.io*), 8
`read_simspec_image()` (*in module specter.io*), 8
`read_simspec_table()` (*in module specter.io*), 8
`rebin_image()` (*in module specter.util.util*), 17
`resample()` (*in module specter.util.util*), 17
`resample()` (*specter.util.pixspline.PixelSpline method*), 15
`resolution_from_icov()` (*in module specter.extract.ex2d*), 7

S

`shift_xy()` (*specter.psf.psf.PSF method*), 12
`sincshift()` (*in module specter.util.util*), 17
`sincshift2d()` (*in module specter.util.util*), 17
`specter` (*module*), 3
`specter.extract` (*module*), 3
`specter.extract.ex1d` (*module*), 4
`specter.extract.ex2d` (*module*), 5
`specter.io` (*module*), 8
`specter.psf` (*module*), 9
`specter.psf.gausshermite` (*module*), 9
`specter.psf.gausshermite2` (*module*), 10
`specter.psf.monospot` (*module*), 11
`specter.psf.pixpsf` (*module*), 11
`specter.psf.psf` (*module*), 11
`specter.psf.spotgrid` (*module*), 14
`specter.throughput` (*module*), 14
`specter.util` (*module*), 15
`specter.util.cachedict` (*module*), 15
`specter.util.pixspline` (*module*), 15
`specter.util.traceset` (*module*), 15
`specter.util.util` (*module*), 16
`split_bundle()` (*in module specter.extract.ex2d*), 8
`SpotGridPSF` (*class in specter.psf.spotgrid*), 14

T

`trapz()` (*in module specter.util.util*), 17

W

`wavelength()` (*specter.psf.psf.PSF method*), 12
`wdisp()` (*specter.psf.psf.PSF method*), 12
`weighted_solve()` (*in module specter.util.util*), 17
`wmax` (*specter.psf.psf.PSF attribute*), 13
`wmax_all` (*specter.psf.psf.PSF attribute*), 13

`wmin` (*specter.psf.psf.PSF attribute*), 13
`wmin_all` (*specter.psf.psf.PSF attribute*), 13
`write_spectra()` (*in module specter.io*), 8

X

`x()` (*specter.psf.psf.PSF method*), 13
`xsigma()` (*specter.psf.gausshermite.GaussHermitePSF method*), 9
`xsigma()` (*specter.psf.psf.PSF method*), 13
`xy()` (*specter.psf.psf.PSF method*), 13
`xypix()` (*specter.psf.psf.PSF method*), 13
`xysrange()` (*specter.psf.psf.PSF method*), 13

Y

`y()` (*specter.psf.psf.PSF method*), 13
`ysigma()` (*specter.psf.gausshermite.GaussHermitePSF method*), 10
`ysigma()` (*specter.psf.psf.PSF method*), 14