crispy Documentation

Release 0.1.0

Maxime J. Rizzo

Mar 08, 2017

This software is designed to simulate lenslet array-based Integral Field Spectrographs and their reduction process. This was developed within the context of NASA's WFIRST Coronagraph mission.

Part I

IFS Simulator

CHAPTER

ONE

INTRODUCTION TO CRISPY

This software is designed to simulate lenslet array-based Integral Field Spectrographs and their reduction process. This was developed within the context of NASA's WFIRST Coronagraph mission.

In this notebook we go straight to the point and illustrate how to use the code. We require astropy, numpy. This notebook was written in a Python 3.5 kernel but should be backward compatible with Python 2.7.

1.1 Create a polychromatic flatfield

1.1.1 Initialization

First we need to load the various modules that we need. Since the notebook can be located anywhere, we need to add the location of the Python code to the path first.

```
In [1]: import sys
        codefolder = '../../code'
        if codefolder not in sys.path: sys.path.append(codefolder)
        import tools
        import glob
        import numpy as np
        from tools.initLogger import getLogger
        log = getLogger('main')
```

All the IFS parameters are contained within a Python class called 'Params'. This class is initialized using only the path the 'code' directory from the repo. In particular, the Params class stores all the relevant paths. All parameters can be changed on the fly.

```
In [2]: from params import Params
        par = Params(codefolder)
```

All the parameters are stored in this class. During the various steps of the software, they are appended to a header file.

```
In [3]: par.hdr
```

```
Out[3]: SIMPLE  =                    T / conforms to FITS standard
        BITPIX  =                    8 / array data type
        NAXIS   =                    0 / number of array dimensions
        EXTEND  =                    T
        COMMENT
        COMMENT ************************************************************
        COMMENT ********************** General parameters ******************
        COMMENT ************************************************************
        COMMENT
        NLENS   =                  108 / # lenslets across array
```

```
PITCH   =             0.000174 / Lenslet pitch (meters)
INTERLAC=                    2 / Interlacing
PHILENS =    26.56505117707799 / Rotation angle of the lenslets (deg)
PIXSIZE =              1.3E-05 / Pixel size (meters)
LENSAMP =                  0.5 / Lenslet sampling (lam/D)
LSAMPWAV=                600.0 / Lenslet sampling wavelength (nm)
FWHM    =                    2 / FHWM of PSFLet at detector (pixels)
FWHMLAM =                660.0 / Wavelength at which FWHM is defined (nm)
NPIX    =                 1024 / Number of detector pixels
DISPDIST=                    F / Use PISCES distortion/dispersion?
```

1.1.2 Create the flatfield

Creating a flatfield is a function built into the unitTests module. In this case, we will construct a cube of nlam wavelength slices, each having 512x512 pixels of value 1. The standard

```
In [5]: from unitTests import testCreateFlatfield
        help(testCreateFlatfield)
```

```
Help on function testCreateFlatfield in module unitTests:

testCreateFlatfield(par, pixsize=0.1, npix=512, pixval=1.0, outname='flatfield.fits')
    Creates a polychromatic flatfield

    Parameters
    ----------
    par :   Parameter instance
        Contains all IFS parameters
    pixsize:   float
       Pixel scale (lam/D)
    npix: int
        Each input frame has a pixel size npix x npix
    pixval: float
        Each input frame has a unform value pixval
```

This test function will create a polychromatic flatfield at the wavelengths provided by the existing wavelength calibration. Those wavelengths can be retrieved as follows.

```
In [6]: from tools.reduction import calculateWaveList
        help(calculateWaveList)
        lam_midpts,lam_endpts = calculateWaveList(par)
        print(lam_midpts)
```

```
Help on function calculateWaveList in module tools.reduction:

calculateWaveList(par, lam_list=None)
    Computes the wavelength lists corresponding to the center and endpoints of each
    spectral bin. Wavelengths are separated by a constant value in log space. Number of
    wavelengths depends on spectral resolution.

    Parameters
    ----------
    par:        Parameter instance
            Contains all IFS parameters
    lam_list:   list of wavelengths
```

```
            Usually this is left to None. If so, we use the wavelengths used for wavelengt
            calibration. Otherwise, we could decide to focus on a smaller/larger region of
            the spectrum to retrieve. The final processed cubes will have bins centered
            on lam_midpts

    Returns
    -------
    lam_midpts: list of floats
            Wavelengths at the midpoint of each bin
    lam_endpts: list of floats
            Wavelengths at the edges of each bin

crispy - INFO - Reduced cube will have 17 wavelength bins
[ 704.42954343  711.94828852  719.54728511  727.22738976  734.9894682
  742.83439535  750.76305552  758.77634242  766.87515933  775.06041915
  783.33304453  791.69396797  800.14413192  808.68448889  817.31600156
  826.03964288  834.85639619]
```

Now let's create our flatfield. By default it will be stored as par.unitTestsOutput/flatfield.fits

```
In [7]: testCreateFlatfield(par)
```

```
crispy - INFO - Reduced cube will have 17 wavelength bins
crispy - INFO - The number of input pixels per lenslet is 6.390626
crispy - INFO - Using PSFlet gaussian approximation
crispy - INFO - Final detector pixel per PSFLet: 40.000000
crispy - INFO - Processing wavelength 0.704430 (0 out of 17)
crispy - INFO - Processing wavelength 0.711948 (1 out of 17)
crispy - INFO - Processing wavelength 0.719547 (2 out of 17)
crispy - INFO - Processing wavelength 0.727227 (3 out of 17)
crispy - INFO - Processing wavelength 0.734989 (4 out of 17)
crispy - INFO - Processing wavelength 0.742834 (5 out of 17)
crispy - INFO - Processing wavelength 0.750763 (6 out of 17)
crispy - INFO - Processing wavelength 0.758776 (7 out of 17)
crispy - INFO - Processing wavelength 0.766875 (8 out of 17)
crispy - INFO - Processing wavelength 0.775060 (9 out of 17)
crispy - INFO - Processing wavelength 0.783333 (10 out of 17)
crispy - INFO - Processing wavelength 0.791694 (11 out of 17)
crispy - INFO - Processing wavelength 0.800144 (12 out of 17)
crispy - INFO - Processing wavelength 0.808684 (13 out of 17)
crispy - INFO - Processing wavelength 0.817316 (14 out of 17)
crispy - INFO - Processing wavelength 0.826040 (15 out of 17)
crispy - INFO - Processing wavelength 0.834856 (16 out of 17)
crispy - INFO - Number of detector pixels per lenslet: 13.384615
crispy - INFO - Rebinning final detector. Image has dimensions 1024x1024
crispy - INFO - Done.
crispy - INFO - Performance: 25 seconds total
main - INFO - Writing data to ../../code/unitTestsOutputs/flatfield.fits
```

1.1.3 Display results

Depending on your version of Python you might see a bunch of VisibleDeprecationWarning. We are working on fixing that. Let's see what we got.

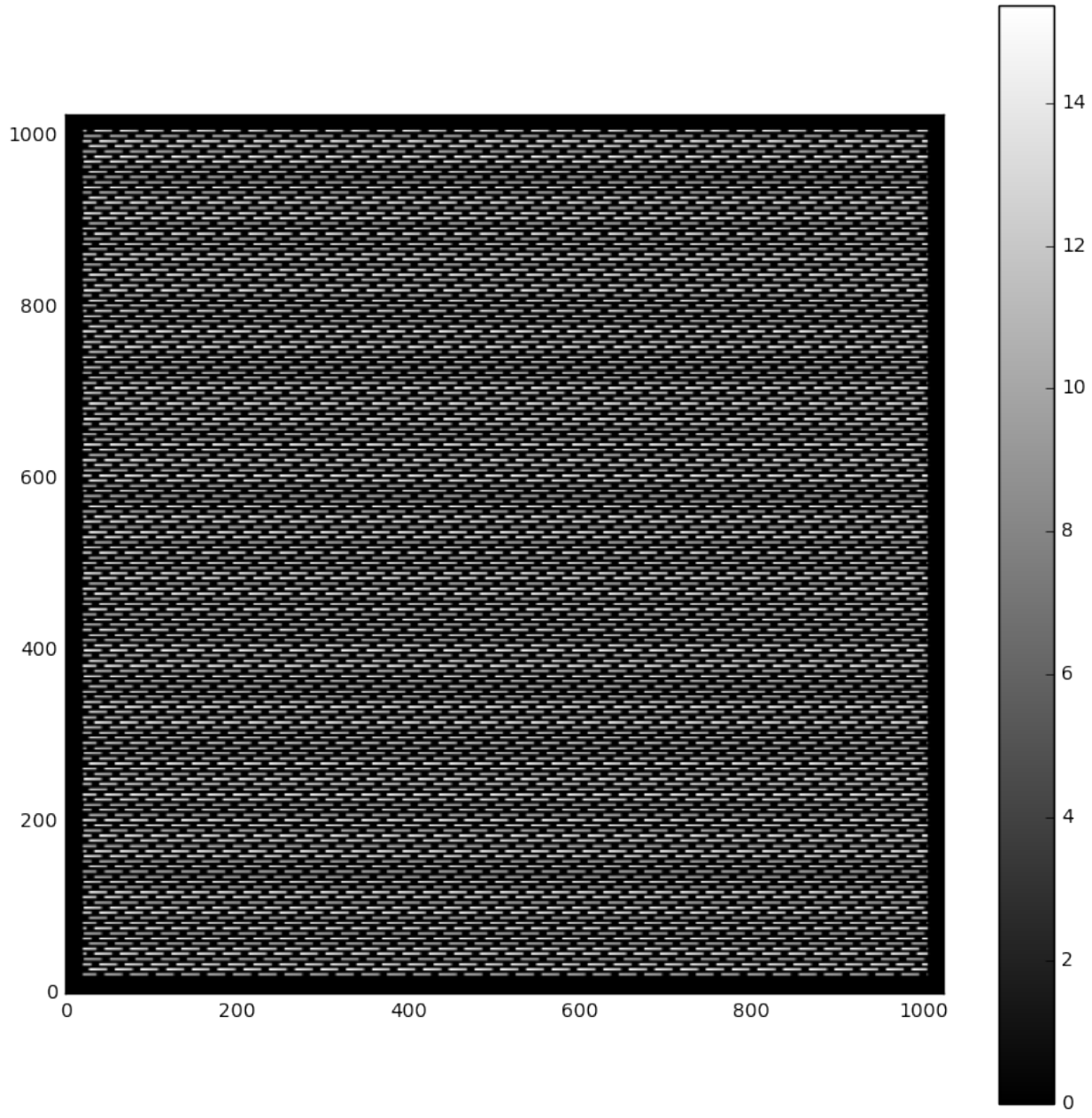
```
In [8]: %pylab inline --no-import-all
        matplotlib.rcParams['image.origin'] = 'lower'
        matplotlib.rcParams['image.interpolation'] = 'nearest'
```

```
Populating the interactive namespace from numpy and matplotlib
```

```
In [9]: from tools.image import Image
        plt.figure(figsize=(10,10))
        plt.imshow(Image(par.unitTestsOutputs+'/flatfield.fits').data,cmap='gray')
        plt.colorbar()
```

```
main - INFO - Read data from HDU 1 of ../../code/unitTestsOutputs/flatfield.fits
```

```
Out[9]: <matplotlib.colorbar.Colorbar at 0x105f1ab38>
```

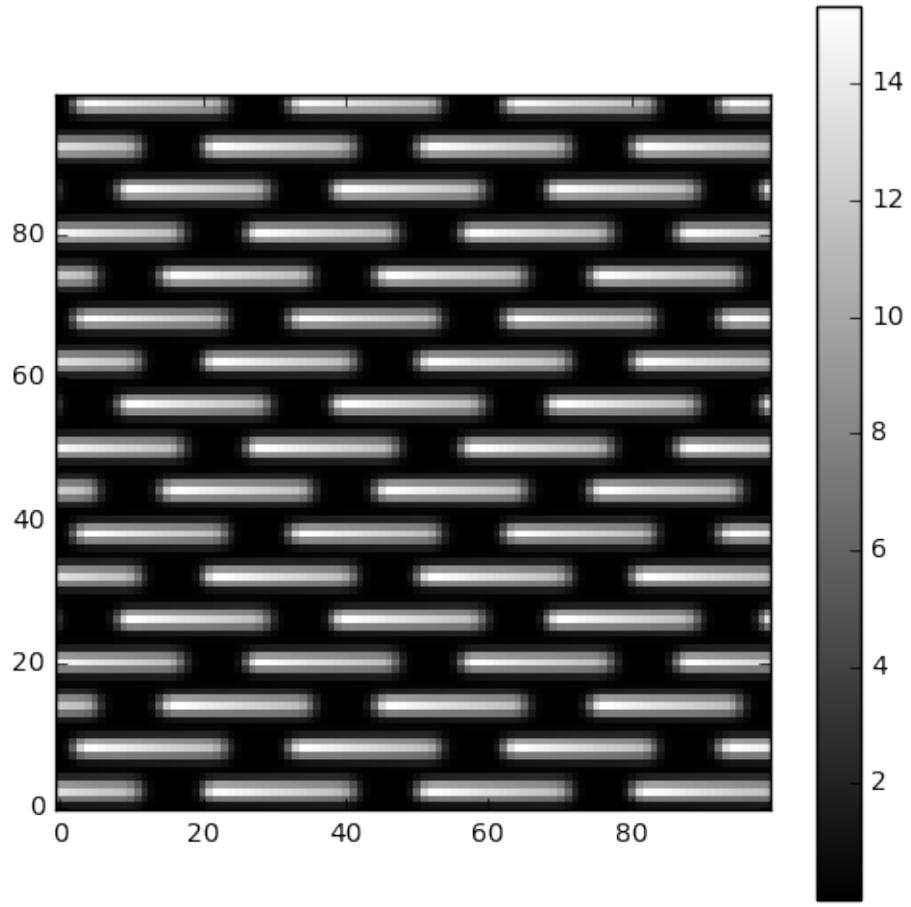


A zoom-in version is visible here:

```
In [10]: plt.figure(figsize=(6,6))
         img = Image(par.unitTestsOutputs+'/flatfield.fits').data
         subsize = 50
         plt.imshow(img[par.npix//2-subsize:par.npix//2+subsize,par.npix//2-subsize:par.npi
         plt.colorbar()
```

```
main - INFO - Read data from HDU 1 of ../../code/unitTestsOutputs/flatfield.fits
```

```
Out[10]: <matplotlib.colorbar.Colorbar at 0x11ac2b198>
```



1.2 Simulate detector readout

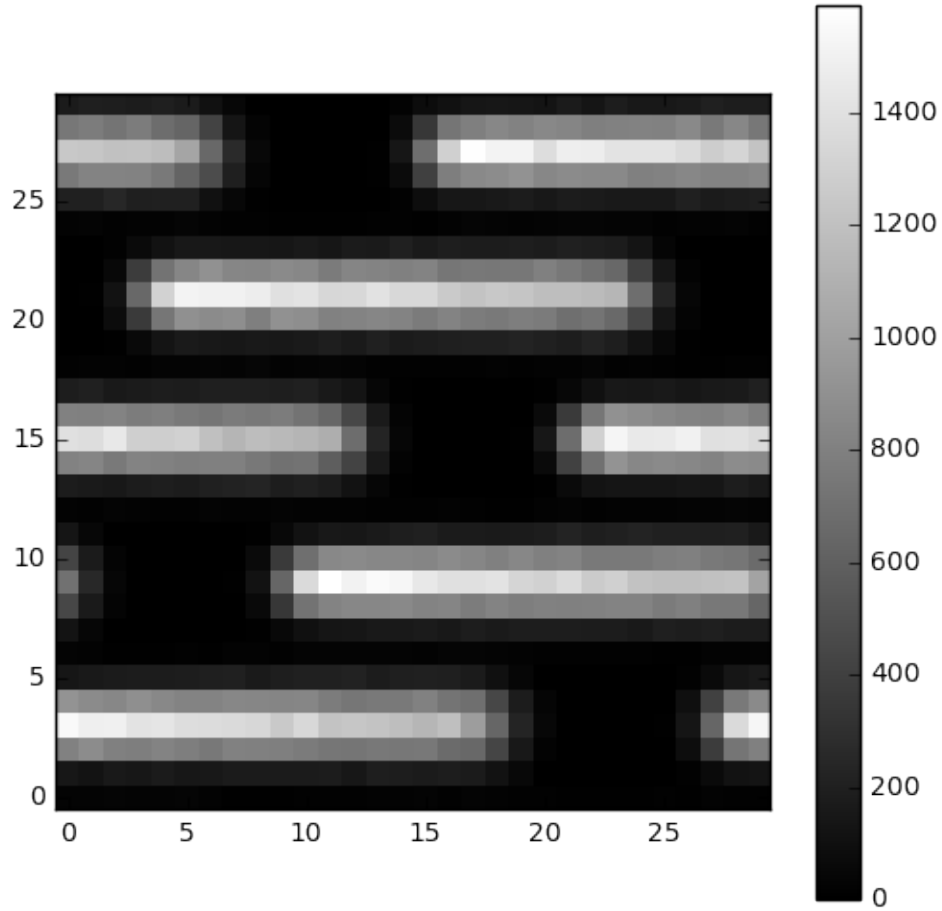
We have also some routines that can add noise to the detector. Assuming that the input detector map is in photons per second, we can 'detectorify' this map by adding Poisson noise, read noise, CIC noise, and dark current noise. In the future, we will also implement Electron-multiplying noise and Traps.

```
In [11]: from tools.image import Image
         from tools.detector import readDetector,averageDetectorReadout
         flat = Image(filename=par.unitTestsOutputs+'/flatfield.fits')
         read=readDetector(par,flat,inttime=100,append_header=True)
```

```
main - INFO - Read data from HDU 1 of ../../code/unitTestsOutputs/flatfield.fits
```

```
In [12]: plt.figure(figsize=(6,6))
         subsize = 15
         plt.imshow(read[par.npix//2-subsize:par.npix//2+subsize,par.npix//2-subsize:par.np
         plt.colorbar()
```

```
Out[12]: <matplotlib.colorbar.Colorbar at 0x11b0dfdd8>
```



Let's save the noisified frame to a new name. This is useful to show since writing to FITS is a very common task.

```
In [13]: newImage = Image(data=read,header=par.hdr)
         newImage.write(par.unitTestsOutputs+'/flatfield_noise.fits',clobber=True)
```

```
main - INFO - Writing data to ../../code/unitTestsOutputs/flatfield_noise.fits
```

1.3 Reduction step

The reduction step is straightforward, as long as the wavelength calibration is good.

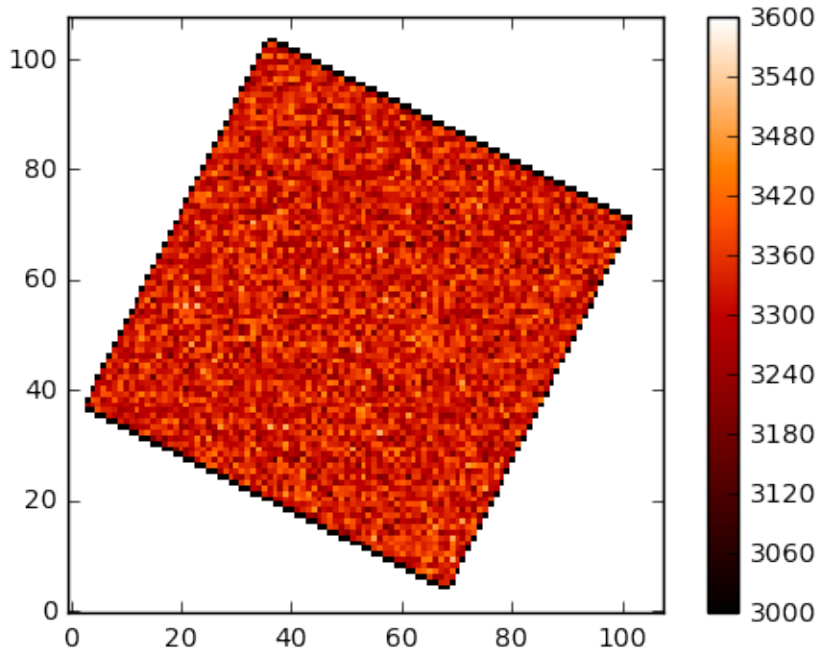
```
In [14]: from IFS import reduceIFSMap
         cube = reduceIFSMap(par,par.unitTestsOutputs+'/flatfield_noise.fits')
```

```
main - INFO - Read data from HDU 1 of ../../code/unitTestsOutputs/flatfield_noise.fits
crispy - INFO - Reduced cube will have 17 wavelength bins
```

```
main - INFO - Writing data to ../../code/SimResults/flatfield_noise_red_optext.fits
```

Now we can display the cube interactively, or look it up with DS9 (it is located in par.exportDir)

```
In [15]: import ipywidgets
         def plt_ifs_optext(wchan):
             plt.imshow(cube.data[wchan-1,:,:],
                        vmin=3000, vmax=3600, cmap='gist_heat')
             plt.colorbar()
         ipywidgets.interact(plt_ifs_optext, wchan=(1,cube.data.shape[0]));
```



Let's look now at the header of the created file

```
In [16]: cube.header
```

```
Out[16]: SIMPLE  =                    T / conforms to FITS standard
         BITPIX  =                    8 / array data type
         NAXIS   =                    0 / number of array dimensions
         EXTEND  =                    T
         COMMENT
         COMMENT ************************************************************
         COMMENT ********************** General parameters ******************
         COMMENT ************************************************************
         COMMENT
         NLENS   =                  108 / # lenslets across array
         PITCH   =             0.000174 / Lenslet pitch (meters)
         INTERLAC=                    2 / Interlacing
         PHILENS =    26.56505117707799 / Rotation angle of the lenslets (deg)
         PIXSIZE =              1.3E-05 / Pixel size (meters)
         LENSAMP =                  0.5 / Lenslet sampling (lam/D)
         LSAMPWAV=                600.0 / Lenslet sampling wavelength (nm)
         FWHM    =                    2 / FHWM of PSFLet at detector (pixels)
         FWHMLAM =                660.0 / Wavelength at which FWHM is defined (nm)
```

```
NPIX    =                 1024 / Number of detector pixels
DISPDIST=                    F / Use PISCES distortion/dispersion?
COMMENT
COMMENT ************************************************************
COMMENT ********************* IFS Simulation *****************
COMMENT ************************************************************
COMMENT
SCALE   =    6.390626327764054 / Factor by which the input slice is rescaled
COMMENT
COMMENT ************************************************************
COMMENT ********************* Innput info ************************
COMMENT ************************************************************
COMMENT
INSLICES=                   17 / Number of wavelengths in input cube
COMMENT
COMMENT ************************************************************
COMMENT ********************* Detector readout ********************
COMMENT ************************************************************
COMMENT
RN      =                  0.2 / Read noise (electrons/read)
CIC     =                0.001 / Clock-induced charge
DARK    =                1E-05 / Dark current
TRAPS   =                    F / Use traps? T/F
INTTIME =                  100 / Integration time (s)
COMMENT
COMMENT ************************************************************
COMMENT ********************* Cube Extraction ********************
COMMENT ************************************************************
COMMENT
R       =                   50 / Spectral resolution of final cube
CALDIR  = 'wavecalR50_770'     / Directory of wavelength solution
CUBEMODE= 'Optimal Extraction' / Method used to extract data cube
LAM_MIN =    704.4295434323802 / Minimum mid wavelength of extracted cube
LAM_MAX =    834.8563961903914 / Maximum mid wavelength of extracted cube
DLOGLAM =  0.01061696327719375 / Log spacing of extracted wavelength bins
NLAM    =                   17 / Number of extracted wavelengths
SMOOTHED=                    F / Cube NOT smoothed over bad lenslets
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [17]: log.shutdown()

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-17-87f260106d49> in <module>()
----> 1 log.shutdown()

AttributeError: 'CharisLogger' object has no attribute 'shutdown'
```

Part II

PISCES

CHAPTER

TWO

PISCES WAVELENGTH CALIBRATION

We have PISCES VARIA calibration sets. We will show how to process them to build a new wavelength calibration

```
In [1]: import numpy as np
        import glob
        import matplotlib
        import matplotlib.pyplot as plt
        matplotlib.rcParams['image.origin'] = 'lower'
        matplotlib.rcParams['image.interpolation'] = 'nearest'
```

```
Populating the interactive namespace from numpy and matplotlib
```

Import crispy-related stuff

```
In [2]: import sys
        codefolder = '../../code'
        if codefolder not in sys.path: sys.path.append(codefolder)
        import tools
        import logging as log
        from tools.initLogger import getLogger
        log = getLogger('main')
        from tools.image import Image
        from PISCESparams import Params
        par = Params(codefolder)
```

2.1 Import files and subtract darks

```
In [3]: VARIA_ON= glob.glob(par.wavecalDir+'IFS???nm.fits')
        VARIA_OFF= glob.glob(par.wavecalDir+'IFSdark???nm.fits')
        par.lamlist = []
        par.filelist = []
        for i in range(len(VARIA_ON)):
            img = Image(VARIA_ON[i]).data - Image(VARIA_OFF[i]).data
            par.lamlist += [float(VARIA_ON[i].split('/')[-1].split('nm')[0].split('IFS')[-1
            par.filelist += [par.wavecalDir+'det'+str(par.lamlist[i])+'.fits']
            Image(data=img).write(par.filelist[i])
```

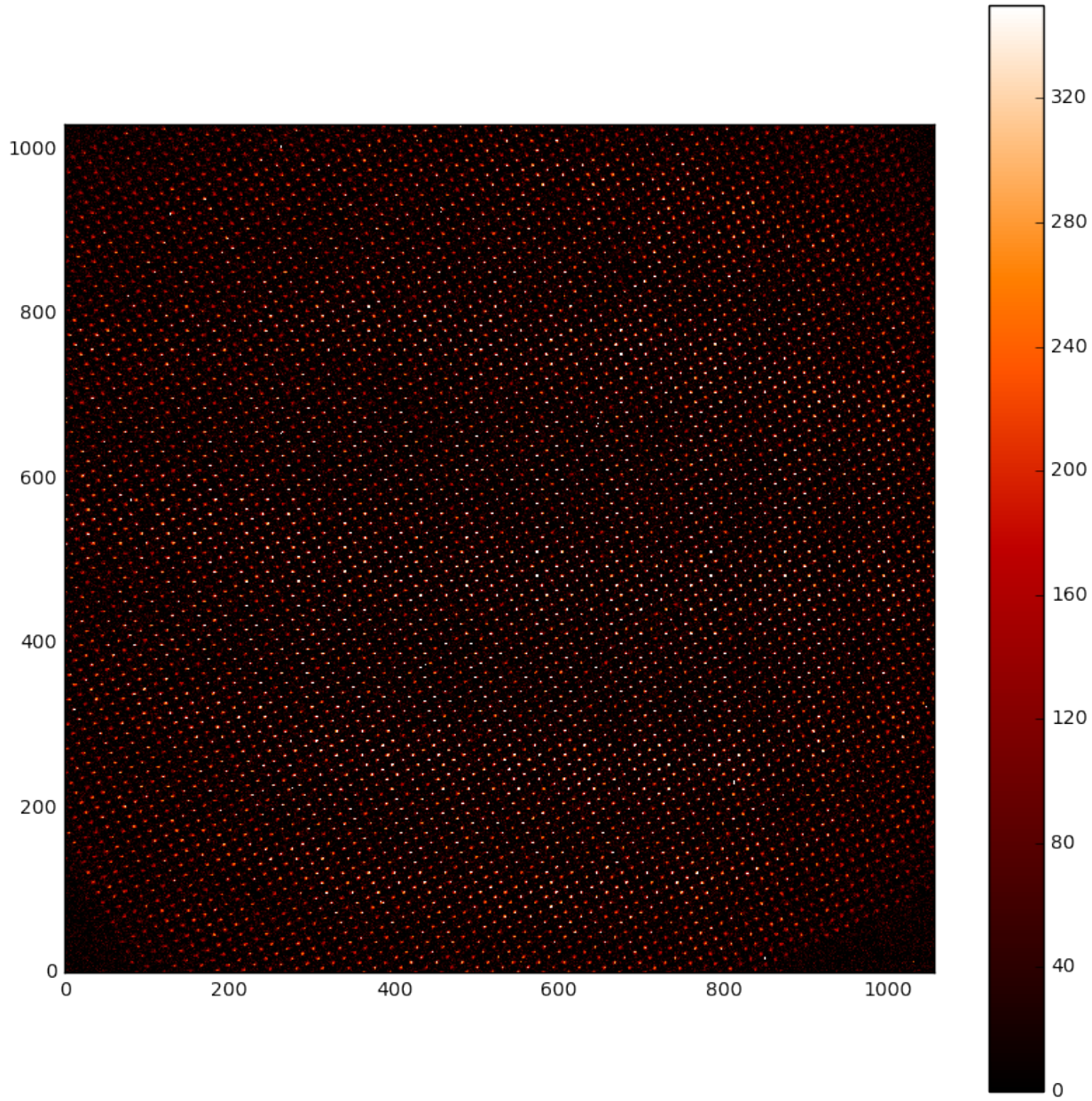
```
crispy - INFO - Read data from HDU 0 of ../../code/ReferenceFiles/Calibra_170306/IFS605nm.f
crispy - INFO - Read data from HDU 0 of ../../code/ReferenceFiles/Calibra_170306/IFSdark605
crispy - INFO - Writing data to ../../code/ReferenceFiles/Calibra_170306/det605.0.fits
crispy - INFO - Read data from HDU 0 of ../../code/ReferenceFiles/Calibra_170306/IFS615nm.f
crispy - INFO - Read data from HDU 0 of ../../code/ReferenceFiles/Calibra_170306/IFSdark615
crispy - INFO - Writing data to ../../code/ReferenceFiles/Calibra_170306/det615.0.fits
```

```
crispy - INFO - Read data from HDU 0 of ../../code/ReferenceFiles/Calibra_170306/IFS625nm.
crispy - INFO - Read data from HDU 0 of ../../code/ReferenceFiles/Calibra_170306/IFSdark625
crispy - INFO - Writing data to ../../code/ReferenceFiles/Calibra_170306/det625.0.fits
crispy - INFO - Read data from HDU 0 of ../../code/ReferenceFiles/Calibra_170306/IFS635nm.
crispy - INFO - Read data from HDU 0 of ../../code/ReferenceFiles/Calibra_170306/IFSdark635
crispy - INFO - Writing data to ../../code/ReferenceFiles/Calibra_170306/det635.0.fits
crispy - INFO - Read data from HDU 0 of ../../code/ReferenceFiles/Calibra_170306/IFS645nm.
crispy - INFO - Read data from HDU 0 of ../../code/ReferenceFiles/Calibra_170306/IFSdark645
crispy - INFO - Writing data to ../../code/ReferenceFiles/Calibra_170306/det645.0.fits
crispy - INFO - Read data from HDU 0 of ../../code/ReferenceFiles/Calibra_170306/IFS655nm.
crispy - INFO - Read data from HDU 0 of ../../code/ReferenceFiles/Calibra_170306/IFSdark655
crispy - INFO - Writing data to ../../code/ReferenceFiles/Calibra_170306/det655.0.fits
crispy - INFO - Read data from HDU 0 of ../../code/ReferenceFiles/Calibra_170306/IFS665nm.
crispy - INFO - Read data from HDU 0 of ../../code/ReferenceFiles/Calibra_170306/IFSdark665
crispy - INFO - Writing data to ../../code/ReferenceFiles/Calibra_170306/det665.0.fits
crispy - INFO - Read data from HDU 0 of ../../code/ReferenceFiles/Calibra_170306/IFS675nm.
crispy - INFO - Read data from HDU 0 of ../../code/ReferenceFiles/Calibra_170306/IFSdark675
crispy - INFO - Writing data to ../../code/ReferenceFiles/Calibra_170306/det675.0.fits
crispy - INFO - Read data from HDU 0 of ../../code/ReferenceFiles/Calibra_170306/IFS685nm.
crispy - INFO - Read data from HDU 0 of ../../code/ReferenceFiles/Calibra_170306/IFSdark685
crispy - INFO - Writing data to ../../code/ReferenceFiles/Calibra_170306/det685.0.fits
crispy - INFO - Read data from HDU 0 of ../../code/ReferenceFiles/Calibra_170306/IFS695nm.
crispy - INFO - Read data from HDU 0 of ../../code/ReferenceFiles/Calibra_170306/IFSdark695
crispy - INFO - Writing data to ../../code/ReferenceFiles/Calibra_170306/det695.0.fits
crispy - INFO - Read data from HDU 0 of ../../code/ReferenceFiles/Calibra_170306/IFS705nm.
crispy - INFO - Read data from HDU 0 of ../../code/ReferenceFiles/Calibra_170306/IFSdark705
crispy - INFO - Writing data to ../../code/ReferenceFiles/Calibra_170306/det705.0.fits
crispy - INFO - Read data from HDU 0 of ../../code/ReferenceFiles/Calibra_170306/IFS715nm.
crispy - INFO - Read data from HDU 0 of ../../code/ReferenceFiles/Calibra_170306/IFSdark715
crispy - INFO - Writing data to ../../code/ReferenceFiles/Calibra_170306/det715.0.fits
crispy - INFO - Read data from HDU 0 of ../../code/ReferenceFiles/Calibra_170306/IFS725nm.
crispy - INFO - Read data from HDU 0 of ../../code/ReferenceFiles/Calibra_170306/IFSdark725
crispy - INFO - Writing data to ../../code/ReferenceFiles/Calibra_170306/det725.0.fits
```

Image the last file

```
In [4]: plt.figure(figsize=(10,10))
        plt.imshow(img,vmin=0, vmax=350, cmap='gist_heat')
        plt.colorbar()
        plt.show()
```

```
Out[4]: <matplotlib.colorbar.Colorbar at 0x10f56e128>
```



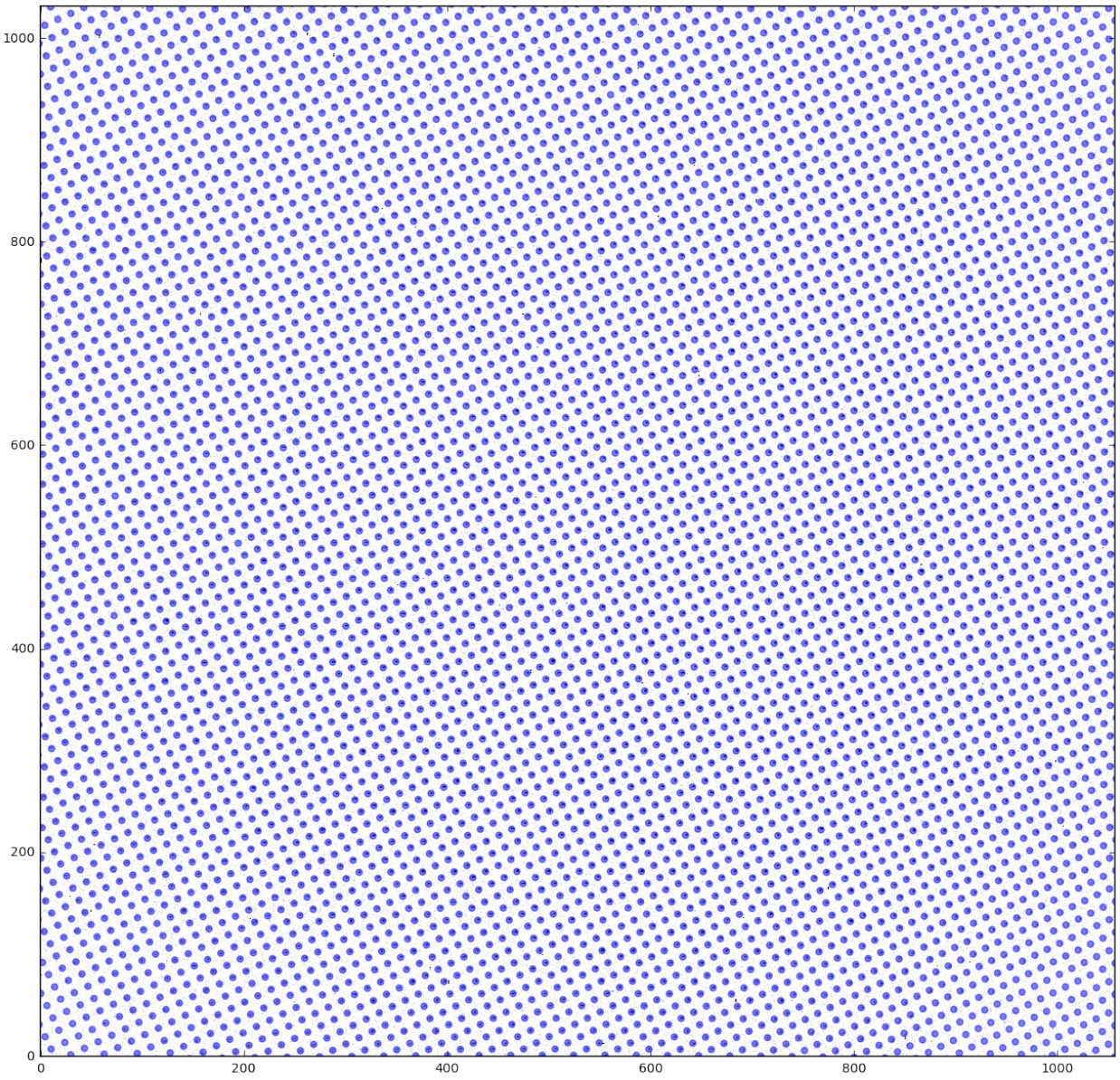
In [5]:
```
print(par.lamlist)
```

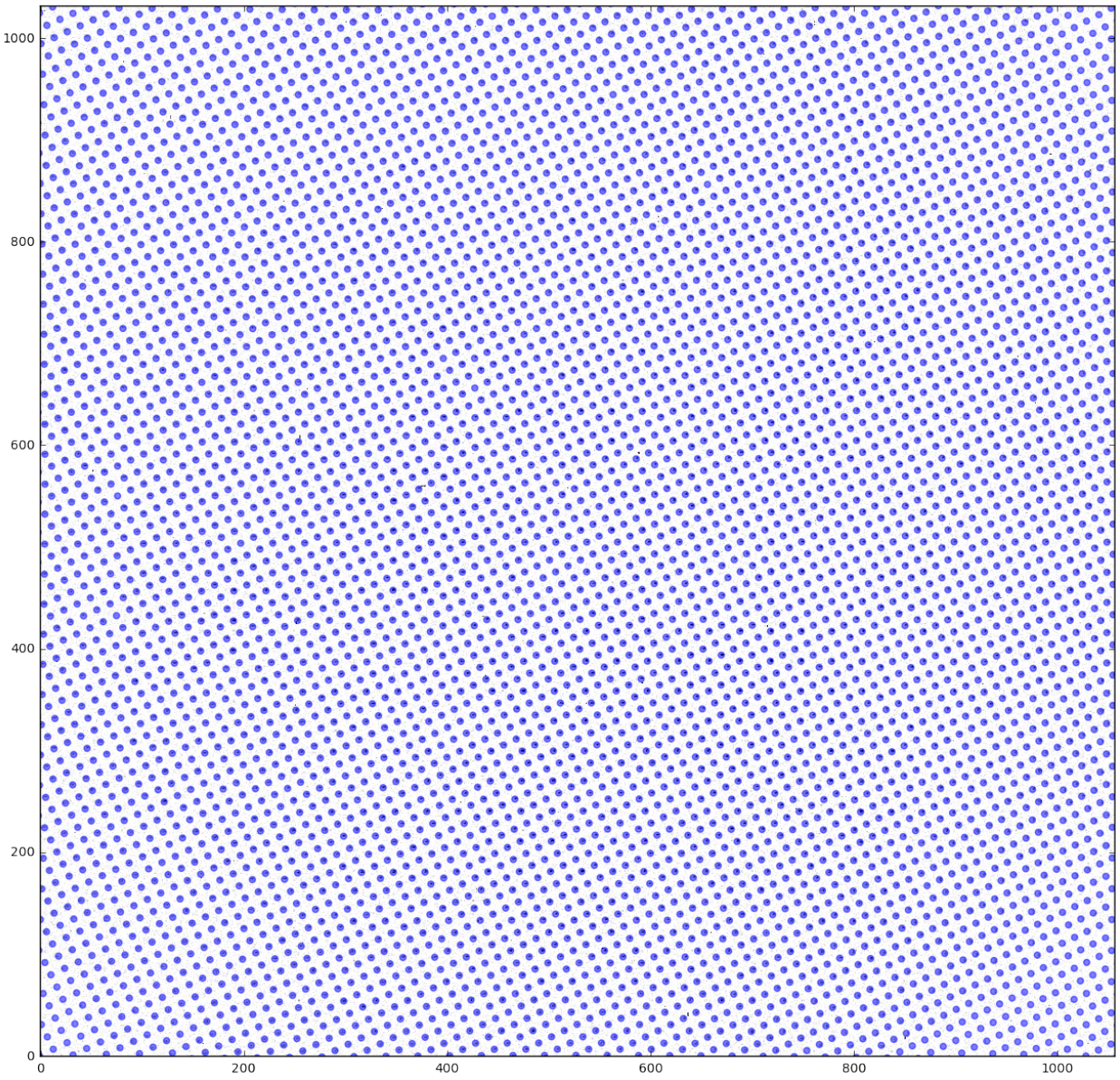
```
[605.0, 615.0, 625.0, 635.0, 645.0, 655.0, 665.0, 675.0, 685.0, 695.0, 705.0, 715.0, 725.0]
```

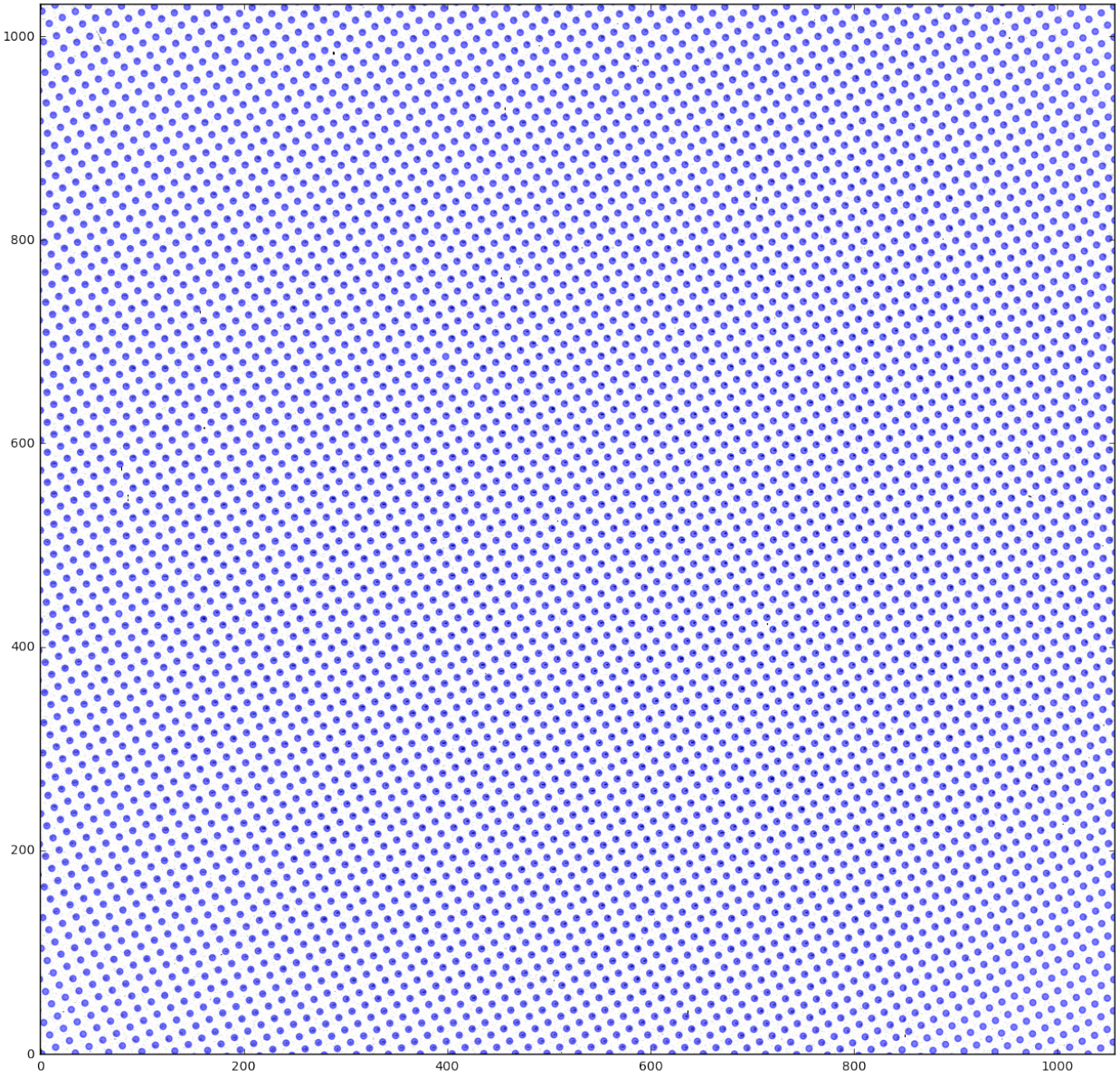
2.2 Build wavelength calibration files

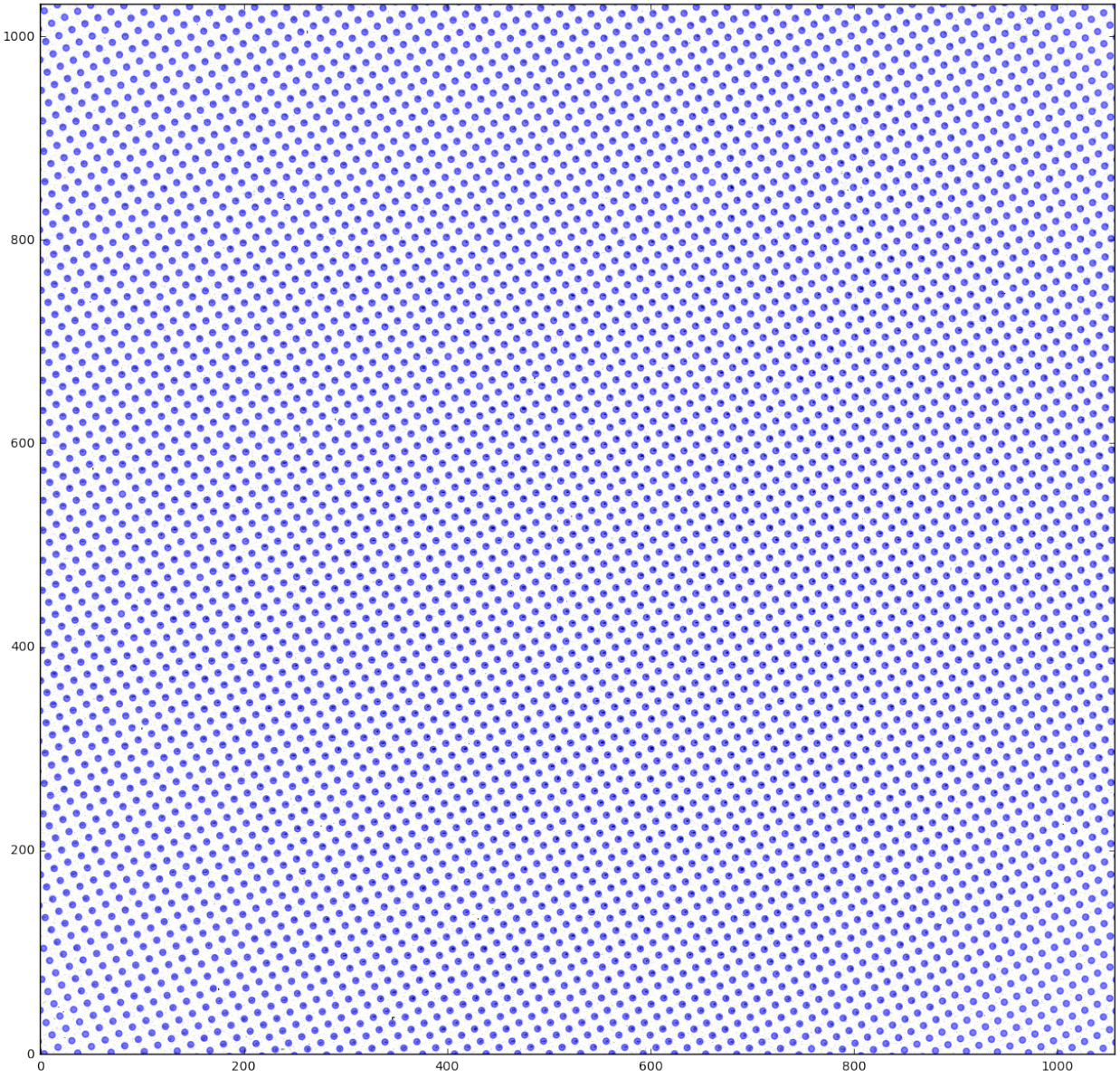
In [6]:
```
from tools.wavecal import buildcalibrations
buildcalibrations(par,
                  genwavelengthsol=True, # Compute wavelength at the center of all
                  makehiresPSFlets=False, # do NOT make high-resolution PSFLets un
                  inspect=True) # build inspection PNGs in par.wavecalDir
```

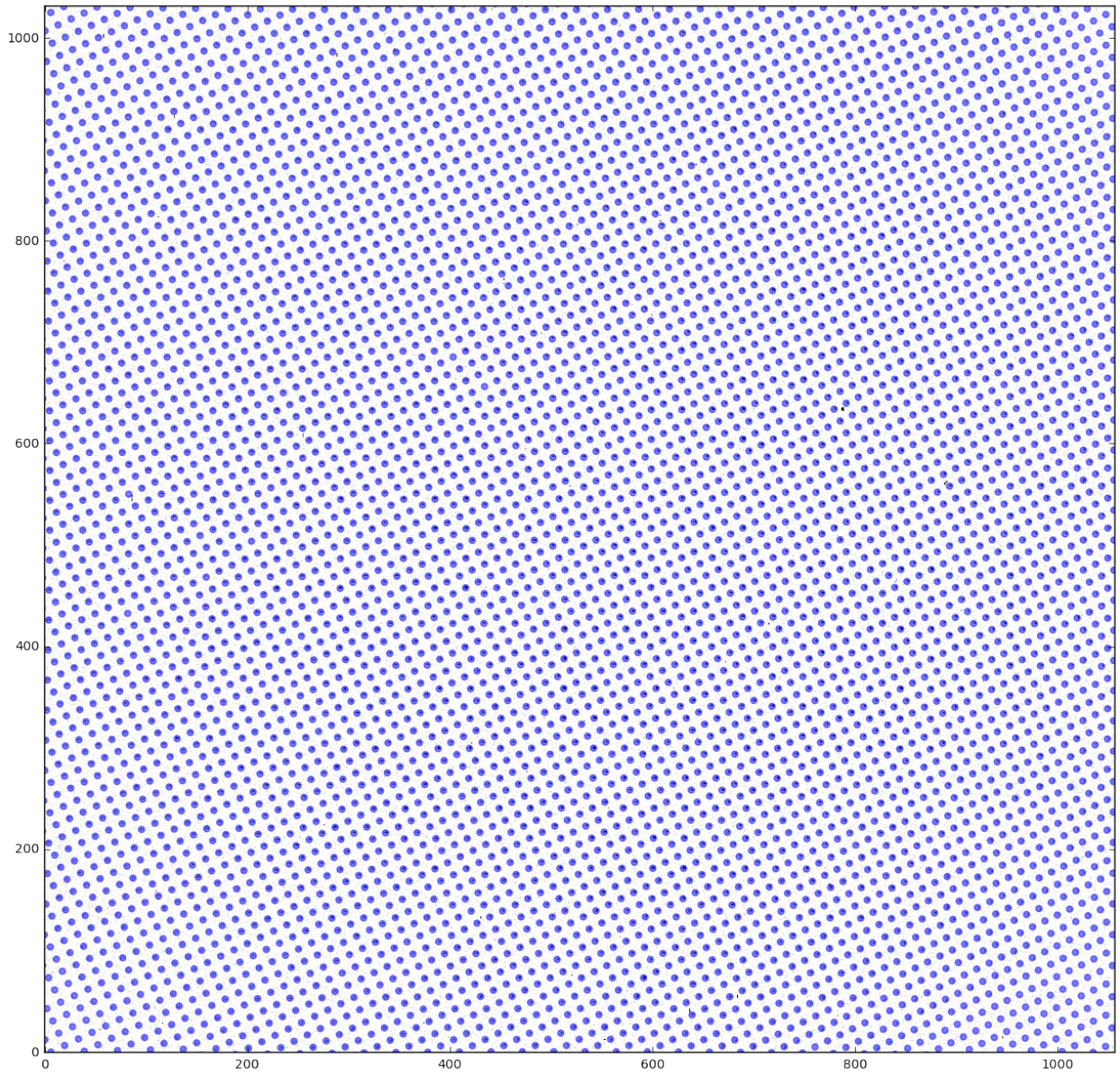
```
crispy - INFO - Building calibration files, placing results in ../../code/ReferenceFiles/Ca
crispy - INFO - Read data from HDU 1 of ../../code/ReferenceFiles/Calibra_170306/det605.0.i
crispy - INFO - Read data from HDU 1 of ../../code/ReferenceFiles/Calibra_170306/det605.0.i
crispy - INFO - Initializing PSFlet location transformation coefficients
crispy - INFO - Performing initial optimization of PSFlet location transformation coefficie
crispy - INFO - Performing final optimization of PSFlet location transformation coefficient
crispy - INFO - Read data from HDU 1 of ../../code/ReferenceFiles/Calibra_170306/det615.0.i
crispy - INFO - Initializing transformation coefficients with previous values
crispy - INFO - Performing final optimization of PSFlet location transformation coefficient
crispy - INFO - Read data from HDU 1 of ../../code/ReferenceFiles/Calibra_170306/det625.0.i
crispy - INFO - Initializing transformation coefficients with previous values
crispy - INFO - Performing final optimization of PSFlet location transformation coefficient
crispy - INFO - Read data from HDU 1 of ../../code/ReferenceFiles/Calibra_170306/det635.0.i
crispy - INFO - Initializing transformation coefficients with previous values
crispy - INFO - Performing final optimization of PSFlet location transformation coefficient
crispy - INFO - Read data from HDU 1 of ../../code/ReferenceFiles/Calibra_170306/det645.0.i
crispy - INFO - Initializing transformation coefficients with previous values
crispy - INFO - Performing final optimization of PSFlet location transformation coefficient
crispy - INFO - Read data from HDU 1 of ../../code/ReferenceFiles/Calibra_170306/det655.0.i
crispy - INFO - Initializing transformation coefficients with previous values
crispy - INFO - Performing final optimization of PSFlet location transformation coefficient
crispy - INFO - Read data from HDU 1 of ../../code/ReferenceFiles/Calibra_170306/det665.0.i
crispy - INFO - Initializing transformation coefficients with previous values
crispy - INFO - Performing final optimization of PSFlet location transformation coefficient
crispy - INFO - Read data from HDU 1 of ../../code/ReferenceFiles/Calibra_170306/det675.0.i
crispy - INFO - Initializing transformation coefficients with previous values
crispy - INFO - Performing final optimization of PSFlet location transformation coefficient
crispy - INFO - Read data from HDU 1 of ../../code/ReferenceFiles/Calibra_170306/det685.0.i
crispy - INFO - Initializing transformation coefficients with previous values
crispy - INFO - Performing final optimization of PSFlet location transformation coefficient
crispy - INFO - Read data from HDU 1 of ../../code/ReferenceFiles/Calibra_170306/det695.0.i
crispy - INFO - Initializing transformation coefficients with previous values
crispy - INFO - Performing final optimization of PSFlet location transformation coefficient
crispy - INFO - Read data from HDU 1 of ../../code/ReferenceFiles/Calibra_170306/det705.0.i
crispy - INFO - Initializing transformation coefficients with previous values
crispy - INFO - Performing final optimization of PSFlet location transformation coefficient
crispy - INFO - Read data from HDU 1 of ../../code/ReferenceFiles/Calibra_170306/det715.0.i
crispy - INFO - Initializing transformation coefficients with previous values
crispy - INFO - Performing final optimization of PSFlet location transformation coefficient
crispy - INFO - Read data from HDU 1 of ../../code/ReferenceFiles/Calibra_170306/det725.0.i
crispy - INFO - Initializing transformation coefficients with previous values
crispy - INFO - Performing final optimization of PSFlet location transformation coefficient
crispy - INFO - Saving wavelength solution to ../../code/ReferenceFiles/Calibra_170306/lams
crispy - INFO - Computing wavelength values at pixel centers
crispy - INFO - Saving wavelength calibration cube
crispy - INFO - Total time elapsed: 401 s
```

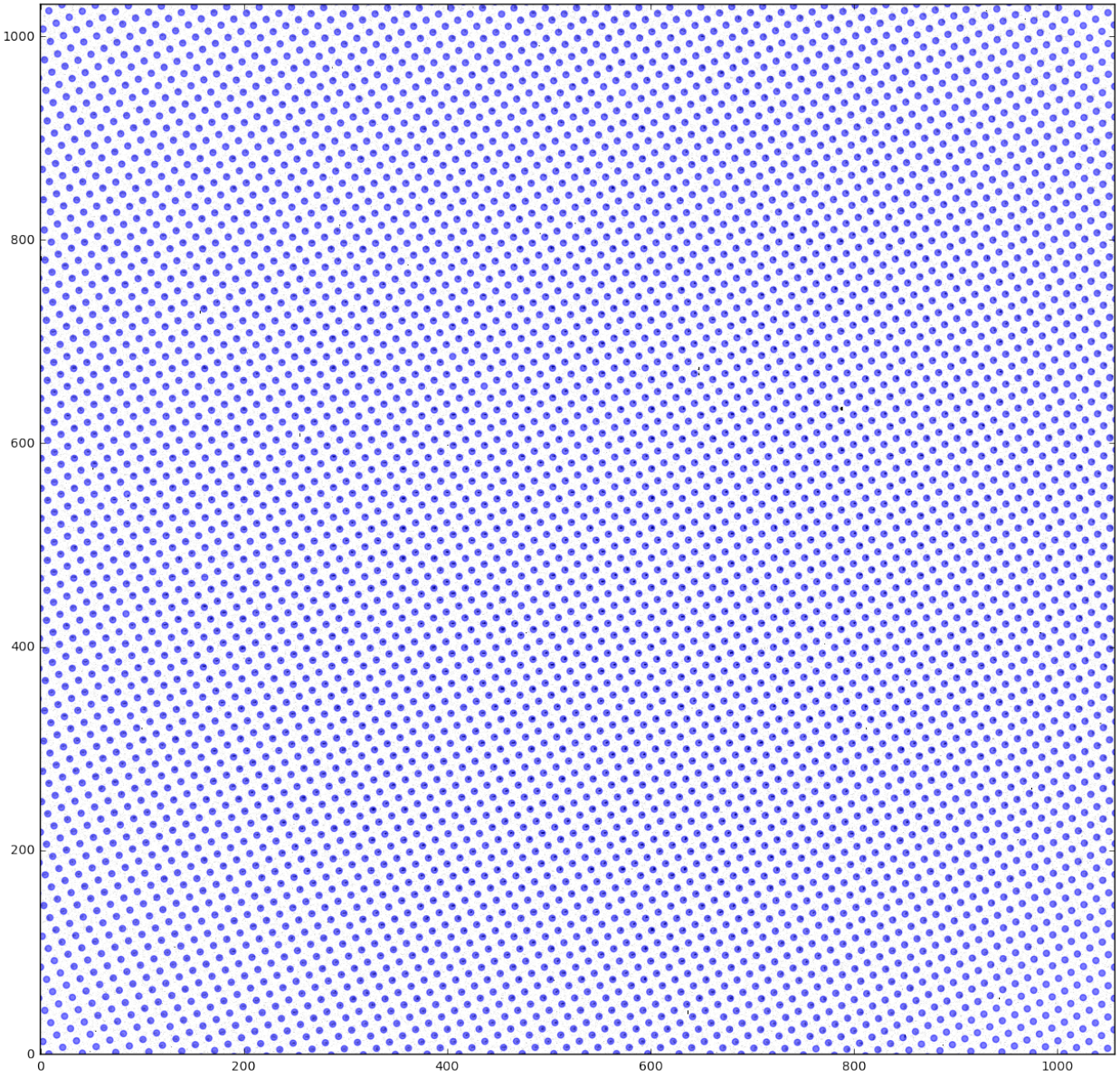

1000
800
600
400
200
0
0
200
400
600
800
1000

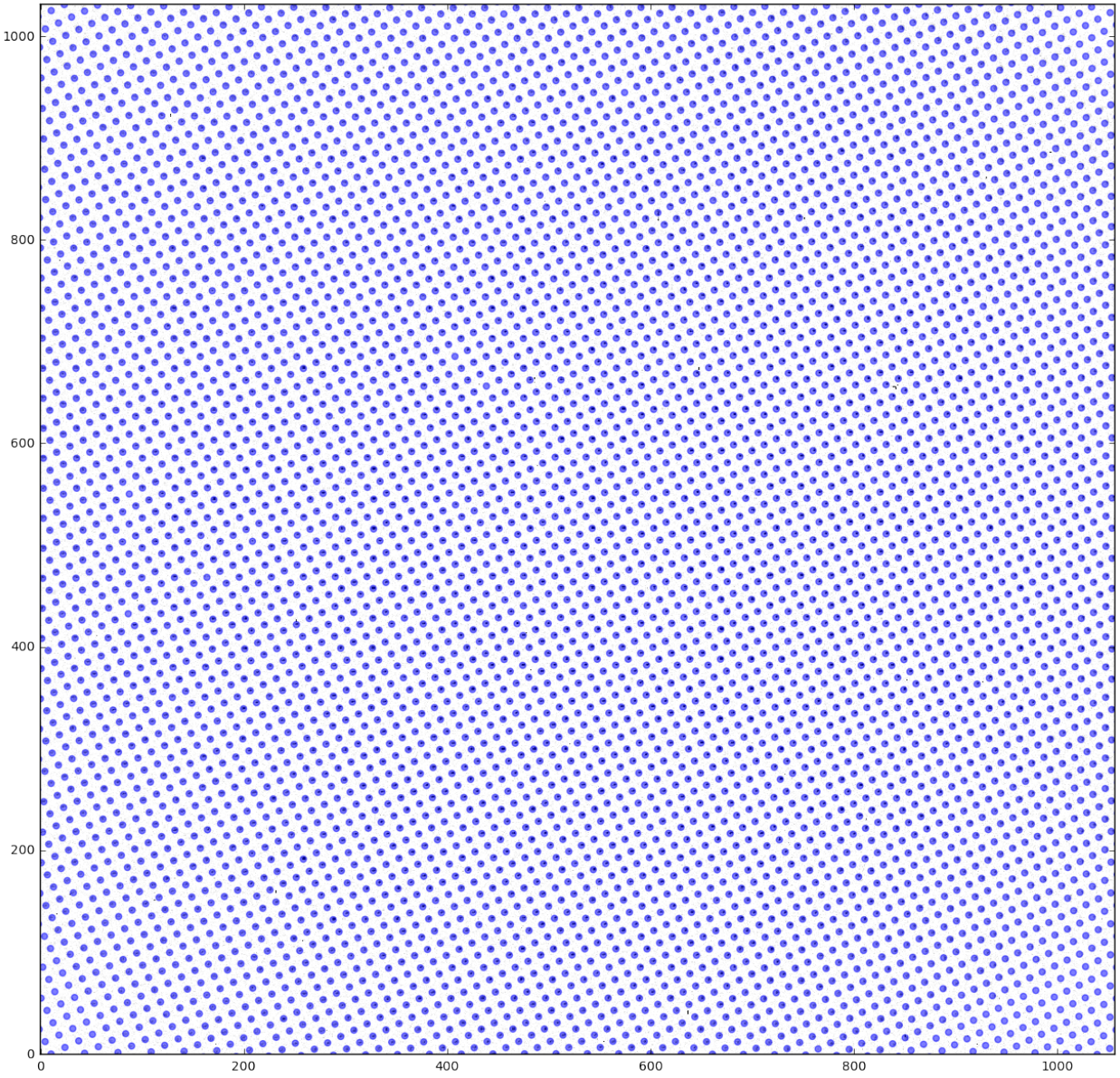

1000
800
600
400
200
0
0
200
400
600
800
1000

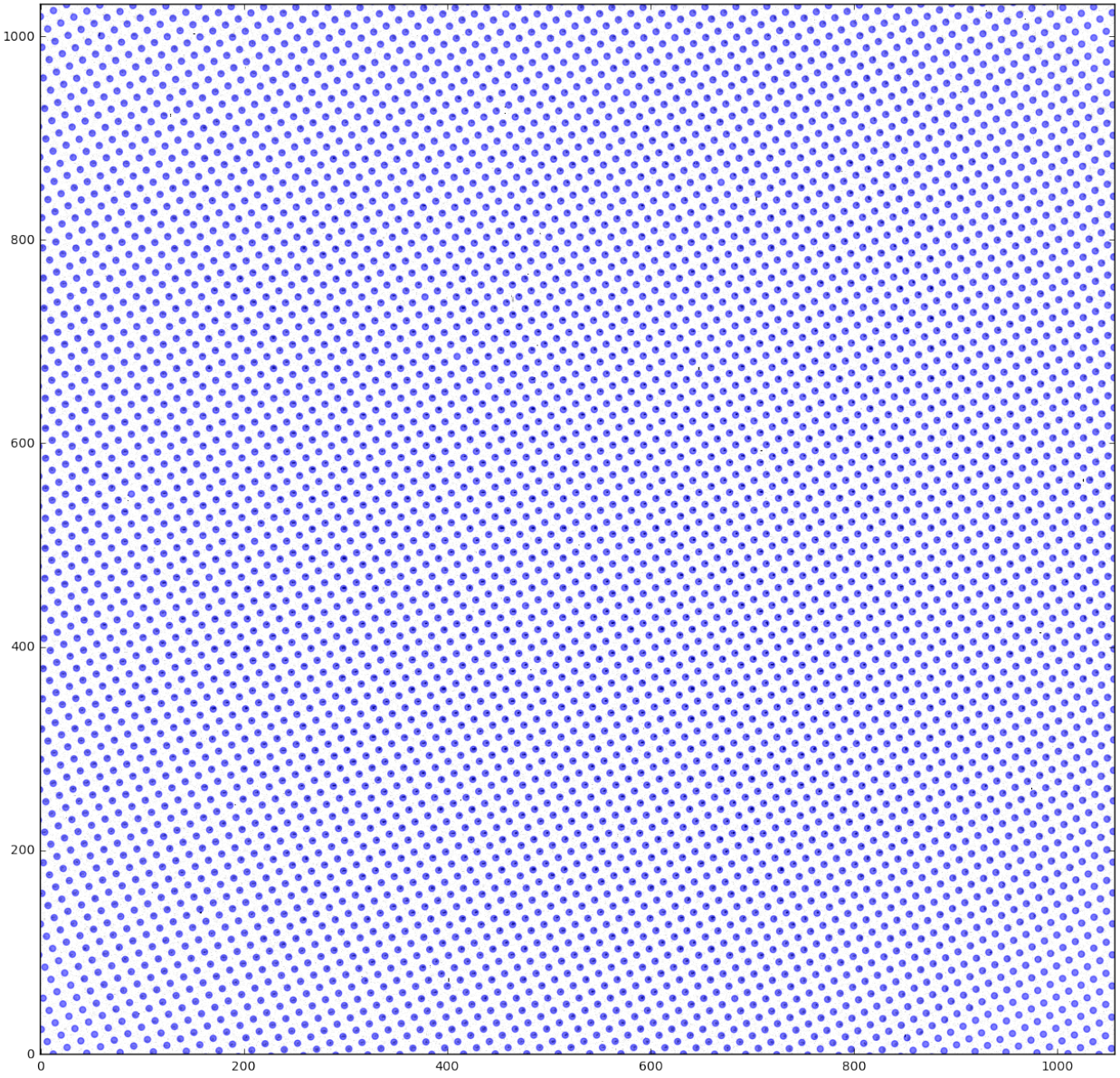

1000
800
600
400
200
0
0
200
400
600
800
1000

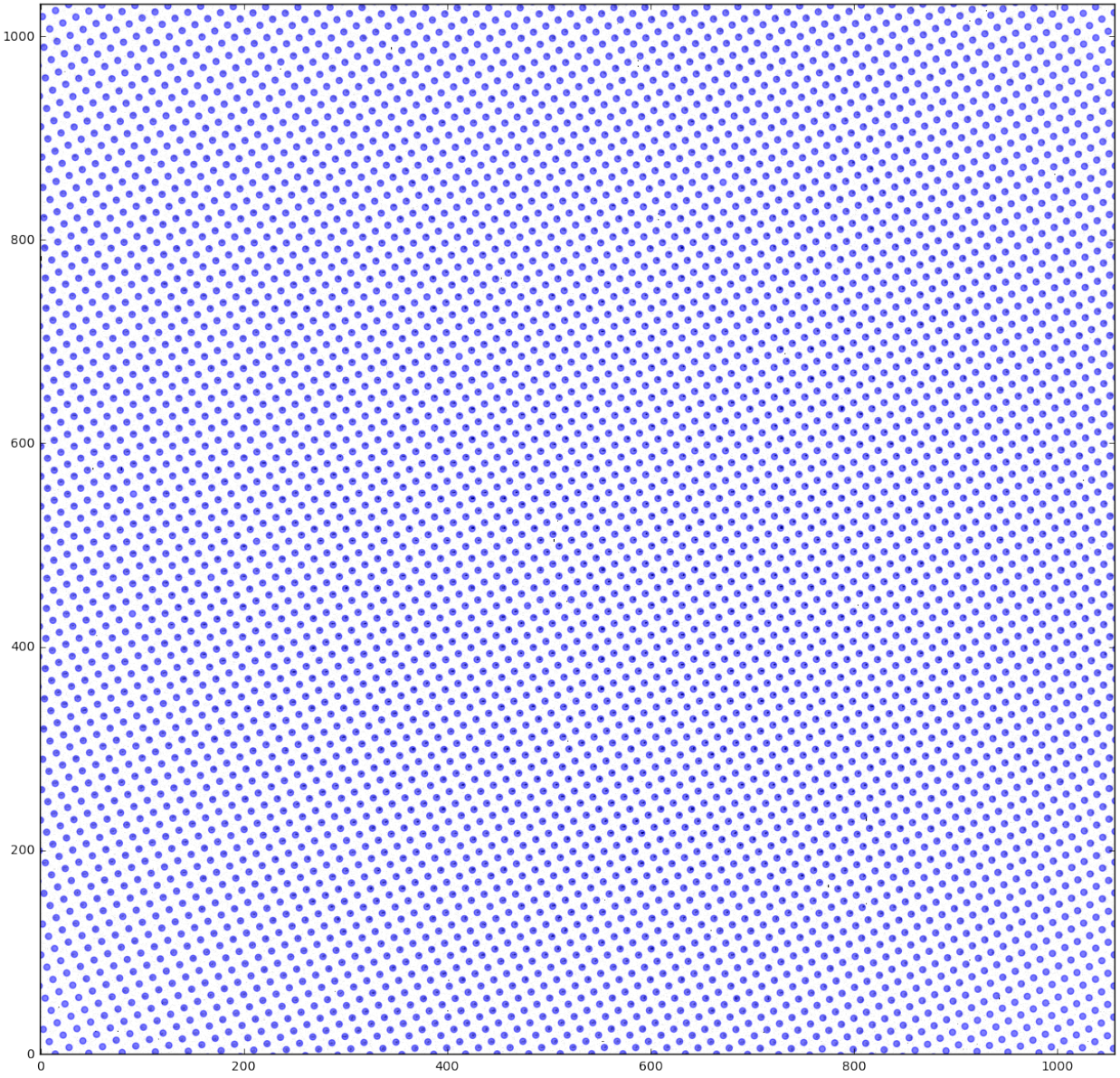

1000
800
600
400
200
0
0
200
400
600
800
1000

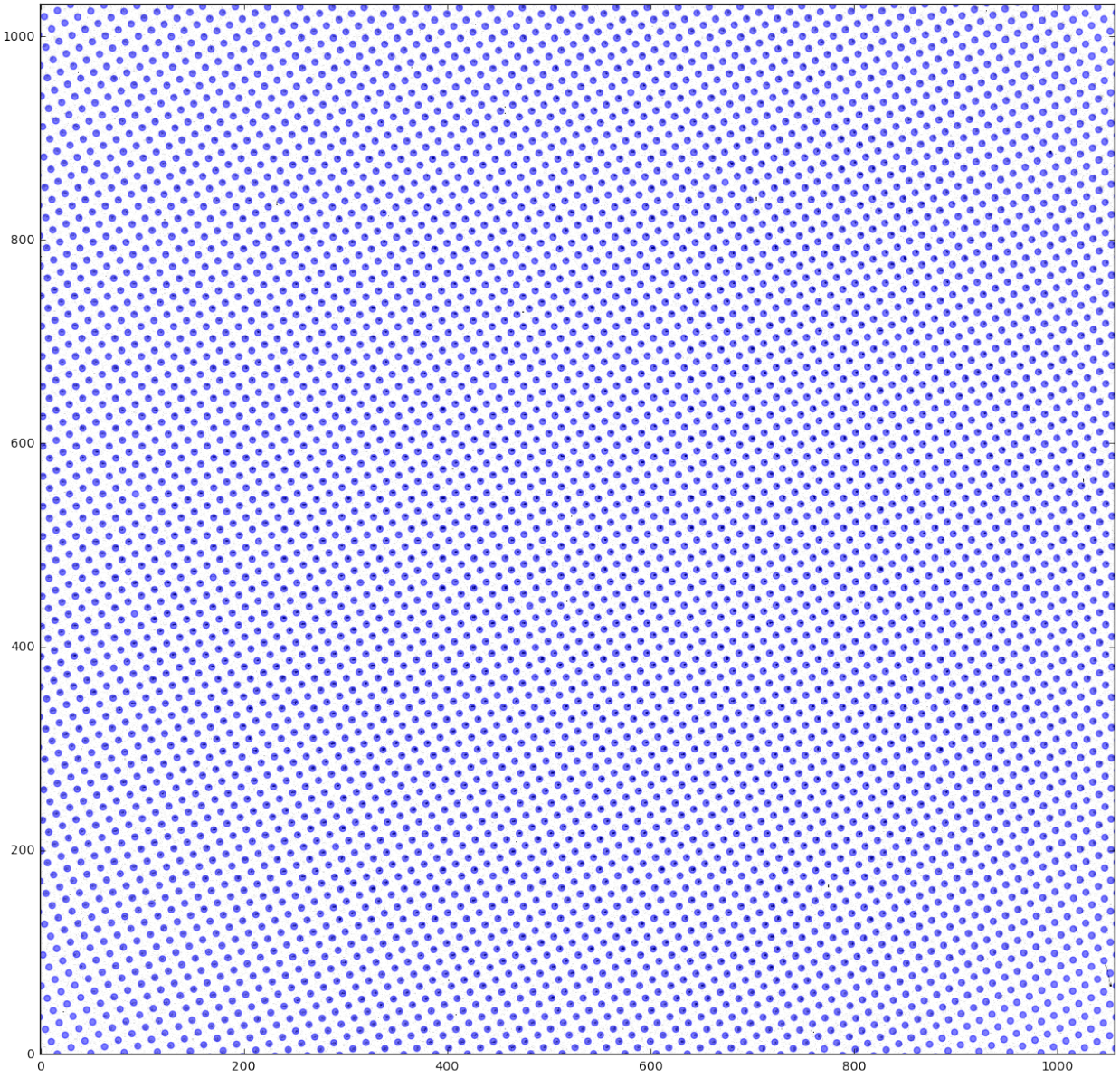

1000
800
600
400
200
0
0
200
400
600
800
1000

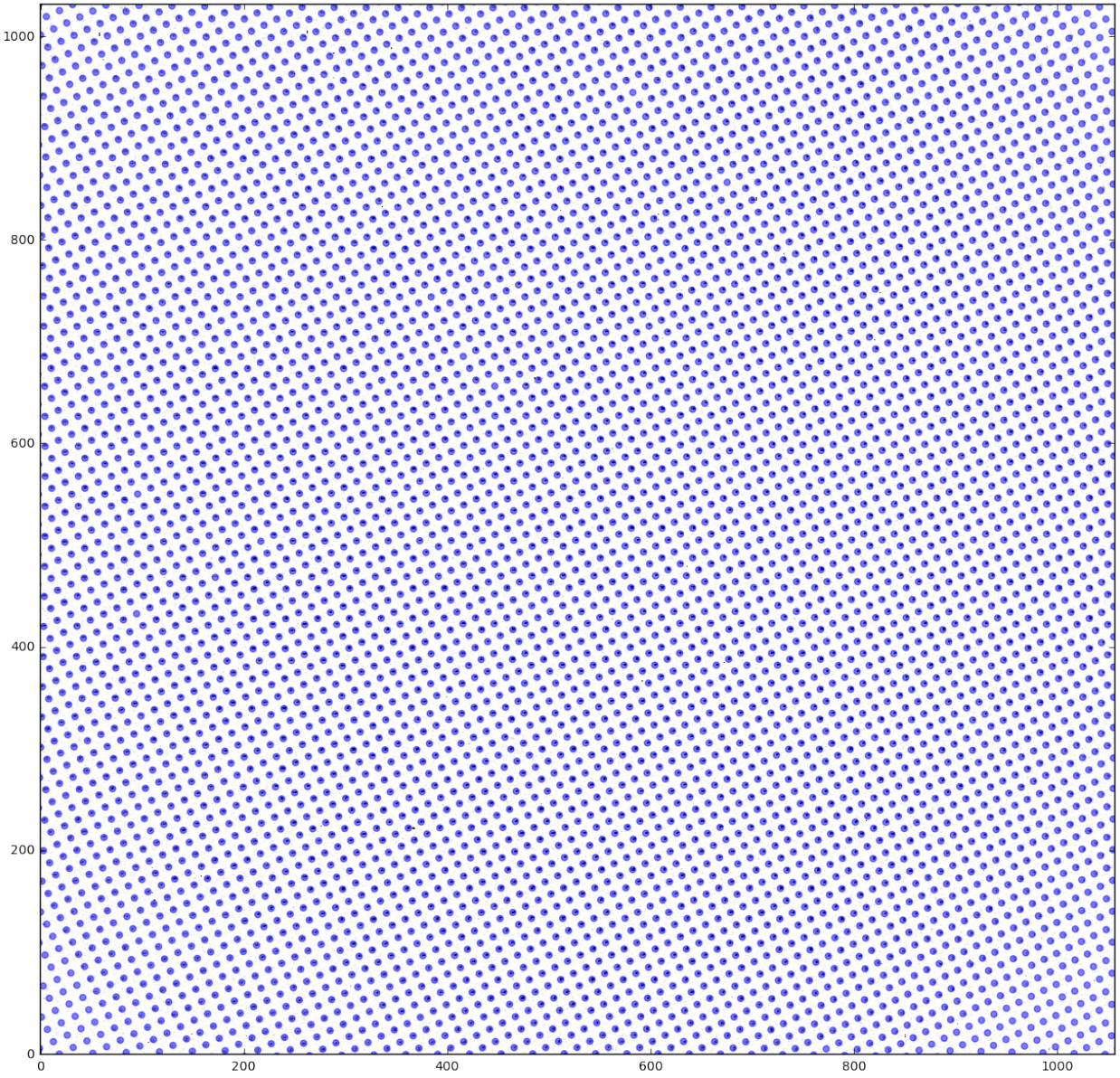

1000
800
600
400
200
0
0
200
400
600
800
1000

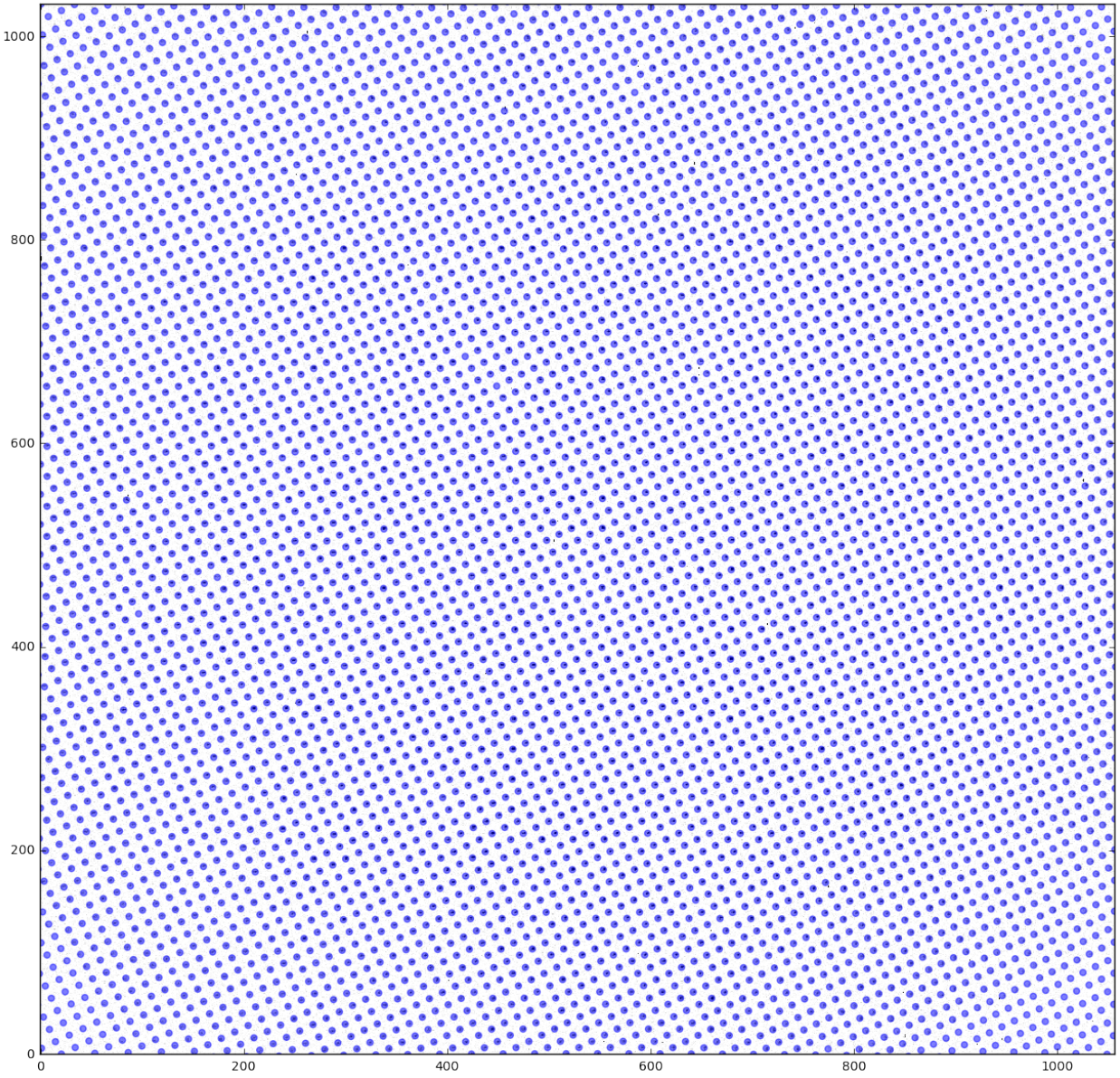

1000
800
600
400
200
0
0
200
400
600
800
1000

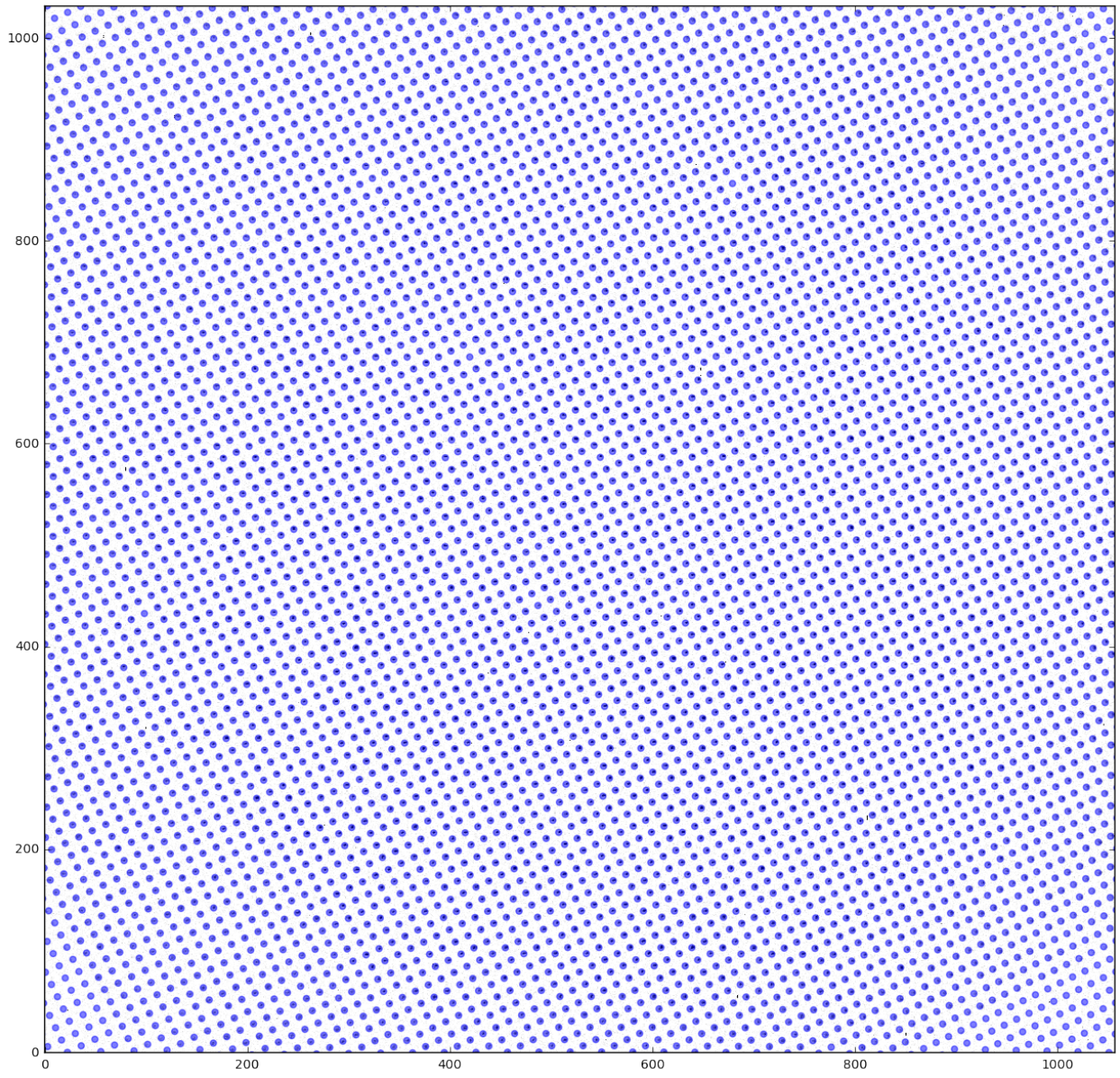

1000
800
600
400
200
0
0
200
400
600
800
1000


1000
800
600
400
200
0
0
200
400
600
800
1000


1000
800
600
400
200
0
0
200
400
600
800
1000


1000
800
600
400
200
0
0
200
400
600
800
1000


1000
800
600
400
200
0
0
200
400
600
800
1000



In [ ]:

In [ ]:

In [ ]:

In [ ]:

CHAPTER

THREE

PISCES DATA REDUCTION

The data reduction pipeline for PISCES is kept as simple as possible for easing integration with JPL's coronagraph control software. It requires an existing wavelength calibration.

```
In [1]: import sys
        codefolder = '../../code'
        if codefolder not in sys.path: sys.path.append(codefolder)
        import tools
        from tools.initLogger import getLogger
        log = getLogger('main')
        from tools.image import Image
        from PISCESparams import Params
        par = Params(codefolder)
```

This is simple: load the reduction function, and call it!

```
In [ ]: from IFS import reduceIFSMap
        reduced_cube= reduceIFSMap(par,MY_FILE_NAME)
```

The files are exported in the folder par.exportDir. Look for the introduction documentation page for more details.

Part III

API

CHAPTER

FOUR

TOOLS PACKAGE

4.1 Submodules

4.2 tools.detector module

tools.detector.**averageDetectorReadout**(*par*, *filelist*, *detectorFolderOut*, *suffix='detector'*, *offaxis=None*)

Process a list of files and creates individual detector readouts If we want only one file, we can just make a list of 1

tools.detector.**readDetector**(*par*, *IFSimage*, *inttime=100*, *append_header=False*)

Read noise, CIC, dark current; NO TRAPS Input is IFSimage in average photons per second Quantum efficiency considerations are already taken care of when generating IFSimage images

tools.detector.**rebinDetector**(*par*, *finalFrame*, *clip=False*)

Rebins the dense detector map with the correct scaling while conserving flux. This also works with non-integer ratios.

Parameters **par** : Parameter instance

Contains all IFS parameters

finalFrame : 2D ndarray

Dense detector map to be rebinned.

Returns **detectorFrame** : 2D array

Return the detector frame with correct pixel scale.

4.3 tools.detutils module

tools.detutils.**frebin**(*array*, *shape*, *total=True*)

Function that performs flux-conservative rebinning of an array. Found at https://github.com/benjaminpope/pysco/blob/master/frebin.py

Parameters **array: ndarray**

Numpy array to be rebinned

shape: tuple

(x,y) of new array size total: Boolean

when True flux is conserved

Returns

———-

new_array: new rebinned array with dimensions: shape

tools.detutils.**rebin**(*a*, *shape*)

Resizes a 2d array by averaging or repeating elements, new dimensions must be integral factors of original dimensions

Parameters **a** : array_like

Input array.

new_shape : tuple of int

Shape of the output array (y, x)

Returns **rebinned_array** : ndarray

If the new shape is smaller of the input array, the data are averaged, if the new shape is bigger array elements are repeated

4.4 tools.image module

class tools.image.**Image**(*filename=''*, *data=None*, *ivar=None*, *header=None*, *extraheader=None*)

Bases: object

Image is the basic class for images

self.data, self.ivar, and self.header should be numpy ndarrays, which can be read from and written to a fits file with the load and write methods. If not ndarrays, they should be None.

Image may be initialized with the name of the raw file to read, through a call to Image.load().

load(*outfilename*)

Read the first HDU with data from filename into self.data, and HDU[0].header into self.header. If there is more than one HDU with data, attempt to read the second HDU with data into self.ivar.

write(*outfilename*, *clobber=True*)

Creates a primary HDU using self.data and self.header, and attempts to write to outfilename. If self.ivar is not None, append self.ivar as a second HDU before writing to a file. clobber is provided as a keyword to fits.HDUList.writeto.

tools.image.**log = <tools.initLogger.CharisLogger object>**

most of this code is due to Tim Brandt

4.5 tools.imgtools module

tools.imgtools.**bowtie**(*image*, *xc*, *yc*, *openingAngle*, *clocking*, *IWApix*, *OWApix*, *export='bowtie'*, *twomasks=False*)

Creates one or two binary mask for a shaped pupil bowtie mask

Parameters **image: 2D ndarray**

Image to which the mask needs to be applied

xc: int

X coordinate of center of bowtie

yc: int

Y coordinate of center of bowtie

openingAngle: float

Angle in degrees representing the opening angle of the bowtie mask

clocking: float

Angle of rotation of the mask in degrees

IWApix: float

Radius in pixels corresponding to the inner working angle of the mask

OWApix: float

Radius in pixels corresponding to the outer working angle of the mask

export: boolean

Whether to export the bowtie or not

twomasks: boolean

If True, returns two masks, one for each side of the bowtie If False, returns one single mask

Returns mask: 2D ndarrays of int

Bowtie mask with 1 inside the mask, zero everywhere else.

mask2: 2D ndarrays of int

If twomasks is True, mask and mask2 are the two sides of the bowtie.

`tools.imgtools.`**`gen_bad_pix_mask`**(*image*, *filsize=3*, *threshold=5.0*, *return_smoothed_image=False*)

Identify and mask bad pixels using median filter

Parameters **image: 2D ndarray**

Image to clean

filsize: int

Size of median filter in pixels

threshold: float

Threshold in terms of standard deviations

return_smoothed_image: boolean

Returns goodpix: 2D int ndarray

Good pixel mask. 1 where pixel is good

image_sm: 2D ndarray, if return_smoothed_image

4.6 tools.initLogger module

class `tools.initLogger.`**`CharisLogger`**(*name*, *level=0*)

Bases: `logging.Logger`

This is the advanced logging object used throughout the CHARIS Data Extraction Pipeline. It inherits from the standard Python library 'logging' and provides added features. The default log level for the output file will be 1, ie ALL messages; while the default for the screen will be INFO, and can be changed easily using the setStreamLevel(lvl) member function.

Initialize the logger with a name and an optional level.

MAINCRITICAL = 80

MAINDEBUG = 60

MAINERROR = 75

MAININFO = 65

MAINWARNING = 70

PRIMCRITICAL = 55

PRIMDEBUG = 35

PRIMERROR = 49

PRIMINFO = 39

PRIMWARNING = 45

SUMMARY = 5

TOOLCRITICAL = 29

TOOLDEBUG = 9

TOOLERROR = 25

TOOLINFO = 15

TOOLWARNING = 19

maincritical (*msg*, *lvl=80*, **args*, ***kws*)

maindebug (*msg*, *lvl=60*, **args*, ***kws*)

mainerror (*msg*, *lvl=75*, **args*, ***kws*)

maininfo (*msg*, *lvl=65*, **args*, ***kws*)

mainwarning (*msg*, *lvl=70*, **args*, ***kws*)

primcritical (*msg*, *lvl=55*, **args*, ***kws*)

primdebug (*msg*, *lvl=35*, **args*, ***kws*)

primerror (*msg*, *lvl=49*, **args*, ***kws*)

priminfo (*msg*, *lvl=39*, **args*, ***kws*)

primwarning (*msg*, *lvl=45*, **args*, ***kws*)

setStreamLevel (*lvl=20*)

Set/change the level for the stream handler for a logging object. Any file handlers will be left alone. All messages of a higher severity level than 'lvl' will be printed to the screen.

Args:

lvl (int): The severity level of messages printed to the screen with the stream handler, default = 20.

Standard Levels		New Levels	
Name	Level	Name	Level
CRITICAL	50	MAINCRITICAL	80
ERROR	40	MAINERROR	75
WARNING	30	MAINWARNING	70
INFO	20	MAININFO	65
DEBUG	10	MAINDEBUG	60
NOTSET	0	PRIMCRITICAL	55
		PRIMERROR	49
		PRIMWARNING	45
		PRIMINFO	39
		PRIMDEBUG	35
		TOOLCRITICAL	29
		TOOLERROR	25
		TOOLWARNING	19
		TOOLINFO	15
		TOOLDEBUG	9
		SUMMARY	5

summary(*msg*, *lvl=5*, **args*, ***kws*)

toolcritical(*msg*, *lvl=29*, **args*, ***kws*)

tooldebug(*msg*, *lvl=9*, **args*, ***kws*)

toolerror(*msg*, *lvl=25*, **args*, ***kws*)

toolinfo(*msg*, *lvl=15*, **args*, ***kws*)

toolwarning(*msg*, *lvl=19*, **args*, ***kws*)

tools.initLogger.**addFileHandler**(*log*, *lvl=1*)

This function will add a file handler to a log with the provided level.

Args:

log (CharisLogger object): A CharisLogger object that was freshly instantiated.

lvl (int): The severity level of messages printed to the file with the file handler, default = 1.

tools.initLogger.**addFitsStyleHandler**(*log*)

This function will add a file handler with a string format ideal for directly loading into a FITS header.

Args:

log (CharisLogger object): A CharisLogger object that was freshly instantiated.

tools.initLogger.**addStreamHandler**(*log*, *lvl=20*)

This function will add a stream handler to a log with the provided level.

Args:

log (CharisLogger object): A CharisLogger object that was freshly instantiated.

lvl (int): The severity level of messages printed to the screen with the stream handler, default = 20.

tools.initLogger.**getLogger**(*name='generalLoggerName'*, *lvl=20*, *addFH=True*, *addSH=True*)

This will either return the logging object already instantiated, or instantiate a new one and return it. **Use this function to both create and return any logger** to avoid accidentally adding additional handlers by using the setUpLogger function instead.

Args:

name (str): The name for the logging object and name.log will be the output file written to disk.

lvl (int): The severity level of messages printed to the screen with the stream handler, default = 20.

addFH (boolean): Add a file handler to this logger? Default severity level for it will be 1, and it will be named following name+'.log'. Default = True.

addSH (boolean): Add a stream handler to this logger? Severity set with the lvl argument. Default = True.

Returns:

log (CharisLogger object): A CharisLogger object that was either freshly instantiated or determined to already exist, then returned.

tools.initLogger.**initLogger**(*logfile*, *levelConsole=20*, *levelLogFile=10*)

tools.initLogger.**logFileProcessInfo**(*log*)

tools.initLogger.**logSystemInfo**(*log*)

A function to be called just after a logging object is instantiated for the DEP to load the log up with info about the computer it is being ran on and the software version. This function utilizes the psutil and platform libraries, so they must be install for it to work. For clarity of the log, it is suggested to perform immediately after instantiation to put it at the top of the log file.

Args:

log (Python logging object): logging object to have the system's info summarized in.

The messages this prints to the log will look like:

System Information Summary:
OS type = Linux
OS Version = 3.9.10-100.fc17.x86_64
Machine UserName = xxxxxx.astron.s.u-tokyo.ac.jp
Machine Processor Type = x86_64
Number of cores = 8
Total RAM [GB] = 23.5403785706, % used = 15.9
Python Version = '2.7.3'

tools.initLogger.**setUpLogger**(*name='generalLoggerName'*, *lvl=20*, *addFH=True*, *addSH=True*)

This function is utilized by getLogger to set up a new logging object. It will have the default name 'generalLoggerName' and stream handler level of 20 unless redefined in the function call. NOTE: If a file handler is added, it will have the lowest severity level by default (Currently no need for changing this setting, so it will stay this way for now). Remember that any messages will be passed up to any parent loggers, so children do not always need their own file handler.

Args:

name (str): The name for the logging object and name.log will be the output file written to disk.

lvl (int): The severity level of messages printed to the screen with the stream handler, default = 20.

addFH (boolean): Add a file handler to this logger? Default severity level for it will be 1, and it will be named following name+'.log'. Default = True.

addSH (boolean): Add a stream handler to this logger? Severity set with the lvl argument. Default = True.

Returns:

log (CharisLogger object): A CharisLogger object that was freshly instantiated.

4.7 tools.inputScene module

tools.inputScene.**calc_contrast**(*wavelist*, *star_T=<Quantity 6000.0 K>*, *planet_type='Jupiter'*, *abundance=1*, *distance=5*, *phase=90*, *mean_contrast=1e-08*, *folder='/local/data/nicolaus2/mrizzo/haystacks/Cahoy_et_al_2010_Albedo_Spectra/albe*

Calculates the contrast curve for a list of wavelengths

tools.inputScene.**convert_krist_cube**(*cubeshape*, *lamlist*, *star_T*, *star_Vmag*, *tel_area*)

This function calculates the number of photons per second entering the WFIRST obscured aperture, given the star Vmag and its temperature

for each slice of the input cube This only works with John Krist's cubes,

4.8 tools.lenslet module

tools.lenslet.**Lenslets**(*par*, *imageplane*, *lam*, *lensletplane*, *allweights=None*, *kernels=None*, *locations=None*)

Function Lenslets

Creates the IFS map on a 'dense' detector array where each pixel is smaller than the final detector pixels by a factor par.pxperdetpix. Adds to lensletplane array to save memory.

Parameters **par** : Parameters instance

Contains all IFS parameters

image : 2D array

Image plane incident on lenslets.

lam : float

Wavelength (microns)

lensletplane : 2D array

Densified detector plane; the function updates this variable

allweights : 3D array

Cube with weights for each kernel

kernels : 3D array

Kernels at locations on the detector

locations : 2D array

Locations where the kernels are sampled

tools.lenslet.**processImagePlane**(*par*, *imagePlane*)

Function processImagePlane

Rotates an image or slice, and rebins in a flux-conservative way on an array of lenslets, using the plate scale provided in par.pixperlenslet. Each pixel represents the flux within a lenslet. Starts by padding the original image to avoid cropping edges when rotating. This step necessarily involves an interpolation, so one needs to be cautious.

Parameters **par** : Parameters instance

Contains all IFS parameters

imagePlane : Image instance containing 3D input cube

Input cube to IFS sim, first dimension of data is wavelength

Returns **imagePlaneRot** : 2D array

Rotated image plane on same sampling as original.

4.9 tools.locate_psflets module

class tools.locate_psflets.**PSFLets**(*load=False*, *infile=None*, *infiledir='.'*)
Bases: object

Helper class to deal with the PSFLets on the detector. Does most of the heavy lifting during the wavelength calibration step.

Initialize the class

Parameters **load: Boolean**

Whether to load an already-existing wavelength calibration file

infile: String

If load is True, this is the name of the file

infiledir: String

If load is True, this is the directory in which the file resides

geninterparray(*lam*, *allcoef*, *order=3*)
Set up array to solve for best-fit polynomial fits to the coefficients of the wavelength solution. These will be used to smooth/interpolate the wavelength solution, and ultimately to compute its inverse.

Parameters **lam: float**

Wavelength in nm

allcoef: list of lists floats

Polynomial coefficients of wavelength solution

order: int

Order of polynomial wavelength solution

Notes

Populates the attribute interp_arr in PSFLet class

genpixsol(*par*, *lam*, *allcoef*, *order=3*, *lam1=None*, *lam2=None*)
Calculates the wavelength at the center of each pixel within a microspectrum

Parameters **lam: float**

Wavelength in nm

allcoef: list of floats

List describing the polynomial coefficients that best fit the lenslets, for all wavelengths

order: int

Order of the polynomical fit

lam1: float

Lowest wavelength in nm

lam2: float

Highest wavelength in nm

Notes

This functions fills in most of the fields of the PSFLet class: the array of xindx, yindx, nlam, lam_indx and nlam_max

loadpixsol(*infile=None*, *infiledir='./calibrations'*)
Loads existing wavelength calibration file

Parameters **infile: String**

Name of the file

infiledir: String

Directory in which the file resides

monochrome_coef(*lam*, *alllam=None*, *allcoef=None*, *order=3*)

return_locations(*lam*, *allcoef*, *xindx*, *yindx*, *order=3*)
Calculates the detector coordinates of lenslet located at `xindx`, `yindx` for desired wavelength `lam`

Parameters **lam: float**

Wavelength in nm

allcoef: list of lists floats

Polynomial coefficients of wavelength solution

xindx: int

X index of lenslet in lenslet array

yindx: int

Y index of lenslet in lenslet array

order: int

Order of polynomial wavelength solution

Returns interp_x: float

X coordinate on the detector

interp_y: float

Y coordinate on the detector

return_locations_short(*coef*, *xindx*, *yindx*)
Returns the x,y detector location of a given lenslet for a given polynomial fit

Parameters **coef: lists floats**

Polynomial coefficients of fit for a single wavelength

xindx: int

X index of lenslet in lenslet array

yindx: int

Y index of lenslet in lenslet array

Returns interp_x: float

X coordinate on the detector

interp_y: float

Y coordinate on the detector

return_res(*lam*, *allcoef*, *xindx*, *yindx*, *order=3*, *lam1=None*, *lam2=None*)

Returns the spectral resolution and interpolated wavelength array

Parameters lam: float

Wavelength in nm

allcoef: list of lists floats

Polynomial coefficients of wavelength solution

xindx: int

X index of lenslet in lenslet array

yindx: int

Y index of lenslet in lenslet array

order: int

Order of polynomial wavelength solution

lam1: float

Shortest wavelength in nm

lam2: float

Longest wavelength in nm

Returns interp_lam: array

Array of wavelengths

R: float

Effective spectral resolution

savepixsol(*outdir='calibrations/'*)

Saves wavelength calibration file

Parameters outdir: String

Directory in which to put the file. The file is name PSFloc.fits and is a multi-extension FITS file, each extension corresponding to: 0. the list of wavelengths at which the calibration is done 1. a 2D ndarray with the X position of all lenslets 2. a 2D ndarray with the Y position of all lenslets 3. a 2D ndarray with the number of valid wavelengths for a given lenslet (some wavelengths fall outside of the detector area)

tools.locate_psflets.**corrval**(*coef*, *x*, *y*, *filtered*, *order*, *trimfrac=0.1*)

Return the negative of the sum of the middle XX% of the PSFlet spot fluxes (disregarding those with the most and the least flux to limit the impact of outliers). Analogous to the trimmed mean.

Parameters coef: list of floats

coefficients for polynomial transformation

x: ndarray

coordinates of lenslets

y: ndarray

coordinates of lenslets

filtered: ndarray

image convolved with gaussian PSFlet

order: int

order of the polynomial fit

trimfrac: float

fraction of outliers (high & low combined) to trim Default 0.1 (5% trimmed on the high end, 5% on the low end)

Returns score: float

Negative sum of PSFlet fluxes, to be minimized

tools.locate_psflets.**initcoef**(*order*, *scale=15.02*, *phi=2.0496941961364943*, *x0=0*, *y0=0*)

Create a set of coefficients including a rotation matrix plus zeros.

Parameters **order: int**

The polynomial order of the grid distortion

scale: float

The linear separation in pixels of the PSFlets. Default 13.88.

phi: float

The pitch angle of the lenslets. Default atan(2)

x0: float

x offset to apply to the central pixel. Default 0

y0: float

y offset to apply to the central pixel. Default 0

Returns coef: list of floats

A list of length (order+1)*(order+2) to be optimized.

Notes

The list of coefficients has space for a polynomial fit of the input order (i.e., for order 3, up to terms like x**3 and x**2*y, but not x**3*y). It is all zeros in the output apart from the rotation matrix given by scale and phi.

tools.locate_psflets.**locatePSFlets**(*inImage*, *polyorder=2*, *sig=0.7*, *coef=None*, *trimfrac=0.1*, *phi=2.0496941961364943*, *scale=15.02*, *nlens=108*)

function locatePSFlets takes an Image class, assumed to be a monochromatic grid of spots with read noise and shot noise, and returns the esimated positions of the spot centroids. This is designed to constrain the domain of the PSF-let fitting later in the pipeline.

Parameters **imImage: Image class**

Assumed to be a monochromatic grid of spots

polyorder: float

order of the polynomial coordinate transformation. Default 2.

sig: float

standard deviation of convolving Gaussian used for estimating the grid of centroids. Should be close to the true value for the PSF-let spots. Default 0.7.

coef: list

initial guess of the coefficients of polynomial coordinate transformation

trimfrac: float

fraction of lenslet outliers (high & low combined) to trim in the minimization. Default 0.1 (5% trimmed on the high end, 5% on the low end)

Returns x: 2D ndarray

Estimated spot centroids in x.

y: 2D ndarray

Estimated spot centroids in y.

good:2D boolean ndarray

True for lenslets with spots inside the detector footprint

coef: list of floats

List of best-fit polynomial coefficients

Notes

the coefficients, if not supplied, are initially set to the known pitch angle and scale. A loop then does a quick check to find reasonable offsets in x and y. With all of the first-order polynomial coefficients set, the optimizer refines these and the higher-order coefficients. This routine seems to be relatively robust down to per-lenslet signal-to-noise ratios of order unity (or even a little less).

Important note: as of now (09/2015), the number of lenslets to grid is hard-coded as 1/10 the dimensionality of the final array. This is sufficient to cover the detector for the fiducial lenslet spacing.

`tools.locate_psflets.`**`transform`**(*x*, *y*, *order*, *coef*)

Apply the coefficients given to transform the coordinates using a polynomial.

Parameters **x: ndarray**

Rectilinear grid

y: ndarray of floats

Rectilinear grid

order: int

Order of the polynomial fit

coef: list of floats

List of the coefficients. Must match the length required by order = (order+1)*(order+2)

Returns _x: ndarray

Transformed coordinates

_y: ndarray

Transformed coordinates

4.10 tools.par_utils module

class tools.par_utils.**Consumer**(*task_queue*, *result_queue*)
Bases: multiprocessing.context.Process

run()

class tools.par_utils.**Task**(*index*, *func*, *args*)
Bases: object

4.11 tools.plotting module

tools.plotting.**plotKernels**(*par*, *kernels*, *locations*, *plot=False*)
Make plots of all the kernels

4.12 tools.postprocessing module

tools.postprocessing.**process_SPC_IFS**(*par*, *psf_time_series_folder*, *offaxis_psf_filename*, *planet_radius=<Quantity 71492000.0 m>*, *mean_contrast=1e-08*, *ref_star_T=<Quantity 9377.0 K>*, *ref_star_Vmag=2.37*, *target_star_T=<Quantity 5887.0 K>*, *target_star_Vmag=5.03*, *lamc=770.0*, *BW=0.18*, *Nlam=45*, *n_ref_star_imgs=30*, *tel_pupil_area=<Quantity 3.650265060424805 m2>*, *outdir_time_series='OS5'*, *outdir_detector='OS5/OS5_detector'*, *outdir_average='OS5/OS5_average'*, *process_cubes=True*, *process_offaxis=True*, *process_detector=True*, *take_averages=True*)
Process SPC PSF cubes from J. Krist through the IFS

4.13 tools.reduction module

tools.reduction.**GPImethod2**(*par*, *name*, *ifsimage*)
Basic cube reduction using an IFS image and a wavecal cube Equivalent to method 2 in the IDL primitive 'pisces_assemble_spectral_datacube'

Parameters par: Parameter instance

Contains all IFS parameters

name: string

Name that will be given to final image, without fits extension

ifsimage: Image instance of IFS detector map, with optional inverse variance

Returns **cube** : 3D array

Return the reduced cube from the original IFS image

tools.reduction.**calculateWaveList**(*par*, *lam_list=None*)

Computes the wavelength lists corresponding to the center and endpoints of each spectral bin. Wavelengths are separated by a constant value in log space. Number of wavelengths depends on spectral resolution.

Parameters **par: Parameter instance**

Contains all IFS parameters

lam_list: list of wavelengths

Usually this is left to None. If so, we use the wavelengths used for wavelength calibration. Otherwise, we could decide to focus on a smaller/larger region of the spectrum to retrieve. The final processed cubes will have bins centered on lam_midpts

Returns lam_midpts: list of floats

Wavelengths at the midpoint of each bin

lam_endpts: list of floats

Wavelengths at the edges of each bin

tools.reduction.**densifiedSimpleReduction**(*par*, *name*, *ifsimage*, *ratio=10.0*)

Use the same method as the 'simple' method but interpolate the original IFS map over a grid with finer sampling

Parameters **par: Parameter instance**

Contains all IFS parameters

name: string

Name that will be given to final image, without fits extension

ifsimage: Image instance of IFS detector map, with optional inverse variance

ratio: int

Ratio by which the original image is densified.

Returns **cube** : 3D array

Return the reduced cube from the original IFS image

tools.reduction.**fit_cutout**(*subim*, *psflets*, *mode='lstsq'*)

Fit a series of PSFlets to an image, recover the best-fit coefficients. This is currently little more than a wrapper for np.linalg.lstsq, but could be more complex if/when we regularize the problem or adopt some other approach.

Inputs: 1. subim: 2D nadarray, microspectrum to fit 2. psflets: 3D ndarray, first dimension is wavelength. psflets[0]

must match the shape of subim.

3.**mode: string, method to use. Currently limited to lstsq (a** simple least-squares fit using linalg.lstsq), this can be expanded to include an arbitrary approach.

Returns: 1. coef: the best-fit coefficients (i.e. the microspectrum).

Note: this routine may also return the covariance matrix in the future. It will depend on the performance of the algorithms and whether/how we implement regularization.

tools.reduction.**fitspec_intpix**(*par*, *im*, *PSFlet_tool*, *lamlist*, *delt_y=6*, *flat=None*, *smoothandmask=False*, *mode='gaussvar'*)

Optimal extraction routine

Parameters **par** : Parameter instance

Contains all IFS parameters

im: Image instance

IFS image to be processed.

PSFlet_tool: PSFLet instance

Inverse wavelength solution that is constructed during wavelength calibration

lamlist: list of floats

List of wavelengths to which each microspectrum is interpolated.

delt_y: int

Width in pixels of each microspectrum in the cross-dispersion direction

flat:

Whether a lenslet flatfield is used (not implemented yet)

smoothandmask: Boolean

Whether to smooth and mask bad pixels

Returns image: Image instance

Reduced cube in the image.data field

tools.reduction.**fitspec_intpix_np**(*par*, *im*, *PSFlet_tool*, *lamlist*, *delt_y=6*, *smoothandmask=False*)

Original optimal extraction routine in Numpy from T. Brand

Parameters **par** : Parameter instance

Contains all IFS parameters

im: Image instance

IFS image to be processed.

PSFlet_tool: PSFLet instance

Inverse wavelength solution that is constructed during wavelength calibration

lamlist: list of floats

List of wavelengths to which each microspectrum is interpolated.

delt_y: int

Width in pixels of each microspectrum in the cross-dispersion direction

Returns image: Image instance

Reduced cube in the image.data field

tools.reduction.**fitspec_intpix_np_old**(*im*, *PSFlet_tool*, *lam*, *delt_x=7*, *header=SIMPLE = T / conforms to FITS standard BITPIX = 8 / array data type NAXIS = 0 / number of array dimensions EXTEND = T*)

tools.reduction.**get_cutout**(*im*, *x*, *y*, *psflets*, *dy=3*)

Cut out a microspectrum for fitting. Return the inputs to linalg.lstsq or to whatever regularization scheme we adopt. Assumes that spectra are dispersed in the -y direction.

Parameters **im: Image intance**

Image containing data to be fit

x: float

List of x centroids for each microspectrum

y: float

List of y centroids for each microspectrum

psflets: PSFLet instance

Typically generated from polychrome step in wavelength calibration routine

dy: int

vertical length to cut out, default 3. This is the length to cut out in the +/-y direction; the lengths cut out in the +x direction (beyond the shortest and longest wavelengths) are also dy.

Returns subim: 2D array

A flattened subimage to be fit

psflet_subarr: 2D ndarray

first dimension is wavelength, second dimension is spatial, and is the same shape as the flattened subimage.

Notes

Both subim and psflet_subarr are scaled by the inverse standard deviation if it is given for the input Image. This will make the fit chi2 and properly handle bad/masked pixels.

`tools.reduction.`**`intOptimalExtract`**(*par*, *name*, *IFSimage*)

Calls the optimal extraction routine

Parameters **par** : Parameter instance

Contains all IFS parameters

name: string

Path & name of the output file

IFSimage: Image instance

Image instance of input image. Can have a .ivar field for a variance map.

Notes

A cube is also written at par.SimResults/name.fits

`tools.reduction.`**`lstsqExtract`**(*par*, *name*, *ifsimage*, *ivar=False*)

Least squares extraction, inspired by T. Brandt and making use of some of his code.

Parameters **par: Parameter instance**

Contains all IFS parameters

name: string

Name that will be given to final image, without fits extension

ifsimage: Image instance of IFS detector map, with optional inverse variance

Returns **cube** : 3D array

Return the reduced cube from the original IFS image

tools.reduction.**simpleReduction**(*par*, *name*, *ifsimage*)

Basic cube reduction using an IFS image and a wavecal cube Equivalent to method 1 in the IDL primitive 'pisces_assemble_spectral_datacube'

Parameters **par: Parameter instance**

Contains all IFS parameters

name: string

Name that will be given to final image, without fits extension

ifsimage: Image instance of IFS detector map, with optional inverse variance

Returns **cube** : 3D array

Return the reduced cube from the original IFS image

tools.reduction.**testReduction**(*par*, *name*, *ifsimage*)

Scratch routine to test various things.

Parameters **par: Parameter instance**

Contains all IFS parameters

name: string

Name that will be given to final image, without fits extension

ifsimage: Image instance of IFS detector map, with optional inverse variance

Returns **cube** : 3D array

Return the reduced cube from the original IFS image

4.14 tools.rotate module

tools.rotate.**Rotate**(*image*, *phi*, *clip=True*, *order=1*)

Rotate the input image by phi about its center. Do not resize the image, but pad with zeros. Function originally from Tim Brandt

Parameters **image** : 2D square array

Image to rotate

phi : float

Rotation angle in radians

clip : boolean (optional)

Clip array by sqrt(2) to remove fill values? Default True.

order : integer (optional)

Order of interpolation when rotating. Default is 1.

Returns imageout: 2D array

Rotated image of the same shape as the input image, with zero-padding

4.15 tools.spectrograph module

tools.spectrograph.**createAllWeightsArray**(*par*, *locations*)
Creates weights for bilinear interpolation

Parameters **par** : Parameter instance

Contains all IFS parameters

locations : 2D ndarray, Nx2

Array of normalized locations on the detector, .

Returns **detectorFrame** : 2D array

Return the detector frame with correct pixel scale.

tools.spectrograph.**distort**(*fx*, *fy*, *lam*)
Apply the distortion and dispersion from pre-determined polynomial This was estimated from Zemax by Qian Gong and Jorge Llop and needs to be revisited

Parameters **fx,fy** : float

Distance between a lenslet and the center of the detector in millimeter.

lam : float

Wavelength in microns.

Returns **x,y** : float

Distance from center of detector at which image from a lenslet falls in mm.

tools.spectrograph.**loadKernels**(*par*, *wavel*)
Loads the kernels that represent the PSFs at different locations on the detector

Parameters **par** : Parameter instance

Contains all IFS parameters

wavel : float

Wavelength at which the kernels are needed

Returns kernels: array of 2D arrays

Represents each 2D kernel at each location

locations: Nx2 ndarray

Location coordinates in detector position ratio (0,0) is bottom left, (1,1) is top right

tools.spectrograph.**selectKernel**(*par*, *lam*, *refWaveList*, *kernelList*)
Select the correct kernel for the current wavlength

Parameters **par** : Parameter instance

Contains all IFS parameters

lam : float

Wavelength at which we want the get the kernel, in microns

refWaveList : list of floats

Wavelengths at which the kernels are defined

kernelList : list of 3D ndarrays

List of the kernels cubes at the wavelengths above

Returns kernels: array of 2D arrays

Represents each 2D kernel at each location for that wavlength

locations: Nx2 ndarray

Location coordinates in detector position ratio (0,0) is bottom left, (1,1) is top right

4.16 tools.wavecal module

tools.wavecal.**buildcalibrations**(*par*, *filelist=None*, *lamlist=None*, *order=3*, *inspect=True*, *genwavelengthsol=True*, *makehiresPSFlets=True*, *savehiresimages=True*, *borderpix=4*, *upsample=3*, *nsubarr=3*, *parallel=True*)

Master wavelength calibration function

Parameters **par** : Parameter instance

Contains all IFS parameters

filelist: list of strings

List of the fits files that contain the monochromatic calibration files

lamlist: list of floats

Wavelengths in nm at which the files are taken

order: int

Order of the polynomial used to fit the PSFLet positions across the detector

genwavelengthsol: Boolean

If True, generate the wavelength calibration. Creates a text file with all polynomial coefficients that best fit the PSFLet positions at each wavelength. If False, then load an already-generated file.

inspect: Boolean

Whether or not to create PNG files that overlay PSFLet fitted position on the monochromatic pictures, to visually inspect the fitting results

makehiresPSFlets: Boolean

Whether or not to do a high-resolution fitting of the PSFs, using the sampling diversity. This requires high-SNR monochromatic images.

savehiresimages: Boolean

Whether to save fits files with the high-res PSFLets

borderpix: int

Number of pixels that are not taken into account towards the edges of the detector

upsample: int

Upsampling factor for each high-resolution PSFLet

nsubarr: int

Detector will be divided into nsubarr x nsubarr regions. A high-resolution PSFLet will be determined in each region from the average of all PSFLets within that region

parallel: Boolean

Whether or not to parallelize the computation for the high-resolution PSFLet and polychrome computation. The wavelength calibration step cannot be parallelized since each wavelength uses the previous wavelength solution as a guess input.

Notes

This function generates all the files required to process IFS cubes: lamsol.dat: contains a list of the wavelengths and the polynomial coefficients that

describe the X,Y positions of all lenslets on the detector as a function of lenslet position on the lenslet array.

polychromekeyRXX.fits: where XX is replaced by the spectral resolution defined in the parameters file. This is a multi-extension fits file with: - a list of the central wavelengths at which the final cube will be reduced to - an array of the X positions of all lenslets - an array of the Y positions of all lenslets - an array of booleans indicating whether that lenslet is good or not (e.g. when it is outside of the detector area)

polychromeRXX.fits: list of 2D arrays of size Npix x Npix which each contain a map with the high-resolution lenslet PSFLets put in their correct position for all the wavelengths that we want in the output cube. Each PSFLet in each wavelength slice is used for least-squares fitting.

PSFLoc.fits: nsubarr x nsubarr array of 2D high-resolution PSFLets at each location in the detector.

polychromeRXX.fits and PSFLoc.fits are only generated if makehiresPSFlets is True.

`tools.wavecal.`**`computeWavecal`**(*par*, *lamlist=None*, *filelist=None*, *order=3*)

Computes a wavelength calibration from a set of fits files Uses Tim Brandt's locate_PSFlets routine with an initial guess. The consecutive solutions use information from previous solutions to improve the initial guess. The process can be optimized somewhat by reducing the number of lenslets within the locatePSFlets function (could make this a parameter!) lamlist and filelist can be defined in the parameters in which case they don't need to be set

Parameters par: Parameter instance

lamlist: list of floats

Wavelengths in nm

filelist: list of filenames

List of fits files to open in corresponding order

order: int

Order of 2d polynomial to be fitted to lenslets

Notes

Obsolete now

`tools.wavecal.`**`createPolychrome`**(*par*)

To be run after generating a wavelength calibration set. This function constructs a cube of (lam_max-lam_min)/par.dlam depth, in which each slice is a monochromatic map at that wavelength. If the wavelength calibration doesn't contain all the required wavelengths, interpolate between wavelengths using map_coordinates.

In the end, one should be able to make a cutout at a given location and get a cube with all the psflets for that lenslet.

For now, this only works with simulated, noiseless data in order to get almost perfect PSFs. In practice, we will have to create this cube from data. For now, already assumes that all wavelengths are available (need to compute these)

tools.wavecal.**createWavecalFiles**(*par*, *lamlist*)
Creates a set of monochromatic IFS images to be used in wavelength calibration step

tools.wavecal.**do_inspection**(*par*, *image*, *xpos*, *ypos*, *lam*)

tools.wavecal.**gethires**(*x*, *y*, *image*, *upsample=5*, *nsubarr=5*, *npix=13*, *renorm=True*)
Build high resolution images of the undersampled PSF using the monochromatic frames.

Inputs: 1.

tools.wavecal.**make_polychrome**(*lam1*, *lam2*, *hires_arrs*, *lam_arr*, *psftool*, *allcoef*, *xindx*, *yindx*, *ydim*, *xdim*, *upsample=5*, *nlam=10*)

4.17 Module contents

CHAPTER

FIVE

UNITTESTS MODULE

unitTests.**testCreateFlatfield**(*par*, *pixsize=0.1*, *npix=512*, *pixval=1.0*, *outname='flatfield.fits'*)
Creates a polychromatic flatfield

Parameters **par** : Parameter instance

Contains all IFS parameters

pixsize: float

Pixel scale (lam/D)

npix: int

Each input frame has a pixel size npix x npix

pixval: float

Each input frame has a unform value pixval

unitTests.**testCutout**(*par*, *fname*, *lensnum=0*)
Testing the cutout function

unitTests.**testFitCutout**(*par*, *fname*, *lensnum*, *mode='lstsq'*, *ivar=False*)
Testing the fit_cutout function

unitTests.**testGenPixSol**(*par*)

unitTests.**testLoadKernels**(*par*)
Make sure the kernel interpolation with wavelength makes sense

Part IV

Indices and tables

- genindex
- modindex
- search

t

u

P

R

S

T

U

W