VO2012

Virtual Observations 2012

WERKCOLLEGE 5: ASTRONOMICAL DATA PROCESSING

Document identifier:	**VO2012-W5-01**
Date:	**October 1, 2012**
Activity:	
Document status:	
Document link:	

Abstract: This werkcollege will go through basic astronomical data processing of images, use of FITS files, python and R.

1. PHOTOMETRY

1. Write down a formula which expresses which fraction of photons emitted by a star are registered on the detector. Assume that photons have wavelength λ and travel through ISM, mirrors, a filter and then hit the detector. Assume that mirrors are identical.
2. Using the formula from previous exesize compute an actual fraction of photons which will reach detector for a chosen wavelength and some typical numbers for the other components.
3. What is the general formula to convert between fluxes in Vega and AB magnitudes?
4. Using the data sources from lecture/werkcollege on SDSS find a conversion between AB and Vega systems for SDSS.

2. INTRODUCTION TO SEXTRACTOR

2.1. CONFIGURATION

Create a simple configuration files for Sextractor:

```
>sex -d > default.sex
```

or download these files from pages of the lecture. You will need `default.sex`, `default.param`, one of convolution matrixes `default.conv` and neural network weights `default.nnw`, if you are going to use sextractor for classification.

2.1.1. DEFAULT.SEX

Following parameters should be checked before running sextractor:

```
CATALOG_NAME    temp.cat        # name of the output catalog
CATALOG_TYPE    FITS_1.0        # "NONE","ASCII_HEAD","ASCII","FITS_1.0"
                                # or "FITS_LDAC"

PARAMETERS_NAME default.param   # name of the file containing catalog contents

#------------------------------- Extraction ----------------------------------

DETECT_TYPE     CCD             # "CCD" or "PHOTO" )
DETECT_MINAREA  4               # minimum number of pixels above threshold
DETECT_THRESH   2.0             # <sigmas> or <threshold>,<ZP> in mag.arcsec-2

FILTER          N               # apply filter for detection ("Y" or "N")?
FILTER_NAME     default.conv    # name of the file containing the filter
```

The output of the sextraction will be put into `temp.cat` in the format you've selected.

2.1.2. DEFAULT.PARAM

```
NUMBER   # running number of object

FLUX_ISO     # flux
FLUXERR_ISO  # with error
MAG_ISO      # and magnitude
MAGERR_ISO   # with error


X_IMAGE      # coordinates of
Y_IMAGE      # the detected object, in px
ALPHA_J2000  # and coordinates in J2000
DELTA_J2000  #

CLASS_STAR   # classification star(1)/galaxy(0)
```

2.2. RUNNING SEXTRACTOR ON IMAGE

The simplest way to run sextractor is to have all configuration files described above in the same directory. In this case

```
>sex 2MASSJ.fits
----- SExtractor 2.5.0 started on 2010-10-05 at 06:50:56 with 1 thread

Measuring from: "Unnamed"  / 512 x 1024 / 32 bits FLOATING POINT data
(M+D) Background: 0.118312   RMS: 1.05576    / Threshold: 3.16729
Objects: detected 114       / sextracted 30
> All done (in 0 s)
```

30 objects were sextracted and stored in `temp.cat` file.

Try to run sextractor on the file with overcrowded field changing minimum detection area and threshold.

2.3. Coordinates and magnitudes

As you can see from the result of sextraction, there is no proper magnitudes (as there is no zero point for magnitudes). Run sextractor on the crowded image with `DETECT_MINAREA=6` and `DETECT_THRESH=5.0` Using the result find an area on the sky and select the same area from 2MASS catalog. Assuming that in the wide range of magnitudes the dependence is linear, find zero point and put your magnitudes on 2MASS magnitudes. Select from sextracted catalog only stars with the best photometry.

3. Introduction to FITS

In this introducstion to FITS file format we will work with pyfits library of python and FITSio library of R.

3.1. Pyfits

The task is to learn basic operations with FITS files: reading, modifying, writing a new image/table.

3.1.1. Read a FITS image

To read a FITS image load pyfits package and use function `read`

```
>>> import pyfits
>>> hdu=pyfits.open("2MASSJ.fits")
>>> hdu.info()
Filename: 2MASSJ.fits
No.     Name          Type       Cards   Dimensions    Format
0     PRIMARY      PrimaryHDU       28   (512, 1024)    float32
```

with `info()` function you can list all HDU units in the file, see their dimensons and data type. In this case image is in primary HDU.

Read the header of the file:

```
>>> hdu[0].header.ascardlist()
SIMPLE   =                    T / file does conform to FITS standard
BITPIX   =                  -32 / number of bits per data pixel
NAXIS    =                    2 / number of data axes
NAXIS1   =                  512 / length of data axis 1
NAXIS2   =                 1024 / length of data axis 2
BZERO    =                   0. / PhysValue = BZERO + BSCALE * ArrayValue
BSCALE   =                   1. / PhysValue = BZERO + BSCALE * ArrayValue
COMMENT   Standard WCS reduction:
CRVAL1   =          11.88798983 / WCS Ref value (RA in decimal degrees)
CRVAL2   =         -25.33718377 / WCS Ref value (DEC in decimal degrees)
CRPIX1   =                256.5 / WCS Coordinate reference pixel
CRPIX2   =                336.5 / WCS Coordinate reference pixel
CD1_1    = -0.000277777783923599 / WCS Coordinate scale matrix
CD1_2    = 1.78947724260645E-08 / WCS Coordinate scale matrix
CD2_1    = 1.78947724260645E-08 / WCS Coordinate scale matrix
CD2_2    = 0.000277777783923599 / WCS Coordinate scale matrix
CTYPE1   = 'RA---SIN'           / WCS Coordinate type
CTYPE2   = 'DEC--SIN'           / WCS Coordinate type
EQUINOX  =                2000. / Equinox
RADESYS  = 'FK5     '           / Coordinate system
MJD-OBS  =     51381.3422136574 / Modified Julian Date at start of observation
EPOCH    =     1999.55329832624 /  Epoch in Julian Year at start of observation
CUNIT1   = 'DEG     '           / RA Coordinate Unit
CUNIT2   = 'DEG     '           / DEC Coordinate Unit
CDELT1   =      0.0002777777845 / WCS Coordinate scale matrix
CDELT2   =      0.0002777777845 / WCS Coordinate scale matrix
CNPIX1   =                    0 / New CNPIX1
CNPIX2   =                    0 / New CNPIX2
```

and change the format of the output:

```
>>> for kw in hdu[0].header.ascardlist().keys():
...     print kw,":",hdu[0].header[kw]
...
SIMPLE : True
BITPIX : -32
NAXIS : 2
NAXIS1 : 512
NAXIS2 : 1024
BZERO : 0.0
```

```
BSCALE : 1.0
COMMENT :   Standard WCS reduction:
CRVAL1 : 11.88798983
CRVAL2 : -25.33718377
CRPIX1 : 256.5
CRPIX2 : 336.5
CD1_1 : -0.000277777783924
CD1_2 : 1.78947724261e-08
CD2_1 : 1.78947724261e-08
CD2_2 : 0.000277777783924
CTYPE1 : RA---SIN
CTYPE2 : DEC--SIN
EQUINOX : 2000.0
RADESYS : FK5
MJD-OBS : 51381.3422137
EPOCH : 1999.55329833
CUNIT1 : DEG
CUNIT2 : DEG
CDELT1 : 0.0002777777845
CDELT2 : 0.0002777777845
CNPIX1 : 0
CNPIX2 : 0
```

Note, that you can access the value of the header using keyword.

Copy an image to a new array and query the size and the data type of the image and image elements

```
>>> image=hdu[0].data
>>> image.shape
(1024, 512)
>>> image.dtype.name
'float32'
```

You can see the whole list of functions you can use on image typing `help(image)`, for example

```
>>> image.min()
-6.290802
>>> image.max()
8683.7422
>>> image.mean()
3.5383291244506836
>>> image.std()
49.280182395696599
```

3.1.2. Modify FITS file

First of all, let us modify an existing field in the header, for example, the reference system

```
>>> hdu[0].header.update('RADESYS','ICRS')
```

or

```
>>> hdu[0].header['RADESYS']='ICRS'
```

or add new tag

```
>>> hdu[0].header.update('OBSERVER','me!')
```

The same can be done with image, but here you should be much more careful to save all changes in the header

```
>>> image.min()
-6.290802
>>> image=image+abs(image.min())
>>> image.min()
0.0
>>> hdu[0].data=image
>>> hdu[0].header.update('BZERO','-6.290802')
```

And, finally, you can save all you've done

```
>>> hdu.writeto("newfile.fits")
```

3.1.3. Create a new image

Let us try to create a new image - 2-dimensional array of (100,100) with exponential distribution of flux inside

```
>>> import numpy, pyfits
>>> from math import exp
>>> s=numpy.zeros((100,100))
>>> for i in range(100):
...  for j in range(100):
...   s[i][j]=exp(-(i-50)*(i-50)/5.0)*exp(-(j-50)*(j-50)/10.0)
...
>>> hdu=pyfits.PrimaryHDU(s)
>>> hdulist=pyfits.HDUList([hdu])
>>> hdulist.writeto("exp.fits")
```

See the resulting file in skycat or Aladin.

3.1.4. Write a table to FITS file

FITS files are very effective way to store table data - the header defines format of the data and the data themselves are compressed.

First we will create a table with 4 columns with identifiers, coordinates and V magnitudes for 5 objects:

```
>>> import numpy, pyfits
>>> inp1=numpy.array(['USNO-B1.0 1525-00212637','QSO B2357-003A','MCG+00-01-015',
'LBQS 2357-0014','TYC 5253-332-1'])
>>> inp2=numpy.array([000.00000,359.99708,000.032671,000.0498754,359.9990329])
>>> inp3=numpy.array([00.00000,-00.03417,-00.040667,+00.0403422,-00.0653422])
>>> inp4=numpy.array([-99,18.0,-99,17.8,10.37])
>>> c1=pyfits.Column(name='ID',format='30A',array=inp1)
>>> c2=pyfits.Column(name='RA',format='D',array=inp2)
>>> c3=pyfits.Column(name='DEC',format='D',array=inp3)
>>> c4=pyfits.Column(name='V',format='E',array=inp4)
```

- see Table 1 for format codes of FITS table. Then put columns together and create a new FITS table

```
>>> cs=pyfits.ColDefs([c1,c2,c3,c4])
>>> table_hdu=pyfits.new_table(cs)
```

and, finally, create a new FITS file which will consist of 2 HDUs: primary and the table

```
>>> hdu=pyfits.PrimaryHDU()
>>> hdulist=pyfits.HDUList([hdu])
>>> hdulist.append(table_hdu)
>>> hdulist.verify()
>>> hdulist.writeto("table.fits")
```

Let us check it:

```
>>> hdu=pyfits.open("table.fits")
>>> hdu.info()
Filename: table.fits
No.    Name         Type      Cards   Dimensions   Format
0    PRIMARY     PrimaryHDU       4  ()            uint8
1                BinTableHDU     16  5R x 4C       [30A, D, D, E]
```

You can see not only the size of the table but format of rows as well. Let us see the data in the table:

Table 1: FITS table data types

FITS format code	Description	8-bit bytes
L	logical (Boolean)	1
X	bit	*
B	Unsigned byte	1
I	16-bit integer	2
J	32-bit integer	4
K	64-bit integer	4
A	character	1
E	single precision floating point	4
D	double precision floating point	8
C	single precision complex	8
M	double precision complex	16
P	array descriptor	8

```
>>> data=hdu[1].data
>>> data
FITS_rec([('USNO-B1.0 1525-00212637', 0.0, 0.0, -99.0),
       ('QSO B2357-003A', 359.99707999999998, -0.034169999999999999, 18.0),
       ('MCG+00-01-015', 0.032670999999999999, -0.040667000000000002, -99.0),
       ('LBQS 2357-0014', 0.0498754, 0.040342200000000002, 17.799999),
       ('TYC 5253-332-1', 359.99903289999997, -0.065342200000000003, 10.37)],
      dtype=[('ID', '|S30'), ('RA', '>f8'), ('DEC', '>f8'), ('V', '>f4')])
```

and browse it:

```
>>> res=data.field('V') > 17.0
>>> res
array([False,  True, False,  True, False], dtype=bool)
>>> faint=data[res]
>>> faint
FITS_rec([('QSO B2357-003A', 359.99707999999998, -0.034169999999999999, 18.0),
       ('LBQS 2357-0014', 0.0498754, 0.040342200000000002, 17.799999)],
      dtype=[('ID', '|S30'), ('RA', '>f8'), ('DEC', '>f8'), ('V', '>f4')])
```

The result is a subset of the whole table with $V > 17.0$ Of course, it is possible to access data as a “normal” array:

```
>>> data[0][0]
'USNO-B1.0 1525-00212637'
>>> data[3][3]
17.799999
```

3.2. Task - create a FITS table

From any table in your database create a FITS file with this table. Use python to access MySQL. (Use python program for cross-dentification from Werkcollege 1 as an example how to retrieve data).

4. FITS IN R

It is possible to read and write FITS images in R as well with the use of library FITSio.

```
R>library(FITSio)
R>?FITSio
```

The basic function is `readFITS`

```
> FI<-readFITS("2MASSJ.fits")
> names(FI)
[1] "imDat" "axDat" "hdr"
```

`FI$imDat` contains the image array:

```
> length(FI$imDat)
[1] 524288
```

`FI$imDat[,34]` will give you 34-th column of this array, for example.

`FI$axDat` - information about axis

```
> FI$axDat
  crpix     crval        cdelt  len    ctype  cunit
1 256.5  11.88799 0.0002777778  512 RA---SIN   DEG
2 336.5 -25.33718 0.0002777778 1024 DEC--SIN   DEG
```

and, finally, `FI$hdr` is a header

```
> FI$hdr
 [1] "SIMPLE"                "T"                     "BITPIX"
 [4] "-32"                   "NAXIS"                 "2"
 [7] "NAXIS1"                "512"                   "NAXIS2"
[10] "1024"                  "BZERO"                 "0."
[13] "BSCALE"                "1."                    "CRVAL1"
[16] "11.88798983"           "CRVAL2"                "-25.33718377"
[19] "CRPIX1"                "256.5"                 "CRPIX2"
[22] "336.5"                 "CD1_1"                 "-0.000277777783923599"
[25] "CD1_2"                 "1.78947724260645E-08"  "CD2_1"
[28] "1.78947724260645E-08"  "CD2_2"                 "0.000277777783923599"
[31] "CTYPE1"                "RA---SIN"              "CTYPE2"
[34] "DEC--SIN"              "EQUINOX"               "2000."
[37] "RADESYS"               "FK5"                   "MJD-OBS"
[40] "51381.3422136574"      "EPOCH"                 "1999.55329832624"
[43] "CUNIT1"                "DEG"                   "CUNIT2"
[46] "DEG"                   "CDELT1"                "0.0002777777845"
[49] "CDELT2"                "0.0002777777845"       "CNPIX1"
[52] "0"                     "CNPIX2"                "0"
[55] "END"
```

Try to see the image:

```
> image(FI$imDat, zlim=c(0,120))
```

Change limits on the flux (`zlim`) to see different details of the image. How the structure you see correlates with the interval of flux you select and why?

5. Proper Motions

The goal of this exercise is to build a simple pipeline which should find proper motions for the field obtained from 2 images with different epochs (2MASS and DSS1). The selection of software used in the task is up to the student (the use of topcat and Aladin is possible for visualization only!).

1. download 2MASS J and DSS1 R images from the page of werkcollege
2. make source extraction
3. find the centers of images using ALPHA_J2000 and DELTA_J2000 coordinates obtained in the previous step. Find radius of each field.
4. download regions of 2MASS and USNO-B1 catalogs from VizieR using parameters obtained in the step 3.
5. cross-identify your catalog for 2MASS J image with 2MASS-PSC catalog from VizieR, cross-identify your catalog for DSS1R image with 2MASS-PSC catalog and USNO-B1 catalog (for crossidentification between 2MASS-PSC catalog and your DSS1R image use a bigger radius)
6. use 2MASS-PSC as a reference catalog to find new coordinates for both images, use 2MASS-PSC catalog to find J magnitudes for your catalog, USNO-B1 catalog to find R1 magnitudes for your image (the linear regression is enough for all cases)
7. cross-identify final catalogs for your images (use a bigger radius - 3 arcsec, for example), calculate and draw proper motions (DSS1R - first epoch, 2MASS- second epoch) Before starting an exercise draw SADT and ER diagrams. On SADT diagram decide which software to use for each operation. It is possible to find coefficients for all equations in MySQL, R or python.