Données slitless CTIO

Time-stamp: <2019-02-12 18:40:35 ycopin>
Authors: Y. Copin J. Neveu

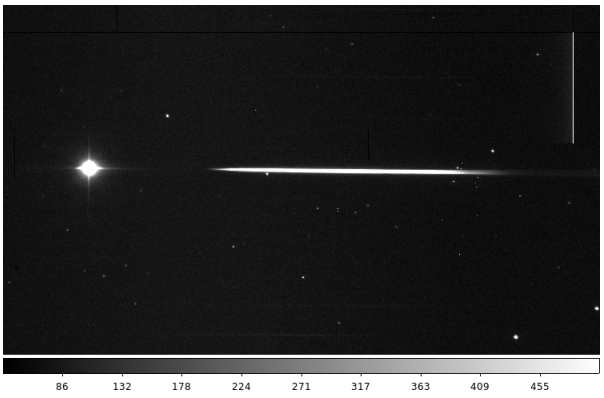
Generic information

Introduction

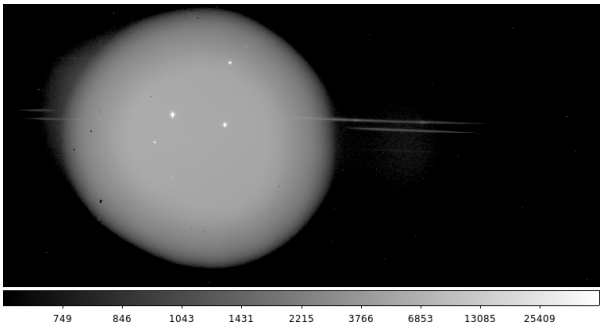
- Telescope: CTIO 0.9m
- Detector: Tek2K_3, pixsize=0.401 arcsec/pix
- Distance CCD-disperseur: environ 56 ± 5 mm

Preprocessed exposures:

- *reduc_20170530_130.fits*: CALSPEC HD111980 (x0, y0 = 791, 703 px) avec Ronchi 400 traits/mm (seeing 0"593, airmass X=1.114)



- *reduc_20170605_005.fits*: nébuleuse planétaire (x0, y0 = 745, 653 px) avec fond bizarre, hologramme ~360 traits/mm (seeing 0"699, airmass X=1.193)



reduc_20170530_130

Ref.: J. Neveu (08-11-18)

Voici en pièce jointe (*dispersion_relation.txt*) le fichier de sortie, une image du spectre et un tableau donnant la conversion pixels<->lambdas (nm), les pixels étant comptés à partir du centroid de l'ordre zéro estimé à:

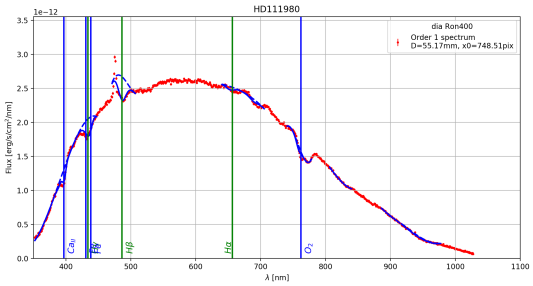
```
2018-11-08 15:03:24  Image      find_target          INFO
    X,Y target position in pixels: 748.005,684.256
```

```
2018-11-08 15:03:24  Image      turn_image           INFO
    Rotate the image with angle theta=-0.63 degree
```

je te donne l'angle de rotation au cas où aussi. Pour charger la relation de dispersion:

```
deltapix, lambdas = np.loadtxt('dispersion_relation.txt').T
```

Couverture spectrale (*reduc_20170530_130_spectrum.fits*): 352.0 -- 1027.5 nm



reduc_20170530_130_table.csv

Ref.: J. Neveu (12-02-19)

Voici ci-joint un fichier qui contient typiquement les éléments à mettre en forme:

- lambdas: la colonne des longueurs d'onde en nm
- Dx: la distance à l'ordre 0 en pixels selon x
- Dy: la distance à l'ordre 0 en pixels selon y
- les 7 dernières colonnes permettent d'alimenter le modèle de ma PSF qui est une Moffat - Gauss avec la classe ci-dessous, les titres des colonnes correspondant exactement aux noms des paramètres à donner à *.evaluate()*

```python
from astropy.modeling import Fittable1DModel, Parameter

class PSF1D(Fittable1DModel):
    inputs = ('x',)
    outputs = ('y',)

    amplitude_moffat = Parameter('amplitude_moffat', default=0.5)
    x_mean = Parameter('x_mean', default=0)
    gamma = Parameter('gamma', default=3)
    alpha = Parameter('alpha', default=3)
    eta_gauss = Parameter('eta_gauss', default=1)
    stddev = Parameter('stddev', default=1)
    saturation = Parameter('saturation', default=1)

    @staticmethod
    def evaluate(x, amplitude_moffat, x_mean, gamma, alpha, eta_gauss, stddev, satura
        rr = (x - x_mean) * (x - x_mean)
        rr_gg = rr / (gamma * gamma)
        # use **(-alpha) instead of **(alpha) to avoid overflow power errors due to h
        a = amplitude_moffat * ( (1 + rr_gg) ** (-alpha) + eta_gauss * np.exp(-(rr /
        if isinstance(x, float) or isinstance(x, int):
            if a > saturation:
```

```
23                 return saturation
24             elif a < 0.:
25                 return 0.
26             else:
27                 return a
28         else:
29             a[np.where(a >= saturation)] = saturation
30             a[np.where(a < 0.)] = 0.
31             return a
```

Generic model

Dispersion law: y [m] = order * rho [lines/m] * wavelength [m] * f [m] with:

- order = +1
- rho = 400 lines/mm
- f = 56 mm

so y [µm] = 1 * 400e3 * 56e-3 * l[µm] = 22.4e3 * l[µm]