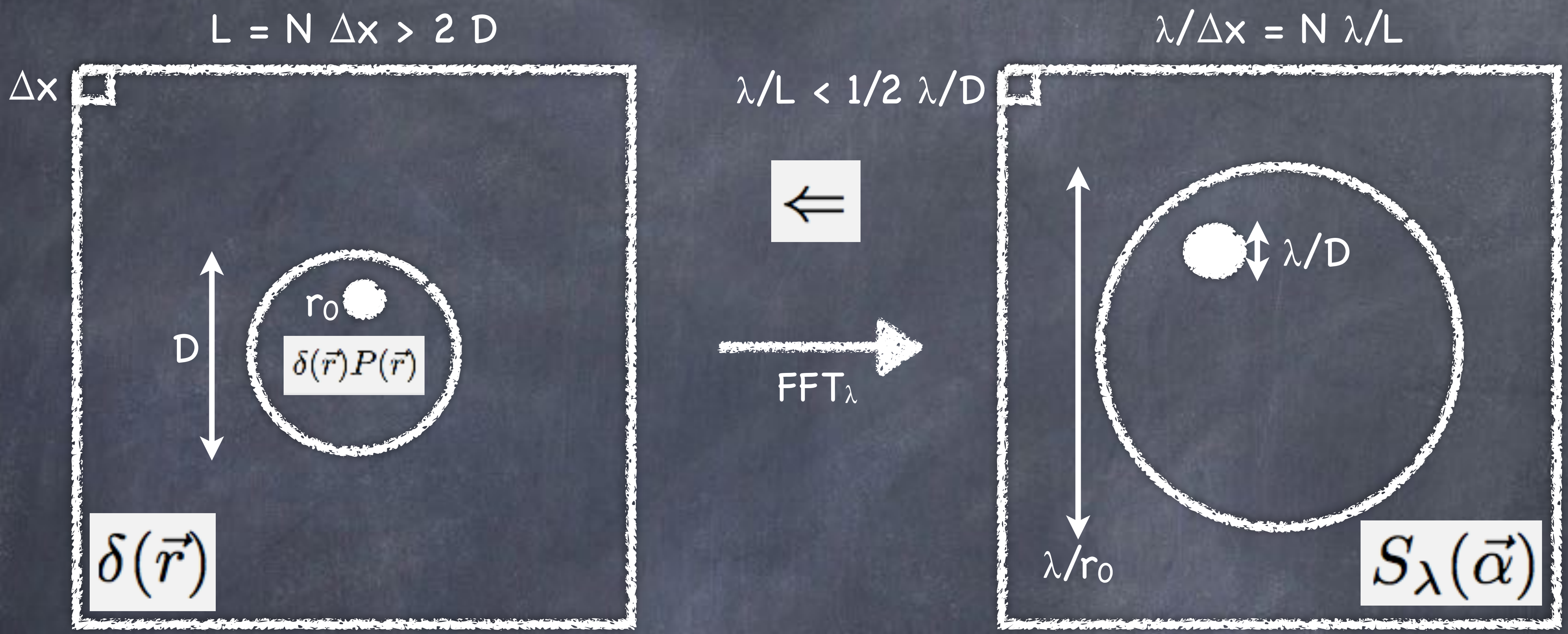
Images & turbulence - 19


$L = N \Delta x > 2D$
Δx
D
r_0
$\delta(\vec{r})P(\vec{r})$
$\delta(\vec{r})$
$\lambda/L < 1/2\ \lambda/D$
$\Leftarrow$
FFT_λ
$\lambda/\Delta x = N\ \lambda/L$
λ/D
λ/r_0
$S_\lambda(\vec{\alpha})$

Shannon (=Nyquist) criterium

=> the image pixel λ/L must be at most half the resolution element (resel!) λ/D
(in other words : one must has AT LEAST 2 image pixels per λ/D)

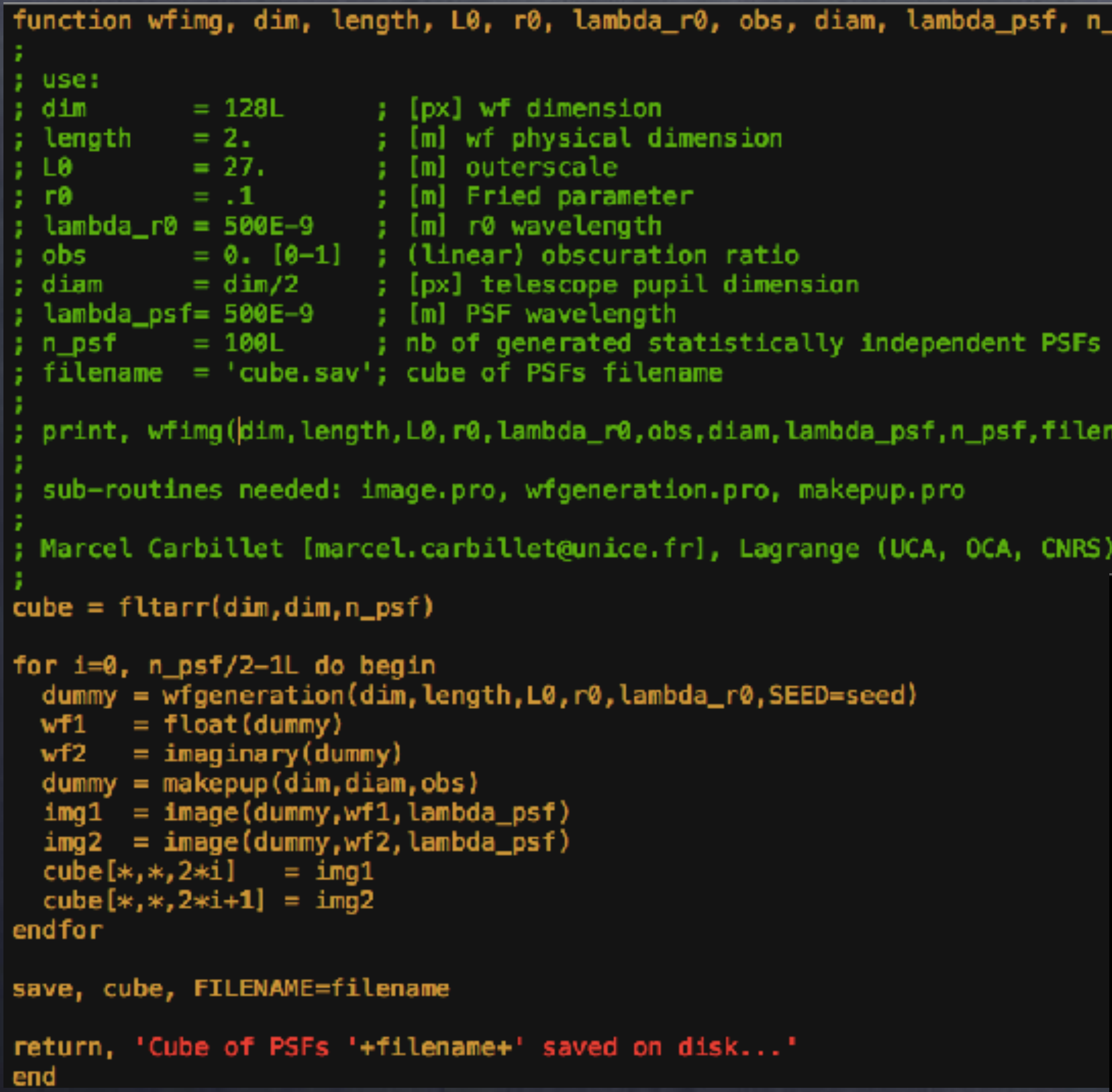
=> the simulated wavefronts must be at least twice the telescope diameter ($L>2D$)

In addition

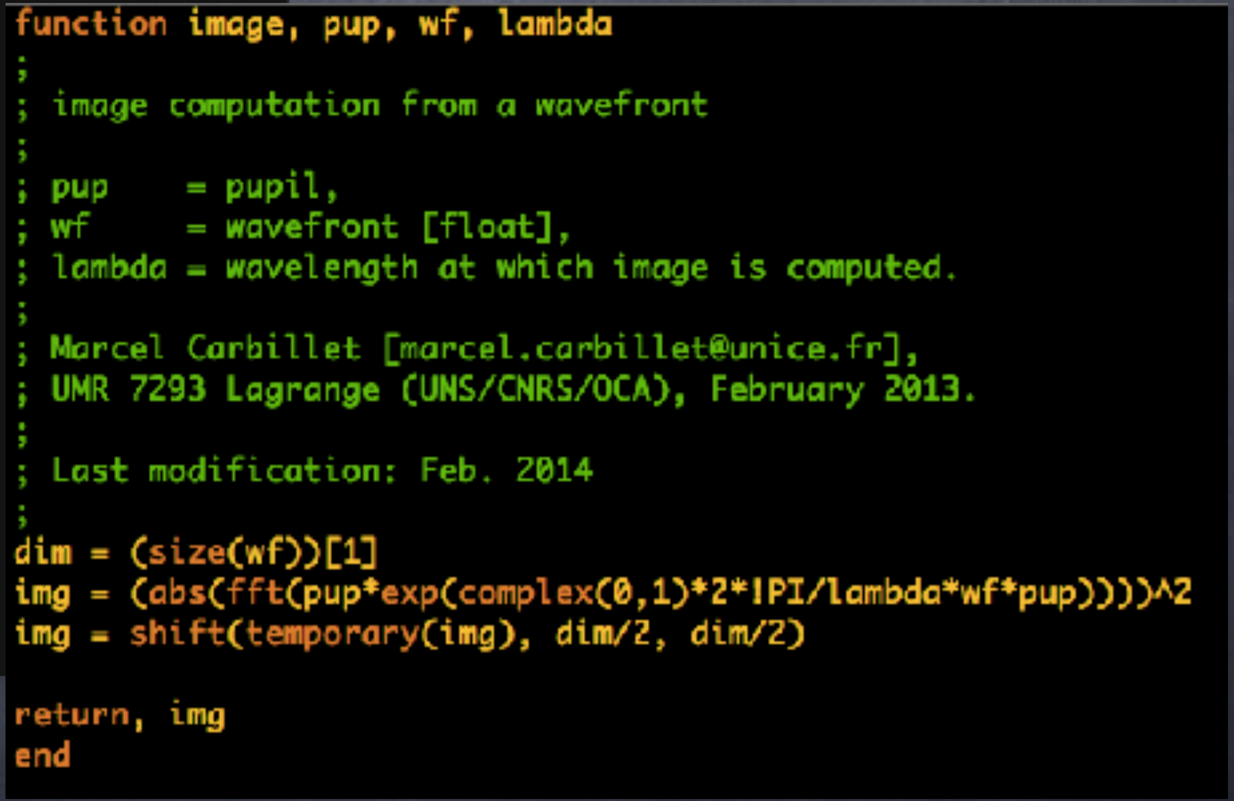
- λ/r_0 should be smaller than $\lambda/\Delta x$ (=> N large enough)

Images & turbulence - 20

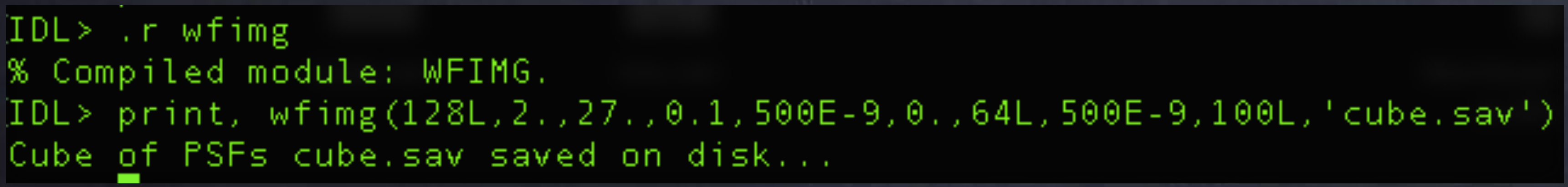
<ins>image formation:</ins>
1- cube of instantaneous PSFs (500nm & H-band)



```
function wfimg, dim, length, L0, r0, lambda_r0, obs, diam, lambda_psf, n_psf, filename
;
; use:
; dim        = 128L      ; [px] wf dimension
; length     = 2.        ; [m] wf physical dimension
; L0         = 27.       ; [m] outerscale
; r0         = .1        ; [m] Fried parameter
; lambda_r0 = 500E-9     ; [m] r0 wavelength
; obs        = 0. [0-1]  ; (linear) obscuration ratio
; diam       = dim/2     ; [px] telescope pupil dimension
; lambda_psf= 500E-9     ; [m] PSF wavelength
; n_psf      = 100L      ; nb of generated statistically independent PSFs
; filename   = 'cube.sav'; cube of PSFs filename
;
; print, wfimg(dim,length,L0,r0,lambda_r0,obs,diam,lambda_psf,n_psf,filename)
;
; sub-routines needed: image.pro, wfgeneration.pro, makepup.pro
;
; Marcel Carbillet [marcel.carbillet@unice.fr], Lagrange (UCA, OCA, CNRS), Feb. 2018.
;
cube = fltarr(dim,dim,n_psf)

for i=0, n_psf/2-1L do begin
  dummy = wfgeneration(dim,length,L0,r0,lambda_r0,SEED=seed)
  wf1   = float(dummy)
  wf2   = imaginary(dummy)
  dummy = makepup(dim,diam,obs)
  img1  = image(dummy,wf1,lambda_psf)
  img2  = image(dummy,wf2,lambda_psf)
  cube[*,*,2*i]   = img1
  cube[*,*,2*i+1] = img2
endfor

save, cube, FILENAME=filename

return, 'Cube of PSFs '+filename+' saved on disk...'
end
```

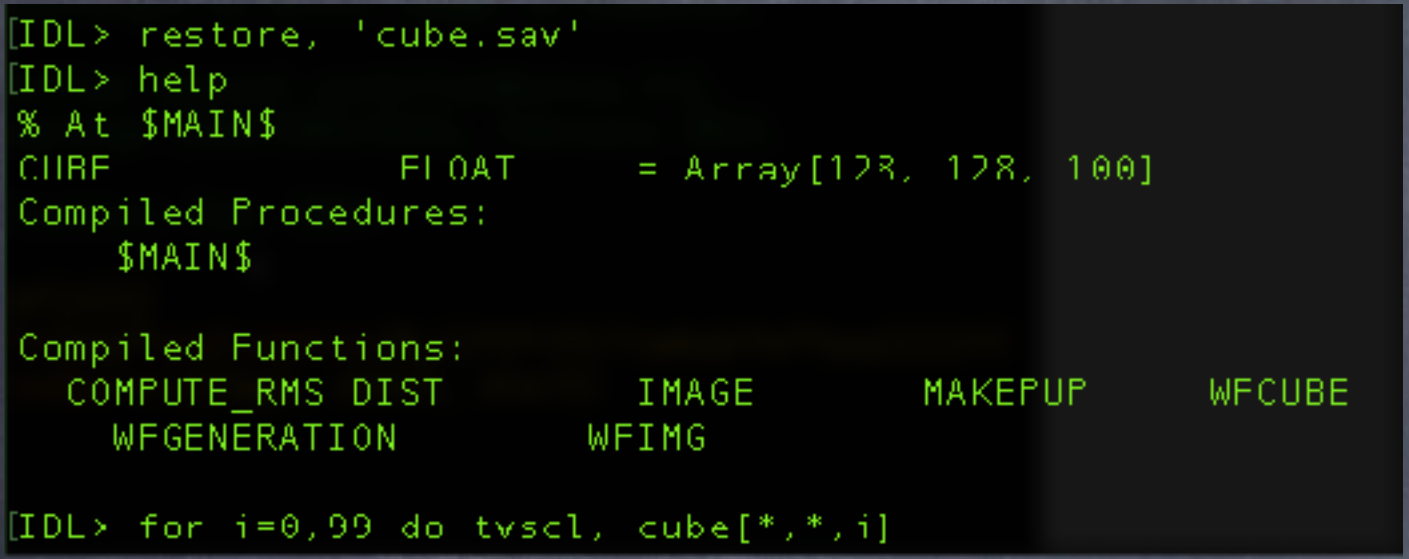


```
function image, pup, wf, lambda
;
; image computation from a wavefront
;
; pup    = pupil,
; wf     = wavefront [float],
; lambda = wavelength at which image is computed.
;
; Marcel Carbillet [marcel.carbillet@unice.fr],
; UMR 7293 Lagrange (UNS/CNRS/OCA), February 2013.
;
; Last modification: Feb. 2014
;
dim = (size(wf))[1]
img = (abs(fft(pup*exp(complex(0,1)*2*!PI/lambda*wf*pup))))^2
img = shift(temporary(img), dim/2, dim/2)

return, img
end
```

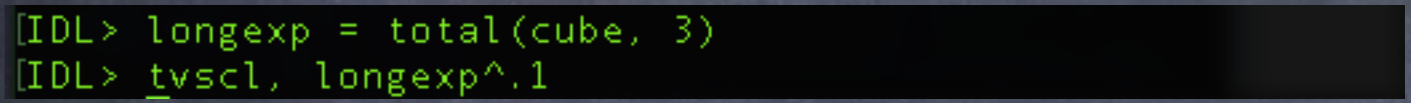


```
IDL> .r wfimg
% Compiled module: WFIMG.
IDL> print, wfimg(128L,2.,27.,0.1,500E-9,0.,64L,500E-9,100L,'cube.sav')
Cube of PSFs cube.sav saved on disk...
```

Images & turbulence - 20

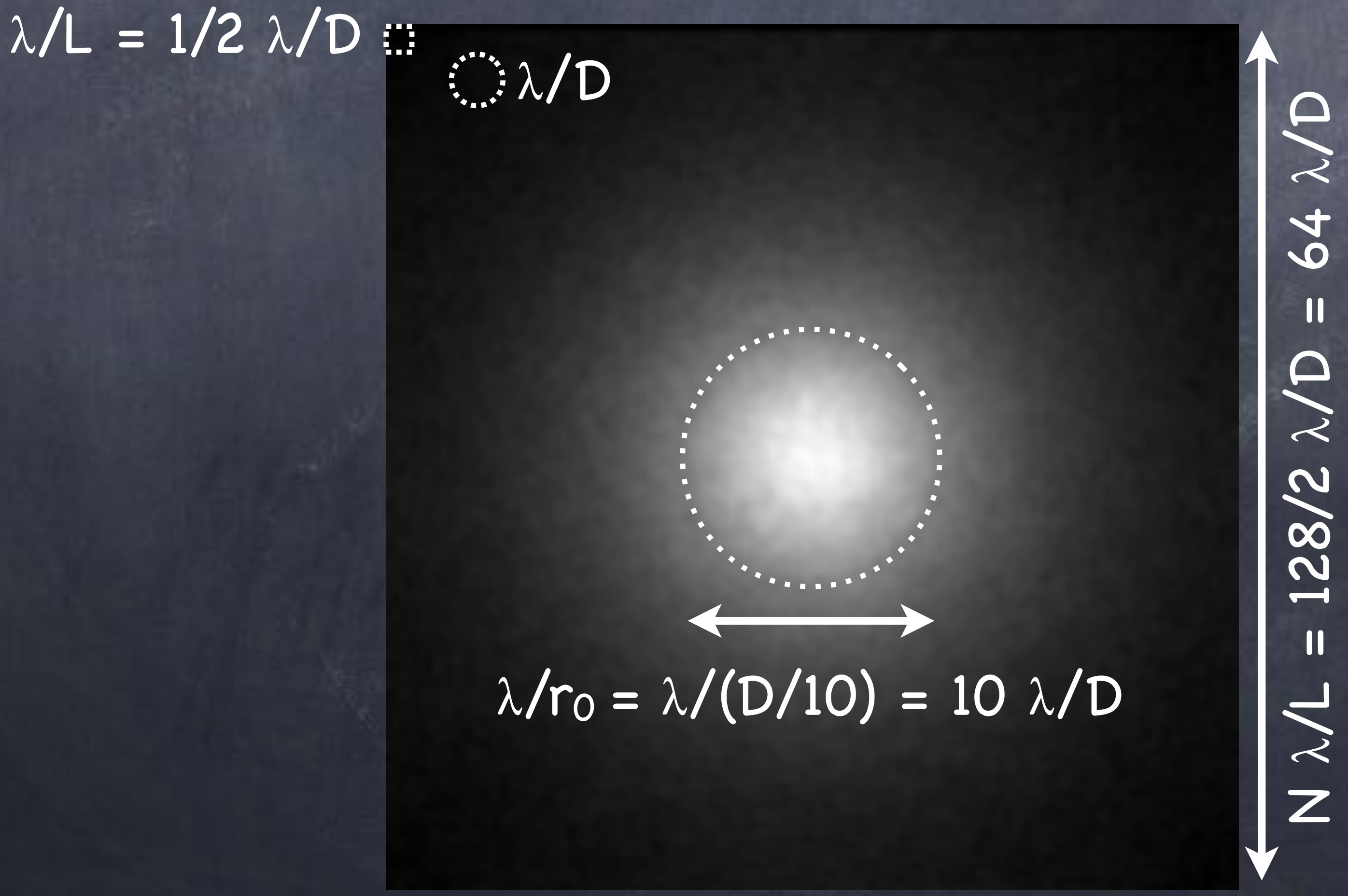


```
[IDL> restore, 'cube.sav'
[IDL> help
% At $MAIN$
CUBE            FLOAT     = Array[128, 128, 100]
Compiled Procedures:
    $MAIN$

Compiled Functions:
  COMPUTE_RMS DIST        IMAGE       MAKEPUP      WFCUBE
    WFGENERATION        WFIMG

[IDL> for i=0,99 do tvscl, cube[*,*,i]
```

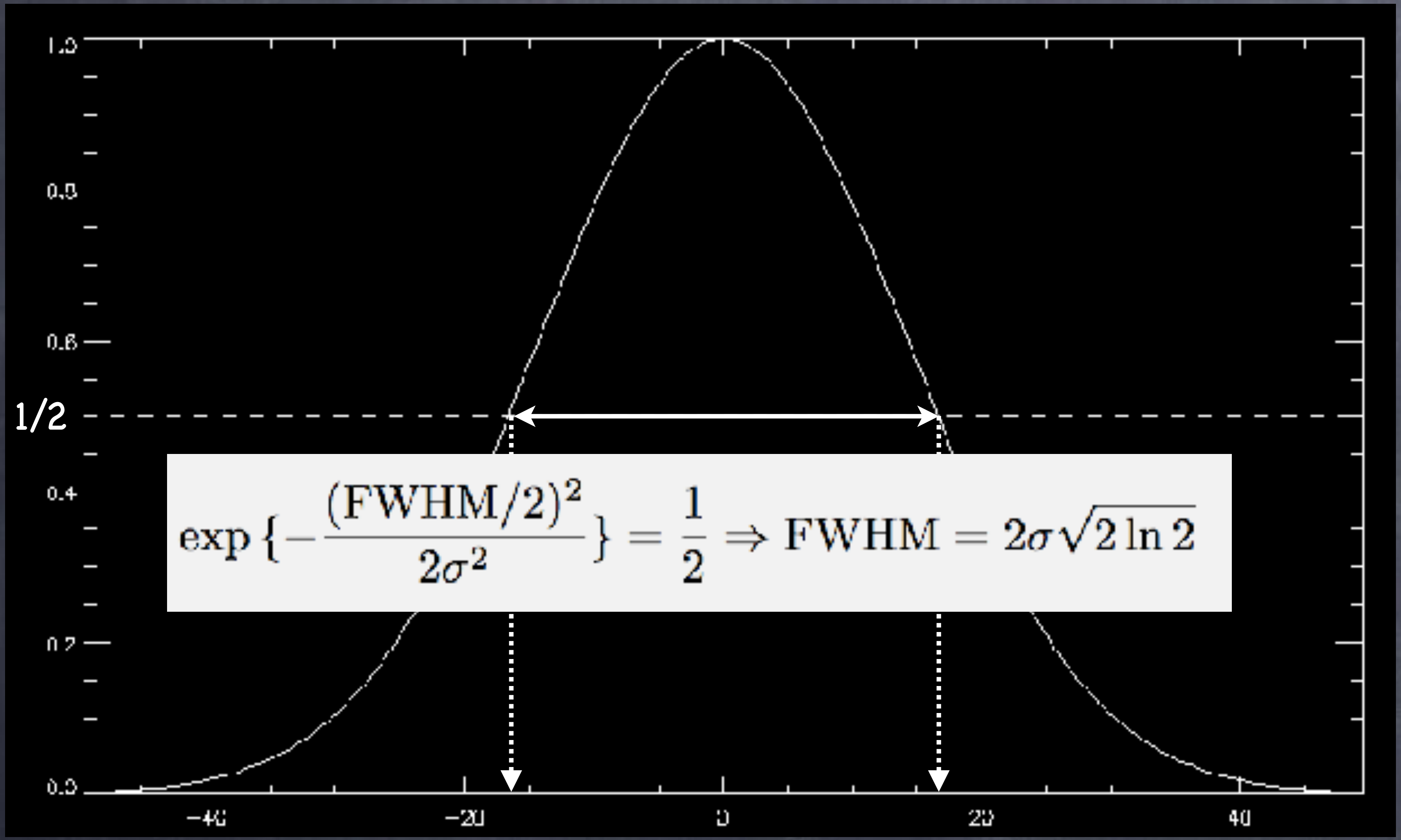


```
[IDL> longexp = total(cube, 3)
[IDL> tvscl, longexp^.1
```

<u>image formation:</u>
1- cube of instantaneous PSFs (500nm & H-band)
2- long-exposure PSF


$\lambda/L = 1/2\ \lambda/D$
λ/D
$\lambda/r_0 = \lambda/(D/10) = 10\ \lambda/D$
$N\ \lambda/L = 128/2\ \lambda/D = 64\ \lambda/D$

Images & turbulence - 21



$$\exp\{-\frac{(\mathrm{FWHM}/2)^2}{2\sigma^2}\} = \frac{1}{2} \Rightarrow \mathrm{FWHM} = 2\sigma\sqrt{2\ln 2}$$

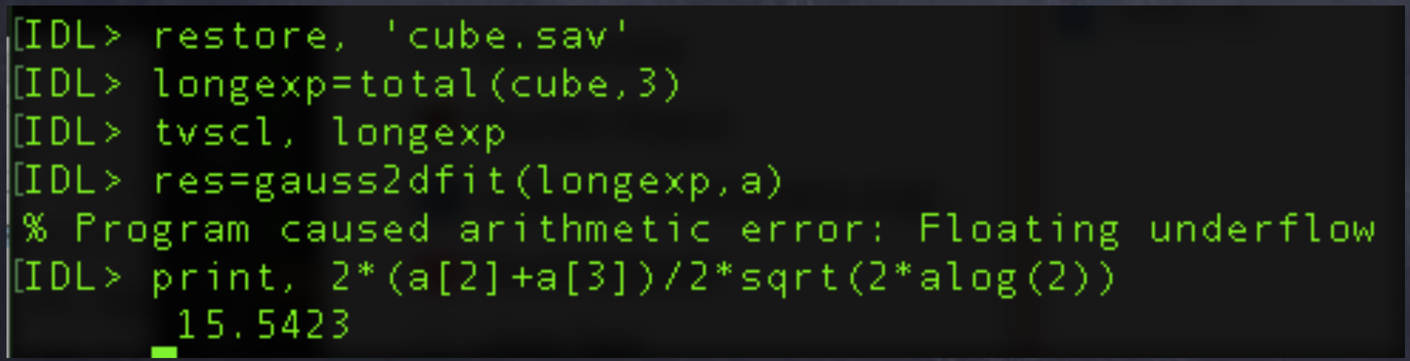
image formation:

1- cube of instantaneous PSFs (500nm & H-band)

2- long-exposure PSFs

3- fit with gaussian and compare FWHM vs. λ/r_0 (seeing), also in function of the outerscale L_0.

-> Also read Martinez...



```
IDL> restore, 'cube.sav'
IDL> longexp=total(cube,3)
IDL> tvscl, longexp
IDL> res=gauss2dfit(longexp,a)
% Program caused arithmetic error: Floating underflow
IDL> print, 2*(a[2]+a[3])/2*sqrt(2*alog(2))
      15.5423
```

In this example, the FWHM is ~15.54px and, since we have here: 1px=(λ/D)/2, we have hence: FWHM~7.77 (λ/D)

Images & turbulence - 22

-> Detector noises:

- At first: *photon noise* (or *shot noise*), poissonian, actually a transformation of the image.

$$p(n) = \frac{N^n e^{-N}}{n!}, \text{with} : N = \frac{L\Delta t}{h\nu}, L = \text{luminosity}, \Delta t = \text{time exp.}$$

p(n) = probability to detect n photons when N are expected

For large N : ~gaussian…

$$p(n) \simeq \exp\left(-\frac{(n-N)^2}{2N}\right)$$

Images & turbulence - 23

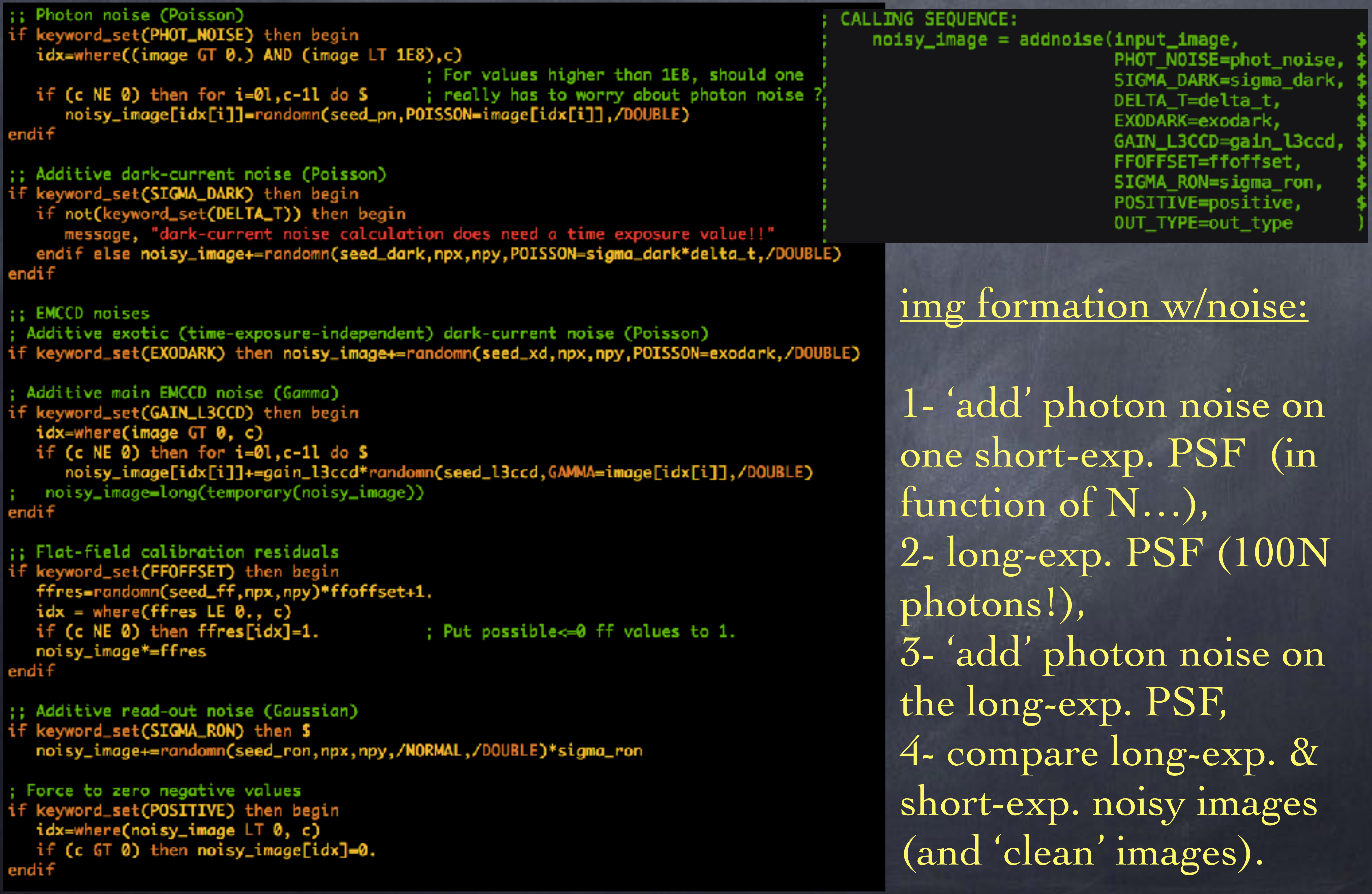
-> Detector noises:

- At first: *photon noise* (or *shot noise*), poissonian, actually a transformation of the image.

- At last: *read-out noise* (*RON*), gaussian with zero mean and rms σ_e [e-/px], additive noise.

- In between: *dark current noise, amplification noise* & *exotic dark current noise* in the case of EMCCDs, noise due to the *calibration* of the *flat field, 'salt & pepper' noise* ('hot' and 'cold' pixels), etc.

Images & turbulence - 24



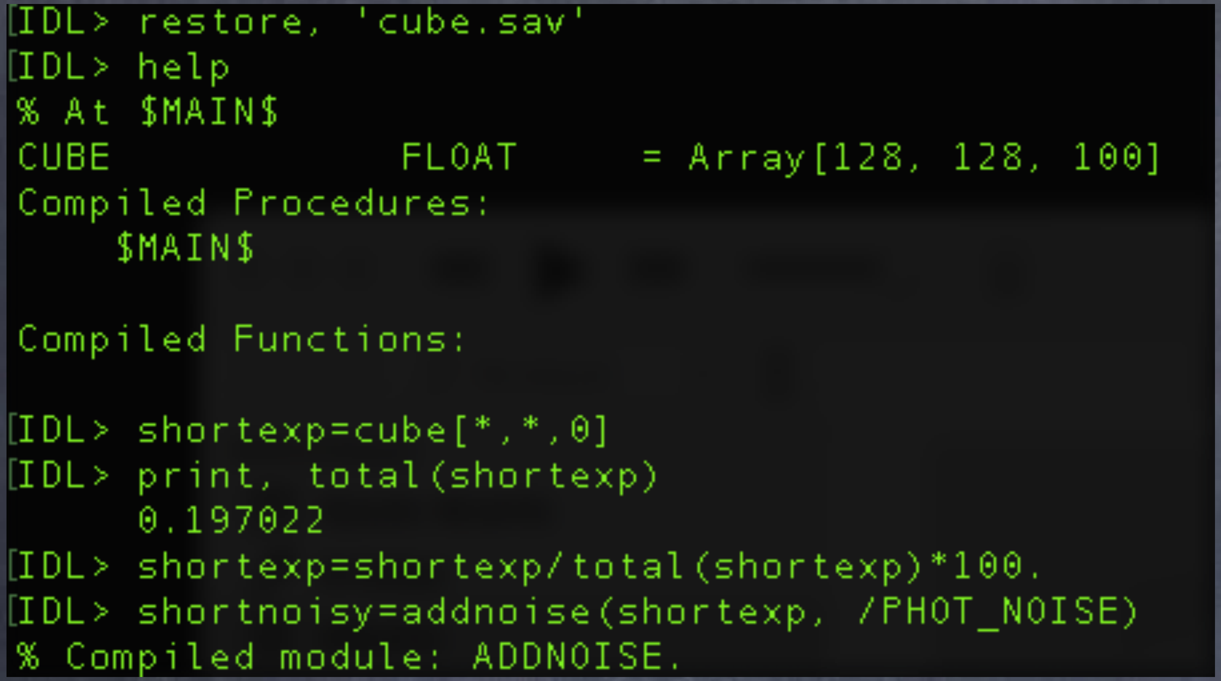
```
;; Photon noise (Poisson)
if keyword_set(PHOT_NOISE) then begin
   idx=where((image GT 0.) AND (image LT 1E8),c)
                                              ; For values higher than 1E8, should one
   if (c NE 0) then for i=0l,c-1l do $        ; really has to worry about photon noise ?
      noisy_image[idx[i]]=randomn(seed_pn,POISSON=image[idx[i]],/DOUBLE)
endif

;; Additive dark-current noise (Poisson)
if keyword_set(SIGMA_DARK) then begin
   if not(keyword_set(DELTA_T)) then begin
      message, "dark-current noise calculation does need a time exposure value!!"
   endif else noisy_image+=randomn(seed_dark,npx,npy,POISSON=sigma_dark*delta_t,/DOUBLE)
endif

;; EMCCD noises
; Additive exotic (time-exposure-independent) dark-current noise (Poisson)
if keyword_set(EXODARK) then noisy_image+=randomn(seed_xd,npx,npy,POISSON=exodark,/DOUBLE)

; Additive main EMCCD noise (Gamma)
if keyword_set(GAIN_L3CCD) then begin
   idx=where(image GT 0, c)
   if (c NE 0) then for i=0l,c-1l do $
      noisy_image[idx[i]]+=gain_l3ccd*randomn(seed_l3ccd,GAMMA=image[idx[i]],/DOUBLE)
;   noisy_image=long(temporary(noisy_image))
endif

;; Flat-field calibration residuals
if keyword_set(FFOFFSET) then begin
   ffres=randomn(seed_ff,npx,npy)*ffoffset+1.
   idx = where(ffres LE 0., c)
   if (c NE 0) then ffres[idx]=1.             ; Put possible<=0 ff values to 1.
   noisy_image*=ffres
endif

;; Additive read-out noise (Gaussian)
if keyword_set(SIGMA_RON) then $
   noisy_image+=randomn(seed_ron,npx,npy,/NORMAL,/DOUBLE)*sigma_ron

; Force to zero negative values
if keyword_set(POSITIVE) then begin
   idx=where(noisy_image LT 0, c)
   if (c GT 0) then noisy_image[idx]=0.
endif
```

```
; CALLING SEQUENCE:
;     noisy_image = addnoise(input_image,                $
;                             PHOT_NOISE=phot_noise,     $
;                             SIGMA_DARK=sigma_dark,     $
;                             DELTA_T=delta_t,           $
;                             EXODARK=exodark,           $
;                             GAIN_L3CCD=gain_l3ccd,     $
;                             FFOFFSET=ffoffset,         $
;                             SIGMA_RON=sigma_ron,       $
;                             POSITIVE=positive,         $
;                             OUT_TYPE=out_type          )
```

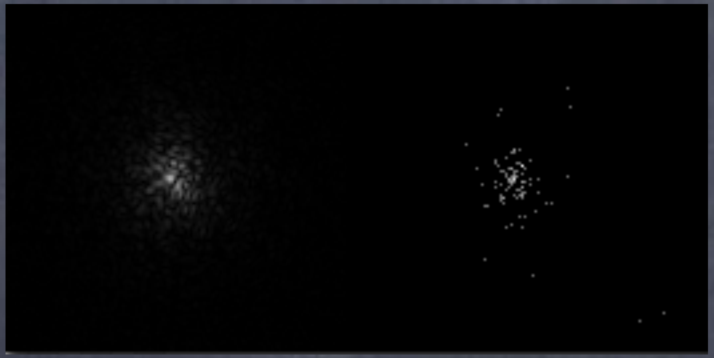
<u>img formation w/noise:</u>

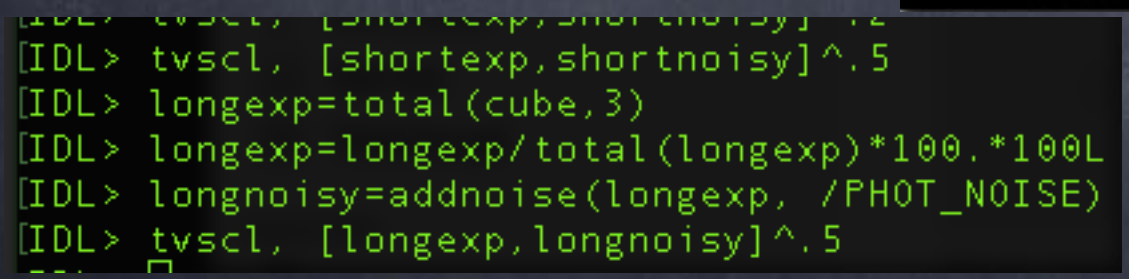
1- ‘add’ photon noise on one short-exp. PSF (in function of N…),
2- long-exp. PSF (100N photons!),
3- ‘add’ photon noise on the long-exp. PSF,
4- compare long-exp. & short-exp. noisy images (and ‘clean’ images).

Images & turbulence - 25

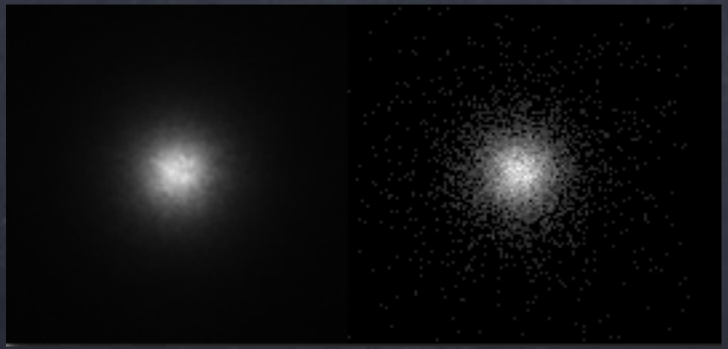


```
[IDL> restore, 'cube.sav'
[IDL> help
% At $MAIN$
CUBE            FLOAT     = Array[128, 128, 100]
Compiled Procedures:
    $MAIN$

Compiled Functions:

[IDL> shortexp=cube[*,*,0]
[IDL> print, total(shortexp)
     0.197022
[IDL> shortexp=shortexp/total(shortexp)*100.
[IDL> shortnoisy=addnoise(shortexp, /PHOT_NOISE)
% Compiled module: ADDNOISE.
```





```
[IDL> tvscl, [shortexp,shortnoisy]^.5
[IDL> longexp=total(cube,3)
[IDL> longexp=longexp/total(longexp)*100.*100L
[IDL> longnoisy=addnoise(longexp, /PHOT_NOISE)
[IDL> tvscl, [longexp,longnoisy]^.5
```



<u>img formation w/noise:</u>

1- 'add' photon noise on one short-exp. PSF (in function of N…),
2- long-exp. PSF (100N photons!),
3- 'add' photon noise on the long-exp. PSF,
4- compare long-exp. & short-exp. noisy images (and 'clean' images).

Speckle ‘interferometry’ - 1

With a simple Fourier transform of the object-image relation:

$$\hat{I}(\vec{f}) = \hat{O}(\vec{f})\ \hat{S}(\vec{f})$$

Where the FT of *S* is the optical transfer function (OTF).

The instantaneous one is random and with complex non-zero values up to the telescope cutting frequency D/λ, and:

$$|\hat{O}(\vec{f})|^2 = \frac{< |\hat{I}(\vec{f})|^2 >}{< |\hat{S}(\vec{f})|^2 >}$$

Speckle ‘interferometry’ - 2

$$|\hat{O}(\vec{f})|^2 = \frac{< |\hat{I}(\vec{f})|^2 >}{< |\hat{S}(\vec{f})|^2 >}$$

It is the spectral density of the object (a statistical invariant), computable from the spectral density of the observed images $I(\alpha)$ and the spectral density of an estimate of the PSF $S(\alpha)$.

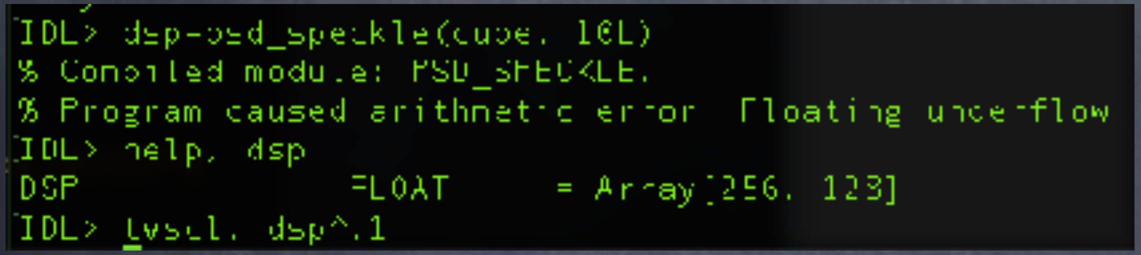
It *almost* permits to deduce the object $O(\alpha)$…

> Actually, it permits to obtain its autocorrelation only (which is the inverse FT of its spectral density), not the object itself, from which one can extract the separation and orientation of the binary, but not the relative positions (who is brighter ?)…

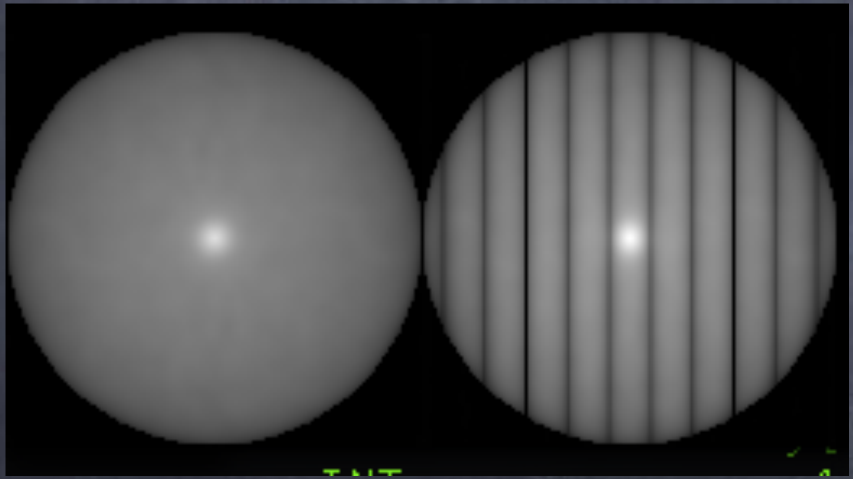
=> this is the well-known problem of quadrant indetermination due to the square power which turns the function even.

Speckle ‘interferometry’ - 3

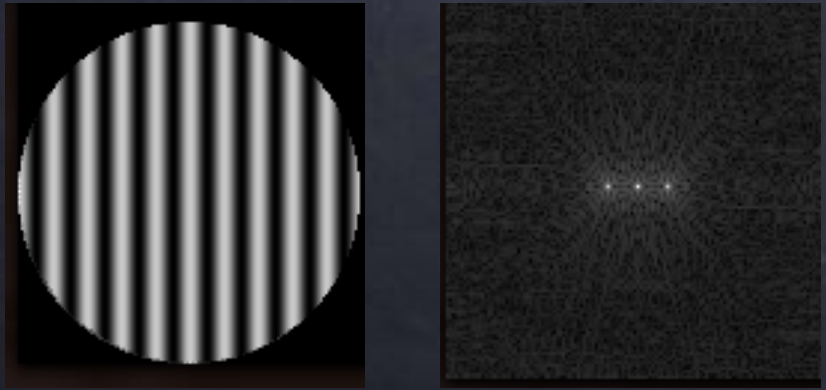
-> Computation of the spectral density of the images of an unresolved single star (hence estimates of the PSF) and a binary star (two points!), with *speckles*, and then *speckles* **and** noise(s)...

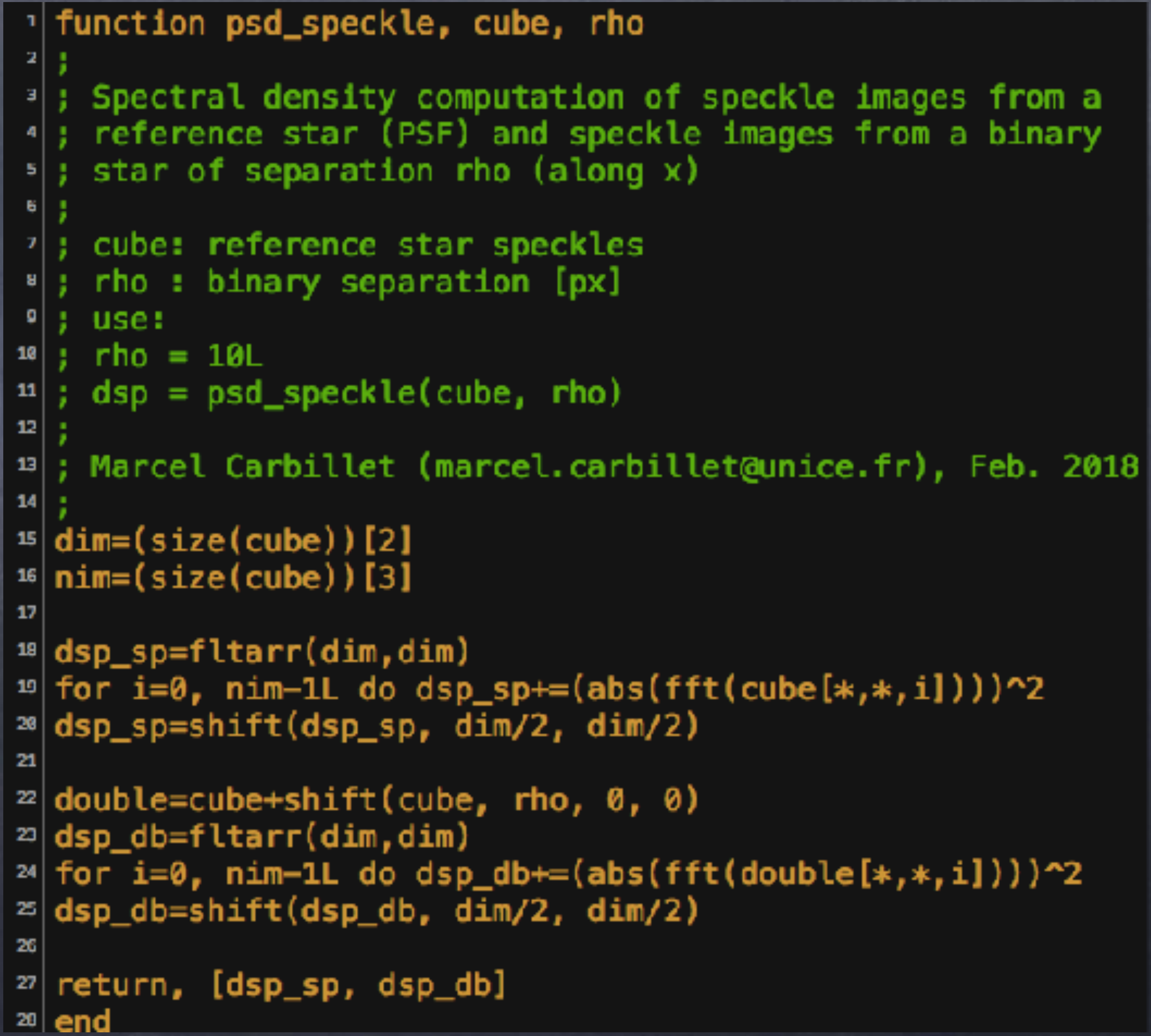


```
IDL> dsp=psd_speckle(cube, 10L)
% Compiled module: PSD_SPECKLE.
% Program caused arithmetic error: Floating underflow
IDL> help, dsp
DSP             FLOAT     = Array[256, 128]
IDL> tvscl, dsp^.1
```



Going further (visibility & autocorrelation) :

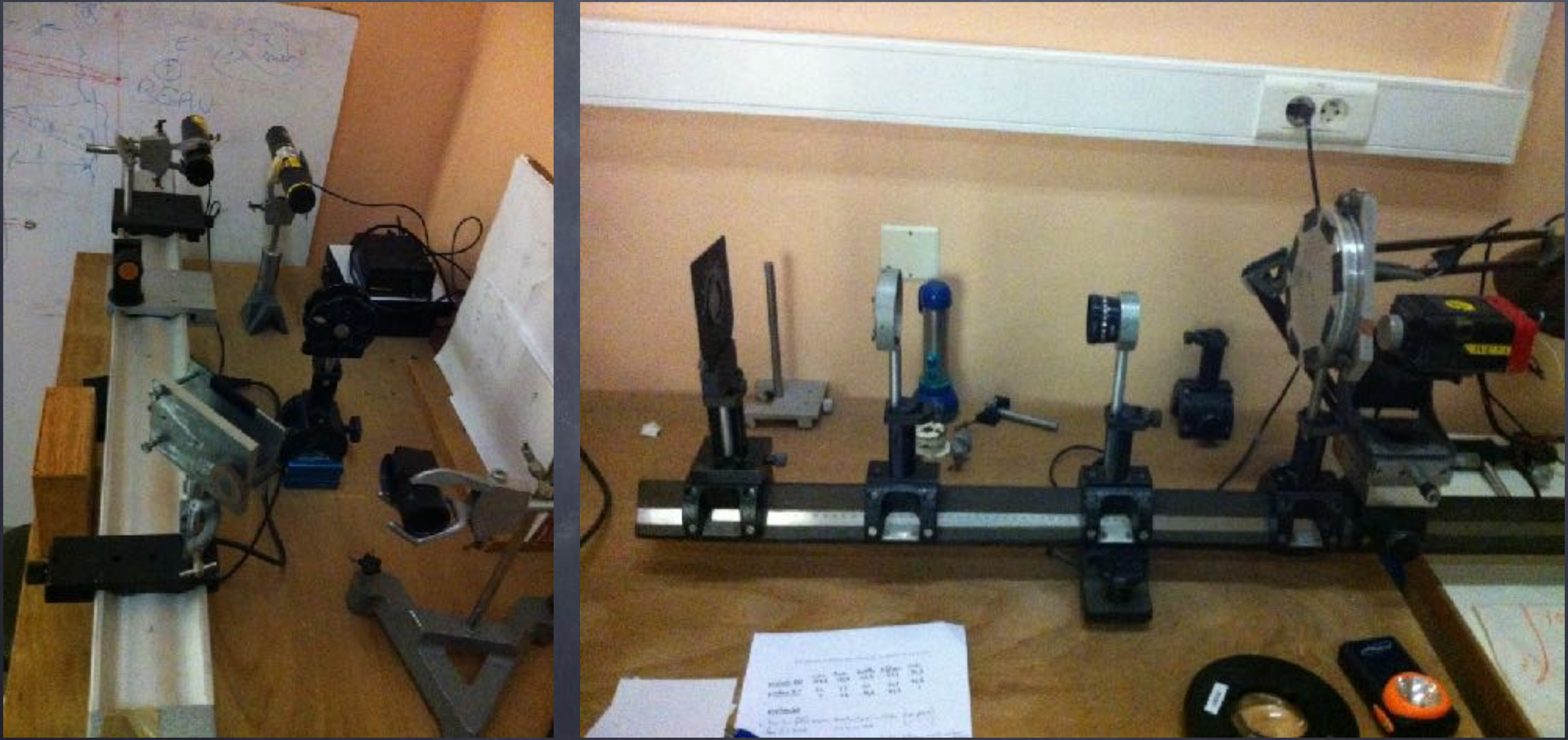




```
function psd_speckle, cube, rho
;
; Spectral density computation of speckle images from a
; reference star (PSF) and speckle images from a binary
; star of separation rho (along x)
;
; cube: reference star speckles
; rho : binary separation [px]
; use:
; rho = 10L
; dsp = psd_speckle(cube, rho)
;
; Marcel Carbillet (marcel.carbillet@unice.fr), Feb. 2018
;
dim=(size(cube))[2]
nim=(size(cube))[3]

dsp_sp=fltarr(dim,dim)
for i=0, nim-1L do dsp_sp+=(abs(fft(cube[*,*,i])))^2
dsp_sp=shift(dsp_sp, dim/2, dim/2)

double=cube+shift(cube, rho, 0, 0)
dsp_db=fltarr(dim,dim)
for i=0, nim-1L do dsp_db+=(abs(fft(double[*,*,i])))^2
dsp_db=shift(dsp_db, dim/2, dim/2)

return, [dsp_sp, dsp_db]
end
```

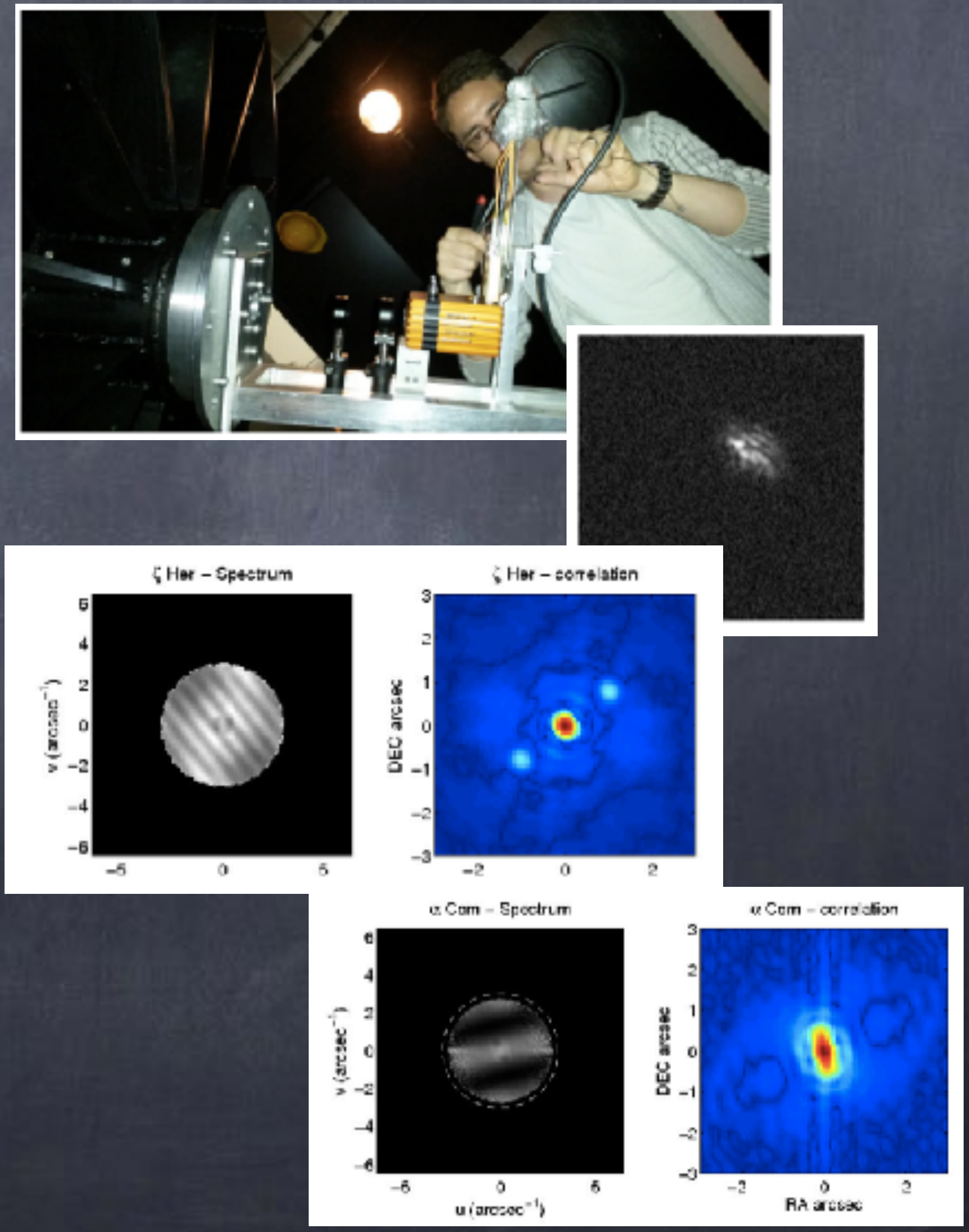
Speckle ‘interferometry’ - 4

-> A simple lab’ experiment *speckle*/binary star



Speckle 'interferometry' - 5

-> Two instruments at C2PU: PISCO (vis.) & HiPIC (nIR)



-> Also read Aime (Sec. 3)...