(IDL: 4 kind of routines/scripts)

```
; call with: IDL> @Exo2
Diam  =1.0
r0    =0.3
N     = 10

J = (N+1)*(N+2)/2-1
Noll = .2944*J^(-sqrt(3)/2)*(Diam/r0)^(5./3)
S = exp(-Noll)
; see result with: IDL> print, S
```

batch: all variables are accessible.

```
; call with: IDL> .rn Exo2_main
Diam  =1.0
r0    =0.3
N     = 10

J = (N+1)*(N+2)/2-1
Noll = .2944*J^(-sqrt(3)/2)*(Diam/r0)^(5./3)
S = exp(-Noll)

end
; see result with: IDL> print, S
```

main: idem (« .r » : run ; « .rn » : run new).

```
; call with: IDL> .rn Exo2_proc
;            IDL> Exo2_proc, Diam, r0, N, S
; with, e.g: Diam=1.0, r0=0.3, N=10, S undefined
pro Exo2_proc, Diam, r0, N, S

J = (N+1)*(N+2)/2-1
Noll = .2944*J^(-sqrt(3)/2)*(Diam/r0)^(5./3)
S = exp(-Noll)

end
; see result with: IDL> print, S
```

procedure: (input/output) parameters are accessible, but variables defined within the procedure are not.

```
; call with: IDL> .rn Exo2_func
;            IDL> print, Exo2_func(Diam, r0, N)
; with, e.g: Diam=1.0, r0=0.3, N=10
function Exo2_func, Diam, r0, N

J = (N+1)*(N+2)/2-1
Noll = .2944*J^(-sqrt(3)/2)*(Diam/r0)^(5./3)
S = exp(-Noll)

return, S
end
```

function: no output parameters, inside variables not accessible, result of the function returned.

Images & turbulence - 21+

Or, more simply if you've saved your own wf cubes (L0,r0):

```
function wfimg2, diam, obs, lambda_psf, filewf, filepsf

;+
; example of use:
;   diam       = 64L           ; [px] telescope pupil dimension
;   obs        = 0. [0-1]      ; (linear) obscuration ratio
;   lambda_psf= 500E-9         ; [m] PSF wavelength
;   filewf     = 'cube.sav'    ; cube of wf filename
;   filepsf    = 'cube_psf.sav'; cube of PSFs filename
;   print, wfimg2(diam,obs,lambda_psf,filewf,filepsf)
;   -> compute the cube of PSFs, save it, and tell how it went
;
; sub-routines needed: make_PSF.pro, wfgeneration.pro, makepup.pro
;
; Marcel Carbillet [marcel.carbillet@unice.fr], Lagrange (UniCA, OCA, CNRS)
; written: Feb. 2018, last modified: March 11th 2024.
;-

; preliminaries
restore, filewf                ; restores variable 'cube' containing nn wf
dim= (size(cube))(1)           ; linear size of wf
nn = (size(cube))(3)           ; nb of wf
cube_psf=fltarr(dim,dim,nn) ; initialize cube of PSFs

; compute and save PSFs
pup = makepup(dim,diam,obs) ; compute entrance pupil
for i=0, nn-1L do cube_psf[*,*,i] = make_PSF(pup,cube[*,*,i],lambda_psf)
                               ; compute the PSF corresponding to each wf
save, cube_psf, FI=filepsf     ; save cube of PSFs to disk

; return back
return, 'Cube of PSFs '+filepsf+' saved on disk...'
end
```

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

```
function make_PSF, pup, wf, lambda

;+
; PSF computation from a wavefront
;
; pup    = input pupil,
; wf     = input wavefront [float],
; lambda = wavelength at which PSF is computed.
; PSF = make_PSF(pup, wf, lambda)
; -> compute the PSF corresponding to wf and pup, at wavelength lambda
;
; Marcel Carbillet [marcel.carbillet@unice.fr],
; UMR 7293 Lagrange (UNS/CNRS/OCA), Feb. 2013.
; Last modification: March 11th 2024
;-

; preliminary
dim = (size(wf))[1]

; compute PSF
psf = (abs(fft(pup*exp(complex(0,1)*2*!PI/lambda*wf*pup))))^2
; NB: (abs(fft(pup*exp(complex(0,1)*2*!PI/lambda*wf))))^2 would suffice
psf = shift(psf, dim/2, dim/2)

; return back
return, psf
end
```

```
IDL> .r lecture-3/makepup
% Compiled module: MAKEPUP.
IDL> .r lecture-4/make_PSF
% Compiled module: MAKE_PSF.
IDL> .r lecture-4/wfimg2
% Compiled module: WFIMG2.
IDL> print, wfimg2(64L, 0., 500E-9, 'cube_r0=0p1_L0=10.sav', 'cube_psf_r0=0p1_L0=10.sav')
Cube of PSFs cube_psf_r0=0p1_L0=10.sav saved on disk...
```

Images & turbulence - 21++

With, for what concerns wf generation...

```
function wfcube2, dim, length, L0, r0, lambda, n_wf, filewf

;+
; example of use:
; dim        = 128L           ; [px] wf dimension
; length     = 2.             ; [m] wf physical dimension
; L0         = 27.            ; [m] outerscale of turbulence
; r0         = .1             ; [m] Fried parameter
; lambda     = 500E-9         ; [m] r0 wavelength
; n_wf       = 100L           ; nb of generated wf
; filewf     = 'cube.sav'     ; cube of wf filename
;
; print, wfcube(dim,length,L0,r0,lambda,n_wf)*1E9
; -> compute the cube of wf, save it, and print the rms value in nm
;
; sub-routines needed:
; wfgeneration.pro, compute_rms.pro
;
; Marcel Carbillet [marcel.carbillet@unice.fr],
; lab. Lagrange (UCA, OCA, CNRS), Feb. 2018.
; Last modification: 11th March 2024
;-

; preliminary
cube = fltarr(dim,dim,n_wf) ; initialize cube of wf

; compute and save cube of wf
for i=0, n_wf/2-1 do begin  ; generate wf
    wf = wfgeneration(dim, length, L0, r0, lambda, SEED=seed)
    cube[*,*,2*i]   = float(wf)
    cube[*,*,2*i+1] = imaginary(wf)
endfor
save, cube, FILE=filewf      ; save cube of wf to disk

; compute mean rms
rms = compute_rms(cube)      ; compute rms

return, rms                  ; return back
end
```

```
IDL> .r lecture-3/compute_rms
% Compiled module: COMPUTE_RMS.
IDL> .r lecture-3/wfgeneration
% Compiled module: WFGENERATION.
IDL> .r lecture-4/wfcube2
% Compiled module: WFCUBE2.
IDL> print, wfcube2(128L,2.,27.,.1,500E-9,100L,'cube_r0=0p1_L0=10.sav')*1E9
       367.953
```

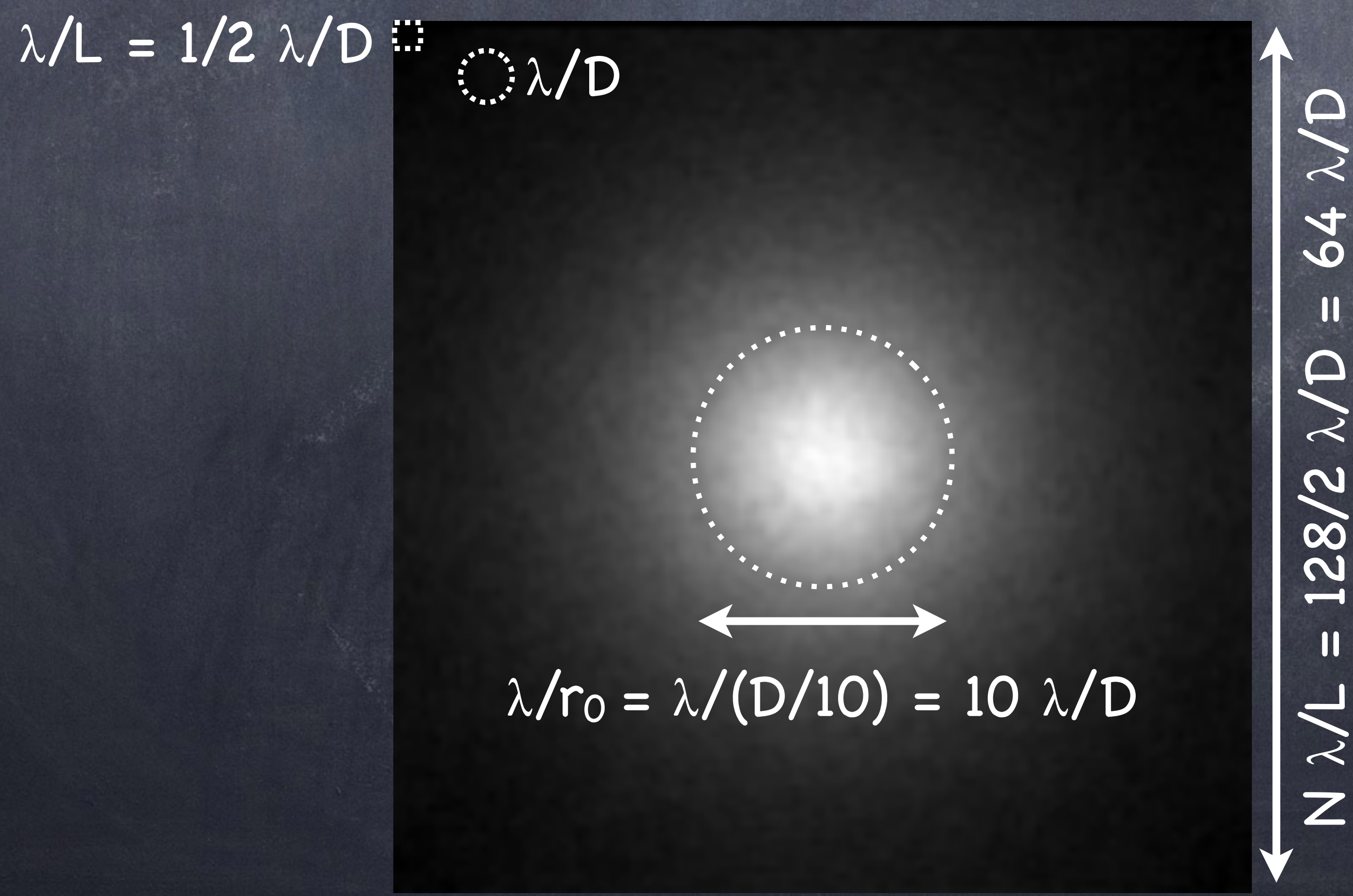
Images & turbulence - 22

```
IDL> $pwd
/Users/marcel/Documents/enseignement/OA-HRA-Fourier/M1-MAUCA/cours-img-turbu/2023-24
IDL> $ls
cube_psf_r0=0p1_L0=10.sav       lecture-1                       lecture-3
cube_r0=0p1_L0=10.sav           lecture-2                       lecture-4
IDL> restore, 'cube_psf_r0=0p1_L0=10.sav'
IDL> help
% At $MAIN$
CUBE_PSF        FLOAT     = Array[128, 128, 100]
Compiled Procedures:
    $MAIN$

Compiled Functions:


IDL> window, XS=512, YS=512, /FREE
IDL> tvscl, rebin(cube_psf[*,*,0], 512, 512, /SAMPLE)
IDL> longexp=total(cube_psf,3)
IDL> tvscl, rebin(longexp, 512, 512, /SAMPLE)^.1
```

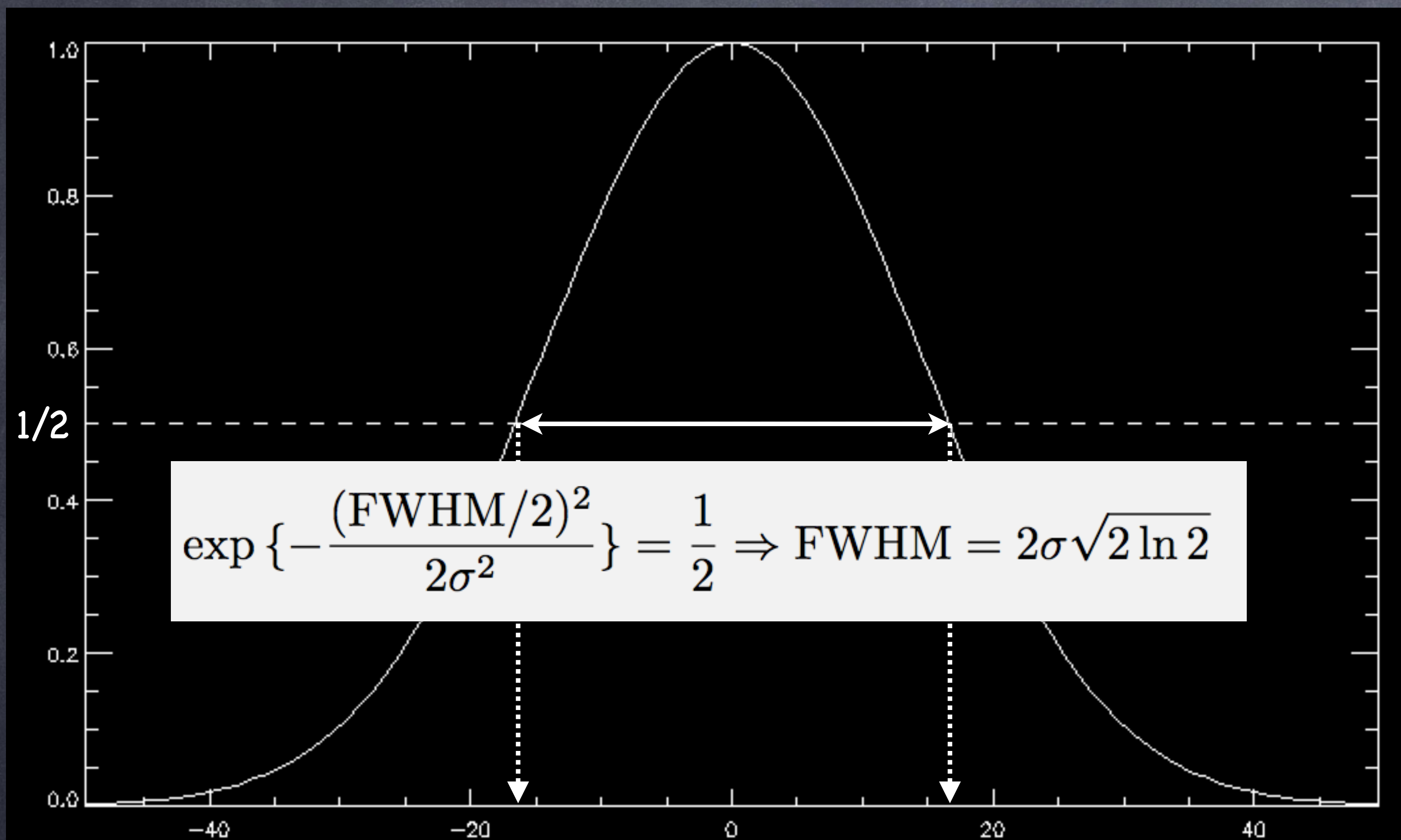
<u>image formation:</u>

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

2- long-exposure PSF



Images & turbulence - 23



<u>image formation:</u>

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

2- long-exposure PSFs

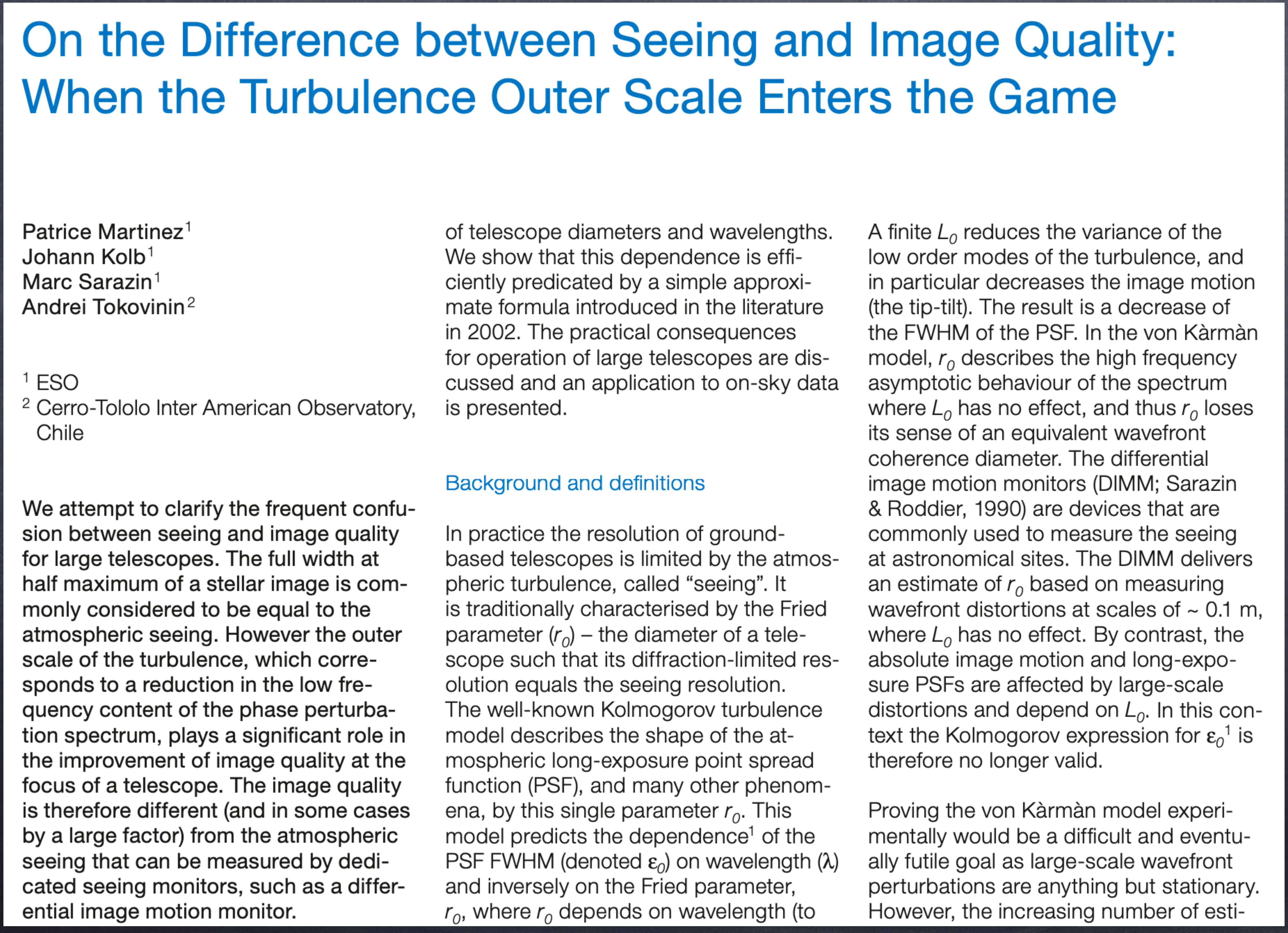
3- fit with gaussian and compare FWHM vs. λ/r_0 (seeing), also in function of the outerscale L_0.

-> Also read Martinez...

```
IDL> res=gauss2dfit(longexp,a)
% Program caused arithmetic error: Floating underflow
IDL> print, 2*((a[2]+a[3])/2)*sqrt(2*alog(2))
      15.9637
IDL>
```

In this example, the FWHM is ≈16px and, since we have here: 1px=(λ/D)/2, we have hence: FWHM≈8 (λ/D) [i.e. 8*0.1″≈0″8 here (@500nm)]

Images & turbulence - 23+



On the Difference between Seeing and Image Quality: When the Turbulence Outer Scale Enters the Game

Patrice Martinez[1]
Johann Kolb[1]
Marc Sarazin[1]
Andrei Tokovinin[2]

[1] ESO
[2] Cerro-Tololo Inter American Observatory, Chile

We attempt to clarify the frequent confusion between seeing and image quality for large telescopes. The full width at half maximum of a stellar image is commonly considered to be equal to the atmospheric seeing. However the outer scale of the turbulence, which corresponds to a reduction in the low frequency content of the phase perturbation spectrum, plays a significant role in the improvement of image quality at the focus of a telescope. The image quality is therefore different (and in some cases by a large factor) from the atmospheric seeing that can be measured by dedicated seeing monitors, such as a differential image motion monitor.

of telescope diameters and wavelengths. We show that this dependence is efficiently predicated by a simple approximate formula introduced in the literature in 2002. The practical consequences for operation of large telescopes are discussed and an application to on-sky data is presented.

Background and definitions

In practice the resolution of ground-based telescopes is limited by the atmospheric turbulence, called "seeing". It is traditionally characterised by the Fried parameter (r_0) – the diameter of a telescope such that its diffraction-limited resolution equals the seeing resolution. The well-known Kolmogorov turbulence model describes the shape of the atmospheric long-exposure point spread function (PSF), and many other phenomena, by this single parameter r_0. This model predicts the dependence[1] of the PSF FWHM (denoted ε_0) on wavelength (λ) and inversely on the Fried parameter, r_0, where r_0 depends on wavelength (to

A finite L_0 reduces the variance of the low order modes of the turbulence, and in particular decreases the image motion (the tip-tilt). The result is a decrease of the FWHM of the PSF. In the von Kàrmàn model, r_0 describes the high frequency asymptotic behaviour of the spectrum where L_0 has no effect, and thus r_0 loses its sense of an equivalent wavefront coherence diameter. The differential image motion monitors (DIMM; Sarazin & Roddier, 1990) are devices that are commonly used to measure the seeing at astronomical sites. The DIMM delivers an estimate of r_0 based on measuring wavefront distortions at scales of ~ 0.1 m, where L_0 has no effect. By contrast, the absolute image motion and long-exposure PSFs are affected by large-scale distortions and depend on L_0. In this context the Kolmogorov expression for ε_0[1] is therefore no longer valid.

Proving the von Kàrmàn model experimentally would be a difficult and eventually futile goal as large-scale wavefront perturbations are anything but stationary. However, the increasing number of esti-

```
REPORT

- Preliminary measures
+ introduction
+ PSD(r0, L0) plot
+ => ccl on the influence of r0 and L0
+ rms(r0, L0) plot or table
+ => ccl on the influence of r0 and L0
+ image formation and FWHM(r0 or lambda, possibly L0)
+ => ccl on the influence of r0 or lambda (and poss. L0)
+ => comparison with the 'seeing' lambda/r0
+ (more to come...)
```

Images & turbulence - 24

-> 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 - 25

-> 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 - 26

```
;; 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 - 27

```
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).

REPORT

```
- Preliminary measures
+ introduction/context
+ PSD(r0, L0)
+ => influence of r0 and L0
+ rms(r0, L0)
+ => influence of r0 and L0
+ FWHM(r0 or lambda=>r0, L0)
+ => influence of r0 and L0
+ => comparison with the "seeing" lambda/r0
+ noisy images
+ personal development ?
```