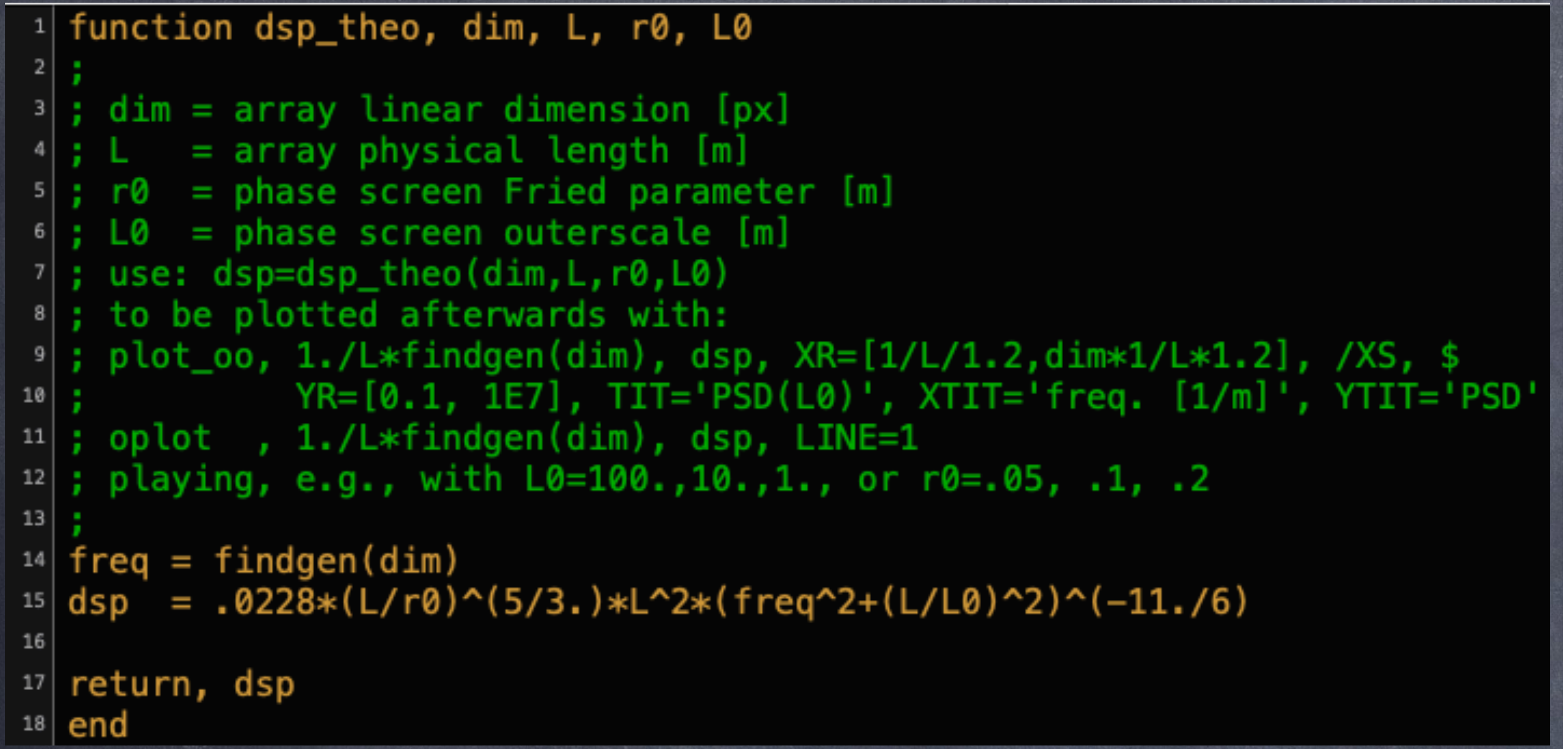
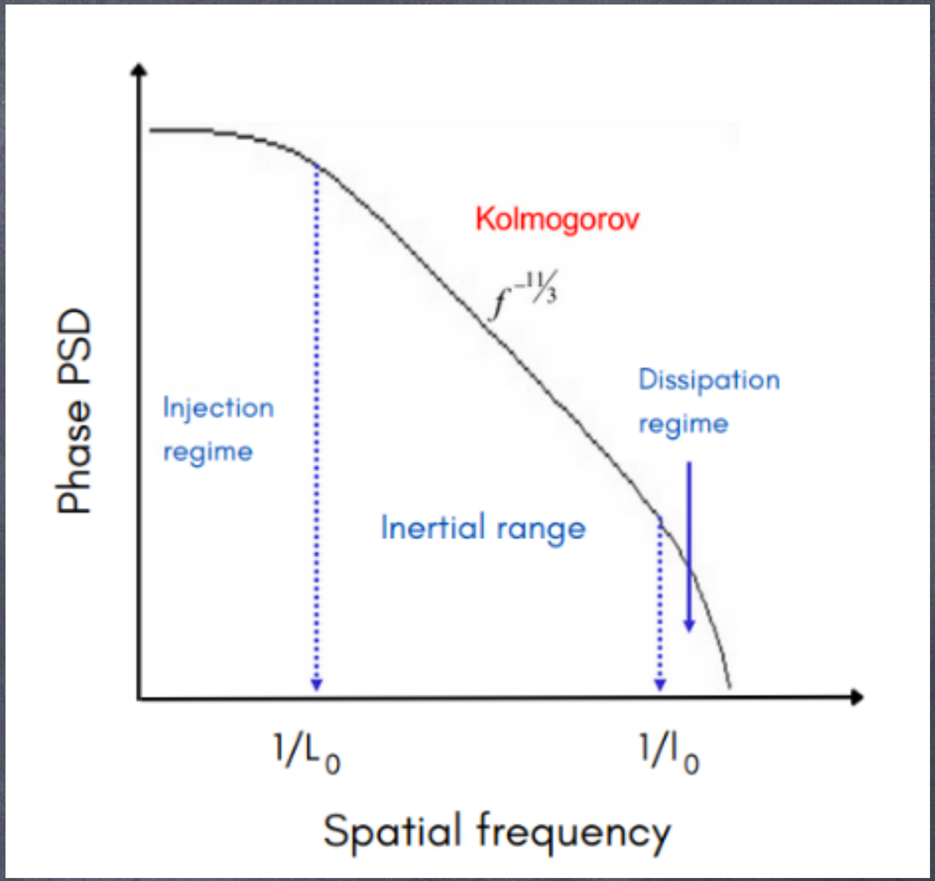
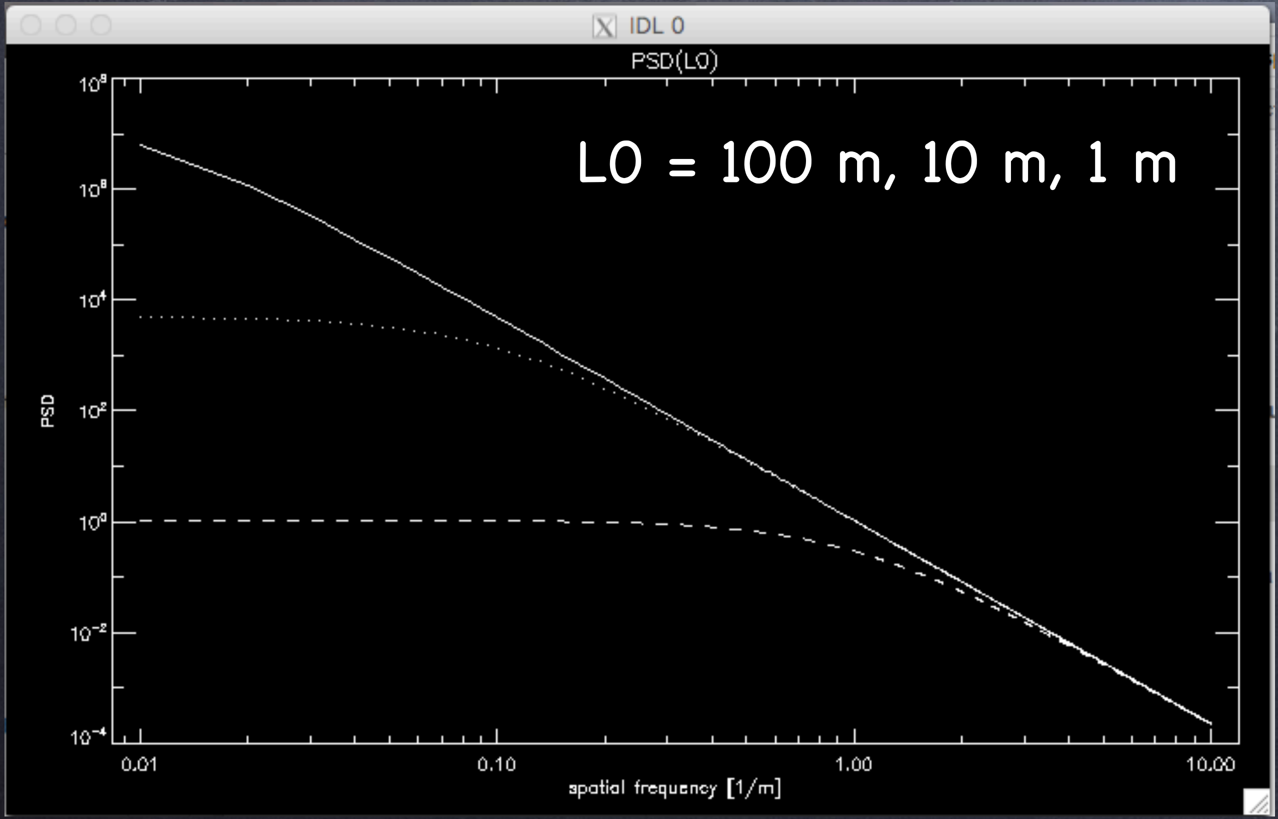
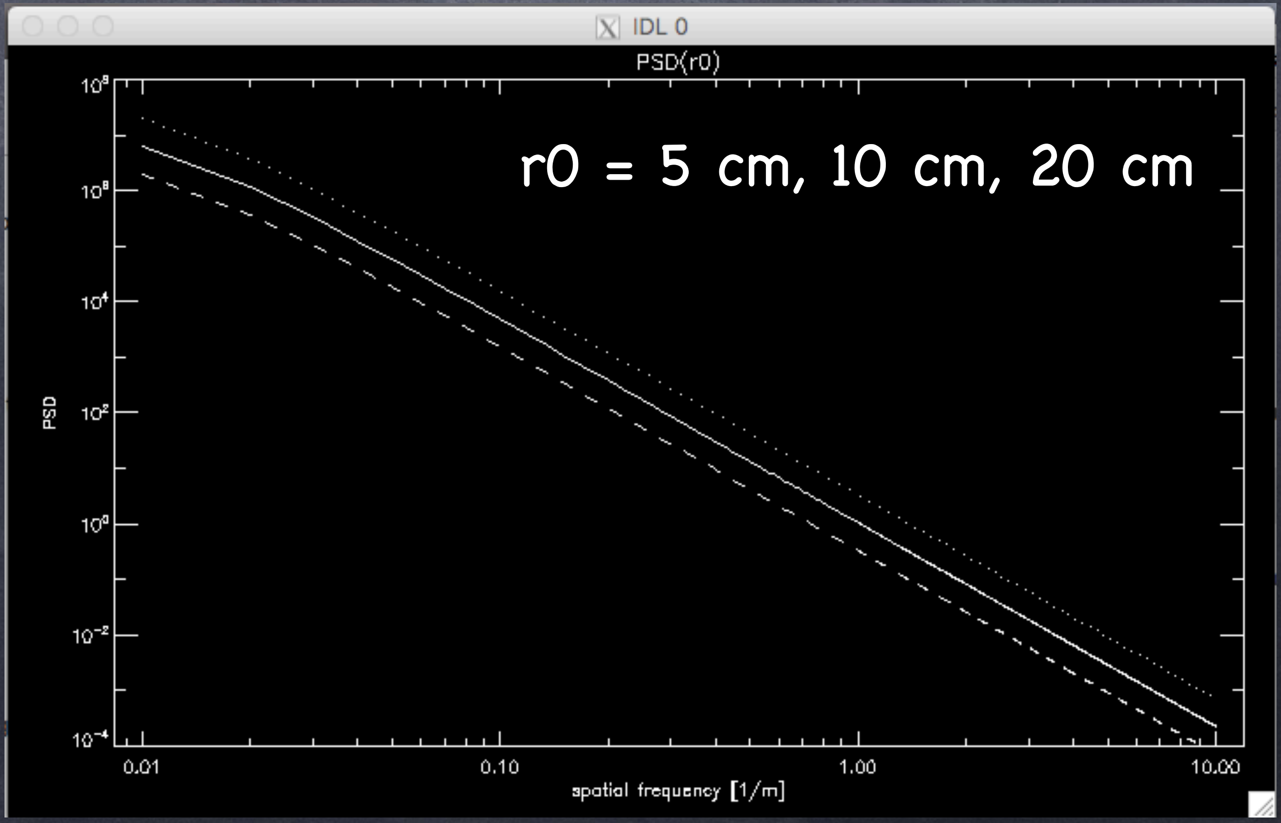
Images & turbulence - 15

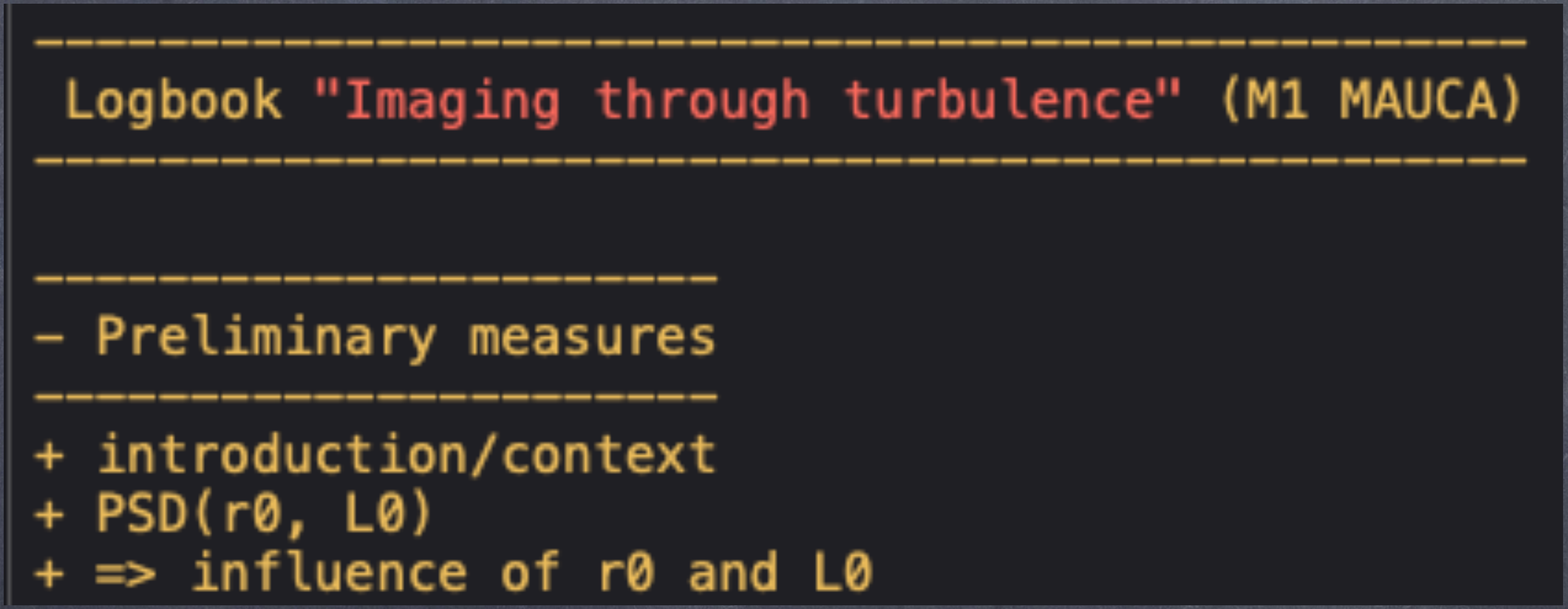


```
function dsp_theo, dim, L, r0, L0
;
; dim = array linear dimension [px]
; L   = array physical length [m]
; r0  = phase screen Fried parameter [m]
; L0  = phase screen outerscale [m]
; use: dsp=dsp_theo(dim,L,r0,L0)
; to be plotted afterwards with:
; plot_oo, 1./L*findgen(dim), dsp, XR=[1/L/1.2,dim*1/L*1.2], /XS, $
;          YR=[0.1, 1E7], TIT='PSD(L0)', XTIT='freq. [1/m]', YTIT='PSD'
; oplot  , 1./L*findgen(dim), dsp, LINE=1
; playing, e.g., with L0=100.,10.,1., or r0=.05, .1, .2
;
freq = findgen(dim)
dsp  = .0228*(L/r0)^(5/3.)*L^2*(freq^2+(L/L0)^2)^(-11./6)

return, dsp
end
```







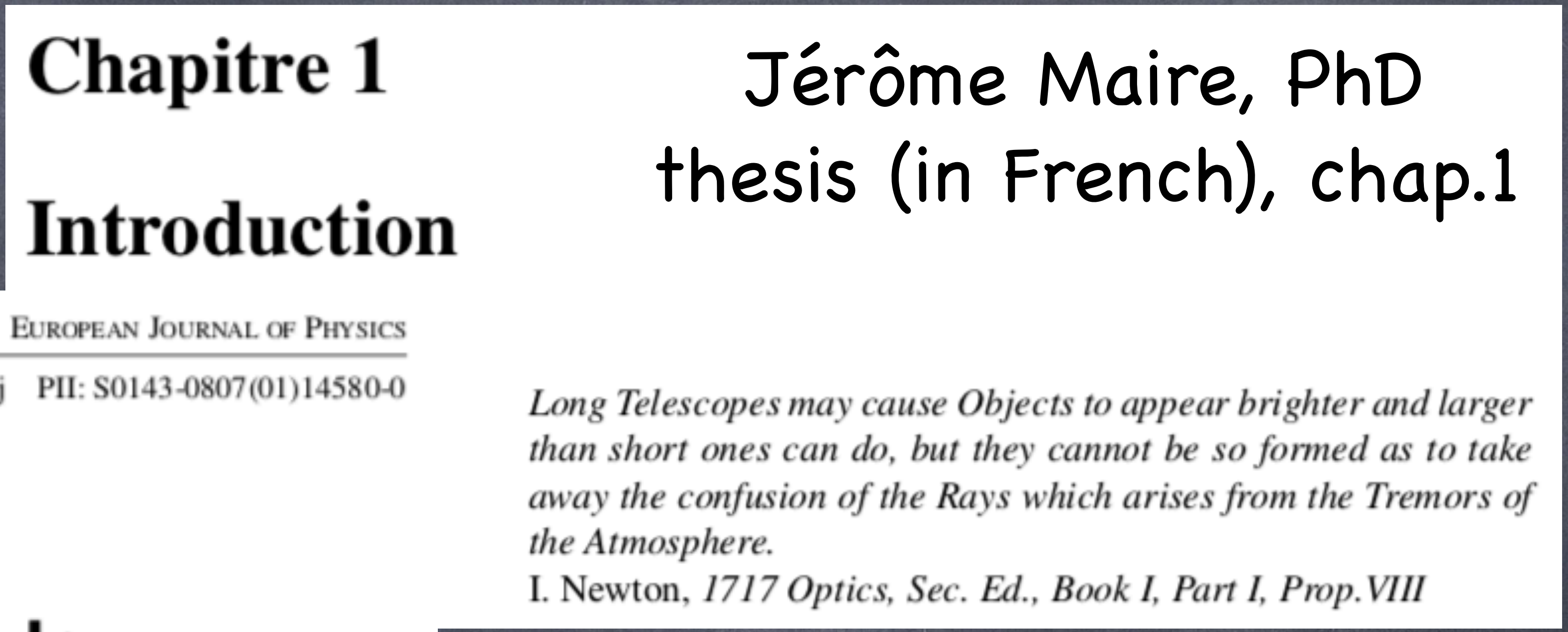


```
------------------------------------------------
 Logbook "Imaging through turbulence" (M1 MAUCA)
------------------------------------------------


----------------------
- Preliminary measures
----------------------
+ introduction/context
+ PSD(r0, L0)
+ => influence of r0 and L0
```

(more to come...)

Images & turbulence - 16

-> For next time: read Aime (Sec. 1 & Sec. 2) and Maire (Chap.1)...

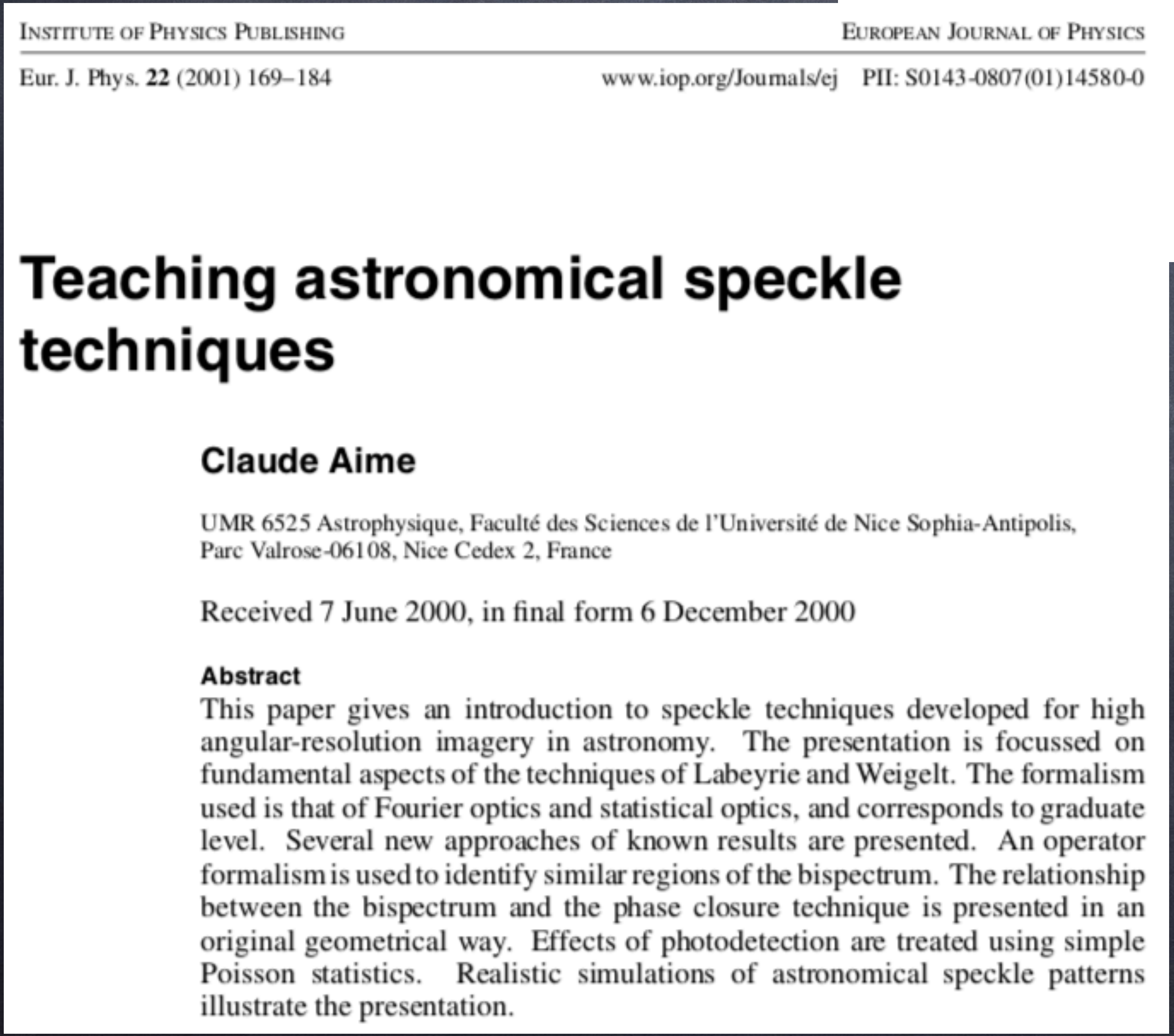


Chapitre 1

Introduction

Jérôme Maire, PhD thesis (in French), chap.1

Long Telescopes may cause Objects to appear brighter and larger than short ones can do, but they cannot be so formed as to take away the confusion of the Rays which arises from the Tremors of the Atmosphere.
I. Newton, *1717 Optics, Sec. Ed., Book I, Part I, Prop.VIII*



INSTITUTE OF PHYSICS PUBLISHING — EUROPEAN JOURNAL OF PHYSICS

Eur. J. Phys. **22** (2001) 169–184 www.iop.org/Journals/ej PII: S0143-0807(01)14580-0

Teaching astronomical speckle techniques

Claude Aime

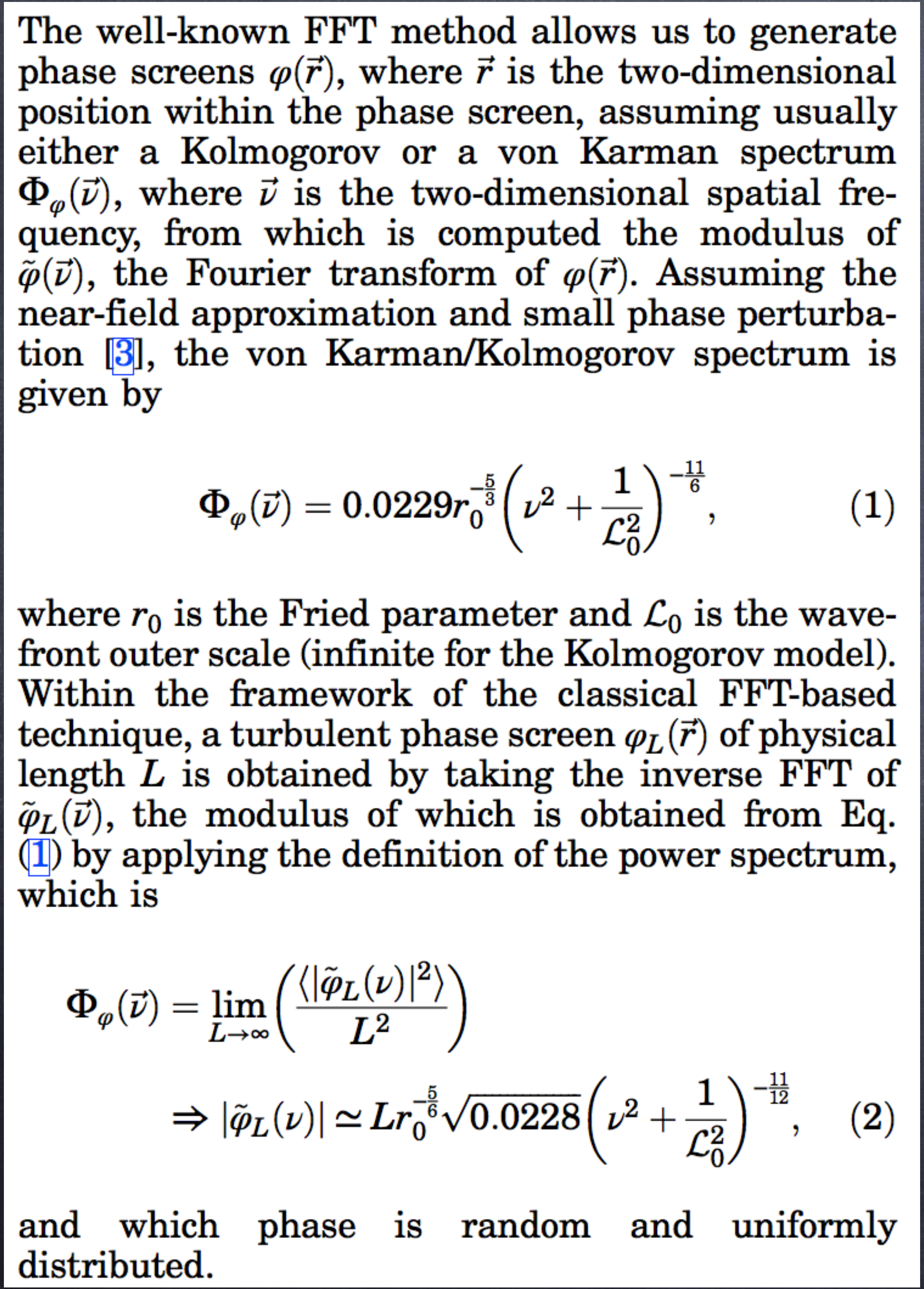
UMR 6525 Astrophysique, Faculté des Sciences de l'Université de Nice Sophia-Antipolis, Parc Valrose-06108, Nice Cedex 2, France

Received 7 June 2000, in final form 6 December 2000

Abstract
This paper gives an introduction to speckle techniques developed for high angular-resolution imagery in astronomy. The presentation is focussed on fundamental aspects of the techniques of Labeyrie and Weigelt. The formalism used is that of Fourier optics and statistical optics, and corresponds to graduate level. Several new approaches of known results are presented. An operator formalism is used to identify similar regions of the bispectrum. The relationship between the bispectrum and the phase closure technique is presented in an original way. Effects of photodetection are treated using simple Poisson statistics. Realistic simulations of astronomical speckle patterns illustrate the presentation.

Images & turbulence - 17

-> Perturbed wavefront generation



The well-known FFT method allows us to generate phase screens $\varphi(\vec{r})$, where $\vec{r}$ is the two-dimensional position within the phase screen, assuming usually either a Kolmogorov or a von Karman spectrum $\Phi_\varphi(\vec{\nu})$, where $\vec{\nu}$ is the two-dimensional spatial frequency, from which is computed the modulus of $\tilde{\varphi}(\vec{\nu})$, the Fourier transform of $\varphi(\vec{r})$. Assuming the near-field approximation and small phase perturbation [3], the von Karman/Kolmogorov spectrum is given by

$$\Phi_\varphi(\vec{\nu}) = 0.0229 r_0^{-\frac{5}{3}} \left(\nu^2 + \frac{1}{\mathcal{L}_0^2}\right)^{-\frac{11}{6}}, \qquad (1)$$

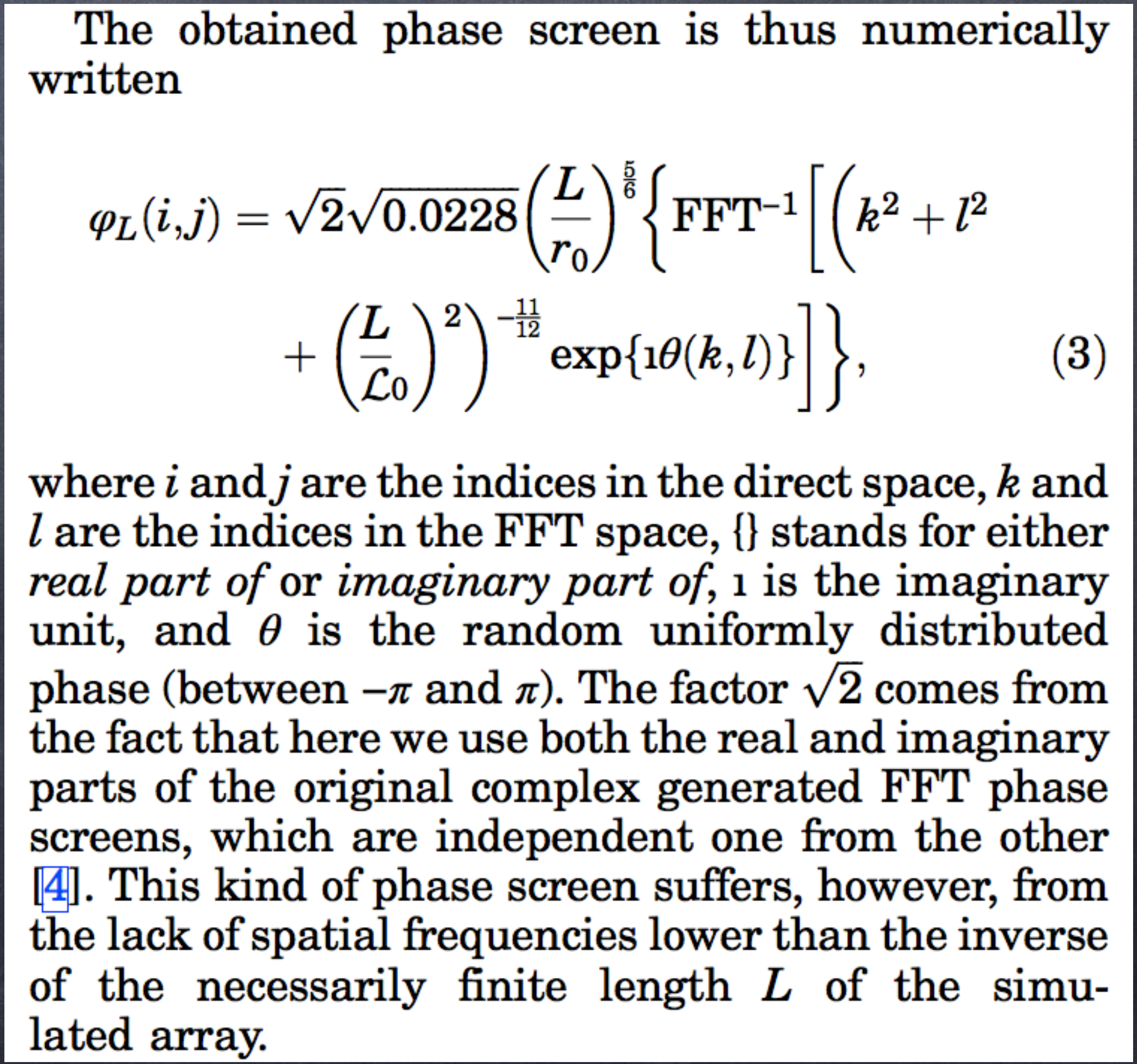
where r_0 is the Fried parameter and $\mathcal{L}_0$ is the wavefront outer scale (infinite for the Kolmogorov model). Within the framework of the classical FFT-based technique, a turbulent phase screen $\varphi_L(\vec{r})$ of physical length L is obtained by taking the inverse FFT of $\tilde{\varphi}_L(\vec{\nu})$, the modulus of which is obtained from Eq. (1) by applying the definition of the power spectrum, which is

$$\Phi_\varphi(\vec{\nu}) = \lim_{L\to\infty}\left(\frac{\langle|\tilde{\varphi}_L(\nu)|^2\rangle}{L^2}\right)$$
$$\Rightarrow |\tilde{\varphi}_L(\nu)| \simeq L r_0^{-\frac{5}{6}} \sqrt{0.0228}\left(\nu^2 + \frac{1}{\mathcal{L}_0^2}\right)^{-\frac{11}{12}}, \qquad (2)$$

and which phase is random and uniformly distributed.

(From Carbillet & Riccardi, sec. 2: read it as well…)

(the same manipulation as before is applied here in order to obtain the numerical formulation here below.)

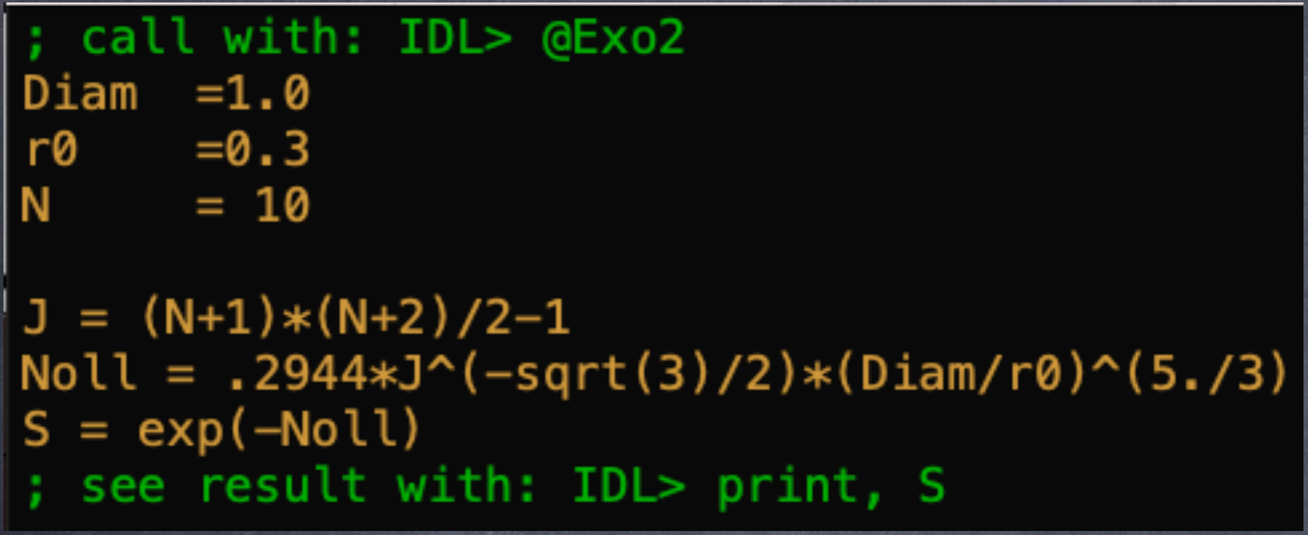


The obtained phase screen is thus numerically written

$$\varphi_L(i,j) = \sqrt{2}\sqrt{0.0228}\left(\frac{L}{r_0}\right)^{\frac{5}{6}} \left\{ \mathrm{FFT}^{-1}\left[\left(k^2 + l^2 + \left(\frac{L}{\mathcal{L}_0}\right)^2\right)^{-\frac{11}{12}} \exp\{\imath\theta(k,l)\}\right]\right\}, \qquad (3)$$

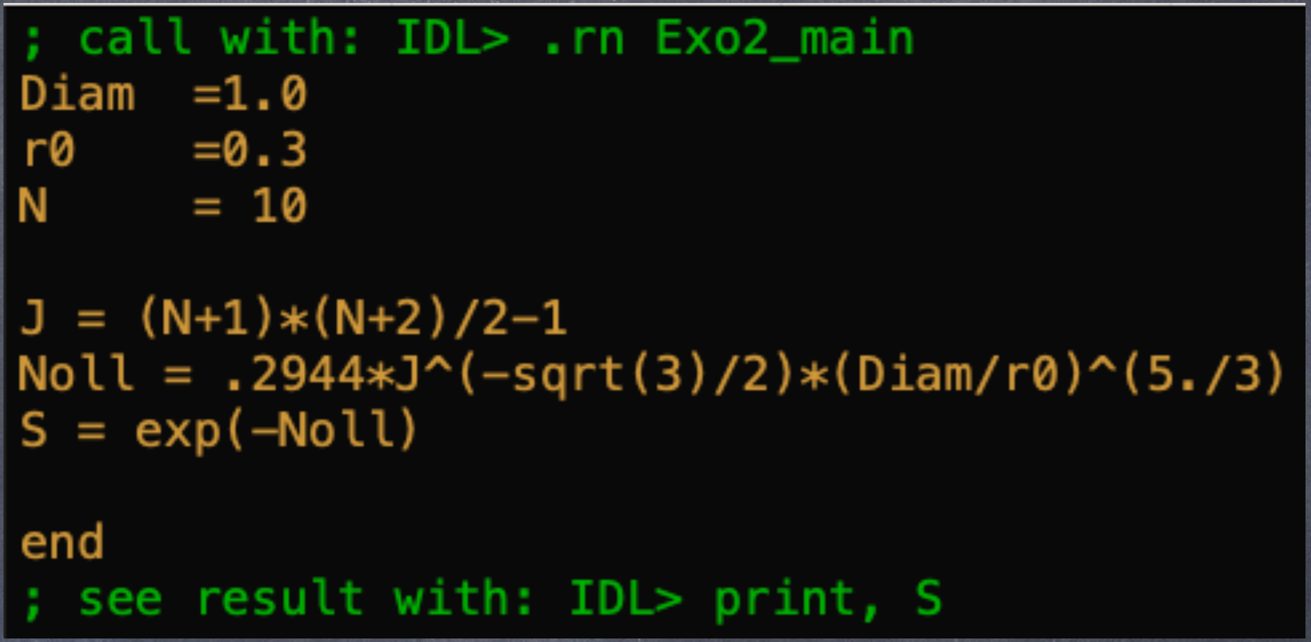
where i and j are the indices in the direct space, k and l are the indices in the FFT space, {} stands for either *real part of* or *imaginary part of*, $\imath$ is the imaginary unit, and θ is the random uniformly distributed phase (between $-\pi$ and π). The factor $\sqrt{2}$ comes from the fact that here we use both the real and imaginary parts of the original complex generated FFT phase screens, which are independent one from the other [4]. This kind of phase screen suffers, however, from the lack of spatial frequencies lower than the inverse of the necessarily finite length L of the simulated array.

(IDL: 4 kind of routines/scripts)



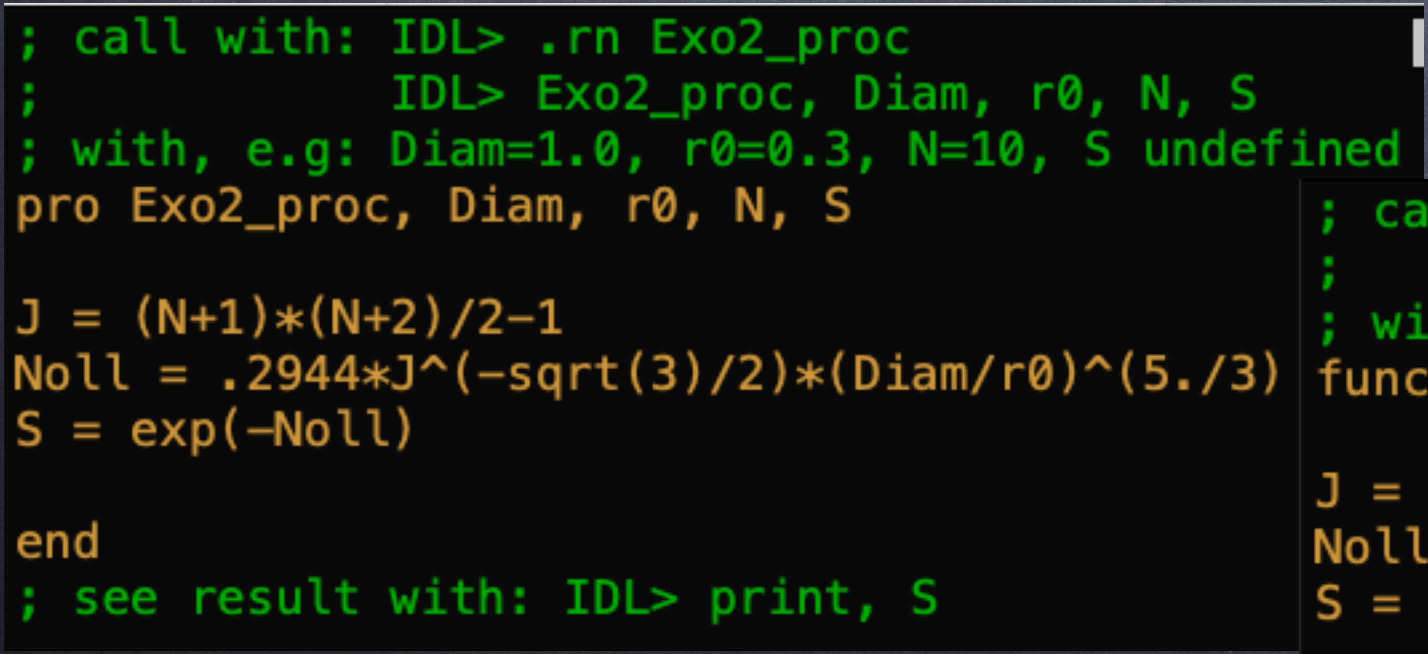
```
; call with: IDL> @Exo2
Diam  =1.0
r0    =0.3
N     = 10

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.



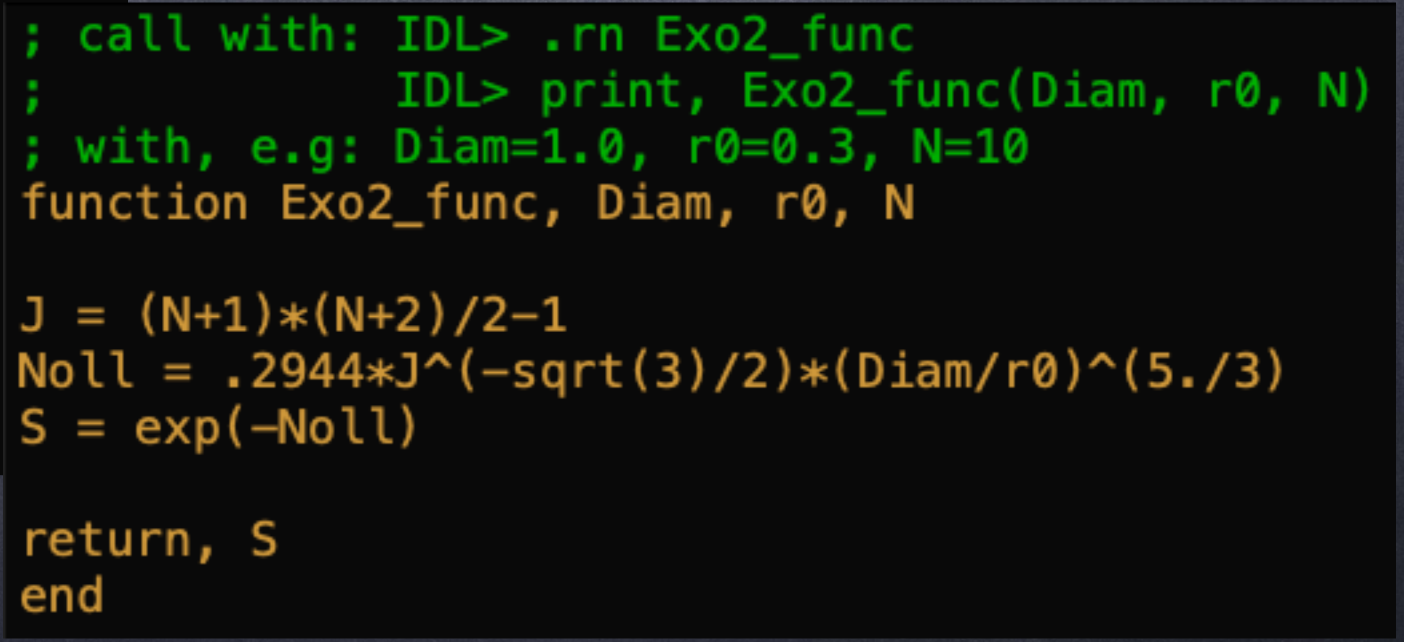
```
; call with: IDL> .rn Exo2_main
Diam  =1.0
r0    =0.3
N     = 10

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.

(IDL: other useful remarks)

- IDL help is called with: IDL>> ?
- '?' opens with a defined browser the file 'idl.htm', which can be also found directly here: **/usr/local/harris/idl89/help/online_help/Subsystems/idl/idl.htm**
- Or also with the help of the unix command 'find':

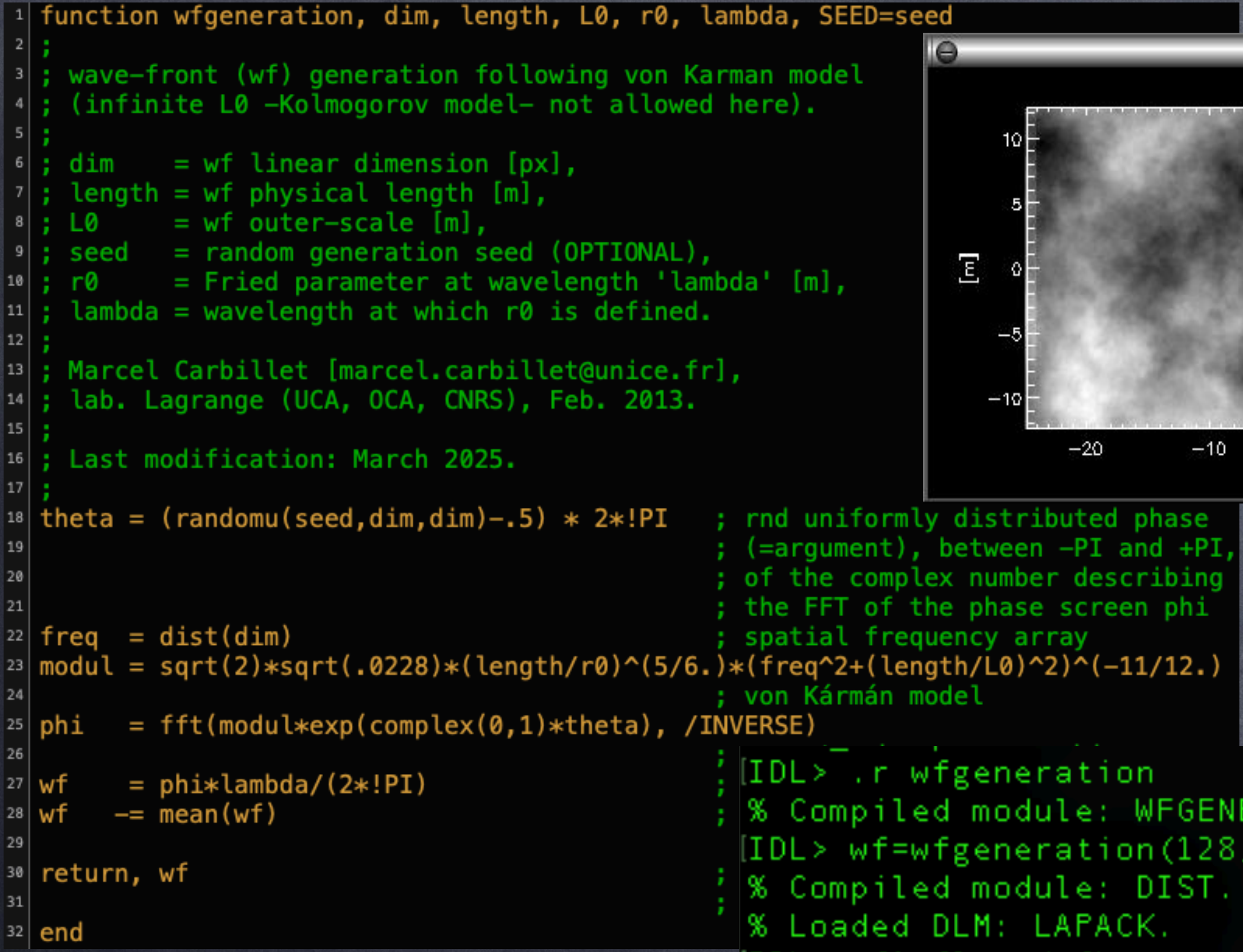
  linux>> cd /

  linux>> find . -name idl.htm
- See also (for routines which are part of a third library):

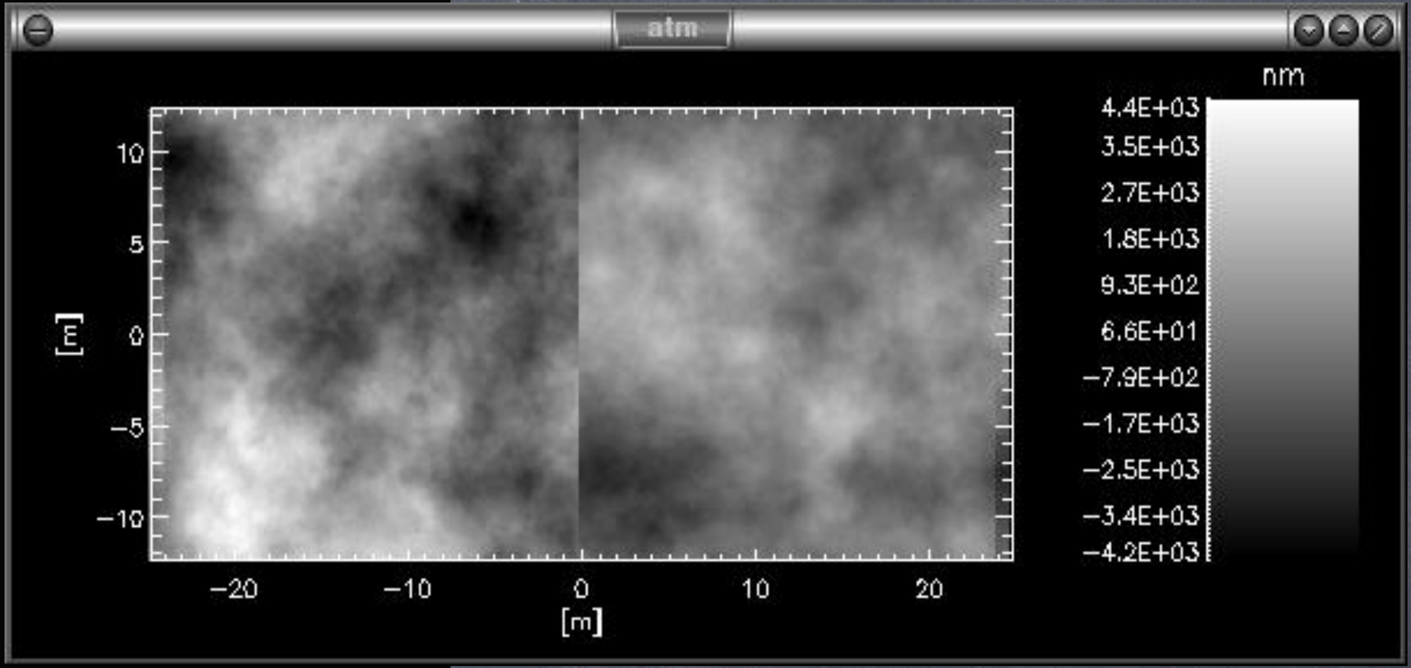
  IDL>> doc_library, 'routine_name'
- Return to main level of programming after a crash:

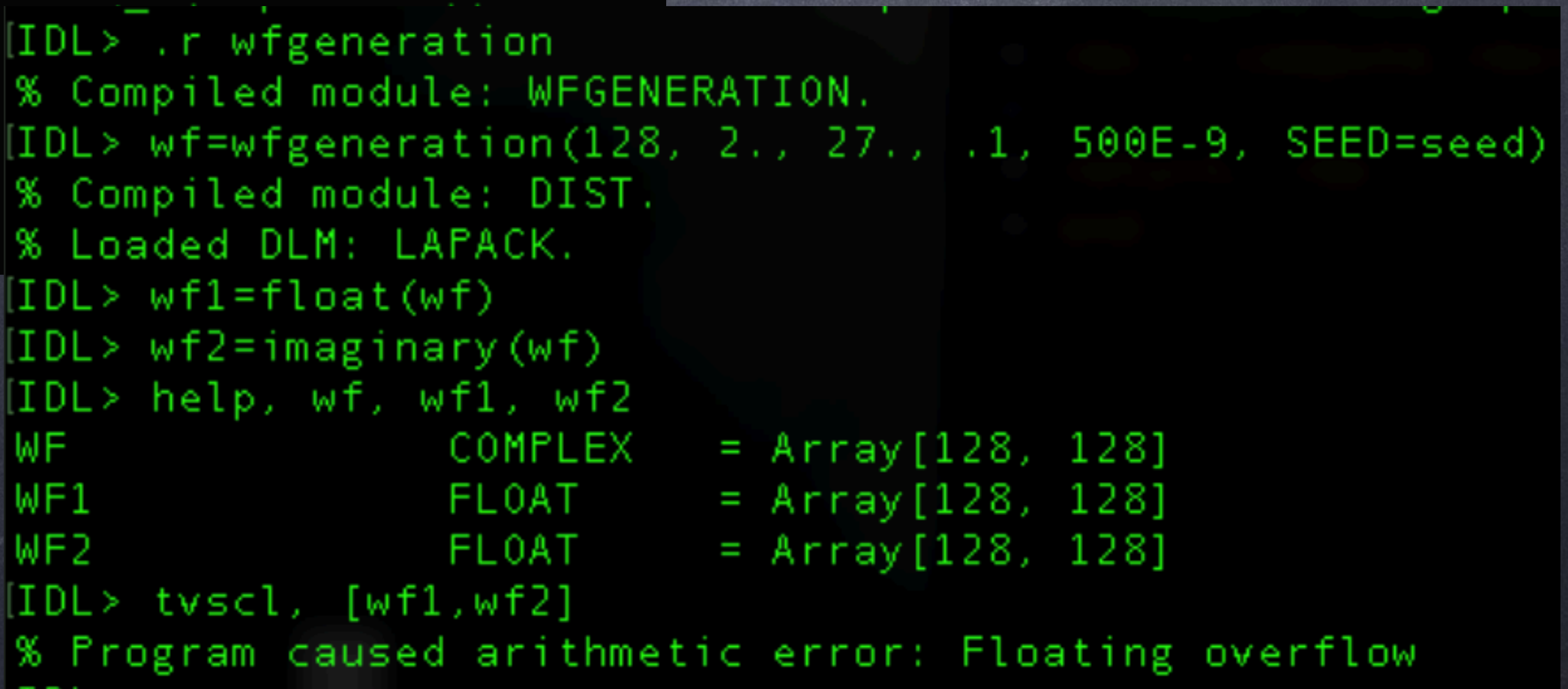
  IDL>> retall
- Details on a var. xxx: **idl> help, xxx** (all var.: **idl> help**)
- Close last opened window: **idl> wdelete**

Images & turbulence — 18



```
function wfgeneration, dim, length, L0, r0, lambda, SEED=seed
;
; wave-front (wf) generation following von Karman model
; (infinite L0 -Kolmogorov model- not allowed here).
;
; dim    = wf linear dimension [px],
; length = wf physical length [m],
; L0     = wf outer-scale [m],
; seed   = random generation seed (OPTIONAL),
; r0     = Fried parameter at wavelength 'lambda' [m],
; lambda = wavelength at which r0 is defined.
;
; Marcel Carbillet [marcel.carbillet@unice.fr],
; lab. Lagrange (UCA, OCA, CNRS), Feb. 2013.
;
; Last modification: March 2025.
;
theta = (randomu(seed,dim,dim)-.5) * 2*!PI    ; rnd uniformly distributed phase
                                              ; (=argument), between -PI and +PI,
                                              ; of the complex number describing
                                              ; the FFT of the phase screen phi
freq  = dist(dim)                             ; spatial frequency array
modul = sqrt(2)*sqrt(.0228)*(length/r0)^(5/6.)*(freq^2+(length/L0)^2)^(-11/12.)
                                              ; von Kármán model
phi   = fft(modul*exp(complex(0,1)*theta), /INVERSE)
                                              ;
wf    = phi*lambda/(2*!PI)                    ;
wf   -= mean(wf)                              ;

return, wf                                    ;
                                              ;
end
```

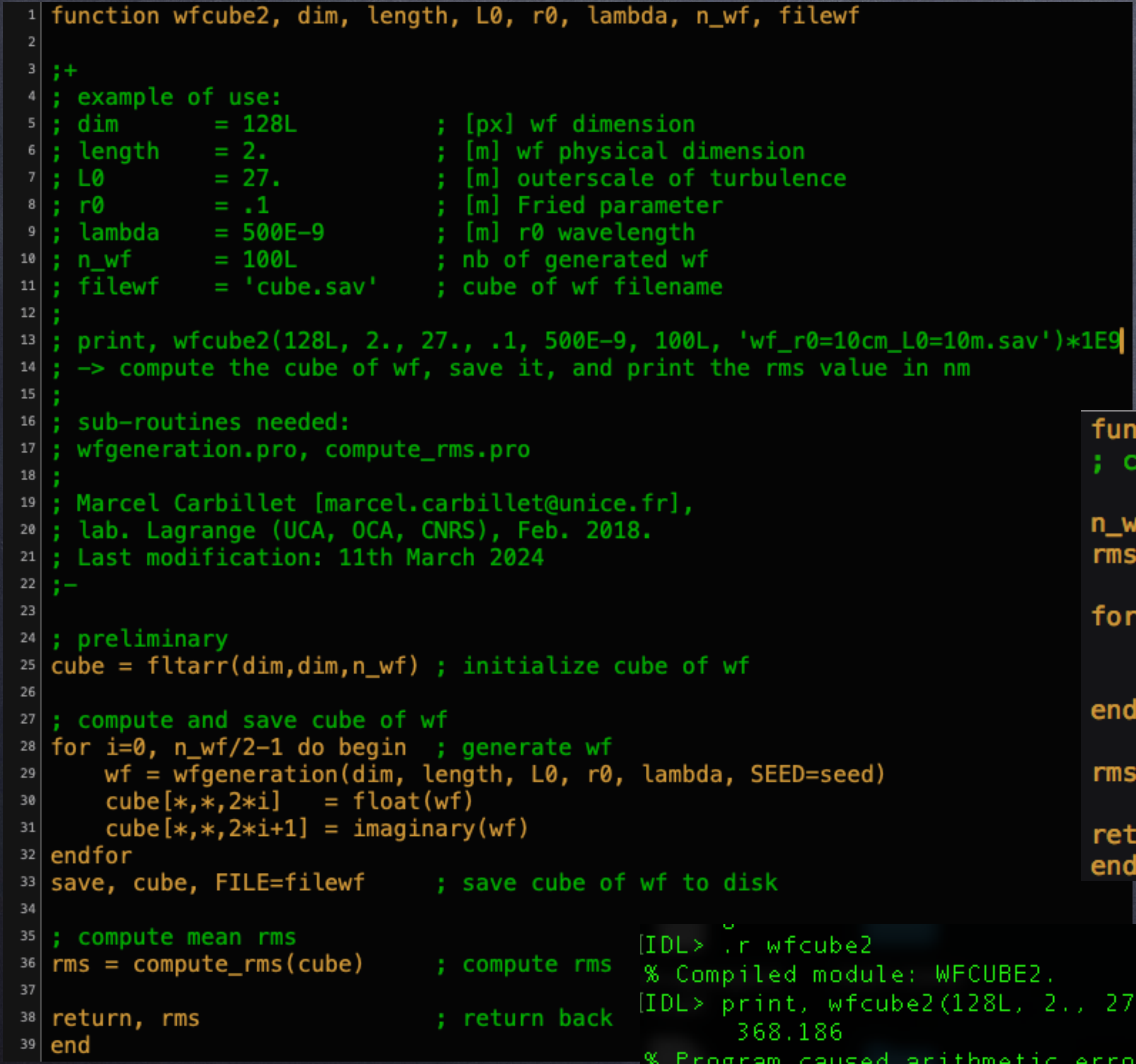




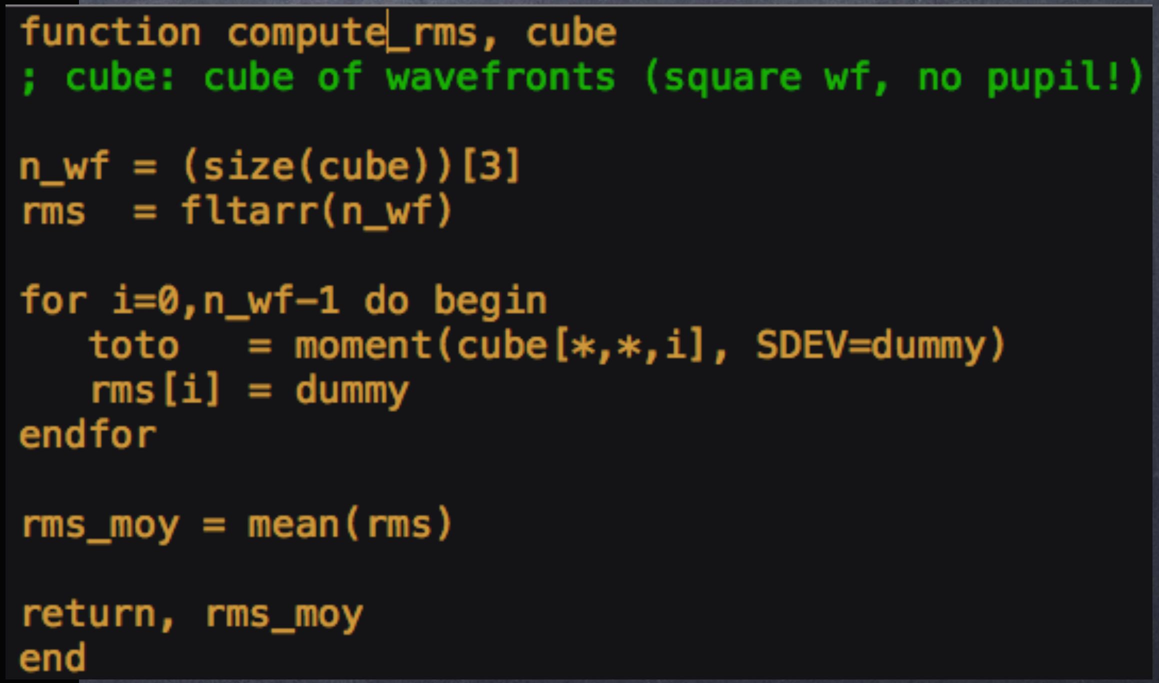
```
IDL> .r wfgeneration
% Compiled module: WFGENERATION.
IDL> wf=wfgeneration(128, 2., 27., .1, 500E-9, SEED=seed)
% Compiled module: DIST.
% Loaded DLM: LAPACK.
IDL> wf1=float(wf)
IDL> wf2=imaginary(wf)
IDL> help, wf, wf1, wf2
WF              COMPLEX   = Array[128, 128]
WF1             FLOAT     = Array[128, 128]
WF2             FLOAT     = Array[128, 128]
IDL> tvscl, [wf1,wf2]
% Program caused arithmetic error: Floating overflow
```

Images & turbulence — 19

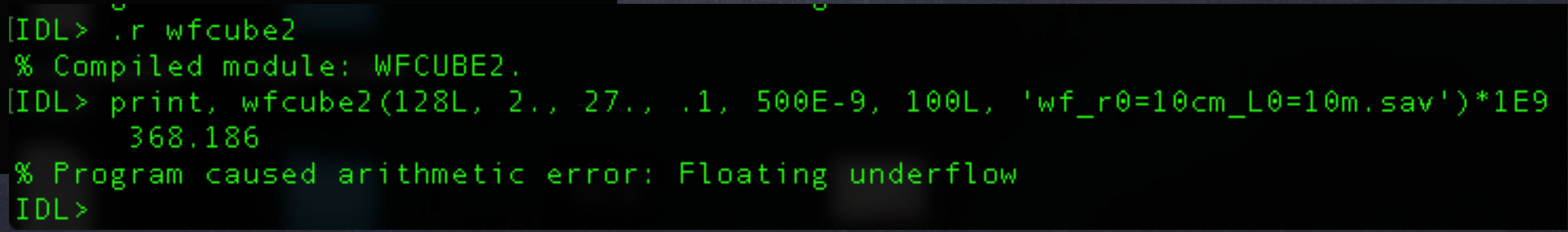
<u>wf generation:</u> generate a cube of statistically independent wf (e.g 100)
=> compute mean *rms* for different [r_0 , L_0]



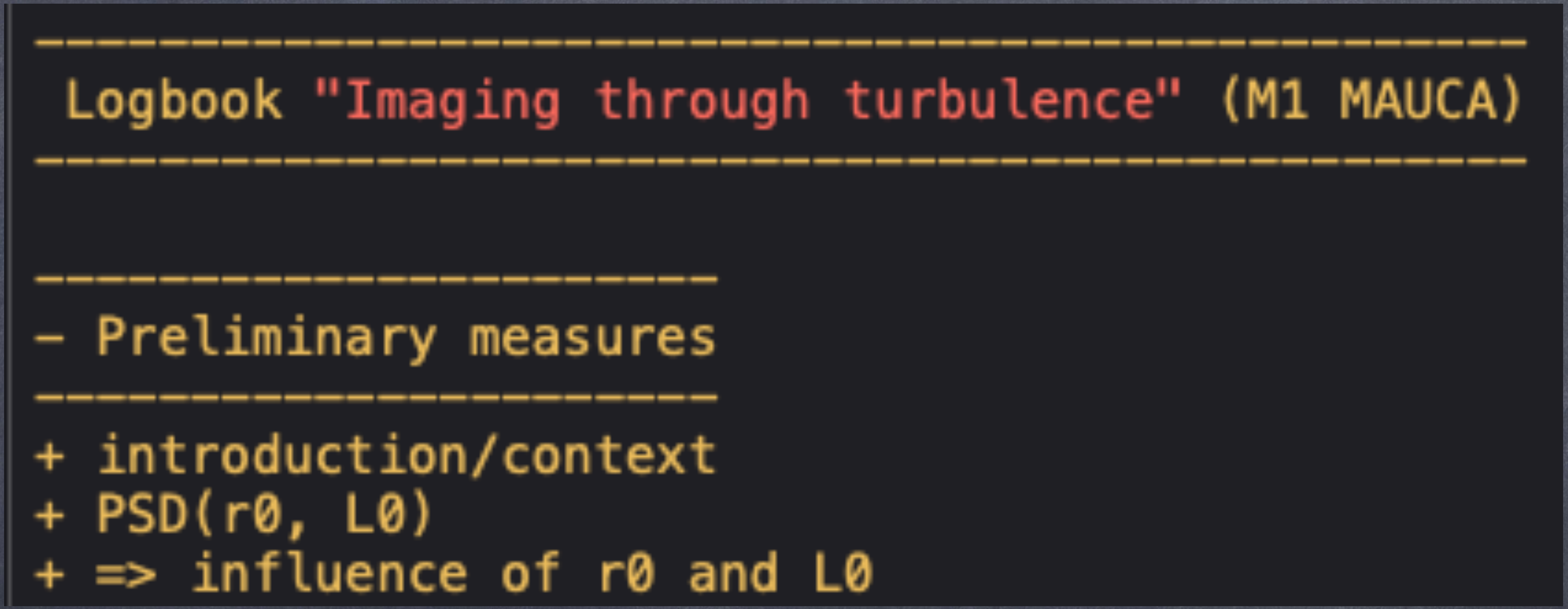
```
function wfcube2, dim, length, L0, r0, lambda, n_wf, filewf

;+
; 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, wfcube2(128L, 2., 27., .1, 500E-9, 100L, 'wf_r0=10cm_L0=10m.sav')*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
```



```
function compute_rms, cube
; cube: cube of wavefronts (square wf, no pupil!)

n_wf = (size(cube))[3]
rms  = fltarr(n_wf)

for i=0,n_wf-1 do begin
    toto   = moment(cube[*,*,i], SDEV=dummy)
    rms[i] = dummy
endfor

rms_moy = mean(rms)

return, rms_moy
end
```



```
IDL> .r wfcube2
% Compiled module: WFCUBE2.
IDL> print, wfcube2(128L, 2., 27., .1, 500E-9, 100L, 'wf_r0=10cm_L0=10m.sav')*1E9
       368.186
% Program caused arithmetic error: Floating underflow
IDL>
```



```
--------------------------------------------------------------
 Logbook "Imaging through turbulence" (M1 MAUCA)
--------------------------------------------------------------


-----------------------------
- Preliminary measures
-----------------------------
+ introduction/context
+ PSD(r0, L0)
+ => influence of r0 and L0
```

(more to come...)