Images & turbulence - 13

$$\Phi_\varphi(\vec{\nu}) = 0.0228\ r_0^{-\frac{5}{3}} \left(\nu^2 + \frac{1}{\mathcal{L}_0^2}\right)^{-\frac{11}{6}}$$

Which, numerically written, and by considering wavefronts made of 'dim' pixels corresponding to 'L' meters, becomes:
(re-writing - ``de-dimensionalizing" - the equation with L0=L0 L/L and ν=ν L/L…)

```
freq = findgen(dim)
dsp  = .0228*(L/r0)^(5/3.)*L^2*(freq^2+(L/L0)^2)^(-11./6)
```

And which (with the right frequency scale) can be plot with:

```
plot_oo, 1./L*findgen(dim), dsp, XR=[1/L/1.2,dim*1/L*1.2], /XS
```

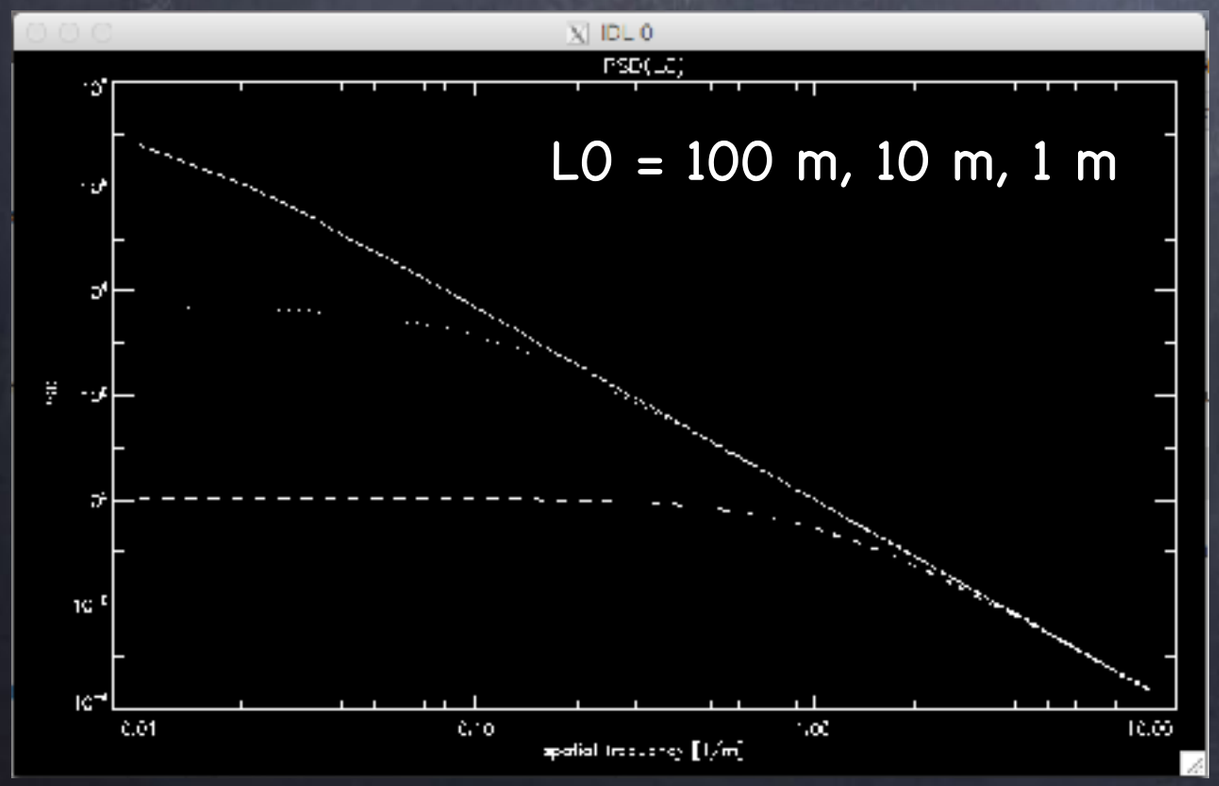
=> make a routine that computes PSD(L0,r0,dim,L) and plot it for different [r_0, L_0]… [with, for example: dim=1000, L=100., r0=0.1, L0=100.,10.,1.]

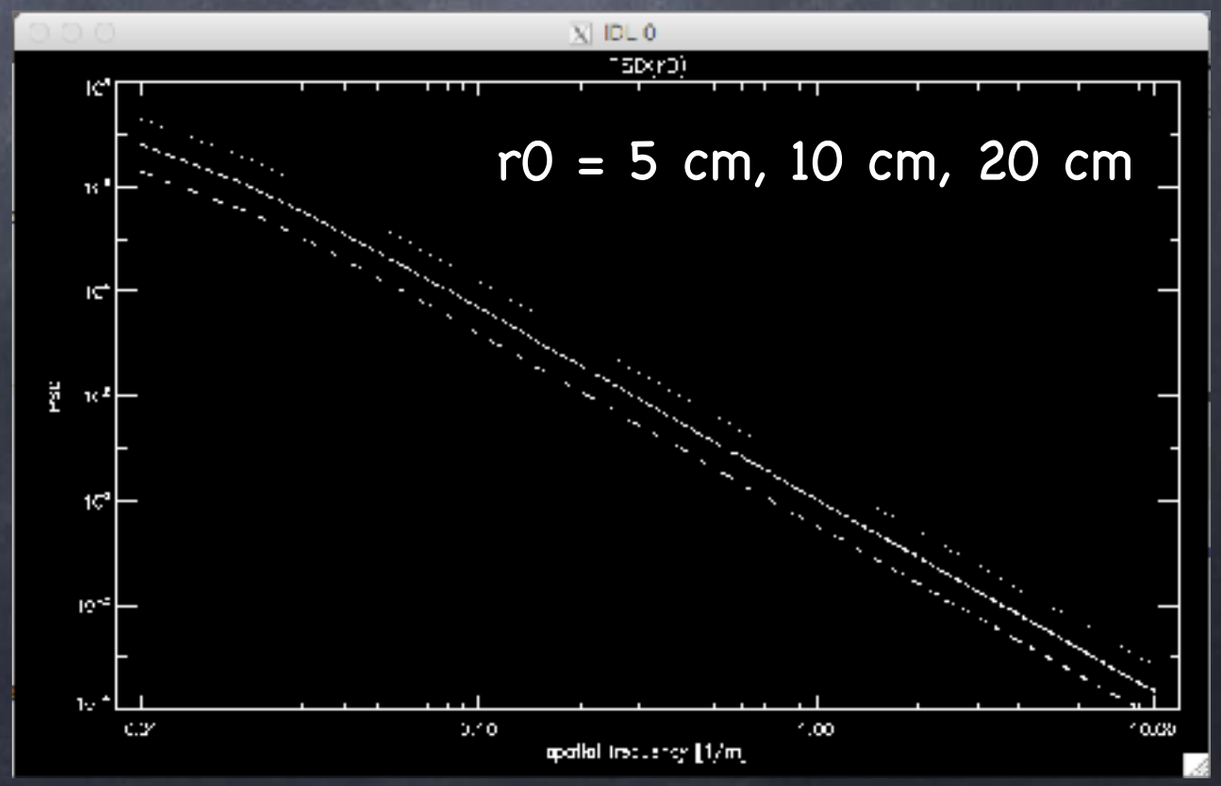
-> Also read Aime (Sec. 1 & Sec. 2) and Maire (Chap.1)…

Images & turbulence - 14

```
marcel@MBA-de-Marcel -> ssh -Y srv-etudiant.oca.eu
marcel@srv-etudiant.oca.eu's password:
Warning: No xauth data; using fake authentication data for X11 forwarding.
Last login: Wed Feb 21 0
[[marcel@srv-etudiant ~]$
IDL Version 8.5.1 (linux
ons, Inc., a subsidiary
Installation number: 358
Licensed for use by: Obs

IDL> plot, [1,2], [0,4]
```

```
1  function dsp_theo, dim, L, r0, L0
2
3  freq = findgen(dim)
4  dsp  = .0228*(L/r0)^(5/3.)*L^2*(freq^2+(L/L0)^2)^(-11./6)
5
6  ;to be plotted afterwards with:
7  ;plot_oo, 1./L*findgen(dim), dsp, XR=[1/L/1.2,dim*1/L*1.2], /XS, $
8  ;         TIT='PSD(L0)', XTIT='spatial frequency [1/m]', YTIT='PSD'
9  ;oplot   , 1./L*findgen(dim), dsp, LINE=1
10 ;playing, e.g., with L0=100.,10.,1., or r0=.05, .1, .2
11
12 return, dsp
13 end
```





Images & turbulence - 15

-> 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.

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

-> (from Carbillet & Riccardi (sec. 2))...

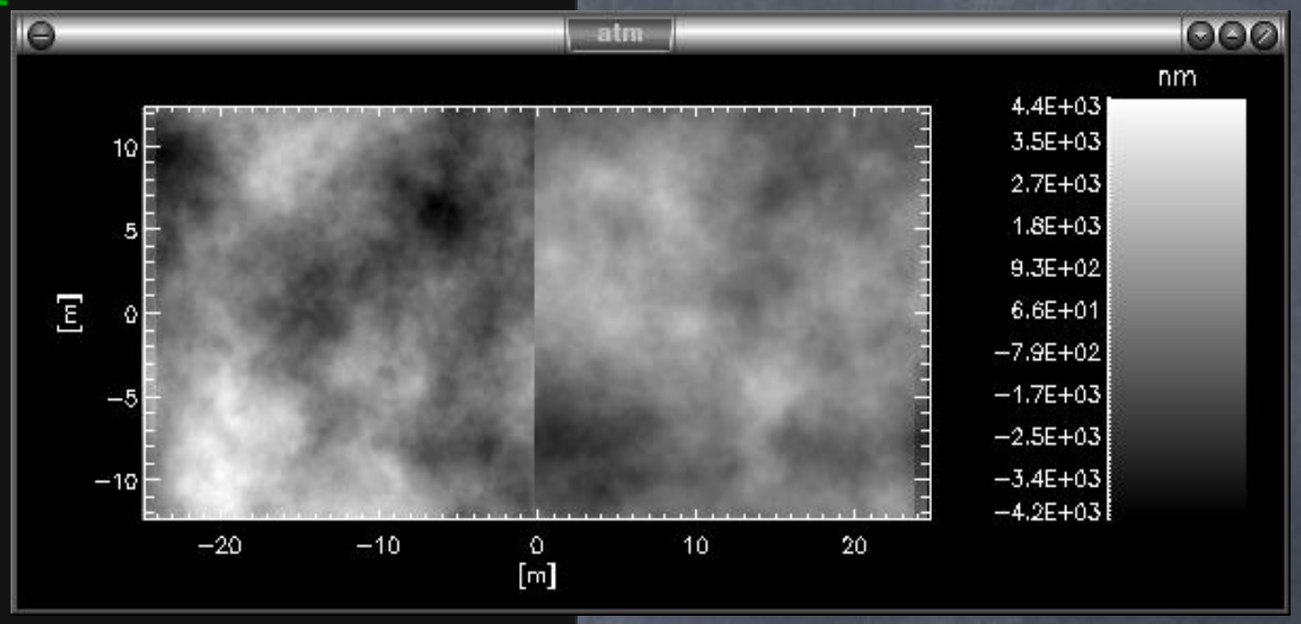
The obtained phase screen is thus numerically written

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

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.

Images & turbulence - 16

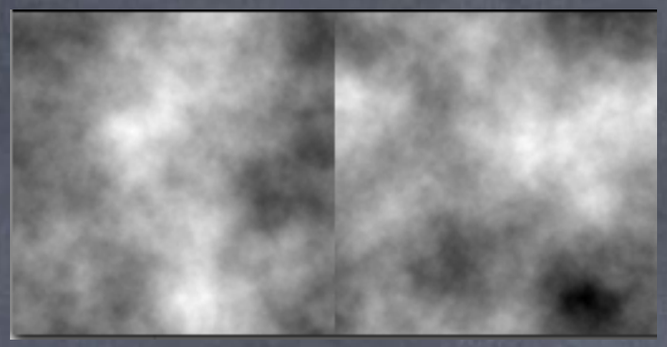
```
function wfgeneration, dim, length, L0, r0, lambda, SEED=seed
;
; wave-front (wf) generation following von Karman model
; (infinite L0 -Kolmogorov model- not allowed here).
;
; dim    = wf linear dimension [px],
; length = wf physical length [m],
; L0     = wf outer-scale [m],
; seed   = random generation 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: Feb. 2018.
;
phase = (randomu(seed,dim,dim)-.5) * 2*!PI    ; rnd uniformly distributed phase
                                              ; (between -PI and +PI)
rr = dist(dim)
modul = (rr^2+(length/L0)^2)^(-11/12.)        ; von Karman model

screen = fft(modul*exp(complex(0,1)*phase), /INVERSE)
                                              ; compute wf
screen *= sqrt(2)*sqrt(.0228)*(length/r0)^(5/6.)*lambda/(2*!PI)
                                              ; proper normalization of wf
screen -= mean(screen)                        ; force mean to zero

return, screen                                ; deliver 2 independent wf:
                                              ; float(screen) & imaginary(screen)
end
```



wf generation:
generate a cube of statistically independent wf (typically 100)...
=> compute mean rms for different [r_0 , L_0]

Images & turbulence - 17

```
IDL> .r wfgeneration
% Compiled module: WFGENERATION.
IDL> wf=wfgeneration(128,2.,27.,.1,500E-9,SEED=seed)
% Compiled module: DIST.
IDL> wf1=float(wf)
IDL> wf2=imaginary(wf)
IDL> tvscl, [wf1,wf2]
IDL> wf=wfgeneration(128,2.,27.,.1,500E-9,SEED=seed)
IDL> wf1=float(wf)
IDL> wf2=imaginary(wf)
IDL> tvscl, [wf1,wf2]
IDL>
```



```
IDL> .rn wfcube
% Compiled module: WFCUBE.
IDL> print, wfcube(128L,2.,27.,.1,500E-9,100L)*1E9
% Compiled module: COMPUTE_RMS.
      367.668
% Program caused arithmetic error: Floating underflow
IDL>
```

```
function wfcube, dim, length, L0, r0, lambda, n_wf
;
; use:
; dim       = 128L      ; [px] wf dimension
; length    = 2.        ; [m] wf physical dimension
; L0        = 27.       ; [m] outerscale
; r0        = .1        ; [m] Fried parameter
; lambda    = 500E-9    ; [m] r0 wavelength
; n_wf      = 100L      ; nb of generated wf
;
; print, wfcube(dim,length,L0,r0,lambda,n_wf,filename,SEED=seed)
; -> prints the rms value
;
; sub-routines needed:
; wfgeneration.pro, calcul_rms.pro
;
; Marcel Carbillet [marcel.carbillet@unice.fr],
; lab. Lagrange (UCA, OCA, CNRS), Feb. 2018.
;
; Last modification: Feb. 2018
;
cube = fltarr(dim, dim, n_wf)

for i=0, n_wf/2-1 do begin
    wf = wfgeneration(dim, length, L0, r0, lambda, SEED=seed)
    cube[*,*,2*i]   = float(wf)
    cube[*,*,2*i+1] = imaginary(wf)
endfor

rms = compute_rms(cube)

return, rms
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
```

-> Also read Carbillet & Riccardi (introduction of Sec. 2)...