Object–Oriented Matlab Adaptive Optics

—

User Guide

R. Conan*

November 22, 2013

Contents

1 Introduction

Object–Oriented *Matlab*® Adaptive Optics (OOMAO) is a library of *Matlab*® classes. Objects from the different classes are assembled to perform the numerical modeling of Adaptive Optics systems. OOMAO can be seen as an extension of the *Matlab*® language. Overloaded *Matlab*® operators are used to propagate the wavefront through the system and to update objects status.

A class is a container for a set of parameters and functions. In the *Matlab*® nomenclature, the parameters and the functions of a class are called the properties and the methods, respectively. A object is created (or instantiated) from a class by calling the class constructor method. The proper syntax of the constructor can be discovered by invoking *help class_name* at the *Matlab*® prompt.

*on–leave from the University of Victoria (Canada) at the "Centro de Radioastronoma y Astrofsica, UNAM" (Morelia, Mich., Mexico)

A more complete description of a class properties and methods is given by *doc class_name*.

The constructor method will set only a limited number of properties, other properties will be set to a default value. All the properties can be altered afterward anyway. A property is read or set through object_name.property_name.

The main classes used during a simulation are

- source,
- atmosphere,
- telescope,
- shackHartmann,
- deformableMirror.

This document is not a full documentation of OOMAO. The complete documentation is embedded into the code following *Matlab*® documentation standard and can be extracted with the *doc* command. This document is an introduction to the use of the library, it doesn't gives for example the list of all the properties and methods of each class. The classes description generated with *doc* is the library reference documentation.

The library is using Git (http://git-scm.com/), a version control system, to keep track of the changes in OOMAO. OOMAO sources are hosted by GitHub (http://github.com/rconan/OOMAO). The last version can be either downloaded from the web–site or a fork of the current version can be created. The changes in the fork version are also managed by git and they can be latter merged into the main repository.

2 The source class

The source class has a very important role in the OOMAO library as it is the link between other classes. A source object carries a wavefront, both amplitude and phase, through different objects representing the atmosphere, the telescope, the wavefront sensor, ... Both natural guide star and Laser guide star of Adaptive Optics can be simulated. A simple on–axis natural guide star object is created by calling the source constructor without parameters:

```
>>  ngs = source;
 @(source)> Created!
___ SOURCE ___
 Obj  zen[arcsec] azim[deg]  height[m]  lambda[micron] magnitude
  1    0.00      0.00       Inf    0.550       0.00
----------------------------------------------------
```
msnippets/s2-m1.mo

The first time a object from an OOMAO class is created a message let the user aware he is using the library. A summary of the object main parameters is also displayed. In this example, zenith and azimuthal angle are both set to zero, the star height is infinite, the wavelength is 550nm and no magnitude has been set.

A more elaborated definition of a source object can be

```
>>  src = source('zenith',30*constants.arcsec2radian,...
          'wavelength',photometry.H,...
          'magnitude',12);
 @(source)> Created!
___ SOURCE ___
 Obj  zen[arcsec] azim[deg]  height[m]  lambda[micron] magnitude
  1   30.00     0.00      Inf    1.654      12.00
----------------------------------------------------
```

msnippets/s2-m2.mo

Here calls to the properties of the static classes

```
>>  constants

ans =

  <a href="matlab:helpPopup constants" style="font-weight:bold">constants</a> with properties

   radian2arcsec: 2.0626e+05
      radian2mas: 2.0626e+08
    radian2arcmin: 3.4377e+03
   arcsec2radian: 4.8481e-06
   arcmin2radian: 2.9089e-04
           plank: 6.6261e-34
               c: 299792458
              Me: 5.9722e+24
              Re: 6378140
               G: 6.6700e-11
```

msnippets/s2-m3.mo

and

```
>>  photometry.H

ans =

   H
```

msnippets/s2-m4.mo

are used to convert arc second to radian and to get the wavelength corresponding to the H band in the standard photometric system.

The propagation of the wavefront is performed with the overloaded *Matlab*® operators *times* or *.** and *mtimes* or *.**. In the following example:

```
>>  disp([src.amplitude src.phase])
>>  wave = sqrt(2)*exp(1i*3/4);
>>  src = src.*wave;
>>  disp([src.amplitude src.phase])
   1.4142   0.7500

>>  src = src*wave;
>>  disp([src.amplitude src.phase])
   2.0000   1.5000

>>  src = src.*wave;
>>  disp([src.amplitude src.phase])
   1.4142   0.7500
```
msnippets/s2-m5.mo

a scalar complex wavefront is created and the amplitude and phase of the src object is set with the .* method. The src is propagated through the *wave* with the * method and finally reset to the original wavefront values. Each time a call to .* is done the amplitude and phase values are reset to they default value 1 and 0, respectively and then * is called. When * is called the amplitude is multiplied by the amplitude of the right hand side object and the phase of the right hand side object is added to the phase of the source object.

The * method is calling the *relay* method of the right hand side object. Most of the OOMAO classes have a *relay* method allowing them to be used with the .* and * operators of the class source. It works too with the *wave* complex number defined above because a *relay* function has been written for complex arrays.

```
>>  type relay

function relay(wave,src)
%% RELAY Source object wave setting
%
% relay(wave,src) sets the amplitude and phase of the source object based
% on wave input. wave can be either a numerical array of complex amplitudes
% or a cell of the form { amplitude , phase }

nSrc = numel(src);
for kSrc = 1:nSrc
   if iscell(wave)
      nWave = size(wave{2},3);
      if nWave==1 || nWave~=nSrc
         src(kSrc).mask      = wave{1}>0;
         src(kSrc).amplitude = wave{1};
         src(kSrc).phase     = wave{2};
      else
         src(kSrc).mask      = wave{1}(:,:,kSrc)>0;
         src(kSrc).amplitude = wave{1}(:,:,kSrc);
         src(kSrc).phase     = wave{2}(:,:,kSrc);
      end
   else
      src(kSrc).mask      = abs(wave)>0;
      src(kSrc).amplitude = abs(wave);
      src(kSrc).phase     = angle(wave);
   end
end
```
msnippets/s2-m6.mo

A OOMAO object is always a handle object meaning that the object variable is the reference to the object not its value. So the command

```
>>  ngs
___ SOURCE ___
 Obj  zen[arcsec] azim[deg]  height[m]  lambda[micron] magnitude
  1     0.00      0.00       Inf    0.550        0.00
----------------------------------------------------------
>>  ngs1 = ngs;
```
msnippets/s2-m7.mo

will create a second pointer to the same object as shown below

```
>>  ngs1.azimuth = pi/4;
>>  ngs
___ SOURCE ___
 Obj   zen[arcsec] azim[deg]  height[m]  lambda[micron] magnitude
  1     0.00      45.00       Inf    0.550        0.00
----------------------------------------------------------
```
msnippets/s2-m8.mo

The second object must be created with a new call to the class constructor

```
>>  ngs1 = source('azimuth',pi/4);
 @(source)> Created!
___ SOURCE ___
 Obj   zen[arcsec] azim[deg]  height[m]  lambda[micron] magnitude
  1     0.00      45.00       Inf    0.550        0.00
----------------------------------------------------------
```
msnippets/s2-m9.mo

OOMAO objects are removed from the *Matlab*® workspace like any other variables using the *clear* command

```
>>  clear src ngs1 ngs
 @(source)> Terminated!
 @(source)> Terminated!
 @(source)> Terminated!
```
msnippets/s2-m10.mo

When no OOMAO objects are left in the workspace, a message inform the user.

A constellation or asterism of sources consists in an array of sources that can be created with a single call to the source constructor:

```
>>  srcs = source('zenith',[0,30]*constants.arcsec2radian,'azimuth',[0,0]);
 @(source)> 2 created!
___ SOURCE ___
 Obj   zen[arcsec] azim[deg]  height[m]  lambda[micron] magnitude
  1     0.00       0.00       Inf    0.550        0.00
  2    30.00       0.00       Inf    0.550        0.00
----------------------------------------------------------
```
msnippets/s2-m11.mo

If the sources are evenly located on a ring, the syntax

```
>>  srcs = source('asterism',{[4,1*constants.arcmin2radian,pi/3]})
 @(source)> 4 created!
___ SOURCE ___
 Obj   zen[arcsec] azim[deg]  height[m]  lambda[micron] magnitude
  1    60.00      60.00      Inf    0.550        0.00
  2    60.00     150.00      Inf    0.550        0.00
  3    60.00     240.00      Inf    0.550        0.00
  4    60.00     330.00      Inf    0.550        0.00
----------------------------------------------------
 @(source)> Terminated!
___ SOURCE ___
 Obj   zen[arcsec] azim[deg]  height[m]  lambda[micron] magnitude
  1    60.00      60.00      Inf    0.550        0.00
  2    60.00     150.00      Inf    0.550        0.00
  3    60.00     240.00      Inf    0.550        0.00
  4    60.00     330.00      Inf    0.550        0.00
----------------------------------------------------
```
msnippets/s2-m12.mo

will create an array of 4 sources on a ring of 2 arc–minute diameter with a 60 degree azimuth offset. One on–axis source is added to the 4 sources asterism above by adding a vector [zenith,azimuth] in the asterism property

```
>>  srcs = source('asterism',{[0,0],[4,1*constants.arcmin2radian,pi/3]})
___ SOURCE ___
 Obj   zen[arcsec] azim[deg]  height[m]  lambda[micron] magnitude
  1     0.00       0.00      Inf    0.550        0.00
  2    60.00      60.00      Inf    0.550        0.00
  3    60.00     150.00      Inf    0.550        0.00
  4    60.00     240.00      Inf    0.550        0.00
  5    60.00     330.00      Inf    0.550        0.00
----------------------------------------------------
 @(source)> Terminated!
___ SOURCE ___
 Obj   zen[arcsec] azim[deg]  height[m]  lambda[micron] magnitude
  1     0.00       0.00      Inf    0.550        0.00
  2    60.00      60.00      Inf    0.550        0.00
  3    60.00     150.00      Inf    0.550        0.00
  4    60.00     240.00      Inf    0.550        0.00
  5    60.00     330.00      Inf    0.550        0.00
----------------------------------------------------
```
msnippets/s2-m13.mo

A Laser guide star is simply created by setting the finite height of the source

```
>>  lgs = source('height',90e3,'wavelength',photometry.Na);
___ SOURCE ___
 Obj  zen[arcsec] azim[deg]  height[m]  lambda[micron] magnitude
  1     0.00       0.00    90000.00    0.589        0.00
----------------------------------------------------------
```
msnippets/s2-m14.mo

This will create a Sodium Laser guide star a 90km. A 10km thick Laser guide star is created with

```
>>  lgs = source('height',linspace(85,95,11)*1e3,...
          'wavelength',photometry.Na);
___ SOURCE ___
 Obj  zen[arcsec] azim[deg]  height[m]  lambda[micron] magnitude
  1     0.00       0.00    [85000 95000]    0.589        0.00
----------------------------------------------------------
 @(source)> Terminated!
```
msnippets/s2-m15.mo

and an asterism of 6 Laser guide stars is obtained with

```
>>  lgs = source('asterism',{[0,0],[5,constants.arcmin2radian,0]},...
          'height',linspace(85,95,11)*1e3,...
          'wavelength',photometry.Na);
 @(source)> 66 created!
___ SOURCE ___
 Obj  zen[arcsec] azim[deg]  height[m]  lambda[micron] magnitude
  1     0.00       0.00    [85000 95000]    0.589        0.00
  2    60.00       0.00    [85000 95000]    0.589        0.00
  3    60.00      72.00    [85000 95000]    0.589        0.00
  4    60.00     144.00    [85000 95000]    0.589        0.00
  5    60.00     216.00    [85000 95000]    0.589        0.00
  6    60.00     288.00    [85000 95000]    0.589        0.00
----------------------------------------------------------
 @(source)> Terminated!
```
msnippets/s2-m16.mo

3 The atmosphere class

The atmosphere class contains all the parameters defining the atmosphere. A single ground layer atmosphere with a 15cm Fried parameter in the visible is created with

```
>>  atm = atmosphere(photometry.V,15e-2);
 @(atmosphere)> Created!
___ ATMOSPHERE ___
 Kolmogorov-Tatarski atmospheric turbulence:
  . wavelength = 0.55micron,
  . r0       = 15.00cm,
  . seeing    = 0.74arcsec,

----------------------------------------------------
  Layer  Altitude[m]  fr0  wind([m/s] [deg])  D[m]  res[px]
   1      0.00    1.00   ( )
----------------------------------------------------
```
msnippets/s3-m17.mo

An outer scale of turbulence can be added to the constructor call

```
>>  atm = atmosphere(photometry.V,15e-2,30);
 @(atmosphere)> Created!
___ ATMOSPHERE ___
 Von Karman atmospheric turbulence
  . wavelength = 0.55micron,
  . r0       = 15.00cm,
  . L0       = 30.00m,
  . seeing    = 0.74arcsec,

----------------------------------------------------
  Layer  Altitude[m]  fr0  wind([m/s] [deg])  D[m]  res[px]
   1      0.00    1.00   ( )
----------------------------------------------------
 @(atmosphere)> Terminated!
```
msnippets/s3-m18.mo

The properties of the layer can be specified with

```
>>  atm = atmosphere(photometry.V,15e-2,30,...
      'altitude',10e3,...
      'windSpeed',10,...
      'windDirection',0);
 @(atmosphere)> Created!
___ ATMOSPHERE ___
 Von Karman atmospheric turbulence
  . wavelength  = 0.55micron,
  . r0          = 15.00cm,
  . L0          = 30.00m,
  . seeing      = 0.74arcsec,
  . theta0(37%) = 1.69arcsec,
  . tau0(37%)   = 8.22millisec
----------------------------------------------------
 Layer  Altitude[m]  fr0  wind([m/s] [deg])  D[m]  res[px]
  1     10000.00    1.00  (10.00  0.00)
----------------------------------------------------
 @(atmosphere)> Terminated!
```
msnippets/s3-m19.mo

When the altitude is different than 0km, the isoplanatic angle θ_0 is computed and if the wind vector is defined too, τ_0 is also computed. Both θ_0 and τ_0 correspond to the $\exp(-1)$ decay of the angular and temporal wavefront covariance, respectively, following Roddier definition. θ_0 and τ_0 can be computed for any other covariance decay value by setting the property *coherenceFunctionDecay*, for example to 50%,

```
>>  atm.coherenceFunctionDecay = 0.5
___ ATMOSPHERE ___
 Von Karman atmospheric turbulence
  . wavelength  = 0.55micron,
  . r0          = 15.00cm,
  . L0          = 30.00m,
  . seeing      = 0.74arcsec,
  . theta0(50%) = 1.34arcsec,
  . tau0(50%)   = 6.52millisec
----------------------------------------------------
 Layer  Altitude[m]  fr0  wind([m/s] [deg])  D[m]  res[px]
  1     10000.00    1.00  (10.00  0.00)
----------------------------------------------------
```
msnippets/s3-m20.mo

Two usual definitions for θ_0 and τ_0 are the aforementioned Roddier $\exp(-1)$ decay and the Fried 1rd^2 structure function value. These definition can be set namely with

```
>>  atm.coherenceFunctionDecay = 'fried'
___ ATMOSPHERE ___
 Von Karman atmospheric turbulence
  . wavelength  = 0.55micron,
  . r0          = 15.00cm,
  . L0          = 30.00m,
  . seeing      = 0.74arcsec,
  . theta0(61%) = 1.09arcsec,
  . tau0(61%)   = 5.31millisec
----------------------------------------------------
  Layer  Altitude[m]  fr0  wind([m/s] [deg])  D[m]  res[px]
   1     10000.00    1.00   (10.00  0.00)
----------------------------------------------------
```
msnippets/s3-m21.mo

and

```
>>  atm.coherenceFunctionDecay = 'roddier'
___ ATMOSPHERE ___
 Von Karman atmospheric turbulence
  . wavelength  = 0.55micron,
  . r0          = 15.00cm,
  . L0          = 30.00m,
  . seeing      = 0.74arcsec,
  . theta0(37%) = 1.69arcsec,
  . tau0(37%)   = 8.22millisec
----------------------------------------------------
  Layer  Altitude[m]  fr0  wind([m/s] [deg])  D[m]  res[px]
   1     10000.00    1.00   (10.00  0.00)
----------------------------------------------------
```
msnippets/s3-m22.mo

Setting the wavelength to another value updates the wavelength dependent properties:

```
>>  atm.wavelength = photometry.J
___ ATMOSPHERE ___
 Von Karman atmospheric turbulence
  . wavelength  = 1.22micron,
  . r0       = 38.83cm,
  . L0       = 30.00m,
  . seeing     = 0.63arcsec,
  . theta0(37%) = 4.69arcsec,
  . tau0(37%)  = 22.74millisec
----------------------------------------------------
  Layer   Altitude[m]  fr0   wind([m/s] [deg])  D[m]   res[px]
   1    10000.00    1.00   (10.00  0.00)
----------------------------------------------------
```
msnippets/s3-m23.mo

A multilayer atmosphere is created by setting the appropriates vectors of altitudes, wind speeds and directions and turbulence strengths:

```
>>  atm = atmosphere(photometry.V,15e-2,30,...
      'altitude',[0, 5,12]*1e3,...
      'fractionnalR0',[0.5,0.3,0.2],...
      'windSpeed',[10,5,15],...
      'windDirection',[0,pi/3,pi]);
 @(atmosphere)> Created!
___ ATMOSPHERE ___
 Von Karman atmospheric turbulence
  . wavelength  = 0.55micron,
  . r0       = 15.00cm,
  . L0       = 30.00m,
  . seeing     = 0.74arcsec,
  . theta0(37%) = 3.22arcsec,
  . tau0(37%)  = 8.33millisec
----------------------------------------------------
  Layer   Altitude[m]  fr0   wind([m/s] [deg])  D[m]   res[px]
   1       0.00     0.50   (10.00  0.00)
   2     5000.00    0.30   ( 5.00  60.00)
   3    12000.00    0.20   (15.00 180.00)
----------------------------------------------------
 @(atmosphere)> Terminated!
```
msnippets/s3-m24.mo

The fractional r_0 is given by

$$f_{r_0} = \left(\frac{r_0(h)}{r_0}\right)^{5/3} = \frac{C_n^2(h)\Delta h}{\int \mathrm{d}h C_n^2(h)}.$$

The atmosphere class alone cannot creates the phase screens in the layers as the phase screen extent is depending on the telescope diameter and field–

of–view. An atmosphere object needs to be coupled with a telescope object to create a 3–D volume of turbulence phase screens.

4 The telescope class

The telescope class has two roles

1. to define the telescope parameters,
2. to define the phase screens in the turbulence layers set by an atmosphere object.

In its simplest form, a 8m diameter telescope is created with

```
>>  tel = telescope(8);
 @(telescope)> Created!
___ TELESCOPE ___
 8.00m diameter full aperture with 50.27m^2 of light collecting area;
----------------------------------------------------
```
msnippets/s4-m25.mo

A 14% central obscuration can be added with

```
>>  tel = telescope(8,'obstructionRatio',0.14);
 @(telescope)> Created!
___ TELESCOPE ___
 8.00m diameter with a 14.00% central obstruction with 49.28m^2 of light collecting area;
----------------------------------------------------
 @(telescope)> Terminated!
```
msnippets/s4-m26.mo

A more complete description of the telescope will be set with

```
>>  tel = telescope(8,'fieldOfViewInArcmin',2,...
      'resolution',64,...
      'samplingTime',1/500);
 @(telescope)> Created!
___ TELESCOPE ___
 8.00m diameter full aperture with 50.27m^2 of light collecting area;
 the field-of-view is 2.00arcmin; the pupil is sampled with 64X64 pixels
----------------------------------------------------
 @(telescope)> Terminated!
```
msnippets/s4-m27.mo

This 8m diameter telescope has a 2 arc–minute field–of–view. The telescope pupil is sampled with 64×64 pixels and the motion of the phase screen in the telescope pupil is sampled at 500Hz.

A 3 layers atmosphere is created

```
>>  atm = atmosphere(photometry.V,15e-2,30,...
     'altitude',[0, 5,12]*1e3,...
     'fractionnalR0',[0.5,0.3,0.2],...
     'windSpeed',[10,5,15],...
     'windDirection',[0,pi/3,pi]);
 @(atmosphere)> Created!
___ ATMOSPHERE ___
 Von Karman atmospheric turbulence
  . wavelength  = 0.55micron,
  . r0       = 15.00cm,
  . L0       = 30.00m,
  . seeing     = 0.74arcsec,
  . theta0(37%) = 3.22arcsec,
  . tau0(37%)  = 8.33millisec
----------------------------------------------------
 Layer  Altitude[m]  fr0   wind([m/s] [deg])  D[m]   res[px]
  1       0.00    0.50   (10.00  0.00)
  2     5000.00    0.30   ( 5.00  60.00)
  3    12000.00    0.20   (15.00 180.00)
----------------------------------------------------
 @(atmosphere)> Terminated!
```

msnippets/s4-m28.mo

and combined with the telescope object:

```
 tel = tel + atm;
```

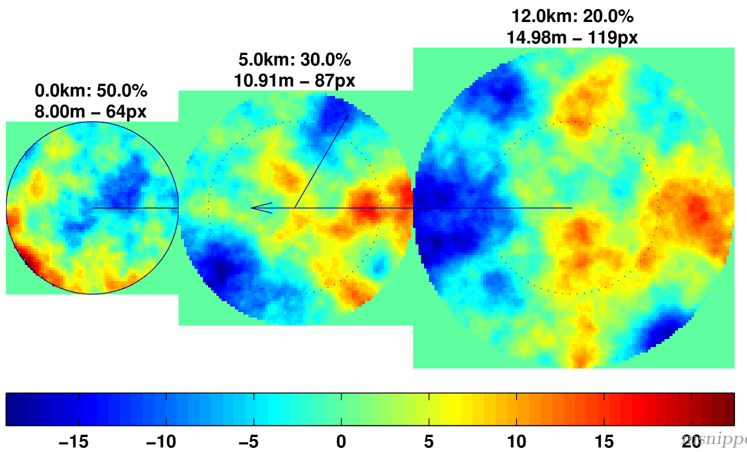
msnippets/s4-m29.mx

The phase screens size D at height h size for a field–of–view α is given by

$$D(h) = D(0) + 2h\tan(\alpha/2)$$

and the sampling mesh size $(D/(n-1))$ is kept constant through the layers. To display, the phase screens the *Matlab*® function *imagesc* can be called with the telescope object as argument:

```
>> figure
>> imagesc(tel)
```



snippets/s4-m30.mo

The first phase screens are computed with the Fourier method on a grid twice the size of the pupil diameter $D(h)$. Then the covariance method is used to derive the phase screens at next time step. The Taylor or frozen flow hypothesis is assumed for the phase screen temporal evolution. Fig. 1 displays the different masks involved in the phase screens temporal generation. The hash zone corresponds to a $n \times n$ phase screen. The red zone has a thickness of 1 pixel around the hash zone and the blue zone has a thickness of 2 pixels on the inner edge of the hash zone. Let's call x the vector of phase values in the red zone, z the vector of phase values in the blue zone and u a random vector following a standard normal distribution. x is given by

$$x = Az + L_B u$$

where

$$\begin{aligned} A &= \Gamma_{zx}^{-T}\Gamma_z \\ B &= \Gamma_x - A\Gamma_{zx} \\ B &= L_B L_B^T \\ \Gamma_x &= \langle xx^T \rangle \\ \Gamma_z &= \langle zz^T \rangle \\ \Gamma_{zx} &= \langle zx^T \rangle. \end{aligned}$$

The original n^2 phase screen is then increased to a $(n+1)^2$ phase screen with x on the rim. In this new phase screen, the phase values at the mesh location shifted of $k\tau\vec{v}$ are linearly interpolated. τ is the turbulence temporal sampling and $k = 1, 2, \ldots$ is the temporal step. The linear interpolation goes on until $k\tau\vec{v}$

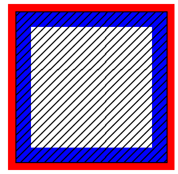


Figure 1: Phase screen zone definition.

is larger than $D/(n-1)$. Then the last n^2 phase screen is augmented of 1 pixel on the edges as previously, k is reset to 1 and the linear interpolation procedure is invoked again. This procedure allows generating infinite size phase screens for any wind speed and directions. This method is similar to the one proposed by Assemat et al. with the difference that the isotropy of the phase statistics regardless of the wind vector is respected here.

The translation of the phase screens of one time step based on the method described previously is obtained with the single command

```
>> +tel;
```

msnippets/s4-m31.mo

The visualization of the phase screens motion is done with

```
while true, imagesc(+tel), drawnow, end
```

msnippets/s4-m32.mx

The infinite while loop is stopped by hitting Ctrl–C.

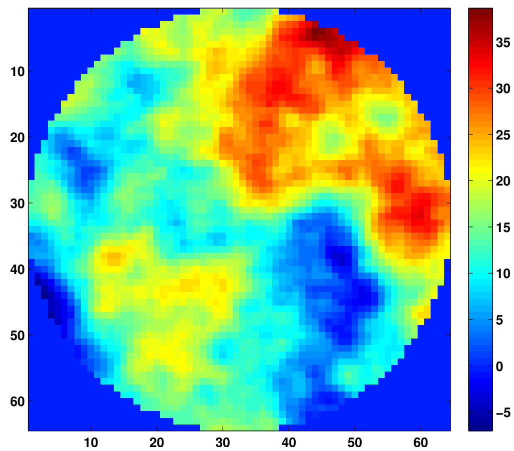
The wavefront in the telescope pupil is obtained propagating a source to the pupil through the atmosphere. Let's create an on–axis source:

```
>> ngs = source;
 @(source)> Created!
___ SOURCE ___
 Obj  zen[arcsec] azim[deg]  height[m]  lambda[micron] magnitude
  1    0.00      0.00       Inf    0.550        0.00
----------------------------------------------------------
```

msnippets/s4-m33.mo

and propagated to the pupil:

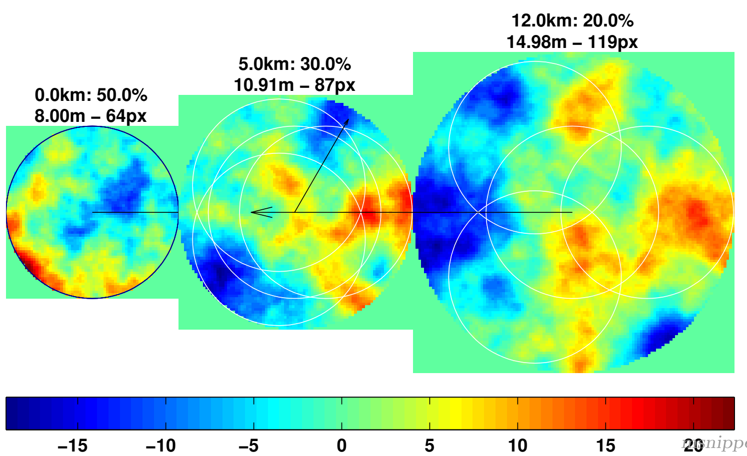
```
>>  ngs = ngs.*tel;
 @(source)> Computing the objective wavefront transmitance ...
>>  figure, imagesc(ngs.phase)
>>  axis square, colorbar
```



msnippets/s4-m34.mo

The same syntax applies for an asterism:

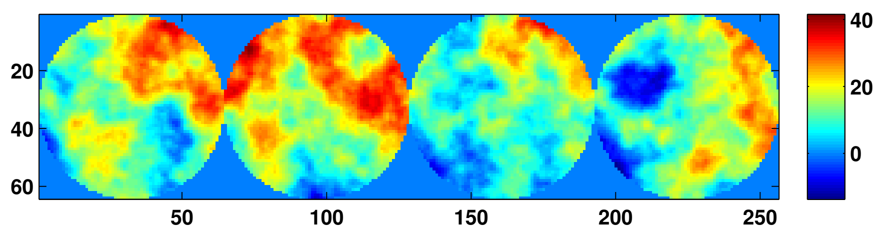
```
>>  ngs = source('asterism',{[0,0],[3,1*constants.arcmin2radian,0]})
 @(source)> 4 created!
___ SOURCE ___
 Obj   zen[arcsec] azim[deg]  height[m]  lambda[micron] magnitude
  1     0.00      0.00       Inf    0.550        0.00
  2    60.00       0.00       Inf    0.550        0.00
  3    60.00     120.00       Inf    0.550        0.00
  4    60.00     240.00       Inf    0.550        0.00
----------------------------------------------------
 @(source)> Terminated!
___ SOURCE ___
 Obj   zen[arcsec] azim[deg]  height[m]  lambda[micron] magnitude
  1     0.00      0.00       Inf    0.550        0.00
  2    60.00       0.00       Inf    0.550        0.00
  3    60.00     120.00       Inf    0.550        0.00
  4    60.00     240.00       Inf    0.550        0.00
----------------------------------------------------
>>  figure, imagesc(tel,ngs)
```



snippets/s4-m35.mo

The footprints of the pupil of the telescope is added on the phase screens when a source array is passed as a second argument of the *imagesc* method of the telescope class.

```
>>  ngs = ngs.*tel;
 @(source)> Computing the objective wavefront transmitance ...
 @(source)> Computing the objective wavefront transmitance ...
 @(source)> Computing the objective wavefront transmitance ...
 @(source)> Computing the objective wavefront transmitance ...
>>  figure, imagesc(ngs.catPhase)
>>  axis equal tight, colorbar
```



msnippets/s4-m36.mo

The *catPhase* property of the source array concatenates the sources *phase* property horizontally.

5 The **deformableMirror** class

In a closed–loop Adaptive Optics Systems, the deformable mirror is the first active component encounter by the wavefront. A numerical deformable mirror is made of a set of influence functions or modes. In OOMAO, a mode shape is derived from two cubic Bézier curves

$$B_1(t) = (1-t^3)P_0 + 3(1-t)^2 tP_1 + 3(1-t)t^2 P2 + t^3 P_3, t \in [0,1] \quad (1)$$
$$B_2(t) = (1-t^3)P_3 + 3(1-t)^2 tP_4 + 3(1-t)t^2 P5 + t^3 P_6, t \in [0,1] \quad (2)$$

$P_k = (x_k, z_k)$ are point in the $x - z$ plane. As t varies from 0 to 1, $B_1(t)$ will go from P_0 to P_3 and $B_2(t)$ will go from P_3 to P_6. P_0 will correspond to the highest point of the mode and is set to the coordinates $(x_0 = 0, z_0 = 1)$. The derivative of the mode at P_0 must be zero, this is ensure by setting $z_1 = 0$. The mode is forced to zero at P_6 by setting $z_6 = 0$. x_6 is set to 2. The derivative of the mode in P_6 is forced to zero by setting $z_5 = 0$. To ensure a smooth junction between both Bézier curves, the following condition is imposed $\vec{P_3P_2} = -\alpha\vec{P_3P_4}$ leading to $P_4 = -P2/\alpha + (1 + 1/\alpha)P_3$.

From the conditions stated above, a deformable mirror mode is set with the following parameters: x_1, (x_2, z_2), (x_3, z_3), α and x_5.

The 1–D half plane mode is obtained by concatenated both Bézier curves

$$B(t) = [B_1(t), B_2(t)] .$$

$B(t)$ is a vector of $x - z$ coordinates, $B(t) = (B_x(t), B_z(t))$. $B_x(t)$ is normalized by $B_x(t_c)$; $B_x(t_c)$ is the x coordinate where $B_z(t) = c$, c is the mechanical coupling of the deformable mirror actuators. The full 1–D mode $M(t)$ is made by concatenating $B(t)$ and its symmetric with respect to the z axis, i.e.

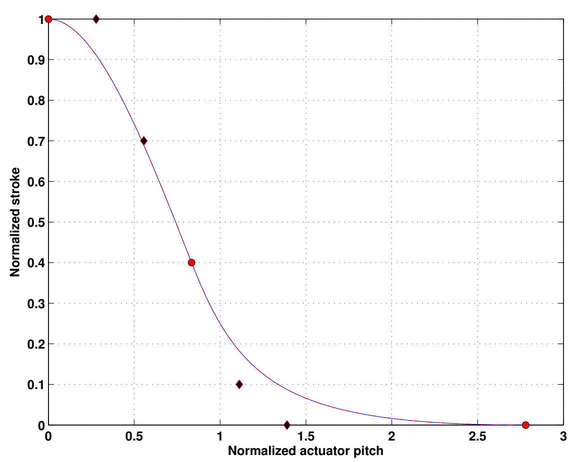
$$M_x(t) = \left([B_{-x}(t), B_x(t)], [B_z(t), B_z(t)] \right).$$

The 2–D mode is the product of the 1-D modes in the $x - z$ plane and in the $y - z$ plane,

$$M(t) = M_x(t) M_y(t).$$

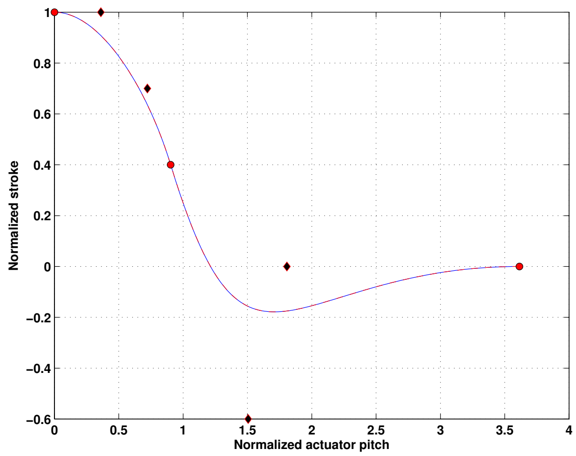
An influence function is created with two arguments passed to its constructor: the list of parameters in a cell and the mechanical coupling value. Instead of the list of parameters, the keywords *'monotonic'* $(0.2, [0.4, 0.7], [0.6, 0.4], 1, 1)$ and *'overshoot'* $(\{0.2, [0.4, 0.7], [0.5, 0.4], 0.3, 1\})$ can be used to call predefined parameter lists, as shown in the next examples:

```
>>  bif = influenceFunction('monotonic',25/100);
 @(bezier influence fun)> Created!
>>  figure, show(bif)
```



msnippets/s5-m37.mo

```
>>  bif = influenceFunction('overshoot',25/100);
 @(bezier influence fun)> Created!
 @(bezier influence fun)> Terminated!
>>  figure, show(bif)
```



msnippets/s5-m38.mo

The markers in the figures correspond to, from left to right, the points P_k from $k = 0$ to 6.

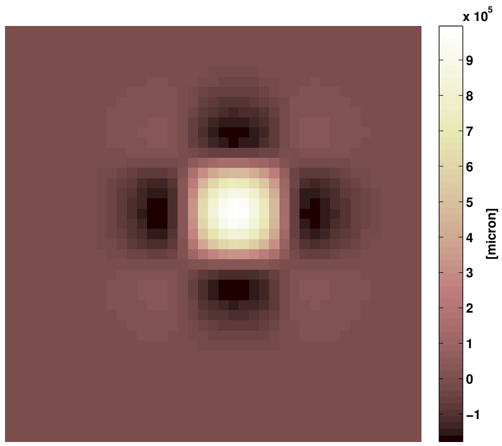
A deformable mirror with 10×10 actuators and the previous influence function sampled with 41×41 pixels is created with

```
dm = deformableMirror(10,...
  'modes',bif,...
  'resolution',41);
```

msnippets/s5-m39.mx

The deformable mirror actuator values are set with the *coefs* property

```
>>  dm.coefs(55) = 1;
>>  figure, imagesc(dm)
```



msnippets/s5-m40.mo

The influence functions of the deformable mirror can be browsed with

```
set(gca,'clim',[-0.2,1])
dm.surfaceListener.Enabled = true;
for k=1:100
   dm.coefs=dm.coefsDefault;
   dm.coefs(k)=1e-6;
   pause(1/10);
end
```

msnippets/s5-m41.mx

When the *Enabled* property of the ***surfaceListener*** is set to true, the deformable mirror display is automatically redrawn when the ***coefs*** property is set to new values.

6 The shackHartmann class

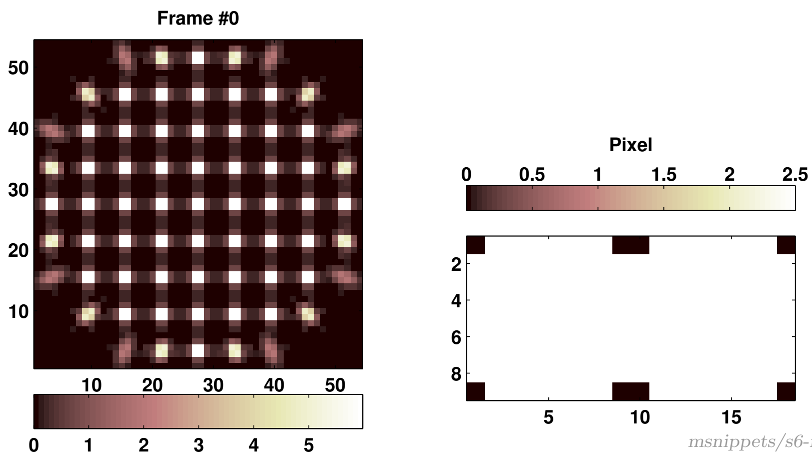
To be complete an Adaptive Optics Systems needs a wavefront sensor. The OOMAO implements the Shack–Hartmann wavefront sensor. A Shack–Hartmann wavefront sensor is made of a lenslet array and a detector. The numerical model follows the physical model by embedding a lenslet array (lensletArray) class and a detector (detector) class in the shackHartmann class. The lensletArray class perform the numerical Fraunhoffer propagation of the wavefront to the detector. The detector class implements a CCD camera detection process including Poisson and read–out noise. The lenslet images are Nyquist sampled per default.

A shackHartmann object with a 9×9 lenslet array and a 54×54 resolution camera is created with

```
>>  wfs = shackHartmann(9,54);
 @(lenslet array)> Created!
Clock rate is: 1.00Hz
 @(detector)> Created!
 @(lensletArray)> Setting the lenslet field stop size!
 @(lensletArray)> Set phasor (shift the intensity of half a pixel
 for even intensity sampling)
___ SHACK-HARTMANN ___
 Shack-Hartmann wavefront sensor:
  . 81 lenslets total on the pupil
  . 6 pixels per lenslet
  . spot algorithm: centroiding, no thresholding!
----------------------------------------------------
___ LENSLET ARRAY ___
 9x9 lenslet array:
  . 2.0 pixels across the diffraction limited spot fwhm
  . 6 pixels across the square lenslet field stop size
  . optical throughput coefficient: 1.0
----------------------------------------------------
___ DETECTOR ___
 54x54 pixels camera
  . quantum efficiency: 1.0
  . photon noise disabled
  . 0.0 photo-events rms read-out noise
  . 1000.0ms exposure time and 1.0Hz frame rate
----------------------------------------------------
 @(shack-hartmann)> Created!
```
msnippets/s6-m42.mo

Usually the lenslet array is located in a conjugated plane of the circular pupil of a telescope. Lets create a 8m diameter telescope and view the light through its pupil

```
tel = telescope(8,'resolution',54);
src = source;
src = src.*tel*wfs;
figure
imagesc(wfs.camera,'parent',subplot(1,2,1))
slopesDisplay(wfs,'parent',subplot(1,2,2))
```



msnippets/s6-m43.mx

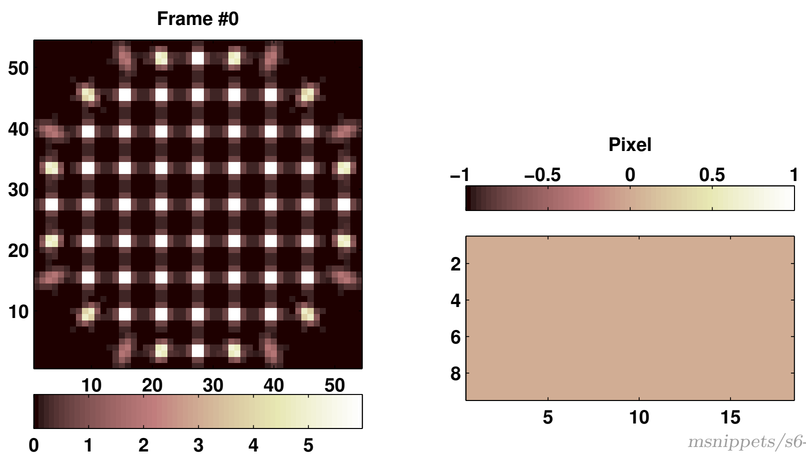
A warning is issued because now the lenslets at the corner of the array are in the dark. This issue is solved by selecting the lenslets that are receiving enough light to be useful to the wavefront detection. The *minLightRatio* property of the lensletArray object set the minimum ratio of light intensity between a partially and fully illuminated lenslet. Setting this property to 60% and asking the wavefront sensor object to select the valid lenslets

```
>>  wfs.lenslets.minLightRatio = 0.6;
>>  wfs.INIT
 @(shack-hartmann)> Setting the valid lenslet and the reference slopes!
@(shackHartmann)> Setting the raster index
```

msnippets/s6-m44.mo

display which lenslets are now selected for the wavefront detection. And now only the slopes corresponding to the valid lenslet are shown on the slopes display

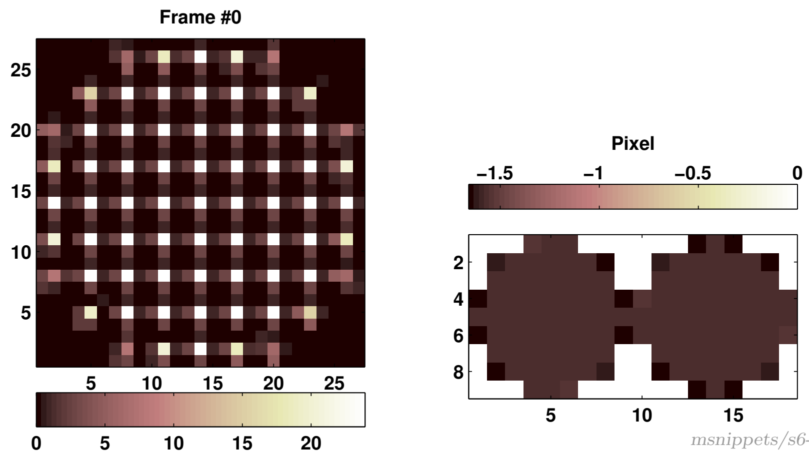
```
>> src = src.*tel*wfs;
>> imagesc(wfs.camera,'parent',subplot(1,2,1))
>> slopesDisplay(wfs,'parent',subplot(1,2,2))
```



msnippets/s6-m45.mo

The sampling of the lenslet images is set by the lensletArray property *nyquistSampling*. The default value is 1 and means that the images are Nyquist sampled. A value of 0.5 means that the images are under sampled by a factor 2 with respect to Nyquist sampling.

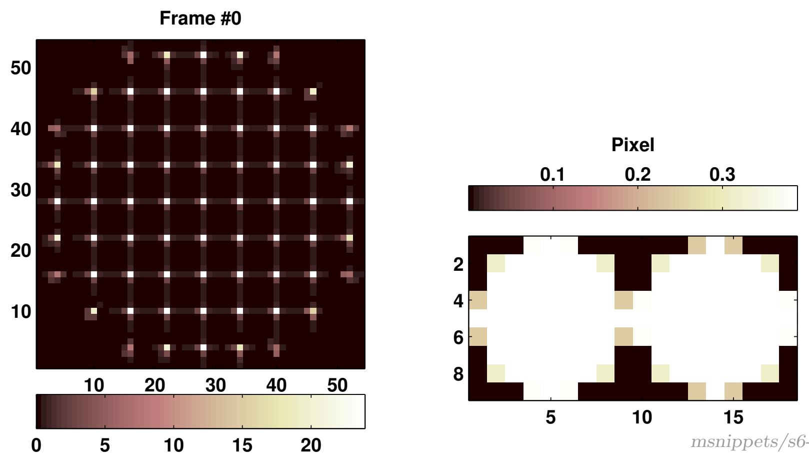
```
>>  wfs.lenslets.nyquistSampling=0.5;
>>  +src;
@(shackHartmann)> Setting the raster index
>>  imagesc(wfs.camera,'parent',subplot(1,2,1))
>>  slopesDisplay(wfs,'parent',subplot(1,2,2))
```



msnippets/s6-m46.mo

After setting the new image sampling value, the source object is re–propagated through the telescope and shackHartmann object using the unary plus (+) operator. Source objects remember their optical path and they can be propagated again simply with the (+) operator. The frame size has been reduced by a factor 2 because the field–of–view of the lenslets has remained the same. The field–of–view is set with the *fieldStopSize* of the lensletArray class. It is expressed in units of the full–width–half–maximum of the diffraction limited images. When the images are Nyquist sampled the full–width–half–maximum is 2 pixels, so *fieldStopSize* was equal to 3. Now with half the sampling, the full–width–half–maximum is 1 pixel and *fieldStopSize* must be 6

```
>>  wfs.lenslets.nyquistSampling=0.5;
>>  wfs.lenslets.fieldStopSize = 6;
>>  +src;
@(shackHartmann)> Setting the raster index
>>  imagesc(wfs.camera,'parent',subplot(1,2,1))
>>  slopesDisplay(wfs,'parent',subplot(1,2,2))
```



msnippets/s6-m47.mo

7 Closed–loop Adaptive Optics

A closed–loop Adaptive Optics Systems is modeled using the classes described in the previous sections. The atmosphere is defined first, it is composed of 3 layers with a r_0 of 15cm and 30m outer scale.

```
>>  atm = atmosphere(photometry.V,0.15,30,...
   'altitude',[0,4,10]*1e3,...
   'fractionnalR0',[0.7,0.25,0.05],...
   'windSpeed',[5,10,20],...
   'windDirection',[0,pi/4,pi]);
 @(atmosphere)> Created!
___ ATMOSPHERE ___
 Von Karman atmospheric turbulence
  . wavelength  = 0.55micron,
  . r0       = 15.00cm,
  . L0       = 30.00m,
  . seeing     = 0.74arcsec,
  . theta0(37%) = 7.01arcsec,
  . tau0(37%)  = 11.01millisec
----------------------------------------------------
 Layer  Altitude[m]  fr0   wind([m/s] [deg])  D[m]   res[px]
  1       0.00    0.70   ( 5.00  0.00)
  2     4000.00    0.25   (10.00  45.00)
  3    10000.00    0.05   (20.00 180.00)
----------------------------------------------------
```

msnippets/s7-m48.mo

The closed–loop system will use one natural guide star on axis

```
>>  ngs = source('wavelength',photometry.R);
 @(source)> Created!
___ SOURCE ___
 Obj  zen[arcsec] azim[deg]  height[m]  lambda[micron] magnitude
  1    0.00      0.00       Inf    0.640        0.00
----------------------------------------------------
 @(source)> Terminated!
```

msnippets/s7-m49.mo

The wavefront sensing will be done at 700nm. The wavefront sensor will use a 9^2 lenslet array with a 54^2 pixels camera. Lenslets will less than 75% of the light of a fully illuminated lenslet with a flat wavefront are discarded.

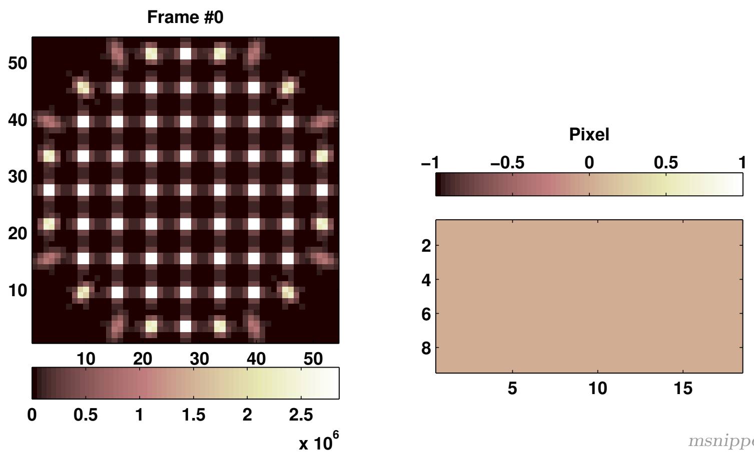
```
>>  wfs = shackHartmann(9,54,0.75);
 @(lenslet array)> Created!
Clock rate is: 1.00Hz
 @(detector)> Created!
 @(lensletArray)> Setting the lenslet field stop size!
 @(lensletArray)> Set phasor (shift the intensity of half a pixel
 for even intensity sampling)
___ SHACK-HARTMANN ___
 Shack-Hartmann wavefront sensor:
  . 81 lenslets total on the pupil
  . 6 pixels per lenslet
  . spot algorithm: centroiding, no thresholding!
----------------------------------------------------
___ LENSLET ARRAY ___
 9x9 lenslet array:
  . 2.0 pixels across the diffraction limited spot fwhm
  . 6 pixels across the square lenslet field stop size
  . optical throughput coefficient: 1.0
----------------------------------------------------
___ DETECTOR ___
 54x54 pixels camera
  . quantum efficiency: 1.0
  . photon noise disabled
  . 0.0 photo-events rms read-out noise
  . 1000.0ms exposure time and 1.0Hz frame rate
----------------------------------------------------
 @(shack-hartmann)> Created!
```
msnippets/s7-m50.mo

The telescope is an 8m diameter with a 2.5 arc–minute field–of–view. The phase screens temporal evolution is sampled at 500Hz.

```
>>  tel = telescope(8,'resolution',54,...
   'fieldOfViewInArcMin',2.5,...
   'samplingTime',1/500);
 @(telescope)> Created!
___ TELESCOPE ___
 8.00m diameter full aperture with 50.27m^2 of light collecting area;
 the field-of-view is 2.50arcmin; the pupil is sampled with 54X54 pixels
----------------------------------------------------
```
msnippets/s7-m51.mo

The reference wavefront is propagated to set wavefront sensor valid lenslets and reference slopes

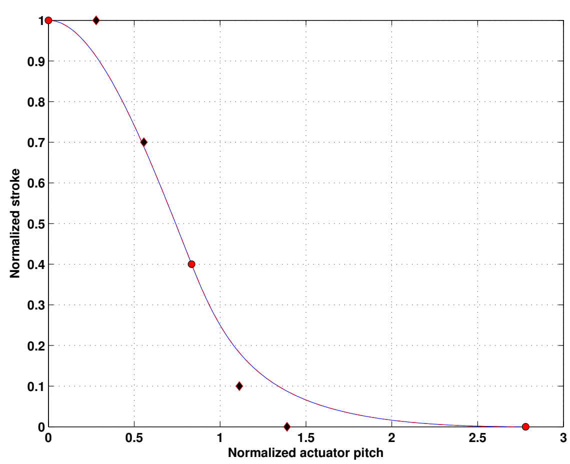
```
ngs = ngs.*tel*wfs;
wfs.INIT;
figure
imagesc(wfs.camera,'parent',subplot(1,2,1))
slopesDisplay(wfs,'parent',subplot(1,2,2))
```



msnippets/s7-m52.mx

The deformable mirror will use 10^2 monotonic influence functions mechanically couple at 25%

```
>>  bif = influenceFunction('monotonic',25/100);
 @(bezier influence fun)> Created!
>>  figure, show(bif)
```



msnippets/s7-m53.mo

The influence function resolution is the same than the wavefront sensor camera. The actuators are located at the corners of the lenslets according to the Fried geometry. Based on the location of the valid lenslets, the locations of the valid actuators in the 10×10 grid is derived from the wavefront sensor object.

```
dm = deformableMirror(10,...
  'modes',bif,...
  'resolution',wfs.camera.resolution(1),...
   'validActuator',wfs.validActuator);
```

msnippets/s7-m54.mx

The influence function are normalized to unity. The actuator stroke set by the *coefs* property is given in meter. The interaction matrix between the wavefront sensor and the deformable mirror is derived by setting first the stroke of the actuators

```
>>  dm.coefs = eye(dm.nValidActuator)*ngs.wavelength/2;
```

msnippets/s7-m55.mo

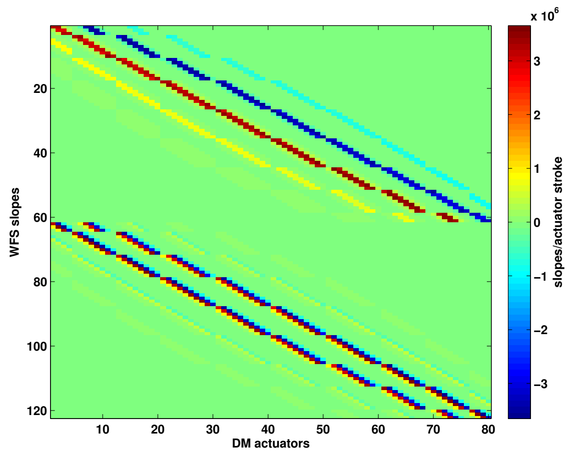
Setting *coefs* to a matrix means that a series a command is going to be sent to the DM, one per column. The stroke is set to half the guide star wavelength. The guide star is propagated through the deformable mirror to the wavefront sensor.

```
>>  ngs = ngs.*tel*dm*wfs;
@(shackHartmann)> Setting the raster index
```

msnippets/s7-m56.mo

The wavefront sensor is now storing the slopes corresponding to the 80 actuators moved and the interaction matrix is simply

```
>> interaction = 2*wfs.slopes./ngs.wavelength;
>> figure
>> imagesc(interaction)
>> xlabel('DM actuators')
>> ylabel('WFS slopes')
>> ylabel(colorbar,'slopes/actuator stroke')
```



msnippets/s7-m57.mo

The command matrix is the pseudo–inverse of the interaction matrix with a 10% threshold on the normalized eigen values of the interaction matrix

```
>> s      = svd(interaction);
>> tol    = s(sum(s/s(1)>0.1));
>> command = pinv(interaction,tol);
```

msnippets/s7-m58.mo

Now the atmosphere and the telescope are combined

```
tel = tel + atm;
```

msnippets/s7-m59.mx

and the loop is closed for 1000 time steps (i.e 2s) with a integrator gain at 0.2

```
nIteration = 1000;
total  = zeros(1,nIteration);
residue = zeros(1,nIteration);
dm.coefs = 0;
loopGain = 0.2;
tic
for kIteration=1:nIteration
   % Propagation throught the atmosphere to the telescope
   ngs=ngs.*+tel;
   % Variance of the atmospheric wavefront
   total(kIteration) = var(ngs);
   % Propagation to the WFS
   ngs=ngs*dm*wfs;
   % Variance of the residual wavefront
   residue(kIteration) = var(ngs);
   % Computing the DM residual coefficients
   residualDmCoefs = command*wfs.slopes;
   % Integrating the DM coefficients
   dm.coefs = dm.coefs - loopGain*residualDmCoefs;
   % Display of turbulence and residual phase
end
toc
```
msnippets/s7-m60.mx

The root–mean–square of the turbulent wavefront (full), the residual wavefront (residue) and the theoretical root–mean–square of the piston–removed wavefront is displayed

```
u = (0:nIteration-1).*tel.samplingTime;
atm.wavelength = ngs.wavelength;
% Piston removed phase variance
totalTheory = phaseStats.zernikeResidualVariance(1,atm,tel);
atm.wavelength = photometry.V;
% Phase variance to micron rms converter
rmsMicron = @(x) 1e6*sqrt(x).*ngs.wavelength/2/pi;
figure(12)
plot(u,rmsMicron(total),u([1,end]),rmsMicron(totalTheory)*ones(1,2),u,rmsMicron(residue))
grid
legend('Full','Full (theory)','Residue',0)
xlabel('Time [s]')
ylabel('Wavefront rms [micron]')
```
msnippets/s7-m61.mo