# Object–Oriented Matlab Adaptive Optics Toolbox

R. Conan[a] and C. Correia[b]

[a]RSAA, The Australian National University, Weston Creek, ACT 2611, Australia [b] Centre for Astrophysics, University of Porto, Rua das Estrelas 4150-762 Porto, Portugal

## ABSTRACT

Object–Oriented Matlab Adaptive Optics (OOMAO) is a Matlab toolbox dedicated to Adaptive Optics (AO) systems. OOMAO is based on a small set of classes representing the source, atmosphere, telescope, wavefront sensor, Deformable Mirror (DM) and an imager of an AO system. This simple set of classes allows simulating Natural Guide Star (NGS) and Laser Guide Star (LGS) Single Conjugate AO (SCAO) and tomography AO systems on telescopes up to the size of the Extremely Large Telescopes (ELT). The discrete phase screens that make the atmosphere model can be of infinite size, useful for modeling system performance on large time scales. OOMAO comes with its own parametric influence function model to emulate different types of DMs. The cone effect, altitude thickness and intensity profile of LGSs are also reproduced. Both modal and zonal modeling approach are implemented. OOMAO has also an extensive library of theoretical expressions to evaluate the statistical properties of turbulence wavefronts. The main design characteristics of the OOMAO toolbox are object–oriented modularity, vectorized code and transparent parallel computing. OOMAO has been used to simulate and to design the Multi–Object AO prototype Raven at the Subaru telescope and the Laser Tomography AO system of the Giant Magellan Telescope. In this paper, a Laser Tomography AO system on an ELT is simulated with OOMAO. In the first part, we set–up the class parameters and we link the instantiated objects to create the source optical path. Then we build the tomographic reconstructor and write the script for the pseudo-open-loop controller.

**Keywords:** adaptive optics, wavefront sensing, laser guide star, numerical modeling

## 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 object properties.

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. An 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. 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 afterwards anyway. A property is read or set through object_name.property_name.

The main classes used during a simulation are

- source,
- atmosphere,
- telescope,
- shackHartmann,

---

Further author information: (Send correspondence to R. Conan)
R. Conan: E-mail: rod.conan@anu.edu.au

Adaptive Optics Systems IV, edited by Enrico Marchetti, Laird M. Close,
Jean-Pierre Véran, Proc. of SPIE Vol. 9148, 91486C ·
CCC code: 0277-786X/14/$18 · doi: 10.1117/12.2054470

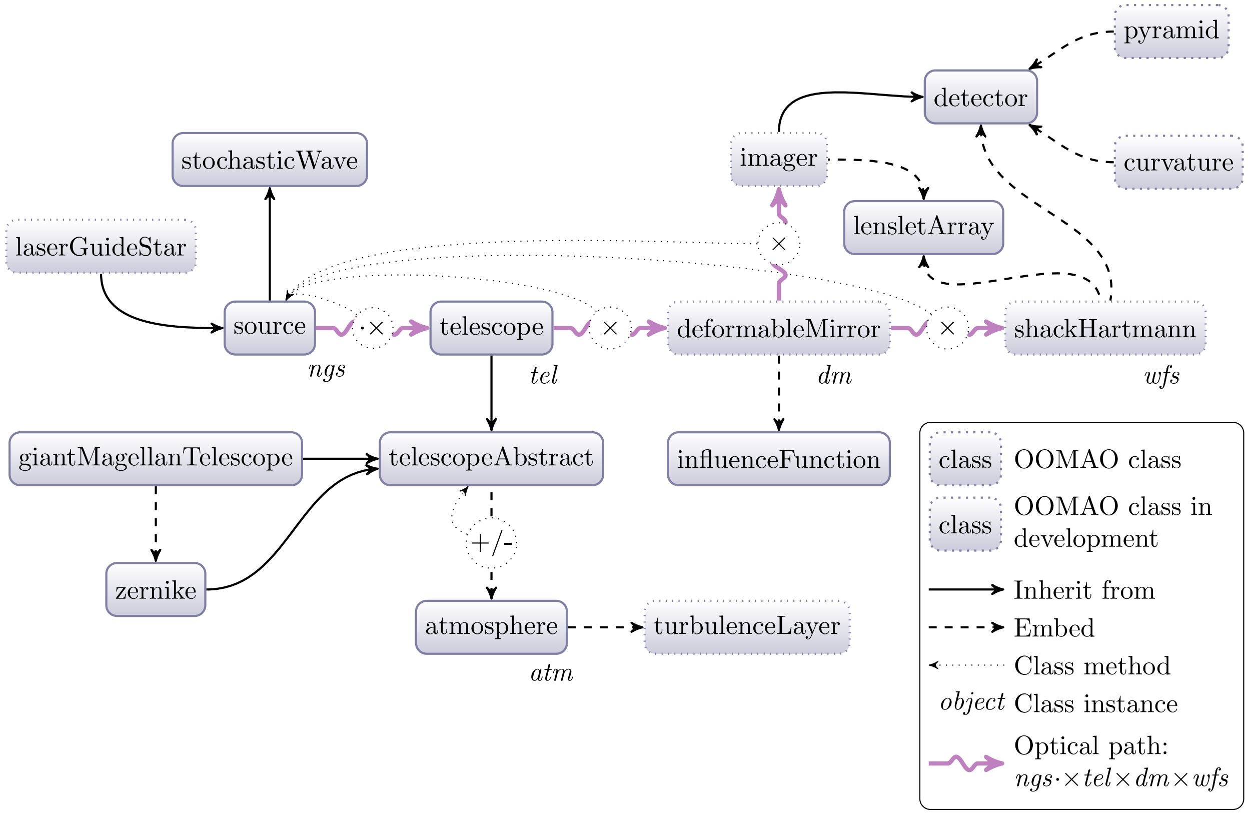


Figure 1. OOMAO class diagram.

- deformableMirror.

Fig. 1 shows the relations between the main classes of OOMAO. The classes are setup along an optical path for a closed–loop AO system. The operators used to optically link the classes are superimposed to the optical link. Classes are documented following *Matlab*® documentation standard and can be extracted with the *doc* command. The classes description generated with *doc* is the library reference documentation.

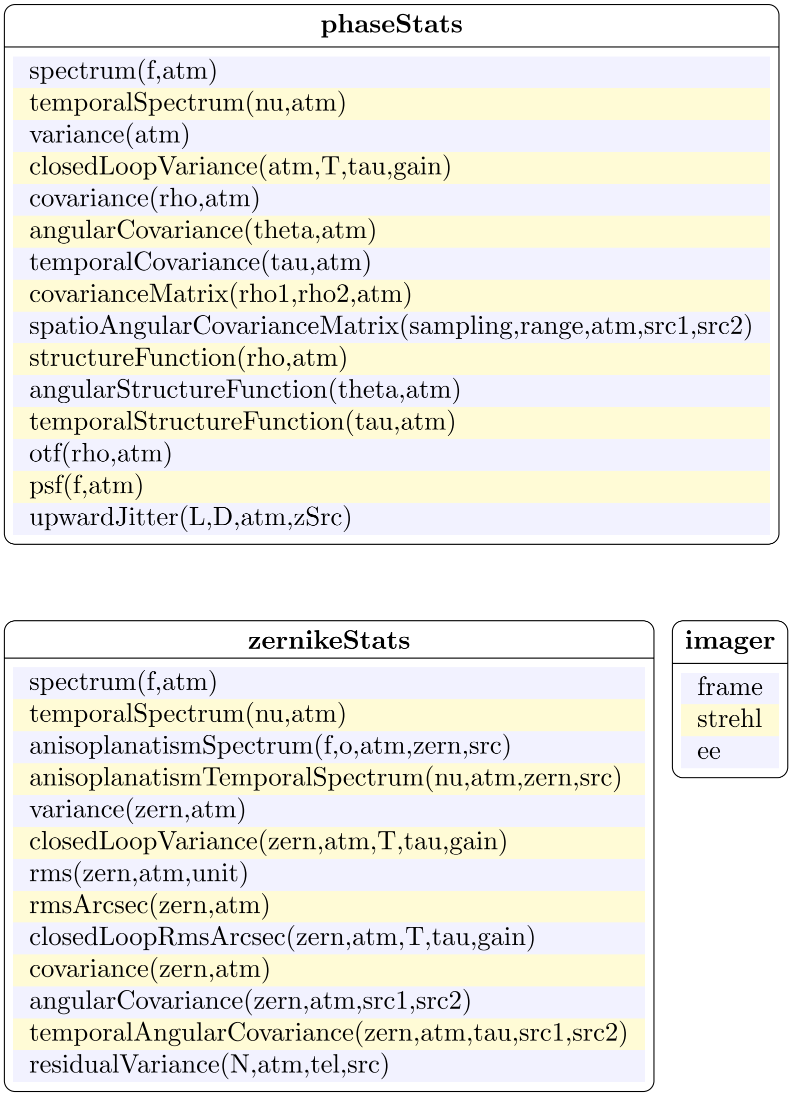
In addition to the classes to model AO systems, OOMAO has a rich library of functions to compute various statistical properties of the wavefront. These functions are contained within the phaseStats and zernikeStats classes and the list of their methods are given in Fig. 2. The figure of merit of an AO system is often the image of a point–like star after correction. The PSF is collected with the imager class (Fig. 1) which properties `frame`, `strehl` and `ee` correspond to the star image and the Strehl ratio and the ensquared energy of the point–spread–function, respectively. OOMAO relies on sparse matrices where possible, in order to minimize the computer memory load, and uses *Matlab*® transparent parallel loop for the heftiest computations. Parallel for loops require the *Parallel Computing Toolbox*®.

OOMAO is used by several projects. For example, the control software of RAVEN[1] is based on OOMAO and OOMAO was used during the conceptual and preliminary design phases of the GMT LTAO system[2] to compare different design options.

## 2. SETTING-UP AN OOMAO SIMULATION

In this section, we describe the different steps to model an AO system. OOMAO does not have configuration files like many other AO simulation software.[3,4] An OOMAO simulation is build in the same way an AO optical bench is build. The main steps in the construction and deployment of an AO system are

1. selection of the components



**phaseStats**

spectrum(f,atm)
temporalSpectrum(nu,atm)
variance(atm)
closedLoopVariance(atm,T,tau,gain)
covariance(rho,atm)
angularCovariance(theta,atm)
temporalCovariance(tau,atm)
covarianceMatrix(rho1,rho2,atm)
spatioAngularCovarianceMatrix(sampling,range,atm,src1,src2)
structureFunction(rho,atm)
angularStructureFunction(theta,atm)
temporalStructureFunction(tau,atm)
otf(rho,atm)
psf(f,atm)
upwardJitter(L,D,atm,zSrc)

**zernikeStats**

spectrum(f,atm)
temporalSpectrum(nu,atm)
anisoplanatismSpectrum(f,o,atm,zern,src)
anisoplanatismTemporalSpectrum(nu,atm,zern,src)
variance(zern,atm)
closedLoopVariance(zern,atm,T,tau,gain)
rms(zern,atm,unit)
rmsArcsec(zern,atm)
closedLoopRmsArcsec(zern,atm,T,tau,gain)
covariance(zern,atm)
angularCovariance(zern,atm,src1,src2)
temporalAngularCovariance(zern,atm,tau,src1,src2)
residualVariance(N,atm,tel,src)

**imager**

frame
strehl
ee

Figure 2. OOMAO classes for performance estimation.

2. initialization of the components
3. calibration of the components
4. link the components with a control system

## 2.1 Component selection

### 2.1.1 The source

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 Adaptive Optics can be simulated. For a NGS, the source height is infinite. For a LGS, the source height is finite and can be set to a vector of heights depending on the thickness of the source. A simple on–axis natural guide star object is created by calling the source constructor without parameters:

```
ngs =source;
```

The first time an 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 azimuth angle are both set to zero, the star height is infinite, the wavelength is 550nm and the magnitude is set to 0.

### 2.1.2 The atmosphere

Next, we create the atmosphere the source will propagate through. The atmosphere class contains all the parameters defining the atmosphere including the turbulence profile. A 3–layer atmosphere with a 20cm Fried parameter in the visible and a 30m outer scale is created with:

```
atm = atmosphere(photometry.V,20e-2,30,...
    'fractionnalR0',[0.5,0.3,0.2],'altitude',[0e3,5e3,12e3],...
    'windSpeed',[10,5,20],'windDirection',[0,pi/2,pi]);
```

The photometry class contains the definition of the photometric systems. If an altered photometric system is needed, modification must be done into the class itself.

### 2.1.3 The telescope

Lest first make the NGSAO model scalable by defining the number of lenslet `nL` of the wavefront sensor, the number of pixel per lenslet `nPx`, the telescope diameter `D` in meter and the sampling frequency `samplingFreq` in Hz.

```
nL   = 60;
nPx  = 10;
nRes = nL*nPx;
D    = 25;
d    = D/nL; % lenslet pitch
samplingFreq = 500;
```

Now, we create a telescope of diameter `D` with a pupil sampled with `nRes` pixel. The telescope field–of–view is 30arcsec and the temporal evolution of wavefront in the telescope pupil will be sampled at `samplingFreq`.

```
tel = telescope(D,'resolution',nRes,...
    'fieldOfViewInArcsec',30,'samplingTime',1/samplingFreq);
```

### 2.1.4 The wavefront sensor

To be complete an Adaptive Optics Systems needs a wavefront sensor (WFS). OOMAO uses the Shack–Hartmann wavefront sensor, the pyramid and the curvature wavefront sensors are currently developed. 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 `nL` × `nL` lenslet array and a `nRes` × `nRes` resolution camera is created. The *`minLightRatio`* property of the lensletArray object set the minimum ratio of light intensity between a partially and fully illuminated lenslet. In the following, *`minLightRatio`* is set to 85%:

```
wfs = shackHartmann(nL,nRes,0.85);
```

**Wavefront sensor initialization** The next component to define is a deformable mirror (DM). The WFS and the DM are optically conjugated to the telescope pupil and the DM actuators are aligned to the WFS lenslet array following the Fried geometry i.e. they are located at the lenslet corners. To fully define the DM, one needs to initialized the WFS first by computing the ”valid lenslet” which are the lenslet receiving sufficient light according to the value of *`minLightRatio`*. The propagation of the wavefront is performed with the overloaded *Matlab*® operators *times* or *.** and *mtimes* or ***.

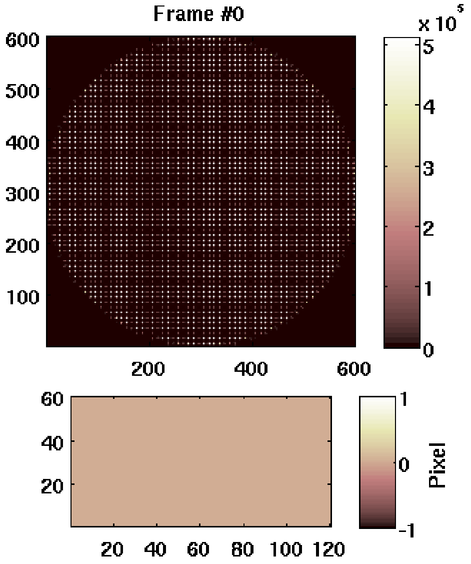
```
ngs = ngs.*tel*wfs;
```

The ngs is propagated through the telescope, the WFS lenslet array and imagelets are formed on the WFS camera. 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.

A warning is issued because 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.

```
wfs.INIT
```



When the WFS detector is read again and a new processing of the frame is done, there is no more warning as only the ”valid lenslet” are now processed; a new read–out and centroiding computation is force by calling the overloaded *uplus* or simply + operator:

```
+wfs;
figure
imagesc(wfs.camera,'parent',subplot(3,2,[1,4]))
slopesDisplay(wfs,'parent',subplot(3,2,[5,6]))
```

The next 2 commands allow the displays of the frame and of the slopes to be updated when a new frame and new slopes are computed

```
wfs.camera.frameListener.Enabled = true;
wfs.slopesListener.Enabled = true;
```

The WFS must be calibrated such as for 1rd of tip–tilt wavefront , it will measure a slope of 1rd. To do so, the *`pointingDirection`* property is set on-axis

```
wfs.pointingDirection = zeros(2,1);
```

whereas the source is progressively moved off-axis

```
pixelScale = ngs.wavelength/...
    (2*d*wfs.lenslets.nyquistSampling);
tipStep = pixelScale/2;
nStep   = floor(nPx/3)*2;
sx      = zeros(1,nStep+1);
u       = 0:nStep;
wfs.camera.frameListener.Enabled = false;
wfs.slopesListener.Enabled = false;
warning('off','oomao:shackHartmann:relay')
for kStep=u
    ngs.zenith = -tipStep*kStep;
    +ngs;
    drawnow
    sx(kStep+1) = median(wfs.slopes(1:end/2));
end
warning('on','oomao:shackHartmann:relay')
Ox_in  = u*tipStep*constants.radian2arcsec;
Ox_out = sx*ngs.wavelength/d/2*constants.radian2arcsec;
%figure
%plot(Ox_in,Ox_out)
%grid
slopesLinCoef = polyfit(Ox_in,Ox_out,1);
wfs.slopesUnits = 1/slopesLinCoef(1);
```

The source is reset on–axis and the WFS is set to always be aligned to the source by setting *`pointingDirection`* to empty.

```
ngs.zenith = 0;
wfs.pointingDirection = [];
```

### 2.1.5 The deformable mirror

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

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

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

$P_k = (x_k, z_k)$ 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 ensured 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 $P_3\vec{P}_2 = -\alpha\vec{P}_3P_4$ 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: $x_1$, $(x_2, z_2)$, $(x_3, z_3)$, $\alpha$ 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 is symmetric with respect to the $z$ axis, i.e.

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

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 $P_k$ coordinates 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:

```
bifa = influenceFunction(’monotonic’,0.75);
figure,show(bifa,’parent’,subplot(1,2,1))
title(’Monototic influence function’)
bifb = influenceFunction(’overshoot’,0.75);
show(bifb,’parent’,subplot(1,2,2))
title(’Overshooted influence function’)
```

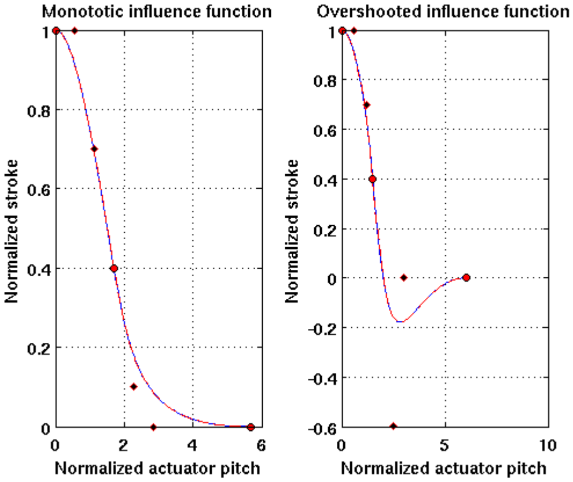
The markers in the figures correspond to, from left to right, the points $P_k$ from $k = 0$ to 6.

In addition to the synthetic influence functions, the measured influence functions of a real deformable mirror can be used in OOMAO by setting the property *`modes`* in a deformableMirror object.

A deformable mirror with $(\mathtt{nL}+1) \times (\mathtt{nL}+1)$ actuators and the previously defined influence function sampled at the same resolution than the telescope is created with

```
dm = deformableMirror(nL+1,’modes’,bifa,...
    ’resolution’,tel.resolution,...
    ’validActuator’,wfs.validActuator);
```

The *`validActuator`* property of the WFS class, derived from the *`validLenslet`* property, is used to define the DM actuators corresponding to the ”valid lenslet”.



**Interaction matrix** Once the WFS and the DM components have been initialized, the calibration matrix can be computed.

Let’s switch off the display automatic update:

```
wfs.camera.frameListener.Enabled = false;
wfs.slopesListener.Enabled = false;
```

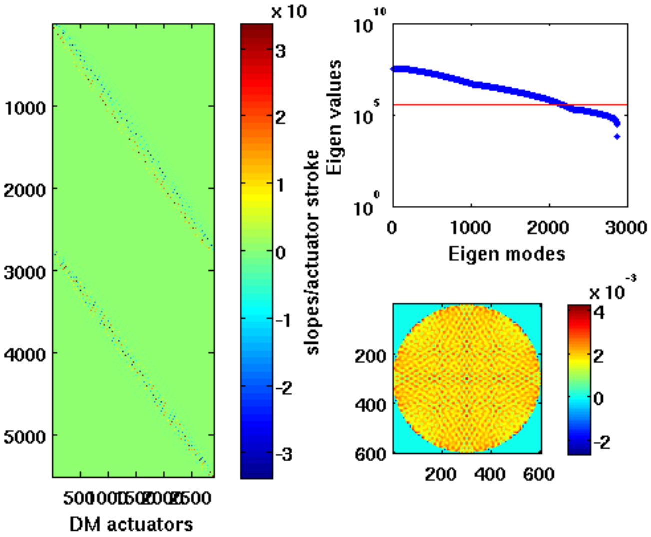
and setup the optical path before the DM/WFS subsystem:

```
ngs = ngs.*tel;
```

The interaction matrix is generated by pushing all the valid actuators sequentially and saving the corresponding WFS measurements in a matrix. The process is done with the *calibration* method of the deformableMirror class. The arguments of the method are the deformable mirror, the wavefront sensor, the calibration source and the actuator amplitude in meter. In OOMAO, the generation of the interaction matrix is vectorized meaning it can be done in 1 step at the expense of requiring a lot of memory. Here the process is divided in as many steps as actuators across the pupil.

```
calibDm = calibration(dm,wfs,ngs,ngs.wavelength,nL+1,'cond',1e2);
```

At the end of the calibration process, the interaction matrix is saved into the object *calibDm* of the class calibrationVault. The interaction matrix is saved into the property *`D`*. The singular value decomposition of *`D`* is immediately computed and the pseudo–inverse of *`D`* is saved into the property *`M`*. A display window shows on the left the interaction matrix *`D`* on the top right the eigen values of *`D`* and in the bottom right the eigen modes corresponding to the eigen values given by the red line in the plot above, which here is essentially the piston mode. The truncated pseudo–inverse of *`D`* is obtained by eliminating the lowest eigen values. The filtering of the lowest eigen values is obtained by changing the value of one of these properties: *`threshold`* setting the minimum eigen values for the computing of *`M`*, *`nThresholded`* setting the number of discarded eigen value starting from the lowest one or *`cond`* forcing the condition number of *`D`*.



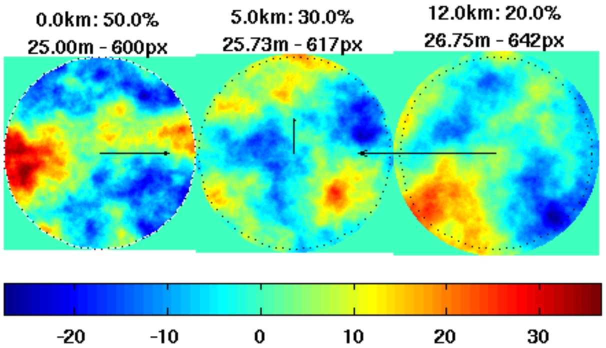
## 3. WAVEFRONT ESTIMATION

In OOMAO, a wavefront can be described in a modal or zonal basis. In this basis, the wavefront can be reconstructed using either a Least-Square reconstructor (LSR) or a Minimum–Mean–Square–Error reconstructor (MMSE). In the following, several wavefront reconstruction methods are compared.
First lets bound the atmosphere to the telescope, without binding the atmosphere object to a telescope object, phase screens cannot be generated. Then the phase screens are displayed and the *ngs* optical path is set up. A source object saves its optical path into the *`opticalPath`* property allowing the source to be propagated again with a simple call to the *uplus* or + operator.

```
tel = tel + atm;
figure
imagesc(tel)
ngs = ngs.*tel*wfs;
```

The wavefront reconstruction is done by estimating the wavefront values at the corner of the lenslets. A new telescope is defined identical to the previous one but with a lower resolution and a pupil defined by the map of "valid actuator".

```
telLowRes = telescope(tel.D,...
    'resolution',nL+1,...
    'fieldOfViewInArcsec',30,...
    'samplingTime',1/500);
telLowRes.pupil = wfs.validActuator;
```



The atmosphere is now bound to the new telescope, the *ngs* is propagated through the atmosphere to the new telescope and the piston free wavefront is saved.

```
telLowRes= telLowRes + atm;
ngs = ngs.*telLowRes;
phase = ngs.meanRmOpd;
```

The first method to estimate the wavefront is the finite difference (FD) method using the inverse of the sparse gradient matrix computed with the method *sparseGradientMatrix*.

```
phaseEst = tools.meanSub( wfs.finiteDifferenceWavefront*ngs.wavelength ,...
    wfs.validActuator);
```

The source is propagated through the finite difference phase screen and the residual wavefront and residual wavefront rms are retrieved from the *ngs* *meanRmOpd* and *opdRms* properties.

```
ngs = ngs.*telLowRes*{wfs.validActuator,-2*pi*wfs.finiteDifferenceWavefront};
phaseEstRes = ngs.meanRmOpd;
phaseEstResRms = ngs.opdRms;
```

The second method uses the interaction matrix to compute DM actuators amplitude in the property *coefs*. As the telescope earlier, a lower resolution DM is created,

```
bifaLowRes = influenceFunction('monotonic',0.75);
dmLowRes = deformableMirror(nL+1,'modes',bifaLowRes,'resolution',nL+1,...
    'validActuator',wfs.validActuator);
```

The *coefs* are derived from the product of the pseudo-inverse of the calibration matrix *D* by the wavefront sensor *slopes*.

```
dmLowRes.coefs = -calibDm.M*wfs.slopes;
```

The DM deformation is read from the *surface* property. The factor 2 takes into account the reflection off the mirror. The source is propagated to the deformable mirror and as previously the residual wavefront and residual wavefront rms are retrieved from the *ngs* *meanRmOpd* and *opdRms* properties.

```
dmSurface = tools.meanSub( dmLowRes.surface.*wfs.validActuator*2 , wfs.validActuator);
ngs = ngs.*telLowRes*dmLowRes;
phaseDmRes = ngs.meanRmOpd;
phaseDmResRms = ngs.opdRms;
```

The last method is the Linear Minimum Mean Square Error (LMMSE) wavefront reconstructor where the wavefront $\varphi$ is derived from the WFS slopes $s$ from

$$\varphi = C_{\varphi s} C_{ss}^{-1} s \tag{3}$$

where $C_{ss}$ is the slope covariance matrix and $C_{\varphi s}$ is the cross–covariance between the wavefront and the slopes. The covariance matrices are computed by the slopesLinearMMSE class.

```
slmmse = slopesLinearMMSE(wfs,tel,atm,ngs,'mmseStar',ngs);
```

The guide star is propagated through the atmosphere and the telescope to the wavefront sensor. The LMMSE wavefront estimate is obtained by multiplying the slopesLinearMMSE object and the shackHartmann object. The wavefront estimate *ps_e* is sampled at the DM actuator locations. The piston is removed from *ps_e* using the *meanSub* method.

```
ngs = ngs.*tel*wfs;
ps_e = slmmse*wfs;
ps_e = tools.meanSub( ps_e , wfs.validActuator);
```

*ps_e* is removed from the NGS wavefront to obtain the zero–piston residual wavefront as well as the residual wavefront error rms.

```
ngs = ngs.*telLowRes*{wfs.validActuator,-ps_e*ngs.waveNumber};
ps_eRes = ngs.meanRmOpd;
ps_eResRms = ngs.opdRms*1e9;
```

A Laser guide star (LGS) and the corresponding slopesLinearMMSE object are created.

```
lgs = source('height',90e3);
lgs_slmmse = slopesLinearMMSE(wfs,tel,atm,lgs,'mmseStar',ngs);
```

The LGS is propagated through the atmosphere and the telescope to the wavefront sensor. The LGS LMMSE wavefront estimate is obtained by multiplying the slopesLinearMMSE object and the shackHartmann object. *lgs_ps_e* is removed from the NGS wavefront to obtain the zero–piston residual wavefront as well as the residual wavefront error rms.

```
lgs = lgs.*tel*wfs;
lgs_ps_e = tools.meanSub( lgs_slmmse*wfs , wfs.validActuator );
ngs = ngs.*telLowRes*{wfs.validActuator,-lgs_ps_e*ngs.waveNumber};
lgs_ps_eRes = ngs.meanRmOpd;
lgs_ps_eResRms = ngs.opdRms*1e9;
```

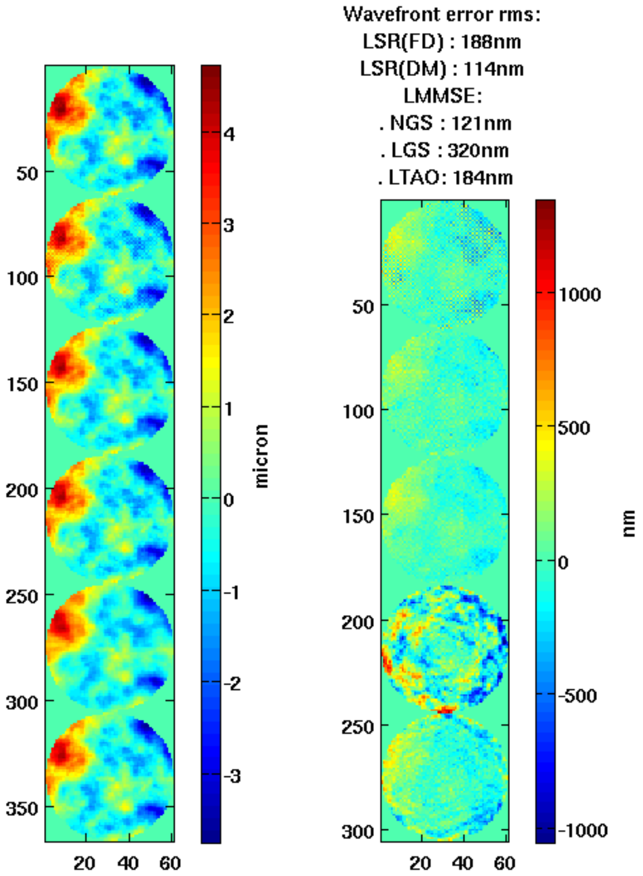
A 3 LGS asterism evenly located on a 20arcsec diameter ring and the corresponding slopesLinearMMSE object are created.

```
lgsAst = source('asterism',{[3,arcsec(10),0]},'height',90e3);
% figure, imagesc(tel, [ngs,lgsAst])
lgsAst_slmmse = slopesLinearMMSE(wfs,tel,atm,lgsAst,'mmseStar',ngs,'NF',1024);
```

The LGSs are propagated through the atmosphere and the telescope to the wavefront sensor. The LMMSE wavefront estimate is obtained by multiplying the slopesLinearMMSE object and the shackHartmann object. *lgsAst_ps_e* is removed from the NGS wavefront to obtain the zero–piston residual wavefront as well as the residual wavefront error rms.

```
lgsAst = lgsAst.*tel*wfs;
lgsAst_ps_e = tools.meanSub( lgsAst_slmmse*wfs , wfs.validActuator );
ngs = ngs.*telLowRes*{wfs.validActuator,-lgsAst_ps_e*ngs.waveNumber};
lgsAst_ps_eRes = ngs.meanRmOpd;
lgsAst_ps_eResRms = ngs.opdRms*1e9;
```

The NGS wavefront is plotted at the top of the left column and the estimates below. The right columns display the corresponding residual wavefront.



```
figure(314)
subplot(6,2,[1,11])
imagesc(1e6*[ phase; phaseEst; dmSurface; ps_e;lgs_ps_e;lgsAst_ps_e] )
axis equal tight
ylabel(colorbar,'micron')
subplot(6,2,[4,12])
imagesc(1e9*[ phaseEstRes; phaseDmRes; ps_eRes;lgs_ps_eRes;lgsAst_ps_eRes] )
axis equal tight
title({sprintf('Wavefront error rms:'),...
    sprintf('LSR(FD) : %3.0fnm',phaseEstResRms*1e9),...
    sprintf('LSR(DM) : %3.0fnm',phaseDmResRms*1e9),...
    sprintf('LMMSE:'),...
    sprintf(' . NGS : %3.0fnm',ps_eResRms),...
    sprintf(' . LGS : %3.0fnm',lgs_ps_eResRms),...
    sprintf(' . LTAO: %3.0fnm',lgsAst_ps_eResRms)},...
    'HorizontalAlignment','Center')
ylabel(colorbar,'nm')
pos = get(314,'pos');
set(314,'pos',[pos(1:3) pos(4)*5/3])
drawnow
```

## 4. CLOSED–LOOP NGS AO SYSTEMS

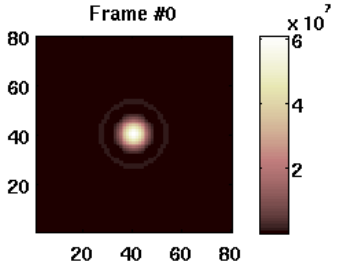
OOMAO can simulate AO systems in closed–loop, open–loop and in pseudo–open loop. A LQG controller[5] for OOMAO is currently in development. In this section, a closed–loop NGS AO is modeled. Lets first create an on–axis science celestial object in J band. Then, the science imaging camera is created with the class imager.

```
science = source('wavelength',photometry.J);
cam = imager(tel);
```

The atmosphere object is detached from the telescope and the science star is propagated through the telescope to the science camera producing a perfect diffraction limited image. The camera display can also be set to update itself when a new camera frame is generated.

```
tel = tel - atm;
science = science.*tel*cam;
figure(31416)
imagesc(cam,'parent',subplot(2,1,1))
%cam.frameListener.Enabled = true;
```

The diffraction limited image is set as the reference frame allowing computing the image Strehl ratio on the subsequent frame captures.



```
cam.referenceFrame = cam.frame;
+science;
fprintf('Strehl ratio: %4.1f\n',cam.strehl)
```

The atmosphere is re–attached to the telescope, the science star is propagated again, this time first through the atmosphere and the atmosphere limited Strehl is obtained.

```
tel = tel + atm;
+science;
fprintf('Strehl ratio: %4.1f\n',cam.strehl)
```

Two types of controller are compared: a closed–loop LSR and a pseudo–open loop LMMSE. The two controllers can be tested side–by–side by using a unique feature of OOMAO: the fact that some classes like shackHartmann or deformableMirror can duplicate some of their properties when they are faced with multiple sources. This means that if one needs to analyze the wavefront of several sources with the same Shack–Hartmann wavefront sensor, a single shackHartmann object is only needed.

Two NGSs and two science objects are created and their optical paths set. Both are going through the atmosphere, the telescope and the DM, but the NGSs end up on the WFS and the science stars are both imaged with the science detector. The WFS display now shows the imagelets and slopes corresponding to the wavefronts of both NGSs.

```
ngsCombo = source('zenith',zeros(1,2),'azimuth',zeros(1,2),'magnitude',8);
ngsCombo = ngsCombo.*tel*dm*wfs;
scienceCombo = source('zenith',zeros(1,2),'azimuth',zeros(1,2),'wavelength',photometry.J);
scienceCombo = scienceCombo.*tel*dm*cam;
```

The science camera clock rate is set to an arbitrary unit of 1 meaning it is slaved to the sampling rate of the telescope i.e. 500Hz. The camera exposure is set to 500 frames at rate of then 500Hz and the camera integration starts after the first 20 frames. The *startDelay* property is reset to 0 when the exposure starts.

```
flush(cam)
cam.clockRate    = 1;
exposureTime     = 100;
cam.exposureTime = exposureTime;
startDelay       = 20;
figure(31416)
imagesc(cam,'parent',subplot(2,1,1))
% cam.frameListener.Enabled = true;
subplot(2,1,2)
h = imagesc(catMeanRmPhase(scienceCombo));
axis xy equal tight
colorbar
```

The closed–loop controller is a simple integrator where the DM coefficients $c$ are given by

$$c_{n+1} = c_n - g_{cl}Ms \tag{4}$$

with $g_{cl}$ the integrator gain, $M$ the truncated pseudo–inverse of the poke matrix and $s$ is the vector of wavefront sensor slopes

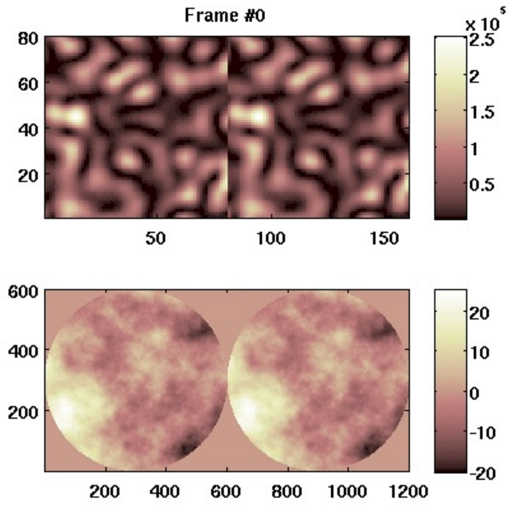
```
gain_cl  = 0.5;
```

The pseudo–open loop controller is given by

$$c_{n+1} = (1-g_{pol})c_n + g_{pol} * F^{-1}\left(C_{\varphi s}C_{ss}^{-1}\left(s - Dc_n\right)\right). \tag{5}$$

$g_{pol}$ is the low–pass filter gain, $F$ is the matrix of the DM influence functions and $D$ is the poke matrix.

```
gain_pol = 0.7;
F = 2*bifaLowRes.modes(wfs.validActuator,:);
iF = pinv(full(F),1e-1);
```

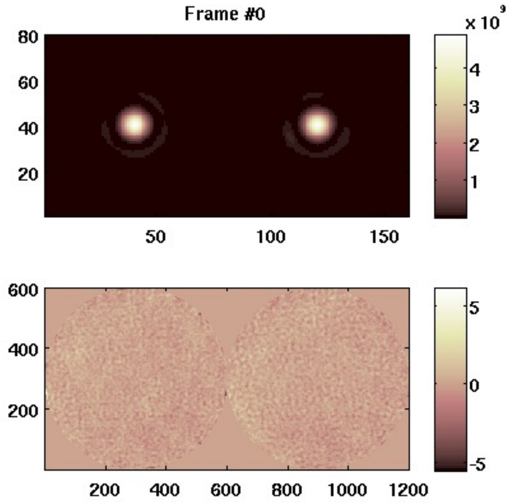


Before closing the loop, the DM coefficients are set to 0 and the science detector is reset. The logging of the wavefront variance of the science object is turned on. The phase variance will be stored into the *phaseVar* property. The LMMSE object is set to return the wavefront estimate in a vector and the iterative solver will use the previous estimate as first guess.

```
dm.coefs = zeros(dm.nValidActuator,2);
flush(cam)
set(scienceCombo,'logging',true)
set(scienceCombo,'phaseVar',[])
slmmse.wavefrontSize = [dm.nValidActuator,1];
slmmse.warmStart = true;
cam.startDelay   = startDelay;
cam.frameListener.Enabled = false;
% set(ngsCombo,'magnitude',8)
% wfs.camera.photonNoise = true;
% wfs.camera.readOutNoise = 2;
% wfs.framePixelThreshold = wfs.camera.readOutNoise;
```

The loop is closed for one full exposure of the science camera.

```
nIteration = startDelay + exposureTime;
for k=1:nIteration
    % Objects update
    +tel;
    +ngsCombo;
    +scienceCombo;
    % Closed-loop controller
    dm.coefs(:,1) = dm.coefs(:,1) - gain_cl*calibDm.M*wfs.slopes(:,1);
    % Pseudo-open-loop controller
    dm.coefs(:,2) = (1-gain_pol)*dm.coefs(:,2) + ...
        gain_pol*iF*( slmmse*( wfs.slopes(:,2) - calibDm.D*dm.coefs(:,2) ) );
    % Display
%     set(h,’Cdata’,catMeanRmPhase(scienceCombo))
%     drawnow
end
imagesc(cam)
set(h,’Cdata’,catMeanRmPhase(scienceCombo))
```



The time series of wavefront variance is saved in the *phaseVar* property of the *scienceCombo* object array. With the *logging* property turned on, each time the *phase* property is updated, its variance is computed and saved into *phaseVar*. The *phase* property is updated twice, first after propagation through the atmosphere+telescope system and second after reflection off the DM. So for each loop iteration, 2 phase variance are saved. The *phaseVar* vector is reshaped in a *nIteration*×2 array, the first column and second columns correspond respectively to the time series of the variance of the full and residual atmospheric aberrations.
The Strehl ratio is estimated from the residual phase variance using the Marechal approximation and it is compared to the Strehl ratio derived from the long exposure image.
For reference, the theoretical piston removed turbulence wavefront rms *wfe_lsq* is also plotted (the black dashed line). *wfe_lsq* is computed with the *residualVariance* method. This method computes the residual variance after removal of Zernike modes from the atmosphere wavefront on a given telescope. The first argument of the method is the number of Zernike modes to be removed starting from the piston mode.

```
var_wfe_lsq = reshape(scienceCombo(1).phaseVar(1:nIteration*2),2,[])’;
wfe_lsq = sqrt(var_wfe_lsq)/scienceCombo(1).waveNumber*1e6;
var_wfe_lmmse = reshape(scienceCombo(2).phaseVar(1:nIteration*2),2,[])’;
wfe_lmmse = sqrt(var_wfe_lmmse)/scienceCombo(1).waveNumber*1e6;
atm_wfe_rms = sqrt(zernikeStats.residualVariance(1,atm,tel))/ngs.waveNumber*1e6;
marechalStrehl_lsq = 1e2*exp(-mean(var_wfe_lsq(startDelay:end,2)));
marechalStrehl_lmmse = 1e2*exp(-mean(var_wfe_lmmse(startDelay:end,2)));
psfStrel = 1e2*cam.strehl;
```

## 5. LASER TOMOGRAPHY ADAPTIVE OPTICS

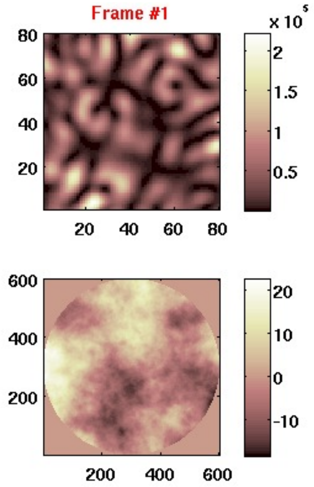
In this section, the performance of an LTAO system on the same telescope is compared to the two NGS AO systems of the former section.

Lets reset the atmosphere to start with the same initial conditions that the NGS AO systems and also the DM.

```
reset(tel)
dm.coefs = zeros(dm.nValidActuator,1);
```

The optical paths of both the LGS constellation and the science star are defined and the display is updated accordingly. Note how the WFS display is now showing the imagelets and slopes corresponding to the 3 LGSs. The WFS frame and slope listeners are turned off to speed up the computation.

```
science = science.*tel*dm*cam;
lgsAst = lgsAst.*tel*dm*wfs;
figure(31416)
imagesc(cam,’parent’,subplot(2,1,1))
subplot(2,1,2)
h = imagesc(catMeanRmPhase(science));
axis xy equal tight
colorbar
wfs.camera.frameListener.Enabled = false;
wfs.slopesListener.Enabled = false;
```

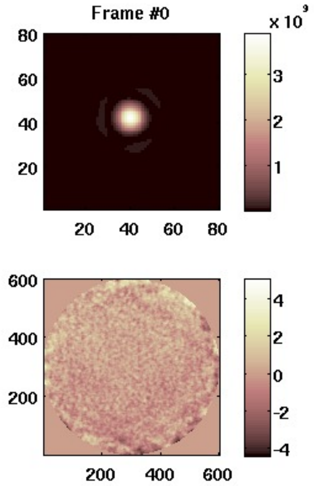


The logging of the wavefront variance of the science object is turned on. The LGS LMMSE object is set to return the wavefront estimate in a vector and the iterative solver will use the previous estimate as first guess.

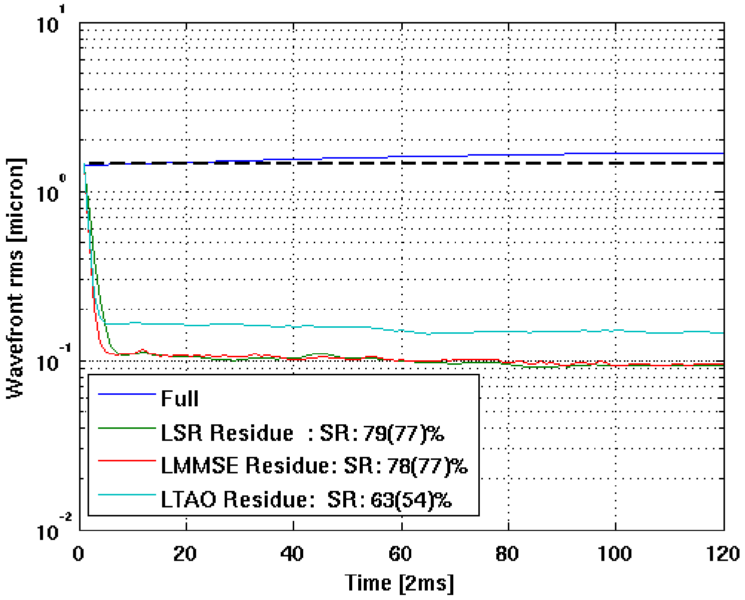
```
flush(cam)
cam.startDelay = startDelay;
set(science,’logging’,true)
set(science,’phaseVar’,[])
lgsAst_slmmse.wavefrontSize = [dm.nValidActuator,1];
lgsAst_slmmse.warmStart = true;
cam.frameListener.Enabled = false;
```

The loop is closed for one full exposure of the science camera.

```
for k=1:cam.startDelay + cam.exposureTime
    % Objects update
    +tel;
    +lgsAst;
    +science;
    % Pseudo-open-loop controller
    dm.coefs = (1-gain_pol)*dm.coefs + ...
        gain_pol*iF*( lgsAst_slmmse*...
( bsxfun( @minus, wfs.slopes, calibDm.D*dm.coefs ) ) );
    % Display
%     set(h,’Cdata’,catMeanRmPhase(science))
%     drawnow
end
imagesc(cam)
set(h,’Cdata’,catMeanRmPhase(science))
```



The time series of wavefront variance is read from the *phaseVar* property of the *science* object. The Strehl ratio is estimated from the residual phase variance using the Marechal approximation and it is compared to the Strehl ratio derived from the long exposure image. The Marechal Strehl is in parentheses in the legend of the plot.



```
var_wfe_ltao = ....
    reshape(science.phaseVar(1:nIteration*2),2,[])';
wfe_ltao = ....
    sqrt(var_wfe_ltao)/science.waveNumber*1e6;
marechalStrehl_ltao = ...
    1e2*exp(-mean(var_wfe_ltao(startDelay:end,2)));
psfStrehl_ltao =1e2*cam.strehl;
```

```
figure(fix(exp(1)*1e2))
u = 1:nIteration;
semilogy(u,wfe_lsq,u,wfe_lmmse(:,2),u,wfe_ltao(:,2))
line([1,nIteration],ones(1,2)*atm_wfe_rms,...
    'color','k','LineStyle','--','linewidth',2)
grid
xlabel('Time [2ms]')
ylabel('Wavefront rms [micron]')
legend('Full',...
    sprintf('LSR Residue  : SR: %2.0f(%2.0f)%%',psfStrel(1),marechalStrehl_lsq),...
    sprintf('LMMSE Residue: SR: %2.0f(%2.0f)%%',psfStrel(2),marechalStrehl_lmmse),...
    sprintf('LTAO Residue:  SR: %2.0f(%2.0f)%%',psfStrehl_ltao, ...
            marechalStrehl_ltao),0)
```

## 6. CONCLUSION

OOMAO is a versatile *Matlab*® toolbox for the simulations of adaptive optics systems. With OOMAO, AO systems are modeled using an object–oriented approach where there is a class for each AO component. The components can then be chained to each other to form the system optical path. Properties inside the classes allow to either get wavefront information or to shape it. The simple framework of OOMAO makes it a versatile simulation tool capable of simulating many systems from eXtreme-AO to LTAO on ELTS. OOMAO is a mature software backed up by several years of development and it is now used by several projects. The source code can be retrieved from GitHub (`http://github.com/rconan/OOMAO`).

## Acknowledgments

C. Correia acknowledges the support of the European Research Council through the Marie Curie Intra-European Fellowship with reference FP7-PEOPLE-2011-IEF, number 300162

## REFERENCES

[1] D. R. Andersen, K. J. Jackson, C. Blain, C. Bradley, C. Correia, M. Ito, O. Lardière, and J.-P. Véran, "Performance Modeling for the RAVEN Multi-Object Adaptive Optics Demonstrator," *PASP* **124**, pp. 469–484, May 2012.

[2] R. Conan, , B. Espeland, S. Parcell, M. V. Dam, A. Bouchez, and P. Piatrou, "Laser Tomography Adaptive Optics for the Giant Magellan Telescope," in *Adaptive Optics for Extremely Large Telescopes III*, May 2013.

[3] F. Rigaut and M. Van Dam, "Simulating Astronomical Adaptive Optics Systems Using Yao," in *Proceedings of the Third AO4ELT Conference*, S. Esposito and L. Fini, eds., Dec. 2013.

[4] M. Carbillet, C. Vérinaud, B. Femenía, A. Riccardi, and L. Fini, "Modelling astronomical adaptive optics - I. The software package CAOS," *Monthly Notices of the RAS* **356**, pp. 1263–1275, February 2005.

[5] C. Correia, K. Jackson, J.-P. Véran, D. Andersen, O. Lardière, and C. Bradley, "Static and predictive tomographic reconstruction for wide-field multi-object adaptive optics systems," *Journal of the Optical Society of America A* **31**, p. 101, January 2014.