



OPTIODELER

**A modular modelling software
for optical interferometry**

Context of the project

oimodel.py

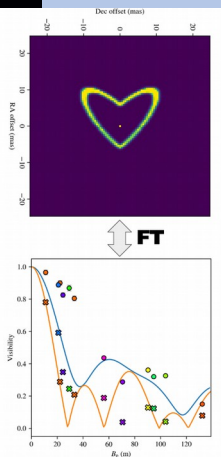
Ideas and skeleton for a general, modular, panchromatic, and hopefully fast model-fitting tool in python

July 2021

The quest for the ultimate modeling tool for optical-IR interferometry

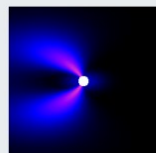
The MATISSE modeling working group:
 Bill Danchi, Julien Drevon, Violeta Gámez Rosas, Michiel Hogerheijde, Jacob Isbell, Julia Kobus, Bruno Lopez, Alexis Matter, **Anthony Meilland**, Florentin Millour, Eric Pantin, Dieter Schertl, Marten Scheuck, Roy van Boekel, **József Varga**, Rens Waters, Gerd Weigelt

MATISSE Science Team meeting, 2021 November 18



MATISSE group

Real time astrophysical models



Kinematic Be disk

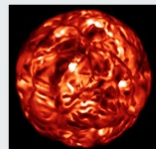
Model of the geometry (size and shape) and kinematics (rotation and expansion) of circumstellar, flat, rotating disks, relevant to Be stars. It is suited to interpret spectro-interferometric data obtained on emission lines formed in the disk.



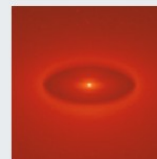
Disk and stellar continuum – DISCO

Model of the continuum emission from a star surrounded by a gaseous circumstellar disk (free-free and bound-free), with partially ionized and geometrically thin disk with a physical structure given by the viscous Keplerian accretion disk model. DISCO is well suited to model Be stars.

Precalculated grids of astrophysical models



Evolved star
 Stellar surface simulation with famous RSG B

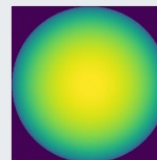


Supergiant B[e] with HDUST

Grid of models for B[e] supergiant stars computed with the 3d Monte Carlo radiative transfer code HDUST. The non-spherical circumstellar envelope (CSE), composed of gas (hydrogen) and dust (silicate), is modelled considering a bimodal outflow description (two-component wind).

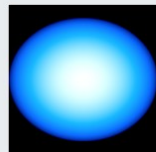


Binary spiral
 Phenomenological model of massive stars



Limb-darkening with SATLAS

Grid of models providing intensity maps for spherically symmetric stars, showing the limb darkening effect. The models were computed with the SATLAS model stellar atmospheres for several spectral bands. Data is provided for FGK dwarfs and red giants.



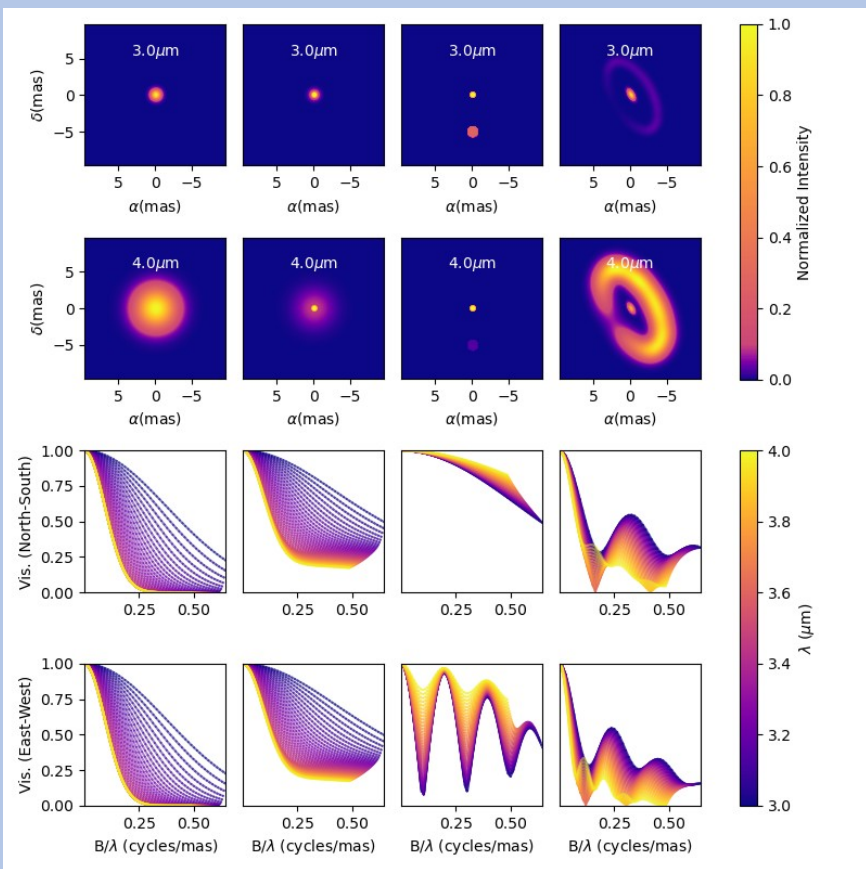
Analytical Limb-darkening Elliptical or Spherical – ALDES

ALDES provides intensity maps (images) or 1d intensity profiles for spherical or elliptical stars showing the limb darkening (LD) effect. Different LD laws are offered: uniform disk, linear, power law, quadratic, square root, logarithmic and four-parameter.

JMMC/AMHRA

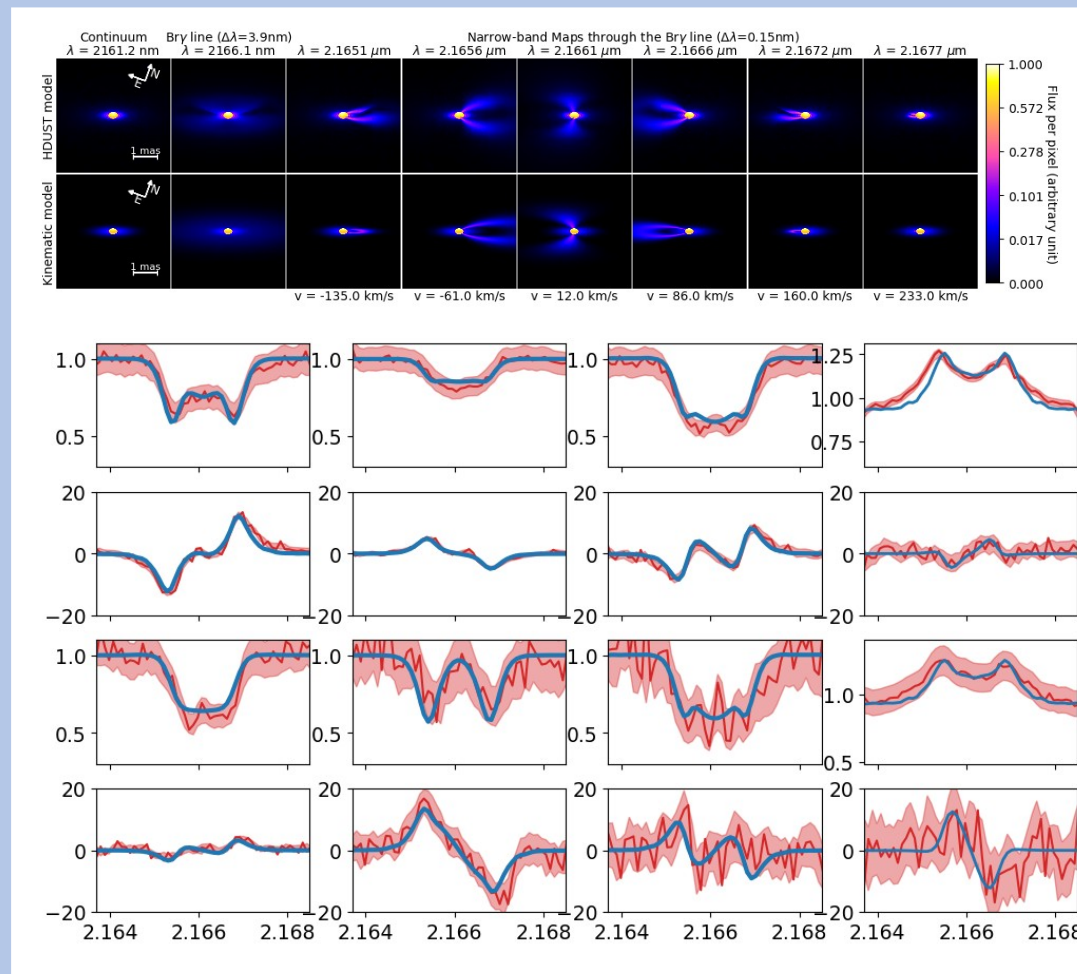


Context of the project



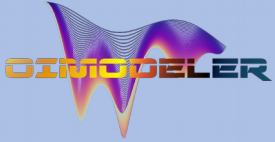
Oimodel.py (2021)

A modular & Fourier-based chromatic modelling tool with a emcee (MCMC) fitter



Cubetools.py (2017)

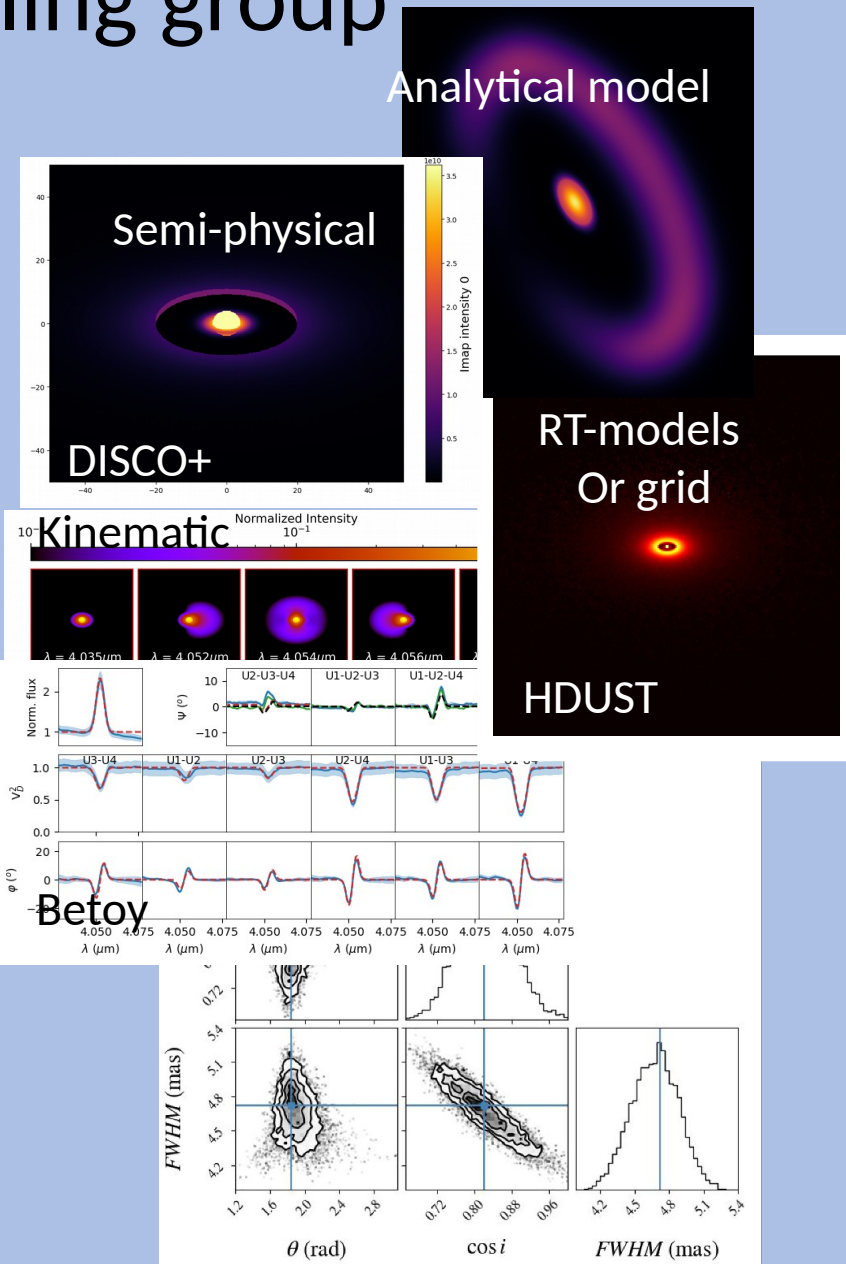
Computing interferometric measurements from chromatic image-cubes



Discussion in MATISSE modelling group

By the end of 2021

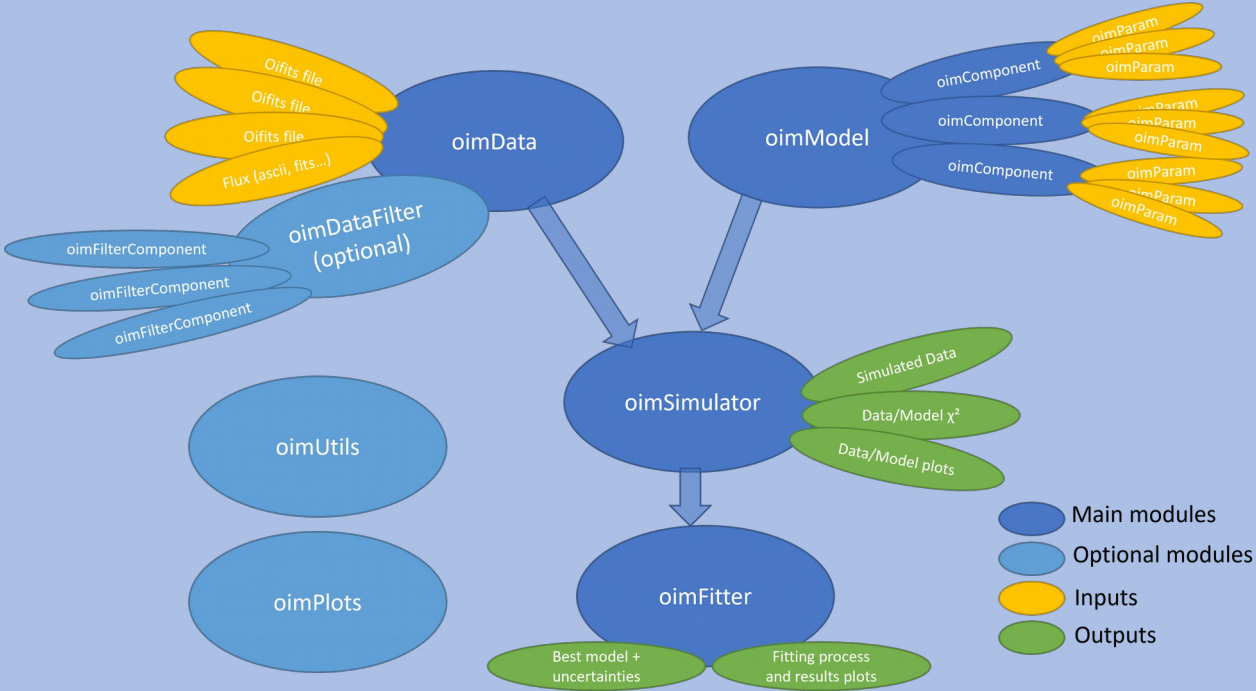
- Python3
- Modularity and flexibility
 - Analytical models in Fourier-plan
 - Analytical & numerical models in Image-plan
 - Use outputs from radiative transfer and explore grids of models
 - Build more complex geometries by mixing components
- Chromaticity and time dependence
 - Of the components parameters
 - Chromatic components (such as temperature gradient, binary)
 - Kinematics through line models
- Ability to use interferometric data from all instruments (oifits)
- Produce high-quality publishable outputs
 - Robust estimation of parameters with uncertainties and correlations
 - Nice customizable plots
 - Export simulated data and images to standard format (oifits and fits images)
- Expandability
 - Easily create new components for models (inheritance, wrapping functions)
 - But also other features: type of data, filters, fitters, plots
- Well documented (and with a test suite, examples, tutorials)
- Open source & easily available (Github)





Coding started in 2022

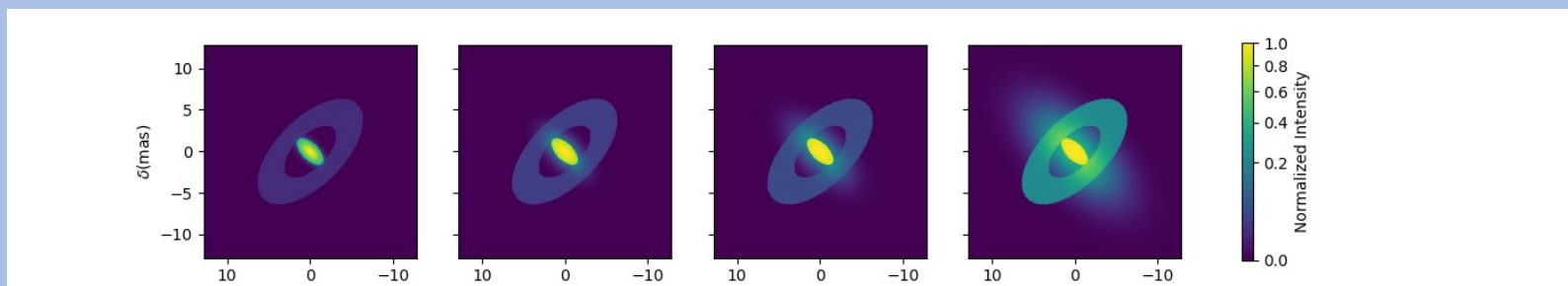
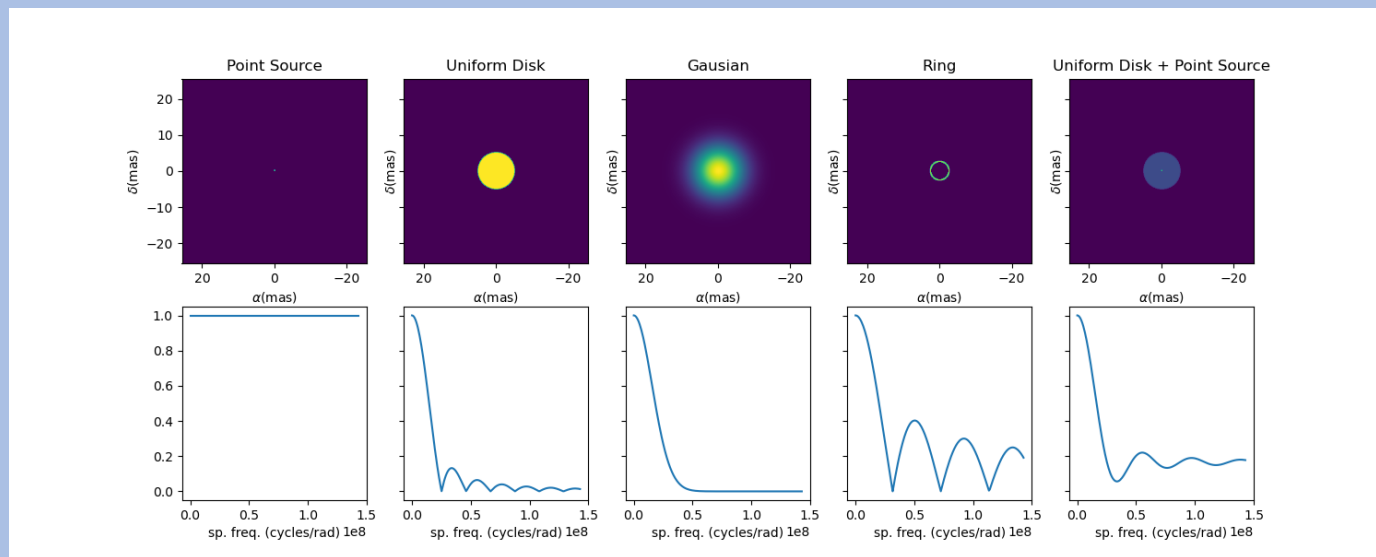
- **January :**
 - Model skeleton : oimParam, oimComponents, oimModel





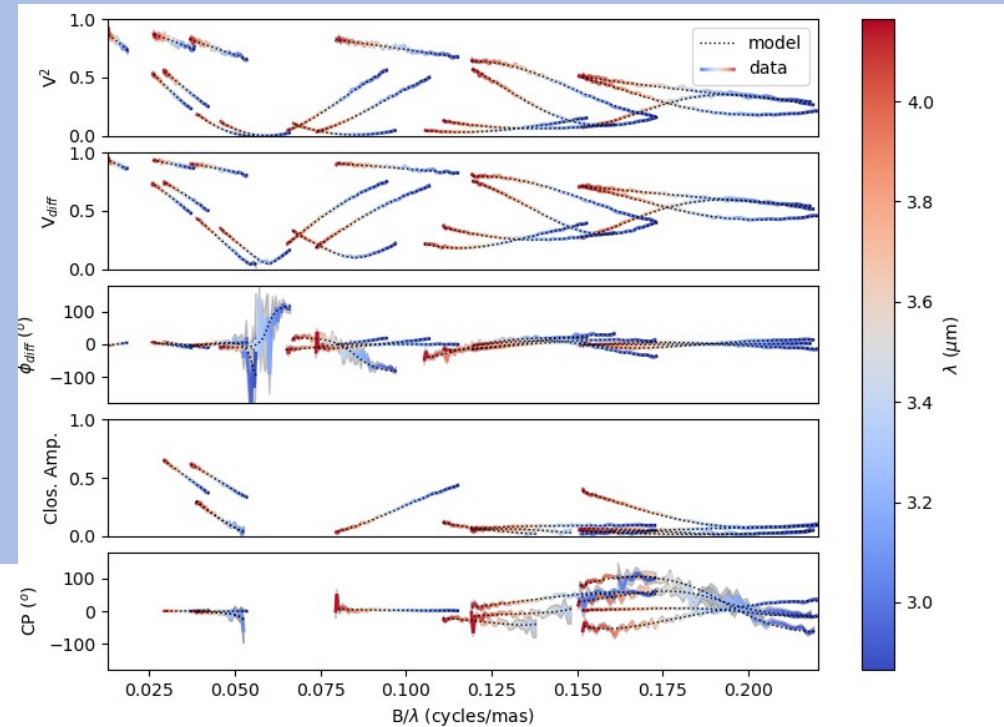
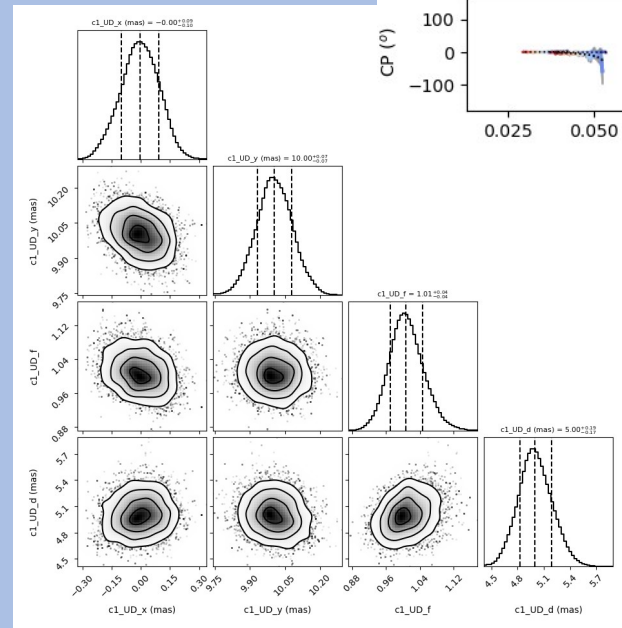
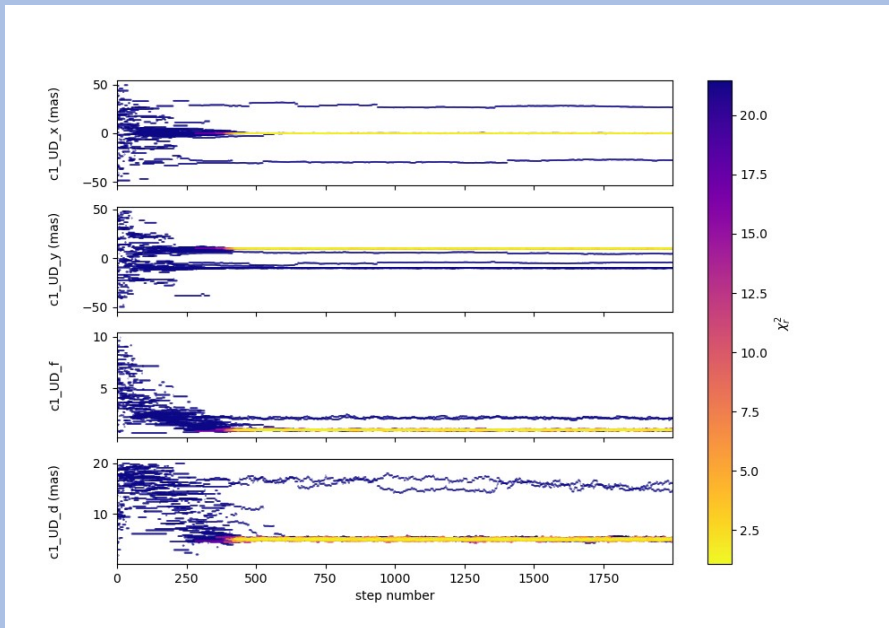
Coding started in 2022

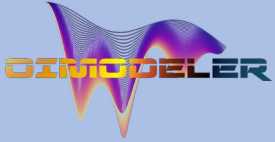
- January :
 - Model skeleton : oimParam, oimComponents, oimModel
- **March-May :**
 - Implementation of Fourier-based components
 - with chromatic parameters (using linear interpolation)
 - Link between parameters



Coding started in 2022

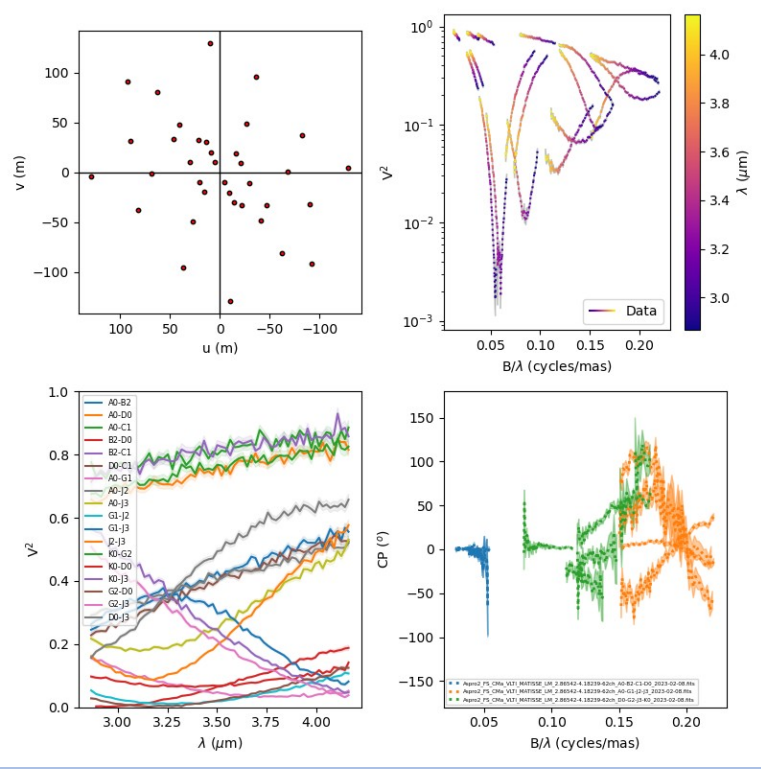
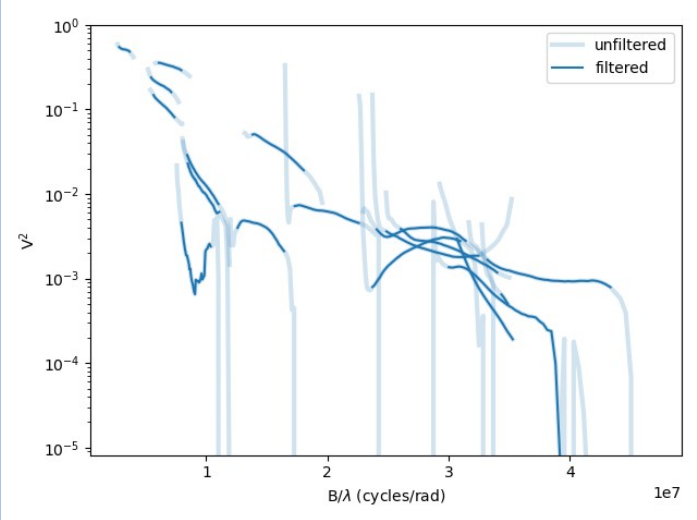
- January :
 - Model skeleton : oimParam, oimComponents, oimModel
- March-May :
 - Implementation of Fourier-based components
 - with chromatic parameters (using linear interpolation)
 - Link between parameters
- **June-August:**
 - oimSimulator class (optimized data, data simulation, chi2)
 - oimFitter class and first emcee Fitter





Coding started in 2022

- January :
 - Model skeleton : oimParam, oimComponents, oimModel
- March-May :
 - Implementation of Fourier-based components
 - with chromatic parameters (using linear interpolation)
 - Link between parameters
- June-August:
 - oimSimulator class (optimized data, data simulation, chi2)
 - oimFitter class and first emcee Fitter
- **September:**
 - data filtering & plots
 - documentation (on readthedoc)



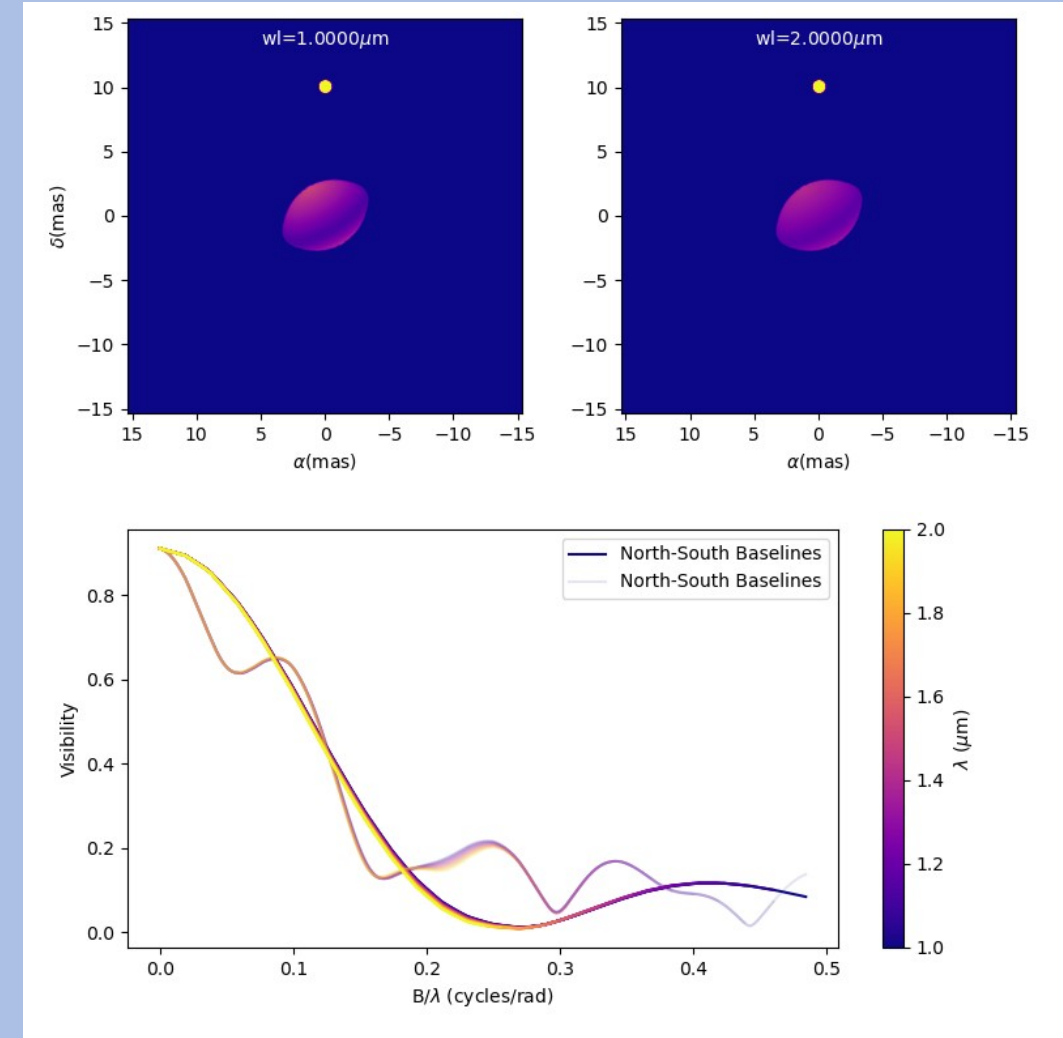
The screenshot shows the GitHub repository for OIMODELER. The main content includes:

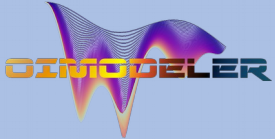
- Repository Name:** oimodeler
- Project Description:** The oimodeler project aims at developing a modular and easily expandable python-based modelling software for optical interferometry. The project started end of 2021, and the software is currently at an early stage of development.
- Usage:** It allows to manipulate data in the oifits format, build complex models from various components, simulate data from the model at the spatial frequencies of your observations, computed chi2, perform model fitting (using mcmc or other fitters), and plot results easily.
- Warning:** The software is in early development.
- Features:**
 - Models : in Fourier and Image plans with chromaticity and time dependence
 - Data : interferometric data only. No photometric or spectroscopic data.
 - Data Filters : Filtering wavelength range, and data type (VIS2DATA, VISAMP...)
 - Fitters : Implementation of a basic emcee-based fitter with plots for results
 - Plots : Basics plots of oifits data and uv-plan plot
 - Utils : miscs utilities for oifits data (creating and modifying array, getting info...)



Coding started in 2022

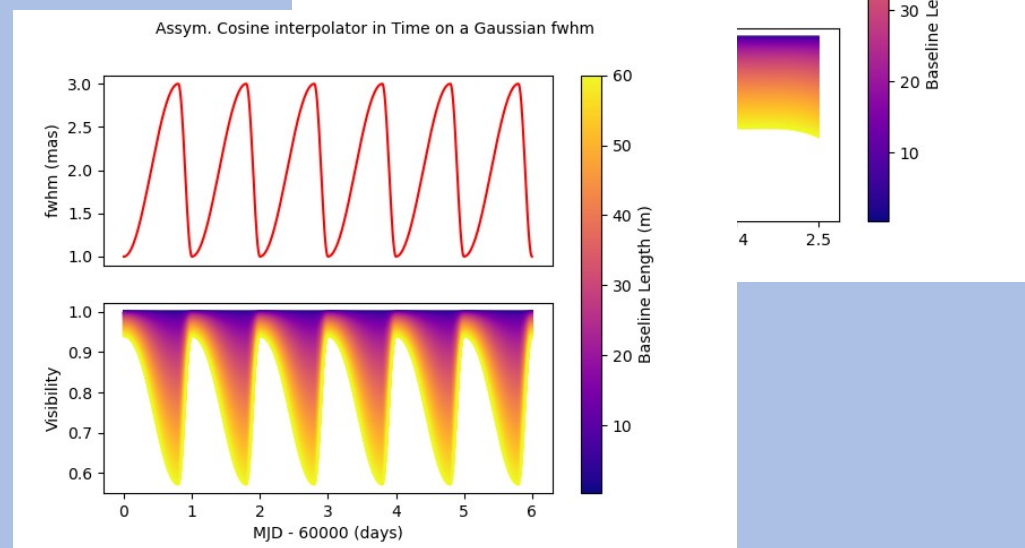
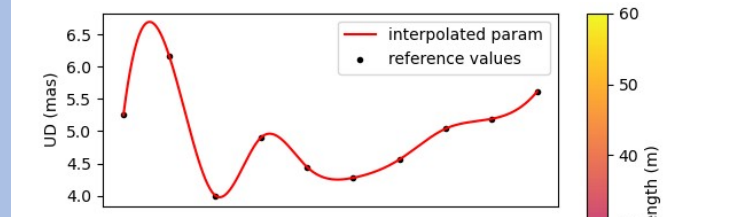
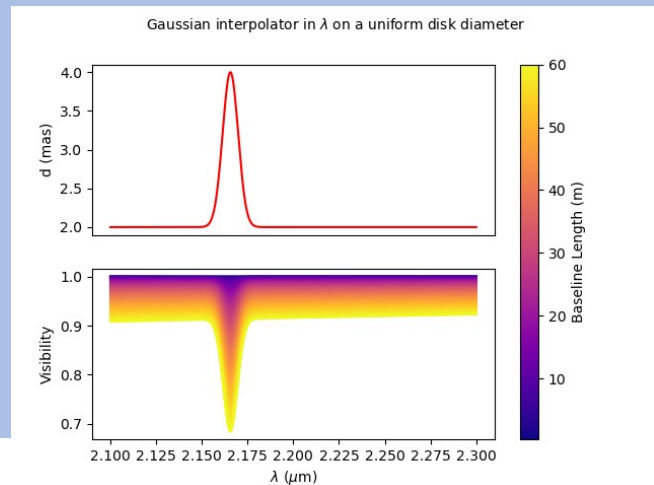
- January :
 - Model skeleton : oimParam, oimComponents, oimModel
- March-May :
 - Implementation of Fourier-based components
 - with chromatic parameters (using linear interpolation)
 - Link between parameters
- June-August:
 - oimSimulator class (optimized data, data simulation, chi2)
 - oimFitter class and first emcee Fitter
- September:
 - data filtering & plots
 - documentation (on readthedoc)
- **October :**
 - image-plan components (FFT, sampling...)





Coding started in 2022

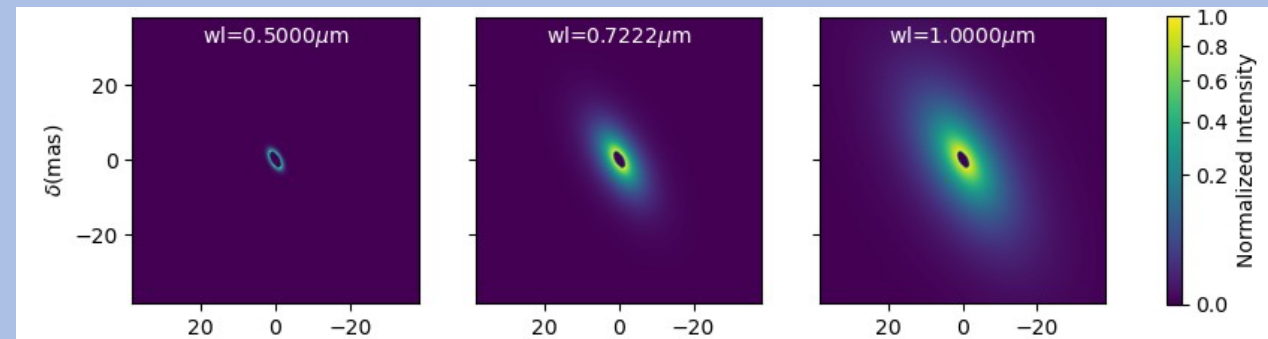
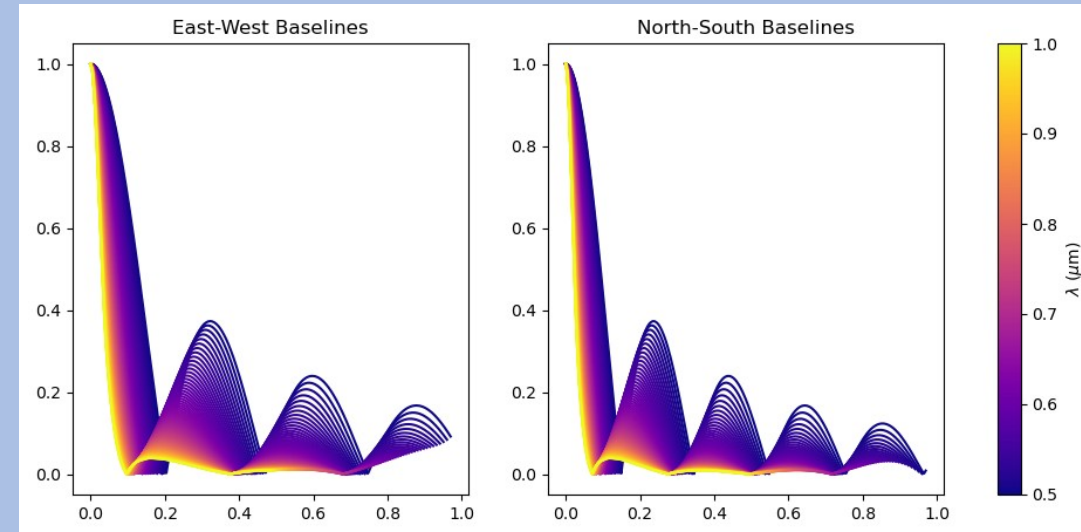
- January
 - Model skeleton : oimParam, oimComponents, oimModel
- March-May
 - Implementation of Fourier-based components
 - with chromatic parameters (using linear interpolation)
 - Link between parameters
- June-August:
 - oimSimulator class (optimized data, data simulation, chi2)
 - oimFitter class and first emcee Fitter
- September:
 - data filtering & plots
 - documentation (on readthedoc)
- **October :**
 - image-plan components (FFT, sampling...)
 - parameter interpolators (time and chromaticity)

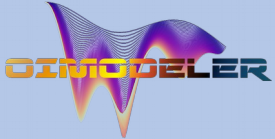




Coding started in 2022

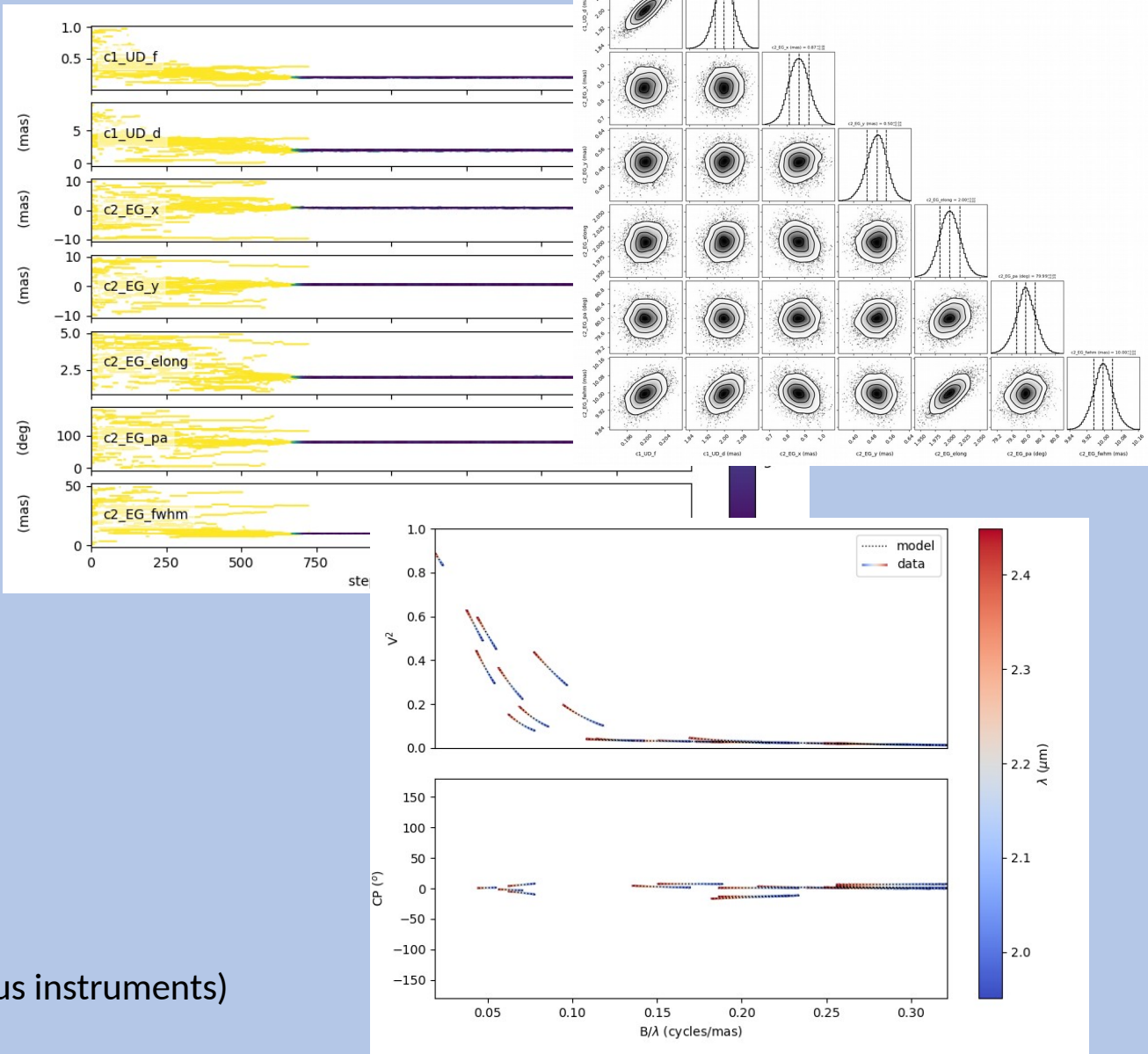
- January
 - Model skeleton : oimParam, oimComponents, oimModel
- March-May
 - Implementation of Fourier-based components
 - with chromatic parameters (using linear interpolation)
 - Link between parameters
- June-August:
 - oimSimulator class (optimized data, data simulation, chi2)
 - oimFitter class and first emcee Fitter
- September:
 - data filtering & plots
 - documentation (on readthedoc)
- October :
 - image-plan components (FFT, sampling...)
 - parameter interpolators (time and chromaticity)
- **November :**
 - radial profile components (Hankel Transform)

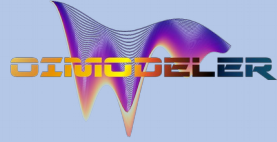




Coding started in 2022

- January
 - Model skeleton : oimParam, oimComponents, oimModel
- March-May
 - Implementation of Fourier-based components
 - with chromatic parameters (using linear interpolation)
 - Link between parameters
- June-August:
 - oimSimulator class (optimized data, data simulation, chi2)
 - oimFitter class and first emcee Fitter
- September:
 - data filtering & plots
 - documentation (on readthedoc)
- October :
 - image-plan components (FFT, sampling...)
 - parameter interpolators (time and chromaticity)
- November :
 - radial profile components (Hankel Transform)
- **December :**
 - Code cleaning + examples
 - Test suite (ASPRO simulated data and real data from various instruments)





Oimodeler on the web

The screenshot shows the GitHub repository for oimodeler. The repository is public and has 2 branches and 3 tags. A notification indicates that the main branch is not protected. A recent merge pull request #3 is shown, merging from oimodeler/anthony-dev. The file list includes docs, examples, images, oimodeler, .gitignore, LICENSE, README.md, and setup.py. The README.md file is open, showing the license (GNU), lifecycle (EarlyDevelopment), and version (0.0.1). The title of the repository is "oimodeler" and the description is "A modular modelling tool for optical interferometry".

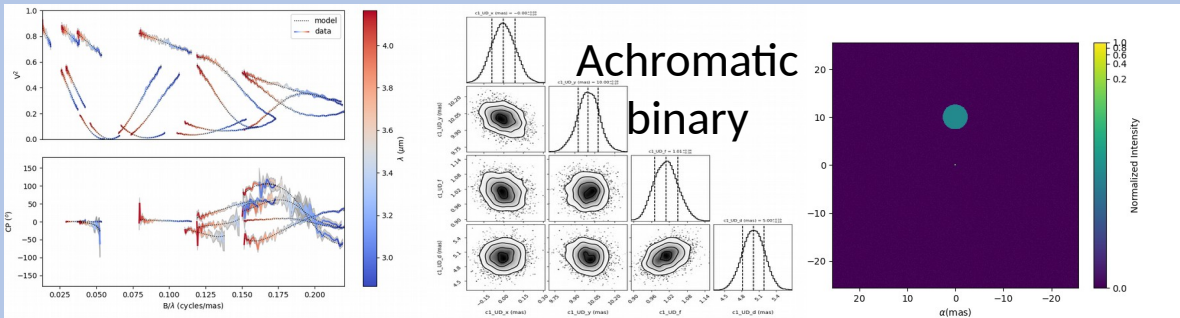
Code available on github
<https://github.com/oimodeler>

The screenshot shows the ReadTheDocs page for oimodeler. The page features the oimodeler logo and a search bar. The left sidebar contains a navigation menu with links to Overview, Installation, Getting Started, Examples, and API. The main content area displays the title "oimodeler" and a description: "The oimodeler project aims at developing a modular and easily expandable python-based modelling software for optical interferometry. The project started end of 2021, and the software is currently at an early stage of development." Below the description, it states that the software allows for manipulating data in the oifits format, building complex models, simulating data, and performing model fitting. A warning box indicates that the software is in early development and lists several limitations and features. A DigitalOcean advertisement is also visible on the left side of the page.

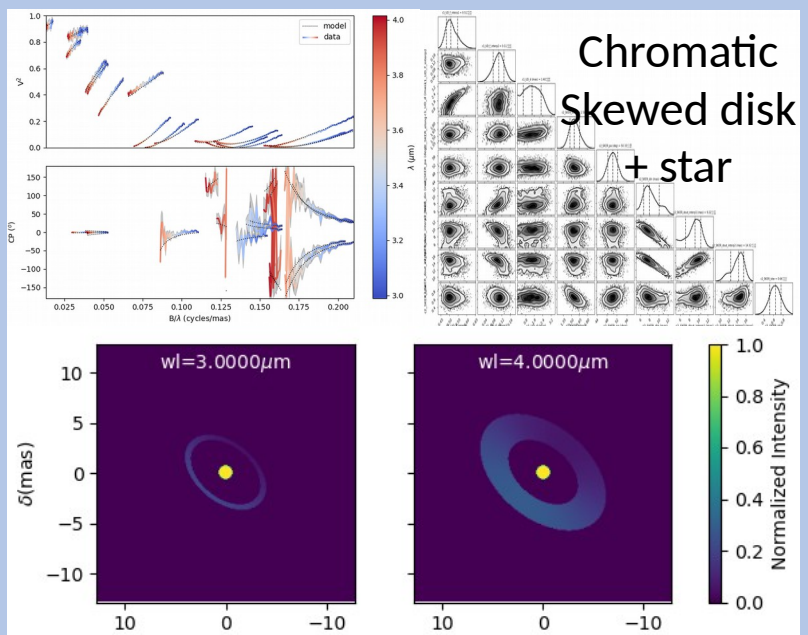
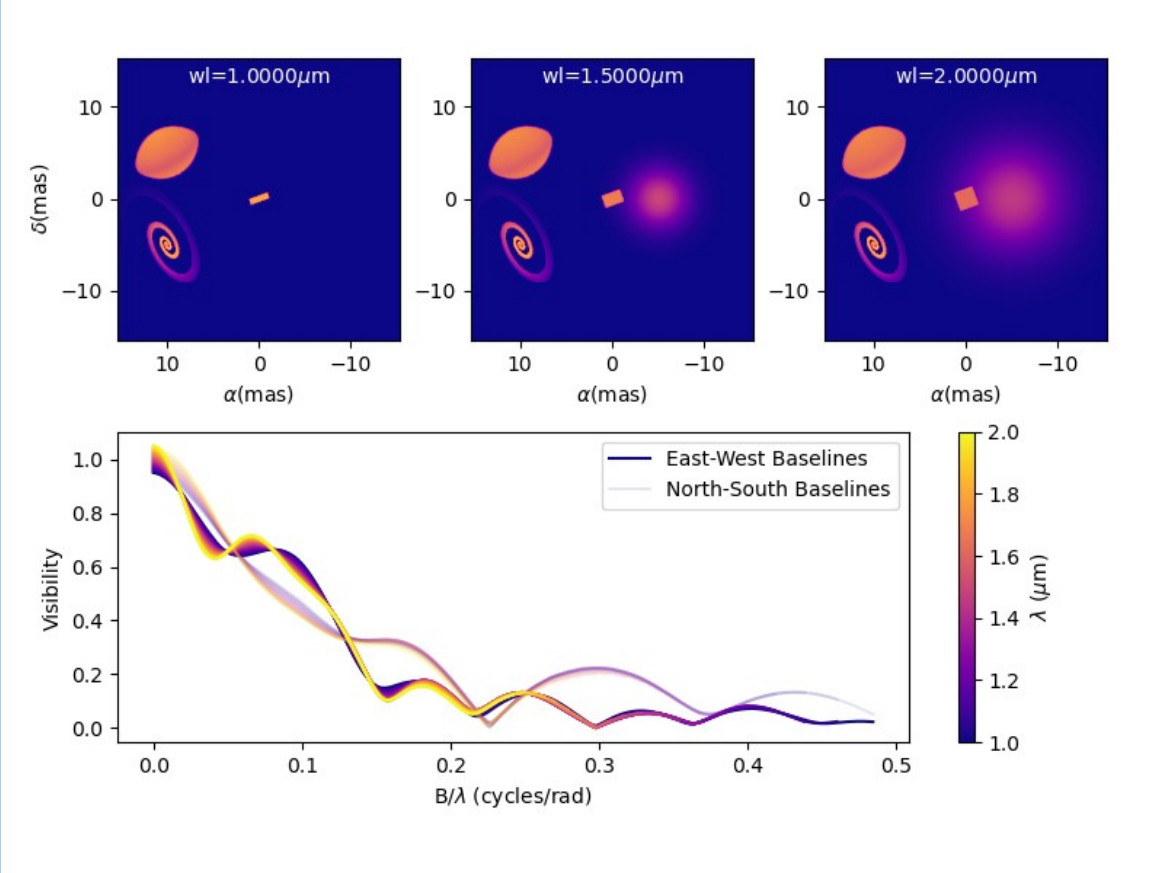
Documentation available on readthedocs
<https://oimodeler.readthedocs.io>

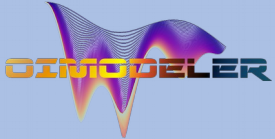
A Few examples

Model-fitting (on simulated data)



Adding new components





Example of simple model-fitting

```
import oimodeler as oim
```

Import oimodeler

```
ud = oim.oimUD(d=3,f=0.5,x=5,y=-5)  
pt = oim.oimPt(f=1)  
model = oim.oimModel(ud,pt)
```

Create a binary model

```
ud.params['d'].set(min=0.01,max=20)  
ud.params['x'].set(min=-50,max=50,free=True)  
ud.params['y'].set(min=-50,max=50,free=True)  
ud.params['f'].set(min=0.,max=10.)
```

Set the parameters space

```
pt.params['f']=oim.oimParamNorm(ud.params['f'])
```

Normalizing flux to 1 (remove one free parameter)

```
fit = oim.oimFitterEmcee(files,model,nwalkers=20)  
fit.prepare(init="random")  
fit.run(nsteps=2000, progress=True)
```

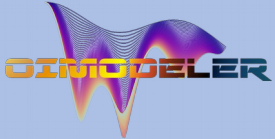
Create, prepare and run the MCMC fitter

```
median,err_l,err_u,err=fit.getResults(discard=1000)
```

Get the results and plots

```
fit.walkersPlot()  
fit.cornerPlot()  
fit.simulator.plot(["VIS2DATA","T3PHI"])  
model.showModel(512,0.1)
```

Make nice plots



Example of simple model-fitting

```
import oimodeler as oim

ud = oim.oimUD(d=3,f=0.5,x=5,y=-5)
pt = oim.oimPt(f=1)
model = oim.oimModel(ud,pt)

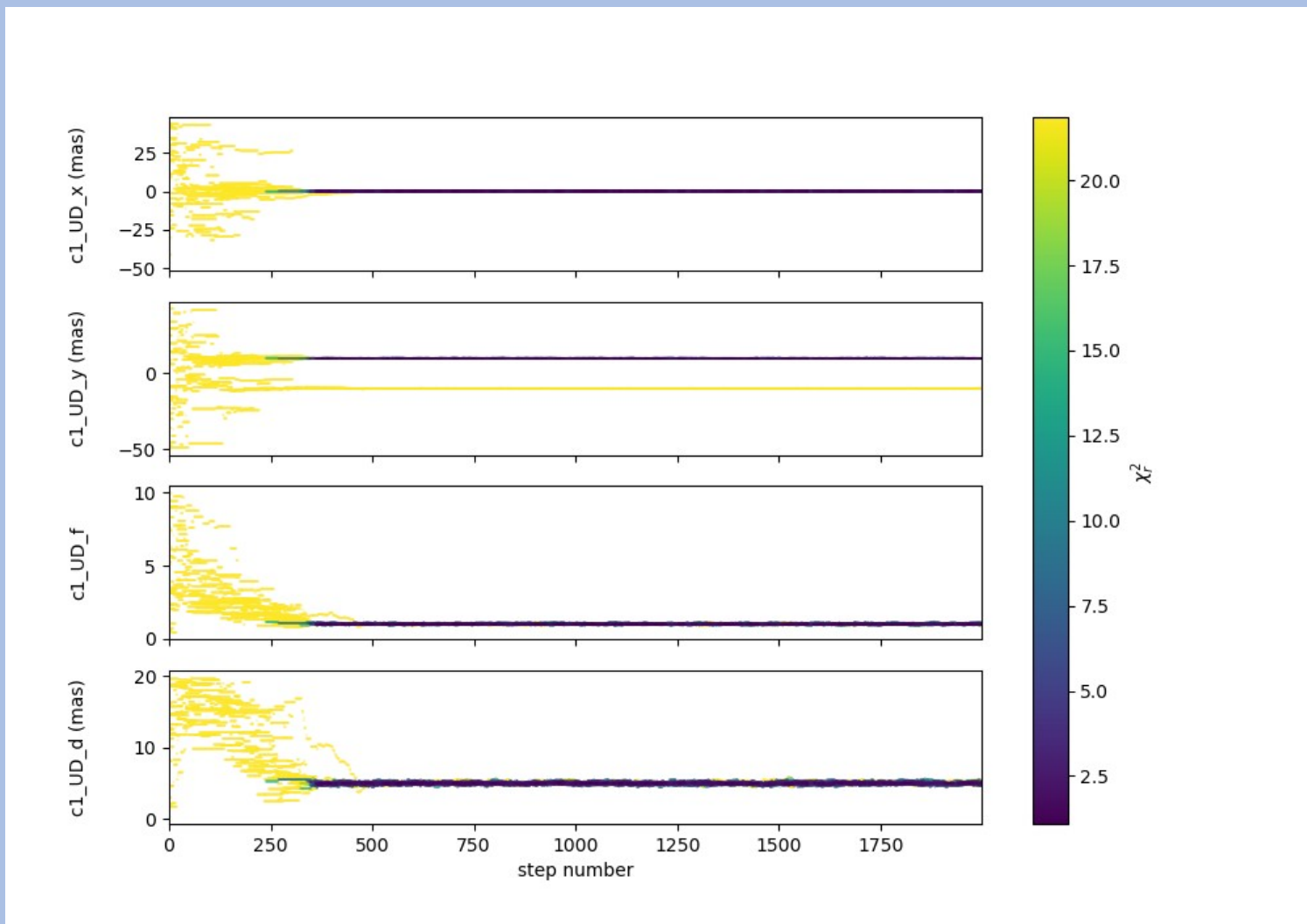
ud.params['d'].set(min=0.01,max=20)
ud.params['x'].set(min=-50,max=50,free=True)
ud.params['y'].set(min=-50,max=50,free=True)
ud.params['f'].set(min=0.,max=10.)

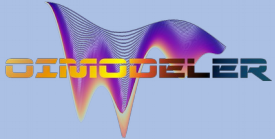
pt.params['f']=oim.oimParamNorm(ud.params['f'])

fit = oim.oimFitterEmcee(files,model,nwalkers=20)
fit.prepare(init="random")
fit.run(nsteps=2000, progress=True)

median,err_l,err_u,err=fit.getResults(discard=1000)

fit.walkersPlot()
fit.cornerPlot()
fit.simulator.plot(["VIS2DATA","T3PHI"])
model.showModel(512,0.1)
```





Example of simple model-fitting

```
import oimodeler as oim
```

```
ud = oim.oimUD(d=3,f=0.5,x=5,y=-5)
```

```
pt = oim.oimPt(f=1)
```

```
model = oim.oimModel(ud,pt)
```

```
ud.params['d'].set(min=0.01,max=20)
```

```
ud.params['x'].set(min=-50,max=50,free=True)
```

```
ud.params['y'].set(min=-50,max=50,free=True)
```

```
ud.params['f'].set(min=0.,max=10.)
```

```
pt.params['f']=oim.oimParamNorm(ud.params['f'])
```

```
fit = oim.oimFitterEmcee(files,model,nwalkers=20)
```

```
fit.prepare(init="random")
```

```
fit.run(nsteps=2000, progress=True)
```

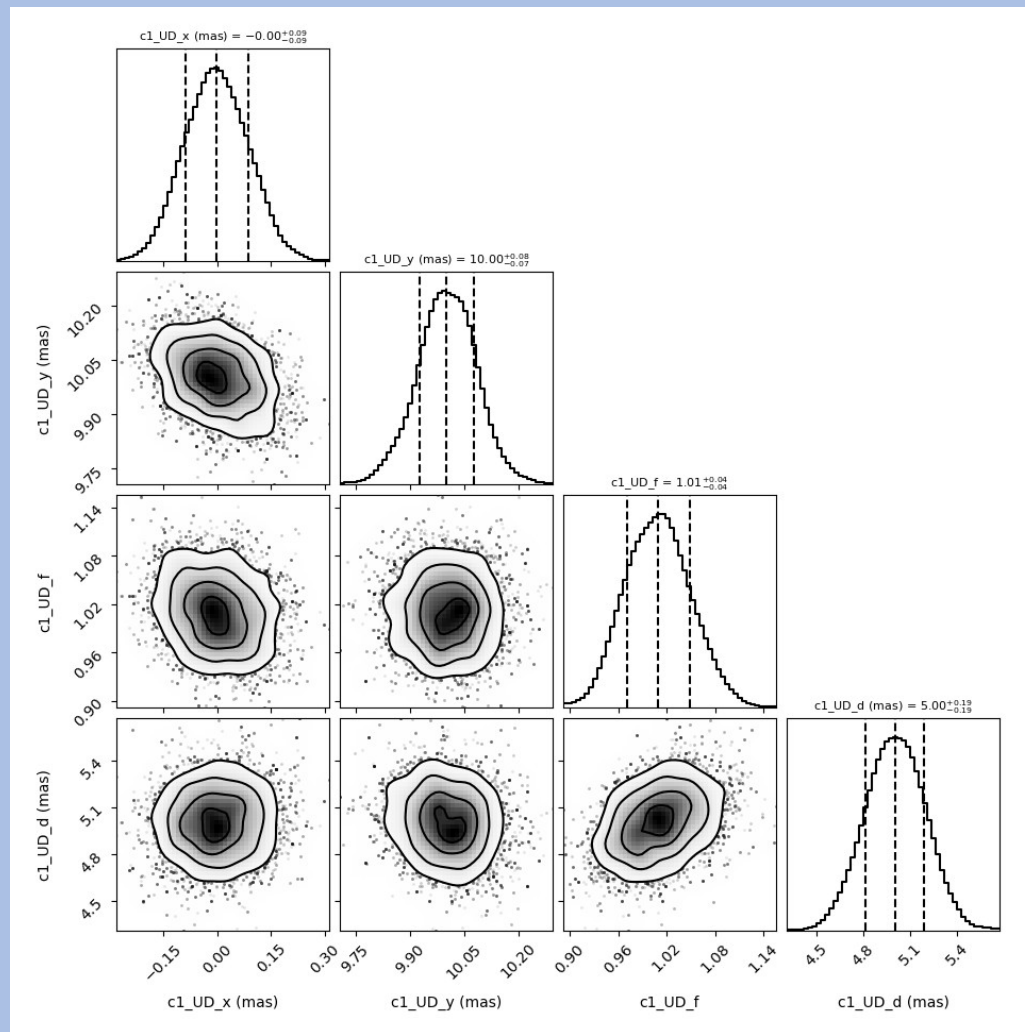
```
median,err_l,err_u,err=fit.getResults(discard=1000)
```

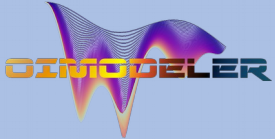
```
fit.walkersPlot()
```

```
fit.cornerPlot()
```

```
fit.simulator.plot(["VIS2DATA","T3PHI"])
```

```
model.showModel(512,0.1)
```





Example of simple model-fitting

```
import oimodeler as oim

ud = oim.oimUD(d=3,f=0.5,x=5,y=-5)
pt = oim.oimPt(f=1)
model = oim.oimModel(ud,pt)

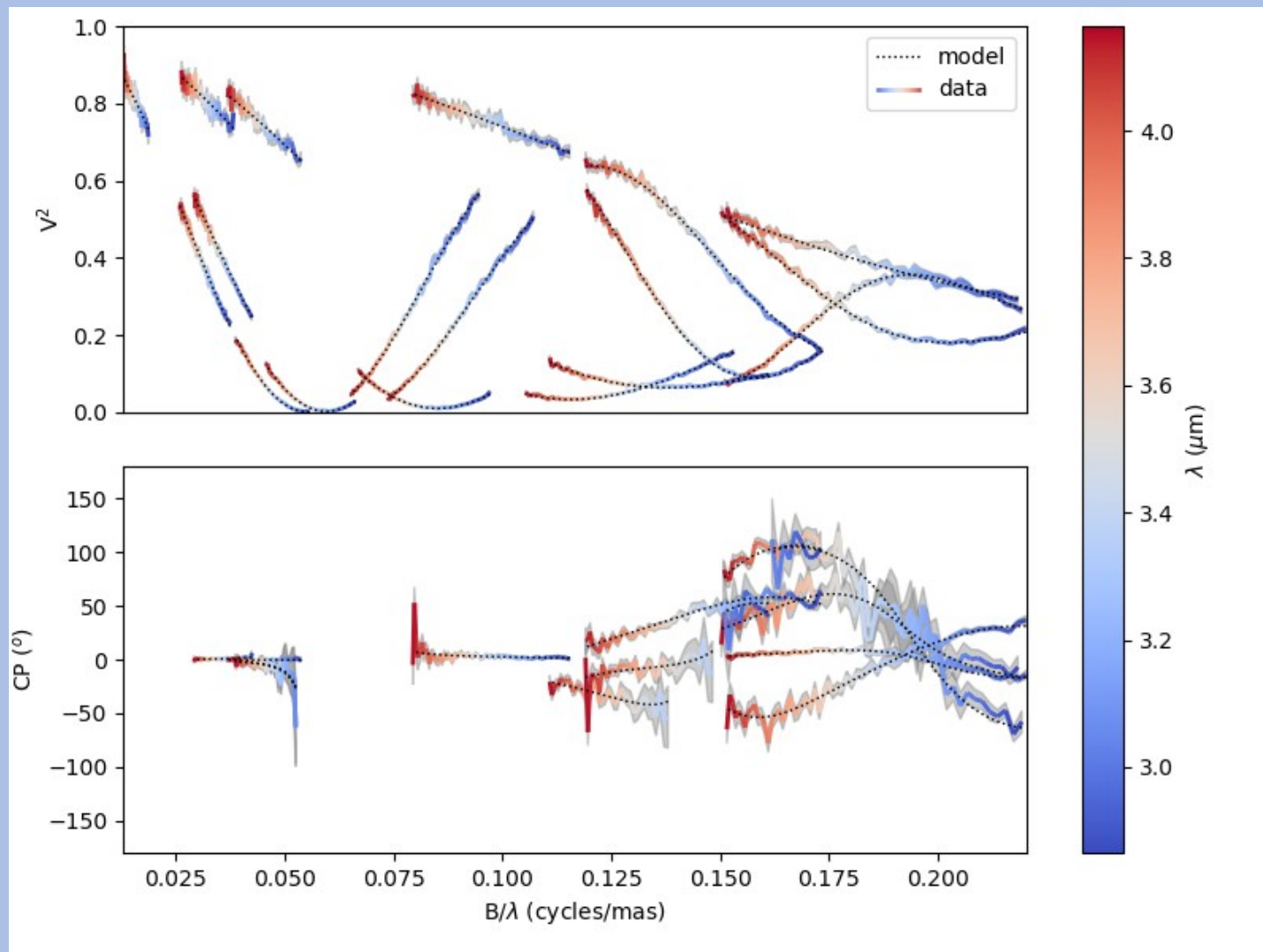
ud.params['d'].set(min=0.01,max=20)
ud.params['x'].set(min=-50,max=50,free=True)
ud.params['y'].set(min=-50,max=50,free=True)
ud.params['f'].set(min=0.,max=10.)

pt.params['f']=oim.oimParamNorm(ud.params['f'])

fit = oim.oimFitterEmcee(files,model,nwalkers=20)
fit.prepare(init="random")
fit.run(nsteps=2000, progress=True)

median,err_l,err_u,err=fit.getResults(discard=1000)

fit.walkersPlot()
fit.cornerPlot()
fit.simulator.plot(["VIS2DATA","T3PHI"])
model.showModel(512,0.1)
```



Example of simple model-fitting

```

import oimodeler as oim

ud = oim.oimUD(d=3,f=0.5,x=5,y=-5)
pt = oim.oimPt(f=1)
model = oim.oimModel(ud,pt)

ud.params['d'].set(min=0.01,max=20)
ud.params['x'].set(min=-50,max=50,free=True)
ud.params['y'].set(min=-50,max=50,free=True)
ud.params['f'].set(min=0.,max=10.)

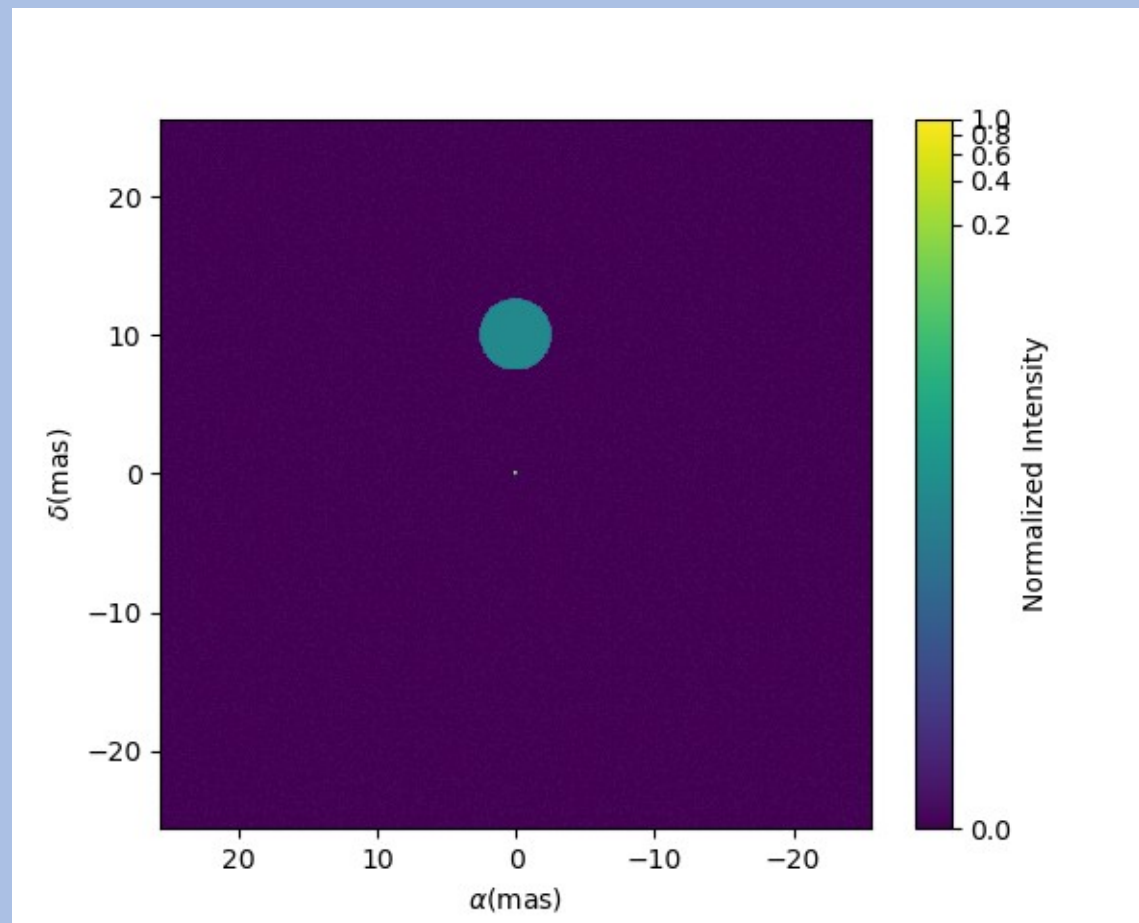
pt.params['f']=oim.oimParamNorm(ud.params['f'])

fit = oim.oimFitterEmcee(files,model,nwalkers=20)
fit.prepare(init="random")
fit.run(nsteps=2000, progress=True)

median,err_l,err_u,err=fit.getResults(discard=1000)

fit.walkersPlot()
fit.cornerPlot()
fit.simulator.plot(["VIS2DATA","T3PHI"])
model.showModel(512,0.1)

```



Chromatic model-fitting

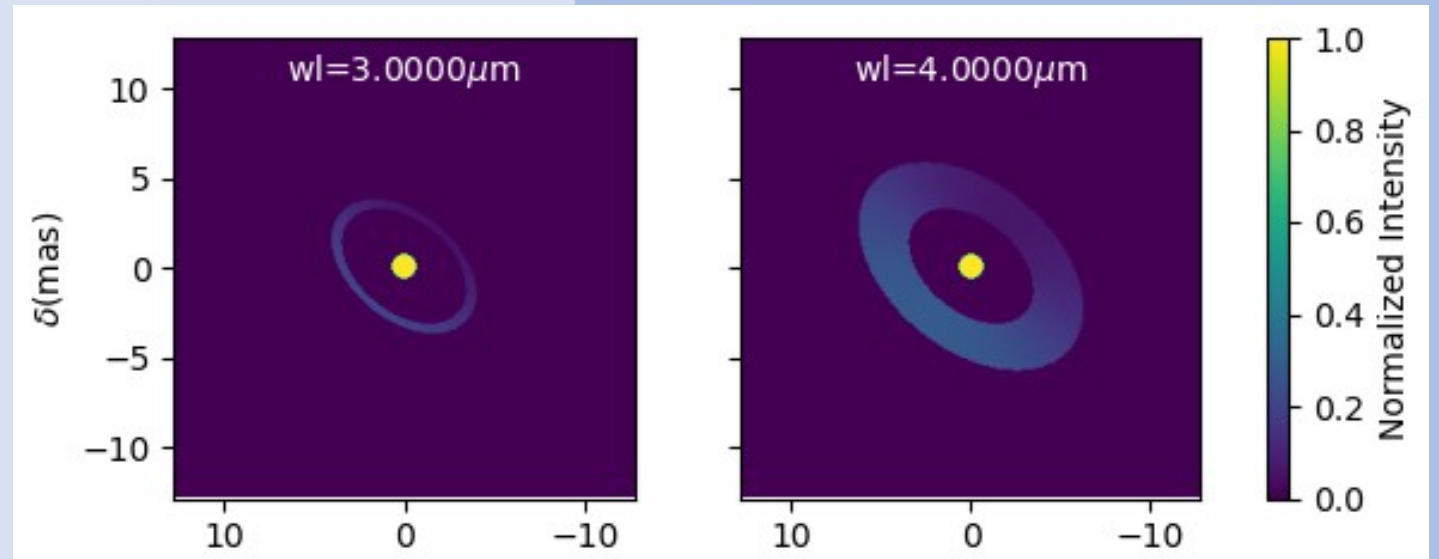
```
import oimodeler as oim
```

```
star=oim.oimUD(d=1,f=oim.oimInterp("wl",wl=[3e-6,4e-6],values=[0.5,0.1]))
disk=oim.oimESKRing(din=8,dout=oim.oimInterp("wl",wl=[3e-6,4e-6],values=[9,14]),elong=1.5,skw=0.8,pa=50)
disk.params["skwPa"]=oim.oimParamLinker(disk.params["pa"],"add",90)
disk.params["f"]=oim.oimParamNorm(star.params["f"])
model=oim.oimModel(star,disk)
```

```
params=model.getFreeParameters()
params['c1_UD_f_interp1'].set(min=0.0,max=1)
params['c1_UD_f_interp2'].set(min=0.0,max=1)
params['c1_UD_d'].set(min=0,max=5,free=True)
params['c2_SKER_pa'].set(min=0.,max=180)
params['c2_SKER_elong'].set(min=1,max=3)
params['c2_SKER_din'].set(min=5,max=20.)
params['c2_SKER_skw'].set(min=0,max=1.)
params['c2_SKER_dout_interp1'].set(min=5.,max=30.)
params['c2_SKER_dout_interp2'].set(min=5.,max=30.)
```

```
fit=oim.oimFitterEmcee(files,model,nwalkers=30)
fit.prepare(init="random")
fit.run(nsteps=2000,progress=True)
```

```
figWalkers,axeWalkers=fit.walkersPlot()
figCorner,axeCorner=fit.cornerPlot(discard=1000)
median,err_l,err_u,err=fit.getResults(mode='median',discard=1000)
figSim,axSim=fit.simulator.plot(["VIS2DATA","T3PHI"])
figImg,axImg,im=model.showModel(256,0.1,wl=[wl[0],wl[-1]])
```





Chromatic model-fitting

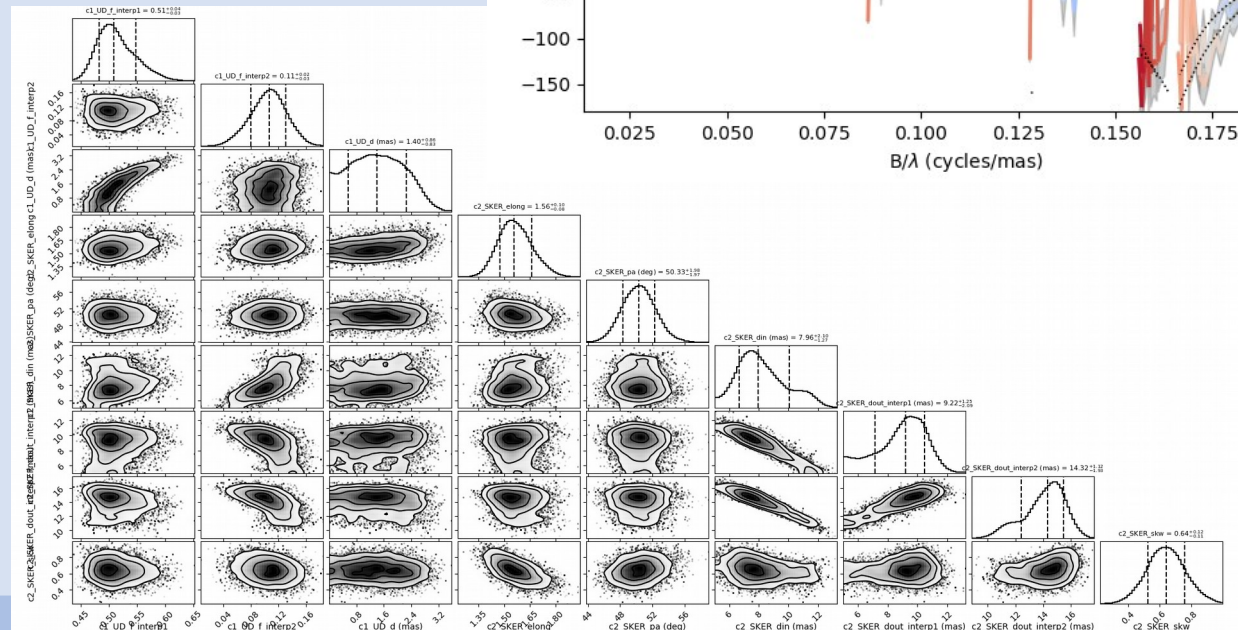
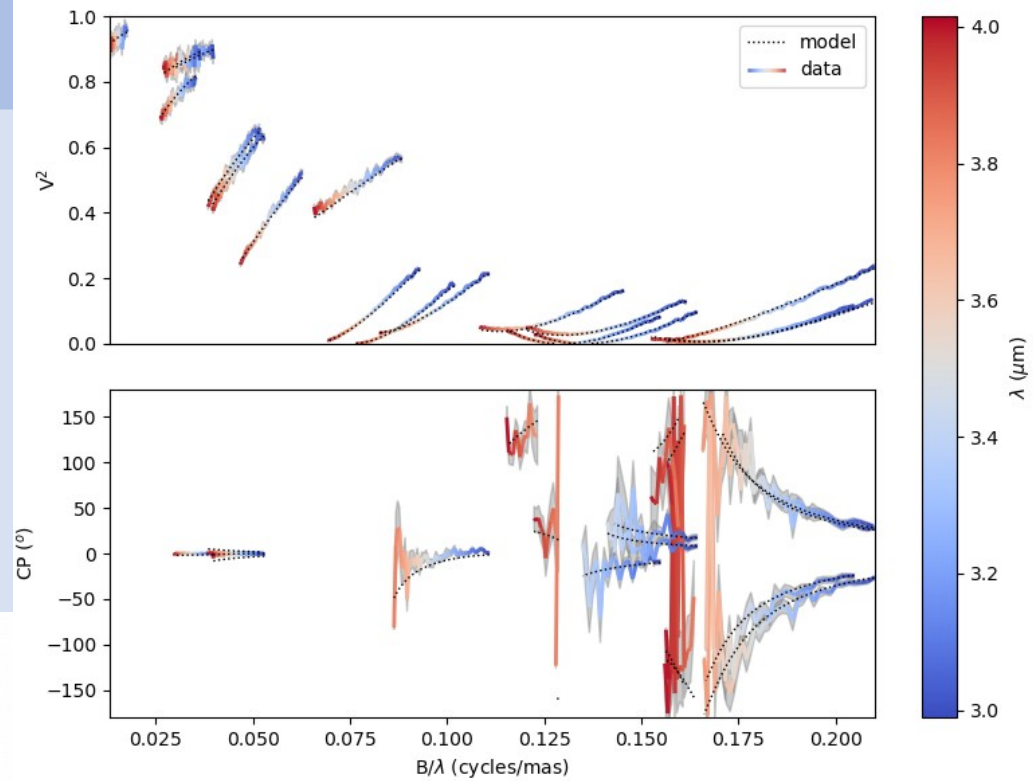
```
import oimodeler as oim
```

```
star=oim.oimUD(d=1,f=oim.oimInterp("wl",wl=[3e-6,4e-6],values=[0.5,0.1]))
disk=oim.oimESKRing(din=8,dout=oim.oimInterp("wl",wl=[3e-6,4e-6],values=[9,14]),elong=1.5,skw=0.8,pa=50)
disk.params["skwPa"]=oim.oimParamLinker(disk.params["pa"],"add",90)
disk.params["f"]=oim.oimParamNorm(star.params["f"])
model=oim.oimModel(star,disk)
```

```
params=model.getFreeParameters()
params['c1_UD_f_interp1'].set(min=0.0,max=1)
params['c1_UD_f_interp2'].set(min=0.0,max=1)
params['c1_UD_d'].set(min=0,max=5,free=True)
params['c2_SKER_pa'].set(min=0.,max=180)
params['c2_SKER_elong'].set(min=1,max=3)
params['c2_SKER_din'].set(min=5,max=20.)
params['c2_SKER_skw'].set(min=0,max=1.)
params['c2_SKER_dout_interp1'].set(min=5.,max=30.)
params['c2_SKER_dout_interp2'].set(min=5.,max=30.)
```

```
fit=oim.oimFitterEmcee(files,model,nwalkers=30)
fit.prepare(init="random")
fit.run(nsteps=2000,progress=True)
```

```
figWalkers,axeWalkers=fit.walkersPlot()
figCorner,axeCorner=fit.cornerPlot(discard=1000)
median,err_l,err_u,err=fit.getResults(mode='median',discard=1000)
figSim,axSim=fit.simulator.plot(["VIS2DATA","T3PHI"])
figImg,axImg,im=model.showModel(256,0.1,wl=[wl[0],wl[-1]])
```



Creating new Fourier components

```
class oimBox(oim.oimComponentFourier):
```

```
    name="2D Box"
```

```
    shortname = "BOX"
```

```
    def __init__(self, **kwargs):
```

```
        super().__init__(**kwargs)
```

```
        self.params["dx"]=oim.oimParam(name="dx", value=1,description="Size in x",unit=u.mas)
```

```
        self.params["dy"]=oim.oimParam(name="dy", value=1,description="Size in y",unit=u.mas)
```

```
        self._eval(**kwargs)
```

```
    def _visFunction(self,ucoord,vcoord,rho,wl,t):
```

```
        x=self.params["dx"](wl,t)*self.params["dx"].unit.to(u.rad)*ucoord
```

```
        y=self.params["dy"](wl,t)*self.params["dy"].unit.to(u.rad)*vcoord
```

```
        return np.sinc(x)*np.sinc(y)
```

```
    def _imageFunction(self,xx,yy,wl,t):
```

```
        return ((np.abs(xx)<=self.params["dx"](wl,t)/2) &
```

```
                (np.abs(yy)<=self.params["dy"](wl,t)/2)).astype(float)
```

Initialization function

- Call parent `__init__`
- Define parameters (oimParam)
- Call eval function

visibility function

- formula as function of u and v or ρ (and optionally λ and t)

Image function

- formula as function of x and y (and optionally λ and t)

Creating new Fourier components

```

class oimBox(oim.oimComponentFourier):
    name="2D Box"
    shortname = "BOX"

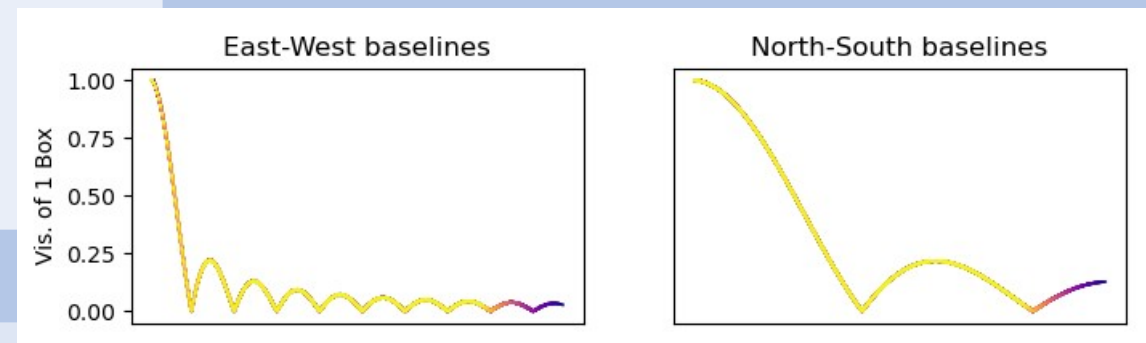
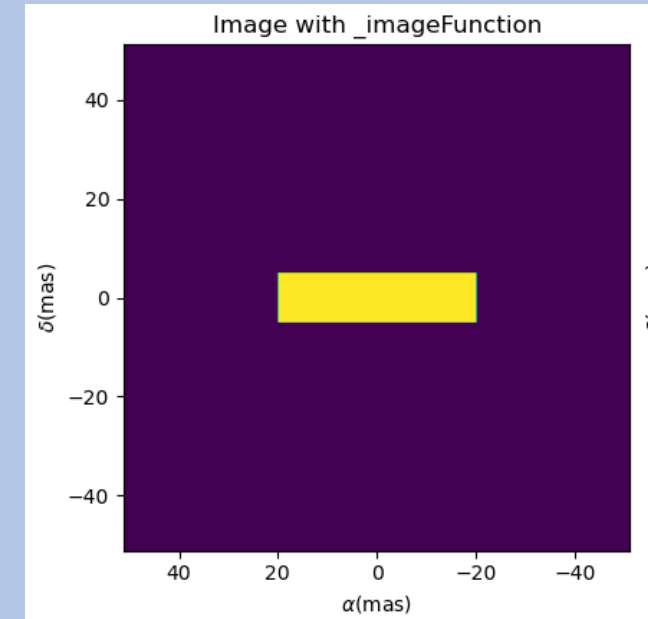
    def __init__(self, **kwargs):
        super().__init__(**kwargs)
        self.params["dx"]=oim.oimParam(name="dx", value=1,description="Size in x",unit=u.mas)
        self.params["dy"]=oim.oimParam(name="dy", value=1,description="Size in y",unit=u.mas)
        self._eval(**kwargs)

    def _visFunction(self,ucoord,vcoord,rho,wl,t):
        x=self.params["dx"](wl,t)*self.params["dx"].unit.to(u.rad)*ucoord
        y=self.params["dy"](wl,t)*self.params["dy"].unit.to(u.rad)*vcoord

        return np.sinc(x)*np.sinc(y)

    def _imageFunction(self,xx,yy,wl,t):
        return ((np.abs(xx)<=self.params["dx"](wl,t)/2) &
                (np.abs(yy)<=self.params["dy"](wl,t)/2)).astype(float)

```



```

b1=oimBox(dx=40,dy=10)
m1=oim.oimModel(b1)

```

```

m1.showModel(512,0.2,axe=ax[0],colorbar=False)
vis=np.abs(m.getComplexCoherentFlux(spfx,spfy)

```

Creating new Image components (external code)

```
class oimFastRotator(oim.oimComponentImage):
```

```
    name="Fast Rotator"
```

```
    shortname="FRot"
```

```
    def __init__(self, **kwargs):
```

```
        super().__init__(**kwargs)
```

```
        self.params["incl"]=oim.oimParam(name="incl",value=0,description="Inclination angle",unit=units.deg)
```

```
        self.params["rot"]=oim.oimParam(name="rot",value=0,description="Rotation Rate",unit=units.one)
```

```
        self.params["Tpole"]=oim.oimParam(name="Tpole",value=20000,description="Polar Temperature",unit=units.K)
```

```
        self.params["dpole"]=oim.oimParam(name="dplot",value=1,description="Polar diameter",unit=units.mas)
```

```
        self.params["beta"]=oim.oimParam(name="beta", value=0.25,description="Gravity Darkening Exponent",unit=units.one)
```

```
        self._t = np.array([0])
```

```
        self._wl = np.linspace(0.5e-6,15e-6,num=10)
```

```
        self._eval(**kwargs)
```

```
    def _internalImage(self):
```

```
        dim=self.params["dim"].value
```

```
        incl=self.params["incl"].value
```

```
        rot=self.params["rot"].value
```

```
        Tpole=self.params["Tpole"].value
```

```
        dpole=self.params["dpole"].value
```

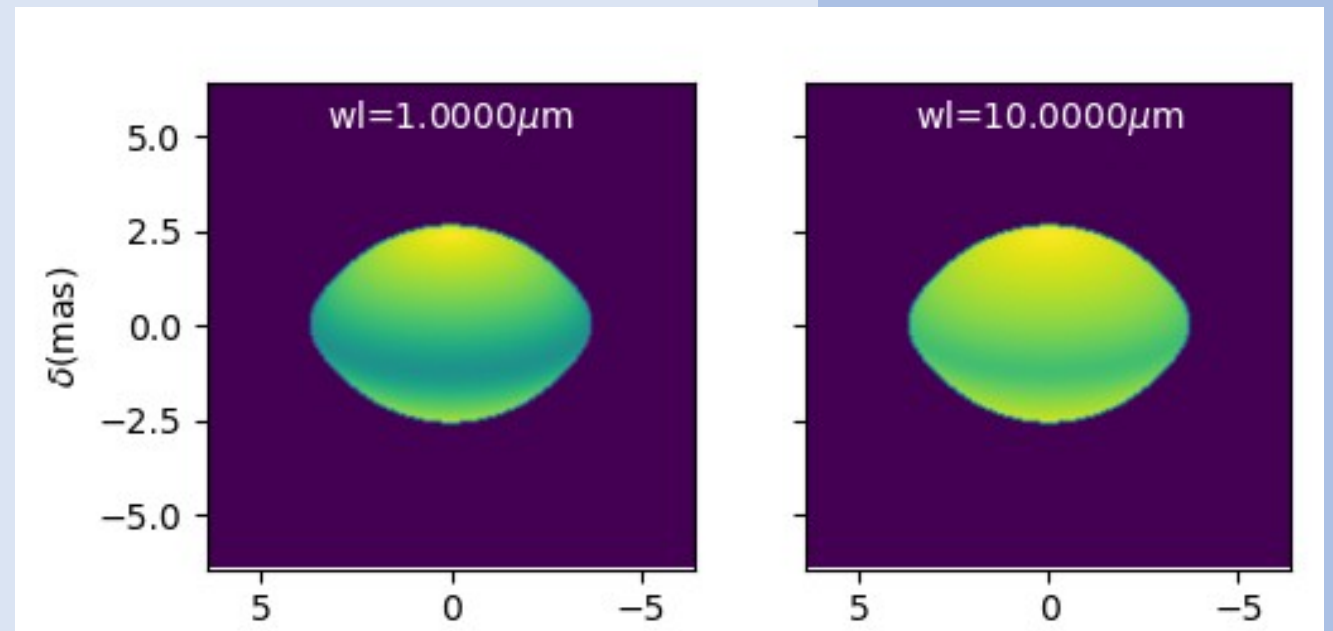
```
        beta=self.params["beta"].value
```

```
        im=fastRotator(dim,1.5,incl,rot,Tpole,self._wl,beta=beta)
```

```
        im=np.tile(np.moveaxis(im,-1,0)[None,:,:,:],(1,1,1,1))
```

```
        self._pixSize=1.5*dpole/dim*units.mas.to(units.rad)
```

```
        return im
```



Creating new Radial Profile components (analytical)

```
class oimExpRing(oim.oimComponentRadialProfile):
```

```
    name="A ring with a decreasing exponential profil"
```

```
    shortname = "E"
```

```
    elliptic=True
```

```
    def __init__(self,
```

```
        super().__ini
```

```
        self.params[
```

```
        self.params[
```

```
        self._dim=25
```

```
        self._t = np.array([0])
```

```
        self._wl = np.array([0.5e-6,1e-6])
```

```
        self._r = np.arange(0, self._dim)*0.05 #in mas
```

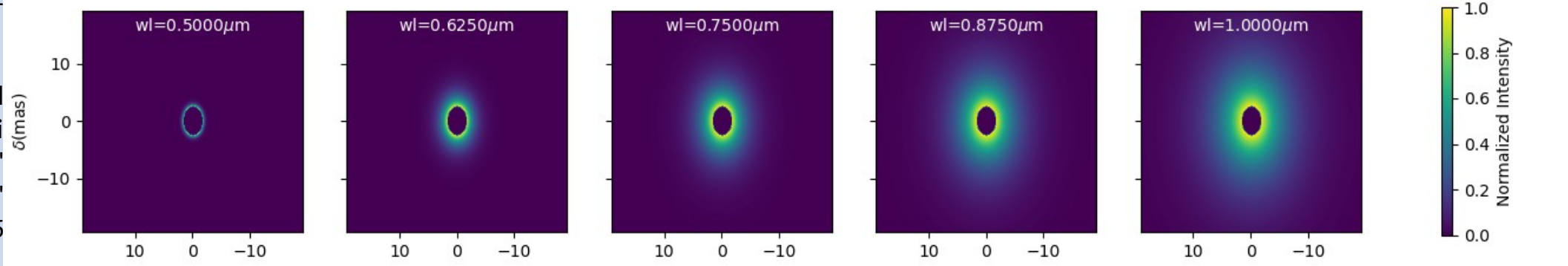
```
        self._eval(**kwargs)
```

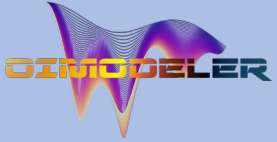
```
    def _radialProfileFunction(self,r,wl,t):
```

```
        r0=self.params["d"](wl,t)/2
```

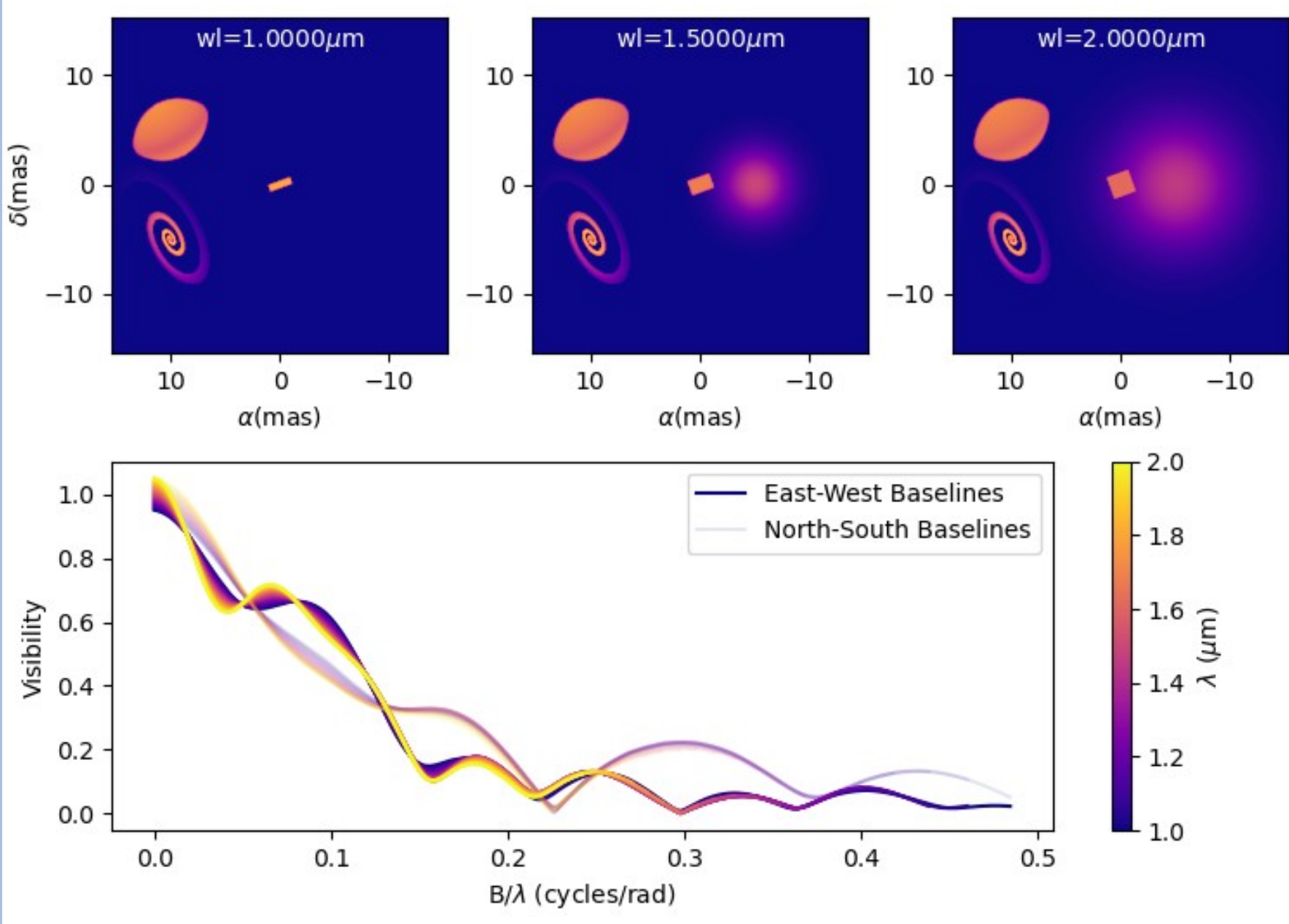
```
        fwhm=self.params["fwhm"](wl,t)
```

```
        return np.nan_to_num((r>r0)*np.exp(-0.692*np.divide(r-r0,fwhm)),nan=0)
```





Combining components



TODO in 2023 ...

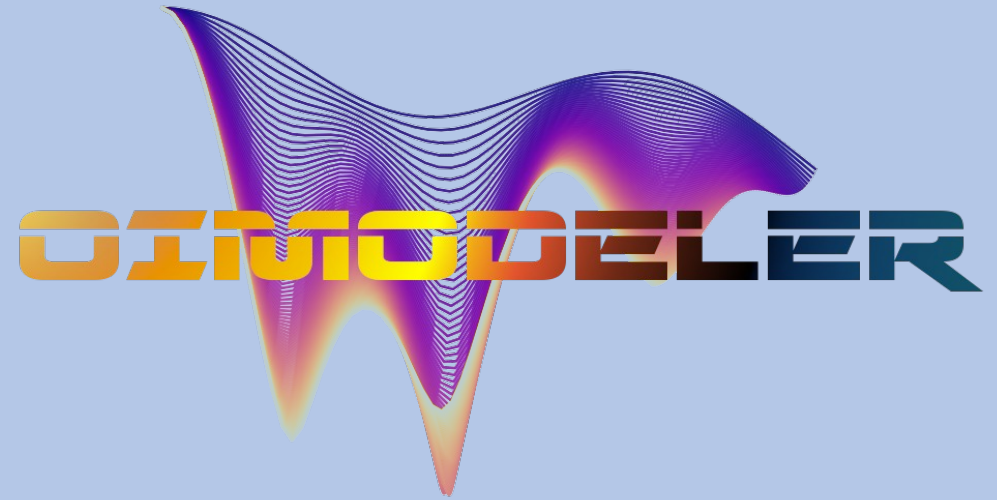
- 0 Implement missing basic features:
 - Create components from fits files and grid
 - Saving (model, fit)
 - Flux normalization (from 1 or ad-hoc to Jy)
 - Photometric and spectroscopic data

- 0 Add a few advanced features
 - models (rot. disk, DISCO+, AMHRA, grids?)
 - “intelligent” sampling for image based models
 - fitters (options, λ -by- λ , Imfit, chain, external constraints...)
 - filters (wl shift, smoothing, binning...)

- 0 Extensive test of the code
 - Unitary tests for all models and features
 - Tests Simulated data (chromatic + time-dependent)
 - Real data from all known instruments

- 0 Start working on optimization
 - Parallelization (model & fitter)
 - FFT & Hankel algorithms
 - Data optimization

- 0 Documentation & project management (GIT...)



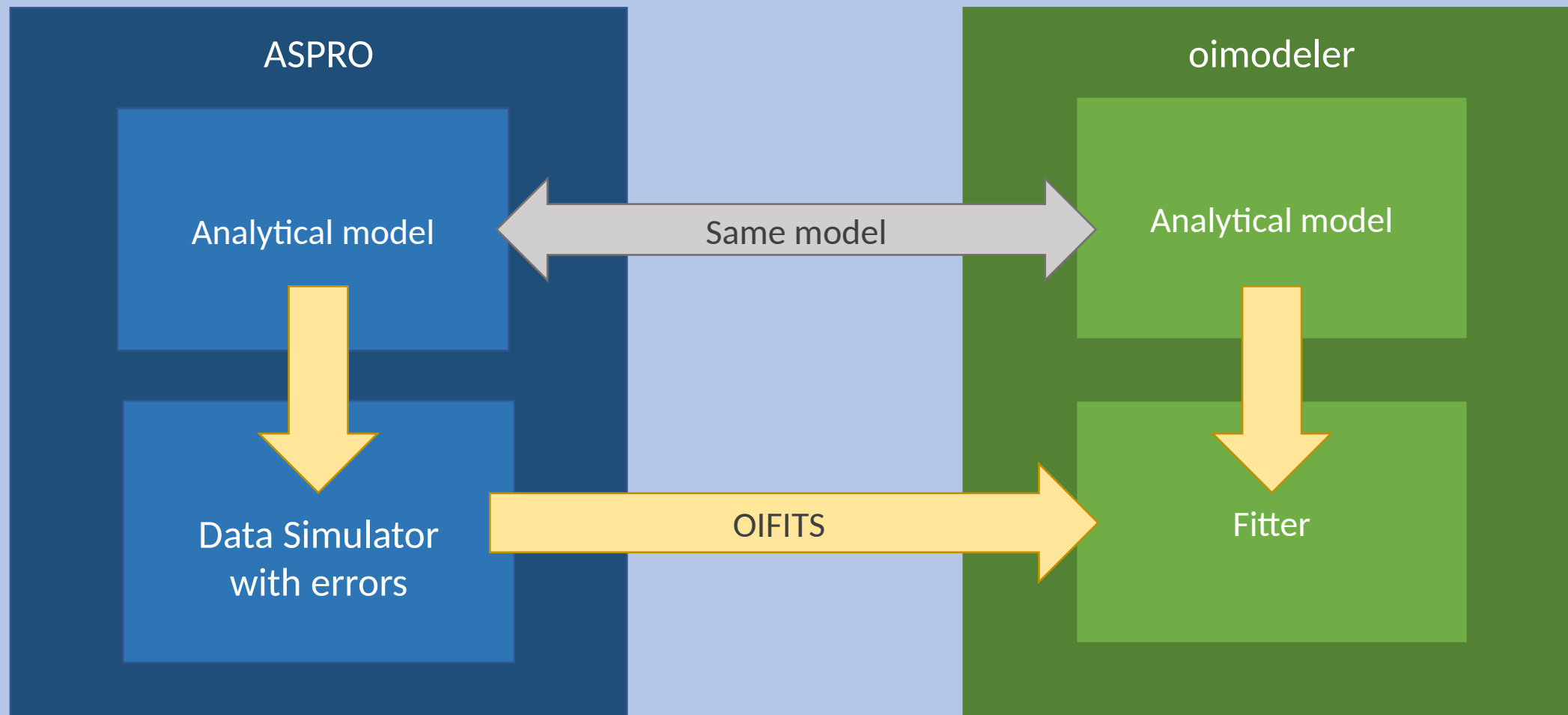
Oimodeler Team is building up...

Anthony Meilland, Jozsef Varga, Alexis Matter, Marten Scheuck, Armando Domiciano de Souza ...

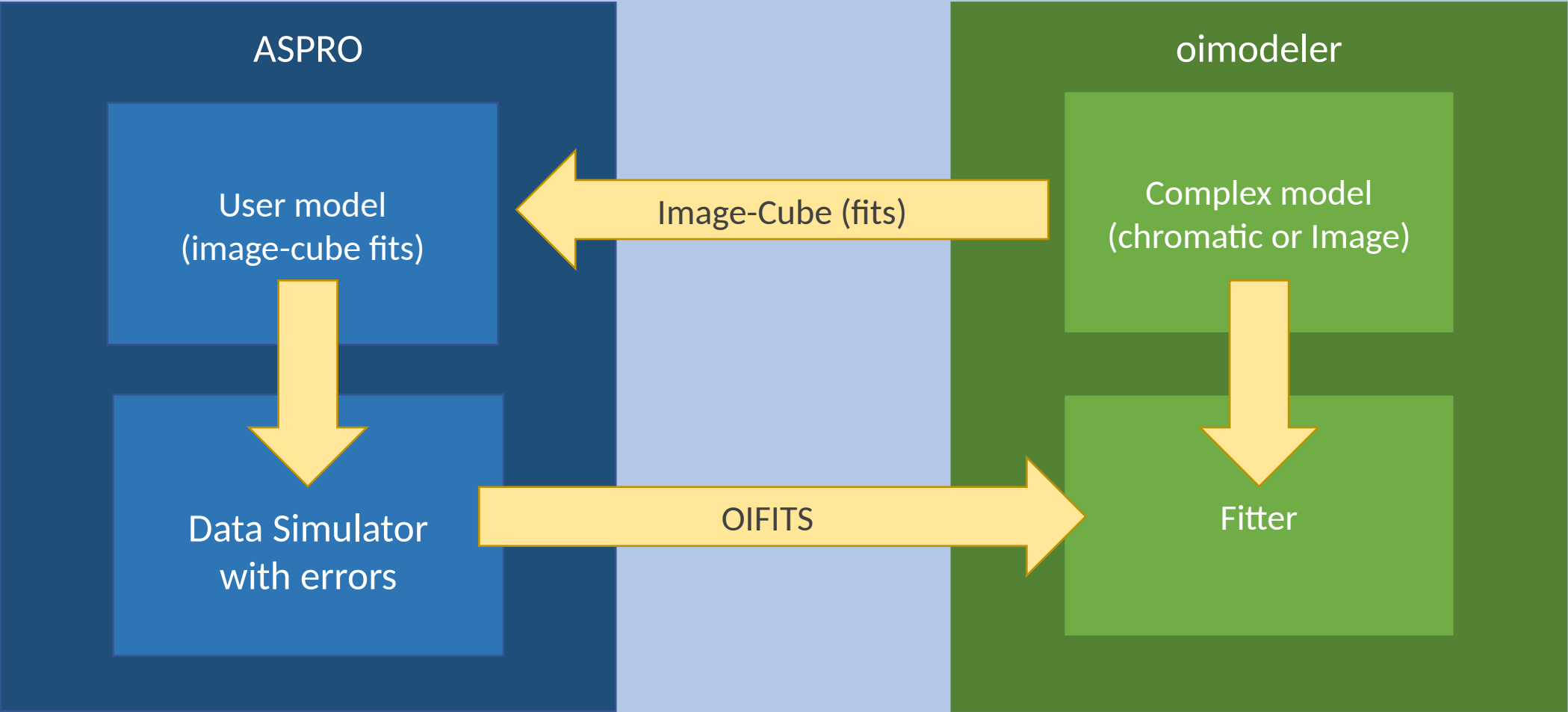
master and PhD students in Nice, Leiden and Heidelberg ...

+ ANR MASSIF

How did I produce simulated data for comparison?



How did I produce simulated data for comparison?



Model-fitting with a image-plan model

