GalFlex: The flexion simulation module

`User guide for using the python GalFlex module`

Justin Bird

January 12, 2015

1 GalFlex concepts

`GalFlex` is an open-source Python module that simulates second-order weak gravitational lensing effects (flexion), along with shear and convergence, on simulated source images. The module is meant to provide a quick, easy, and intuitive user interface for simulating images from input catalogs or directly from a file. By keeping dependencies strictly in Python, `GalFlex` forgoes a building process, and does not have a complicated structure. `GalFlex` operates in real-space, drawing a finely meshed matrix, and performing the second-order lensing effect based on input shear and flexion values. The images are represented in the form of a 2-dimensional, NxN `numpy` array. These allow greater transparency in the heart of the code, with operations running almost as quickly as the equivalent C code.

1.1 baseImage Class

Most of the functionality of the `GalFlex` module is in the `baseImage` class. This is the base class for defining the interface with which all GalFlex image classes access their shared methods and attributes.

Images are composed of a 2-d, row-contiguous numpy array of NxN shape and a pixel scale. Representing sky coordinates, this scale is defined by a parameter `xmax`, and the image x,y axes run from `-xmax` to `xmax`, with the origin centered at (0,0).

An Image can be constructed from either an existing NxN `numpy` array, or created from a number of predefined profiles (such as a Sersic profile) which inherit all the `baseImage` methods:

```python
import galflex

xmax = 20.0                          #Coordinates range from -20.0 to 20.0 (arcsec) on both axes
array = numpy.ones((100,100))        #Sample NxN array
gal1 = galflex.baseImage(array,xmax)#Create instance

xmax = 15.0
n = 1.3                              #Sersic index
re = 2.3                             #Half-light-radius (arcsec)
gal2 = galflex.Sersic(xmax,n,N=500,flux=1.e5,re=re) #Create Sersic intensity profile
```

```
# Copyright 2013 Justin Bird:
# http://physics.drexel.edu/~jbird/galflex/
#
# This file is part of GalFlex: The flexion simulation module
#
# GalFlex is free software: you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation, either version 3 of the License, or
# (at your option) any later version.
#
# GalFlex is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
# GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with GalFlex. If not, see <http://www.gnu.org/licenses/>
#


"""
Example code demonstrating GalFlex being used within GalSim's pipeline.


This is an example showing the method of integration of GalFlex with the GREAT3 code GalSim.

An base galaxy image is produced from GalSim, and drawn to a numpy NxN array with high precision.
    This is lensed by GalFlex, including 2nd-order flexion terms, and returned to GalSim for
    further manipulation.
"""

import galsim
import galflex

#Image properties
N = 500                                    # Subdivisions for image (pixels)
pixel_scale = 0.2                   # arcsec / pixel
xmax=N*pixel_scale/2.0       # Largest coordinate in image (from -xmax to xmax)

#Source properties
gal_flux = 1.e6                     # total counts on the image
gal_r0 = 2.7                        # scale radius (arcsec)

#Create source
gal = galsim.Exponential(flux=gal_flux, scale_radius=gal_r0)

#Draw intensity distribution to double precision.
#A finer mesh will result in a more accurate lensing.

#Define draw image
image = galsim.ImageD(N,N)

#Draw image
gal = gal.draw(image, dx=pixel_scale,normalization="flux").array

#Create GalFlex instance
gal = galflex.baseImage(gal,xmax)
```

Further details can be found in the API at the end of this reference guide.

The actual numpy array can be accessed by using the instance '.image' attribute. The individual elements in the '.image' attribute are indexed as gal.image[y,x], matching the standard numpy convention of [row,column].

1.2 Lensing routine

The galflex lensing function represents a second-order lensing effect on a 2-dimensional image. Operating in real space, the class method `.lens()` acts on a `baseImage` class NxN `numpy` matrix, and performs the second-order lensing effect based on input convergence, shear, and flexion values. For an accurate lensing application, an image must be finely meshed into the NxN array.

The process consists of a linear interpolation scheme, as follows:

- Convert the index value of the array to the appropriate coordinate value.
- Apply the second order Taylor expansion, using input lensing values, to the image coordinates:

$$x_b = A_{11}x_t + A_{12}y_t + 0.5(D_{111}x_t^2 + 2D_{112}x_t y_t + D_{221}y_t^2)$$
$$y_b = A_{22}y_t + A_{12}x_t + 0.5(D_{222}y_t^2 + 2D_{221}x_t y_t + D_{112}x_t^2)$$

- Each coordinate pixel will map to a new location on our foreground plane, not necessarily coincident upon the index coordinates of the background plane. As such, a weighting factor is introduced, where the intensity contributions from surrounding background cells are summed to the new coordinate pixel.
- For foreground pixels that would require background pixels as contributors, but those pixels are outside the range of the image provided, an approximation to the background sky intensity is assumed. This approach requires the galaxy image to be fully encompassed by the postage stamp domain.

Lensing parameters used are:

- κ convergence (in %)
- γ_1 shear1 (in %)
- γ_2 shear2 (in %)
- $\mathcal{F}_1$ spin-1 flexion1 (units of arcseconds^{-1})
- $\mathcal{F}_2$ spin-1 flexion2 (units of arcseconds^{-1})
- $\mathcal{G}_1$ spin-3 flexion1 (units of arcseconds^{-1})
- $\mathcal{G}_2$ spin-3 flexion2 (units of arcseconds^{-1})

These are specified in one of two ways.

- Each parameter can be set individually when calling the method [default=0.0]

```
gal.lens(kap=0.0, gamma1=0.0, gamma2=0.0, f1=0.0, f2=0.0, g1=0.0, g2=0.0)
```

- A list of 7 (float) parameters can be specified as: `params=list_name`:

```
params = [kap,gamma1,gamma2,f1,f2,g1,g2]
gal.lens(params = params)
```

The following is an example usage:

```python
import galflex

xmax = 20.0    #Largest image axis coordinate (arcsec)
n = 1.4        #Sersic profile index
gal1 = galflex.Sersic(xmax,n,N=500,flux=1.e5,re=3.4)  #Create 1rst Sersic image profile
#Convergence = 20%, shear1 = 25%, Spin-1 flexion1 = 0.001 arcsec^-1
params = [0.20,0.25,0.0,0.001,0.0,0.0,0.0]
gal1.lens(params=params)

gal2 = galflex.Sersic(xmax,1.6,N=500,flux=2.3e6,re=2.7)   #Create 2nd Sersic image profile
gal2.lens(kap=0.47, gamma1=-0.3, f1=0.02)
```

It is important to keep in mind that flexion signal strength is dependent on the scale of the image. If the magnitude of flexion multiplied by the image scale is comparable to 1 or greater, there will be unphysical multiple image effects. This can be quantified as:

$$(|\mathcal{F}| \quad \text{or} \quad |\mathcal{G}|) * \texttt{xmax} << 1$$

Also note:
The gravitational lensing of an image is a noncommutative process, so both shear and flexion must be added to an image using one routine.

2 API

2.1 Image handling

class `baseImage()`

Initialization:

```
gal = galflex.baseImage(array, xmax)
```

This is the base class for defining the interface with which all GalFlex image classes access their shared methods and attributes.
Images are composed of a 2-d, row-contiguous `numpy` array of NxN shape and a pixel scale. Representing sky coordinates, this scale is defined by a parameter `xmax`, and the image x,y axes run from `-xmax` to `xmax`, with the origin centered at (0,0). An image can be constructed from either an existing NxN `numpy` array, or created from a number of profiles (such as a Sersic profile) which inherit all the `baseImage` methods.
Multiple images may be added together by simple `+` operation, and multiplied by a scalar by simple `*` operation.

Class methods:

- `gal.copy()`

  Return a deep copy of instance.

- `gal.getFlux()`

  Return sum of pixel values of image. This may not be equal to the 'true' flux if the intensity profile is poorly sampled or is rapidly varying (i.e. high Sersic index).

- `gal.setFlux(new_flux)`

  Set flux of image to the specified value

- `gal.addGaussianNoise(sigma=1.0,mean=0.0)`

  Add Gaussian noise to the image, centered at `mean` with standard deviation `sigma`. Performs on a per-pixel basis.

- `gal.addPoissonNoise(sky_level)`

  Add Poisson noise to the image, based on the pixel fluxes. A sky_level can be specified per pixel, where the Poisson noise is derived from the sum of the image and the sky, and then the sky_level is subtracted from the image again.

- `gal.plot(title='Lensed image', units='Arcseconds')`

  Convenience method for simple viewing of images. Using `matplotlib.pyplot`, intended as a visual aid in comparing images. Accepts simple arguments for changing the graph title and coordinate units if necessary. Returns a `matplotlib figure` object. The user can also use the class method `gal.plot(...)` or the function `galflex.plot(instance,...)`

- `gal.writeFITS(filename,odir=None,clobber=True)`

  Writes image to a FITS file using `pyfits`. This can be called directly as a function: `galflex.writeFITS(image,...)` or as a `baseImage` method. If called directly, accepts either a single instance or a list of instances as the first argument.
  Automatically creates a directory 'output' in the run directory where the FITS file is saved. If `odir` is specified, that directory will replace 'output'.

  Setting `clobber=True` when `filename` is given will silently overwrite existing files.

- `gal.add(*args)`

  Another way for adding images together. Accepts `baseImage` class instances. Takes arguments in comma format, i.e. `gal.add(img1,img2,img3...)` or list format.

- `gal.lens(kap=0.0,gamma1=0.0,gamma2=0.0,f1=0.0,f2=0.0,g1=0.0,g2=0.0,cx=0.0,cy=0.0,**kwargs)`

  Performs lensing on an input image. Can either specify parameters individually when calling, or provide a list `params=[]` of 7 lensing parameters: convergence, shear1, shear2, F-flexion1, F-flexion2, G-flexion1, G-flexion2 (flexion in inverse arcseconds). Accepts positional arguments for expanding about the center point of the galaxy. See source file for complete details.

- `gal.shift(cx,cy)`

  Shifts the entire image according to entered coordinates.

- `gal.SISlens(einRad,rc,beta)`

  Performs a fully analytic gravitational lensing by a singular isothermal sphere (SIS) lens profile. This functionality is separate from the lensing by other GalFlex halo profiles, which only simulate distortion to a second order Taylor expansion of the deflection angle.

class `Sersic(baseImage)`

Initialization:

```
gal = galflex.Sersic(xmax, n, N=500, flux=1.0, q=1.0, phi=0.0, cx=0.0,cy=0.0, **kwargs)
```

Initialization class for a Sersic profile baseImage. Inherits all baseImage methods. Supports an elliptical Sersic intensity profile of form:

$$I(r)\ e^{-b_n((r/r_e)^{1/n}-1)}$$

or

$$I(r)\ e^{-(r/r_0)^{1/n}}$$

depending on input arguments, with:

`xmax` - Largest coordinate value along an array axis which runs from: (-xmax) to (xmax).

`n` - The exponent of intensity profile.

`N` - Integer length of NxN numpy array. [default=500]

`flux` - Total photon flux in image. [default=1.0]

`q` - Axis ratio of galaxy image (0.0 to 1.0), defined by x:(y*q). [default=1.0]

`phi` - Orientation angle of galaxy image, in radians. [default=0.0]

`cx` - Centroid offset in x-direction, units of xmax [default=0.0]

`cy` - Centroid offset in y-direction, units of xmax [default=0.0]

User must also specify one of the following after positional arguments:

`re` - Effective radius of the isophote containing half the total flux (half-light-radius).

`r0` - Scale radius where the intensity drops to 1/e the central peak value of the spherical profile.

b_n as a variable that depends on n, and can be modeled in the range $0.5 < n < 8$ as $b_n = 1.992n - 0.327$. r is defined as $r = \sqrt{x^2 + (y * q)^2}$.

The intensity profile takes on a few familiar forms at certain exponent values:

- n = 0.5 specifies a Gaussian profile
- n = 1.0 specifies an exponential profile
- n = 4.0 specifies a deVaucoulers profile

2.2 Lensing halos

---

class `ellipticalHalo()`

Initialization:

```
halo = galflex.ellipticalHalo(einRad, exponent=0.5, softening=0.0, q_lens=1.0, beta=0.0)
```

This is a class for a softened, elliptical, power-law potential halo profile.
Physical properties of the halo (such as mass, concentration, etc.) are contained within the Einstein radius term, so these are not specified. The NFWlens class, however, does use the physical properties of the lens directly.

The potential is of the form:

$$\phi = \theta_E \sqrt{s^2 + x^2 + (y/q)^2}^{\alpha}$$

with the (float) parameters specifying

`einRad` (θ_E) - Einstein radius of the lensing cluster (in arcseconds).

`exponent` (α) - Power of the potential. Defined between 0.0 < exponent < 1.0 [default=0.5 for SIS]

`softening` (s) - Softening term for lensing potential (arcsec). [default=0.0]

`q_lens` (q) - Axis ratio of the projected equipotentials of the lens, defined by x:(y/q). Must be defined between 0.0 < q <= 1.0 [default=1.0]

`beta` - Standard rotation angle of the lens profile (radians). [default=0.0]

Class methods:

- `params = halo.getParams(**kwargs)`

  Calculates and returns the lensing values `params =` $[\kappa, \gamma_1, \gamma_2, \mathcal{F}_1, \mathcal{F}_2, \mathcal{G}_1, \mathcal{G}_2]$ as a list of 7 values with flexion in units of inverse arcseconds, at a specified position around the lens.
  Must specify either a pair of x,y coordinates, or a radial distance and position angle.

  - `xc=` Position in units of the image coordinates (usually arcsec).
  - `yc=`

  –or–

  - `rc=` The radial distance from halo center (usually arcsec)
  - `angle=` Position angle around lens from x-axis
  - `angle_unit=` The units of angle as a string ('degrees' or 'radians')

---

class `NFWlens()`

Initialization:

```
halo = galflex.NFWlens(mass,conc,z,omega_m=0.3,omega_lam=0.7,omega_r=0.0,omega_c=0.0)
```

Class for spherical NFW halos. Can compute the lensing fields convergence, shear, and flexion of a NFW halo of given mass, concentration, redshift, and cosmology parameters. No mass-concentration relation is employed.

Based on the GREAT3 GalSim NFW halo profile.

The cosmology to compute angular diameter distances can be set by providing Ω_m, Ω_Λ, Ω_r, Ω_c. By default, $\Omega_m = 0.3$, $\Omega_\Lambda = 0.7$, all else = 0.0.

Initialization parameters:

`mass` - Mass of halo defined using a spherical overdensity of 200 times the critical density of the universe, in units of M_solar/h.

`conc` - Concentration parameter of halo, i.e., ratio of virial radius to NFW scale radius.

`z` - Redshift of the halo.

`omega_m` - Ω_{matter} [default=0.3]

`omega_lam` - Ω_Λ [default=0.7]

`omega_r` - $\Omega_{\text{radiation}}$ [default=0.0]

`omega_c` - $\Omega_{\text{curvature}}$ [default=0.0]

Class methods:

- `halo.getKs(z_s)`

  Returns lensing strength as a utility. Takes the redshift of the source.

- `halo.Da(z,z_ref=0.0)`

  Used internally, but can be used as a utility. Computes angular diameter distance between two redshifts in units of c/H0. In order to get the distance in Mpc/h, multiply by 3000. Takes 2 redshifts.

- `halo.getConvergence(r,phi,z_s)`

  Calculates convergence field of the halo at a specified radial position (arcseconds), assumed to be post-lensing, and an angular position around the lens (in radians). Also takes the redshift of the source. Returns κ

- `halo.getShear(r,phi,z_s)`

  Calculate shear of halo at specified positions (not reduced). Returns γ_1, γ_2

- `halo.getFlexion(r,phi,z_s)`

  Calculate flexion of halo at specified positions (units of inverse length, not reduced). Returns $\mathcal{F}_1, \mathcal{F}_2, \mathcal{G}_1, \mathcal{G}_2$

- `halo.getParams(r,phi,z_s)`

  Calculate convergence, shear, and flexion of halo at specified positions, return as a list of 7 parameters in order $[\kappa, \gamma_1, \gamma_2, \mathcal{F}_1, \mathcal{F}_2, \mathcal{G}_1, \mathcal{G}_2]$.

---

class `SIS()`

Initialization:

```
halo = galflex.SIS(einRad)
```

Class for a singular isothermal sphere halo potential profile.

Physical properties of the halo (such as mass, concentration, etc.) are contained within the Einstein radius term, so these are not specified. The NFWlens class does use the physical properties of the lens directly.

The spherically symmetric potential is of form:

$$\phi = \theta_E \theta$$

Class methods:

- `params = halo.getParams(**kwargs)`

  Calculates and returns the lensing values `params =` $[\kappa, \gamma_1, \gamma_2, \mathcal{F}_1, \mathcal{F}_2, \mathcal{G}_1, \mathcal{G}_2]$ as a list of 7 values with flexion in units of inverse arcseconds, at a specified position around the lens.
  Must specify either a pair of x,y coordinates, or a radial distance and position angle.

  - `xc=` Position in units of the image coordinates (usually arcsec).
  - `yc=`

  –or–

  - `rc=` The radial distance from halo center (usually arcsec)
  - `angle=` Position angle around lens from x-axis
  - `angle_unit=` The units of angle as a string ('degrees' or 'radians')

2.3 I/O support

---

function `writeFITS(arg, filename, odir=None, clobber=True)`

Call signature:

```
galflex.writeFITS(arg, filename, odir=None, clobber=True)
```

Function that will write images to a FITS file using `pyfits`. This can be called directly as a function: `galflex.writeFITS(image,...)` or as a `baseImage` method. If called directly, accepts either a single instance or a list of instances as the first argument.

Automatically creates a directory 'output' in the run directory where the FITS file is saved. If `odir` is specified, that directory will replace 'output'.

Setting `clobber=True` when `filename` is given will silently overwrite existing files.

---

function `inputParams(filename)`

Call signature:

```
params = galflex.inputParams(filename)
```

Inputs the seven lensing parameters from a specified file: $\kappa, \gamma_1, \gamma_2, \mathcal{F}_1, \mathcal{F}_2, \mathcal{G}_1, \mathcal{G}_2$ with flexion in units of inverse length.
The (float) parameters should be listed one per line in a simple text file. Function returns a list of the 7 parameters in the same order.

---

function `outputParams(params, filename)`

Call signature:

```
params = galflex.outputParams(params,filename)
```

Ouputs the seven lensing parameters to a specified file: $\kappa, \gamma_1, \gamma_2, \mathcal{F}_1, \mathcal{F}_2, \mathcal{G}_1, \mathcal{G}_2$ with flexion in units of inverse length.
The (float) parameters will be listed one per line in a simple text file, and will by default create a directory "output" where the simple text file is saved.

---

function `plot(img, xmax, title='Lensed image', units='Arcseconds')`

Call signature:

```
fig = galflex.plot(Img, xmax, title='Lensed Image', units='Arcsec')
```

Convenience method for simple viewing of images. Using `matplotlib.pyplot`, intended as a visual aid in comparing images. Accepts simple arguments for changing the graph title and coordinate units if necessary. Returns a `matplotlib.pyplot figure()` object. The user can also use the class method `gal.plot(...)`.