## BAGLE

Release 0.1

## Niranjan Bhatia, Jessica Lu

## CONTENTS

1 BAGLE TUTORIAL ..... 3
1.1 PSPL Model - No Parallax ..... 3
1.2 PSPL Model - With parallax ..... 4
1.3 PSPL Model From Belokurov and Evans 2002 (Figure 1) ..... 4
2 Overview of bagle.model ..... 7
2.1 Example ..... 7
2.2 Point source, point lens, photometry only: ..... 8
2.3 Point source, point lens, photometry and astrometry: ..... 8
2.4 Point source, point lens, astrometry only ..... 9
2.5 Point soruce, binary lens, photometry only ..... 9
2.6 Point source, binary lens, photometry and astrometry ..... 9
2.7 Binary source, point lens, photometry and only ..... 10
2.8 Binary source, point lens, photometry and astrometry ..... 10
2.9 Finite source, point lens, photometry and astrometry (broken) ..... 10
3 Developers ..... 11
3.1 Class Families ..... 12
3.1.1 Model Class Family ..... 12
3.1.2 Data Class Family ..... 12
3.1.3 Parallax Class Family ..... 12
3.1.4 Parameterization Class Family ..... 12
3.1.5 Making a New Model ..... 14
4 Instantiable Model Classes ..... 15
4.1 PSPL Model Classes ..... 15
4.1.1 PSPL ..... 15
4.1.2 PSPL Parallax ..... 17
4.1.3 PSPL_parallax2 / PSPL_multiphot_parallax ..... 25
4.1.4 PSPL_phot ..... 39
4.1.5 PSPL_phot_parallax / PSPL_phot_multiphot_parallax ..... 42
4.1.6 PSPL Phot parallax with GP ..... 46
4.1.7 PSPL Phot, no parallax with GP ..... 56
4.1.8 PSPL PhotAstrom, parallax with GP ..... 60
4.1.9 PSPL PhotAstrom, no parallax with GP ..... 80
4.2 PSBL User Classes ..... 85
4.2.1 PSBL ..... 85
4.2.2 PSBL Parallax ..... 96
4.2.3 PSBL_Phot ..... 120
4.2.4 PSBL_phot_parallax ..... 125
4.2.5 PSBL PhotAstrom, no parallax with GP ..... 131
4.2.6 PSBL PhotAstrom, parallax with GP ..... 142
4.2.7 PSBL Phot, no parallax with GP ..... 154
4.2.8 PSBL Phot, parallax with GP ..... 160
4.3 FSPL User Class ..... 166
4.3.1 FSPL_parallax ..... 166
5 POINT SOURCE POINT LENS (PSPL) CLASSES ..... 171
5.1 Parameterization Class Family ..... 171
5.1.1 PSPL Models ..... 171
5.1.2 GP Models ..... 180
5.2 Data Class Family ..... 184
5.3 GP Class Family ..... 187
5.4 Parallax Class Family ..... 191
6 POINT SOURCE BINARY LENS (PSBL) CLASSES ..... 201
6.1 Data Class Family ..... 201
6.2 Parallax Class Family - PSBL ..... 213
6.3 Parameterization Class Family - PSBL ..... 216
6.3.1 PSBL Models ..... 216
6.3.2 GP Models ..... 224
7 BINARY SOURCE POINT LENS (BSPL) CLASSES ..... 225
7.1 Parameterization Class Family - BSPL ..... 225
7.1.1 BSPL Models ..... 225
7.1.2 GP Models ..... 230
7.2 Data Class Family ..... 236
7.3 Parallax Class Family ..... 245
8 FSPL(Finite-Source Point Lens) Models - NOT DONE YET... place holders ..... 249
8.1 FSPL Classes ..... 249
8.2 FSPL Limb Classes ..... 260
9 GENERAL USE AND SHARED FUNCTIONS ..... 265
9.1 Shared Functions ..... 265
10 Model Fitter ..... 269
10.1 PSPL_Solver ..... 269
10.2 Prior Generators ..... 286
10.2.1 Parameters ..... 288
10.3 GENERAL USE AND SHARED FUNCTIONS ..... 289
10.3.1 Shared Functions ..... 289
Index ..... 297

BAGLE allows modeling of gravitational microlensing events both photometrically and astrometrically. Supported microlensing models include:

- PSPL: point-source, point-lens with parallax
- PSBL: point-source, binary-lens
- FSPL: finite-source, point-lens (currently testing further)

All models support fitting data with single or multi-band photometry only, astrometry only, or joint fitting of photometry and astrometry (recommended).

## CHAPTER

ONE

## BAGLE TUTORIAL

To learn to use BAGLE models, make microlensing events, and make photometric or astrometric plots, we have created a Jupyter Notebook tutorial.

### 1.1 PSPL Model - No Parallax

The first step in the tutorial is to generate a PSPL model with no parallax and do the following:

- Get amplification of event over time
- Plot the astrometric shift over time
- Animate the microlensing event




### 1.2 PSPL Model - With parallax

The second step is to generate a PSPL model with parallax adding the $r a$ (right ascention of lens) and dec (declination of lens).

### 1.3 PSPL Model From Belokurov and Evans 2002 (Figure 1)

Then a series of plots are made of the lens, source, and image positions on the sky for a set of microlensing event parameters previously published.



## OVERVIEW OF BAGLE.MODEL

model.py is a module that contains a set of classes and functions that allow the user to construct microlensing models. The available classes for instantiating a microlensing event are shown in the list below. See the API documentation for each class for details.

### 2.1 Example

To instantiate a model:

```
from bagle import model
mL}=10.0 # msu
t0 = 57000.00
xS0 = np.array([0.000, 0.000])
beta = 1.4 # mas
muS = np.array([8.0, 0.0])
muL = np.array([0.00, 0.00])
dL = 4000.0
dS = 8000.0
b_sff = [1.0] # one for each filter
mag_src = [19.0] # one for each filter
event1 = model.PSPL_PhotAstrom_noPar_Param1(mL,
    tQ, beta, dL, dL / dS,
    xSO[0], xSO[1], muL[0], muL[1],
    muS[0], muS[1],
    b_sff, mag_src)
# Get time range for event
t = np.arange(event1.t0 - 3000,
            event1.t0 + 3000, 1)
dt = t - event1.t0
# Quanties you can print
A = event1.get_amplification(t)
shift = event1.get_centroid_shift(t)
shift_amp = np.linalg.norm(shift, axis=1)
```

Note, each model class has a name that typically has a structure of
<ModelDataType>_<Parallax>_<GP>_<Parameterization>

For example, PSPL_Phot_noPar_Param2 has a data and model class type of PSPL_Phot, which contains a point-source, point-lens event with photometry only. The model has no parallax, no GP and uses parameterization \#2.

The complete list of instantiable model classes is:

### 2.2 Point source, point lens, photometry only:

- PSPL_Phot_noPar_Param1
- PSPL_Phot_noPar_Param2
- PSPL_Phot_Par_Param1
- PSPL_Phot_Par_Param2
- PSPL_Phot_Par_Param1_geoproj
- PSPL_Phot_noPar_GP_Param1
- PSPL_Phot_noPar_GP_Param2
- PSPL_Phot_Par_GP_Param1
- PSPL_Phot_Par_GP_Param1_2
- PSPL_Phot_Par_GP_Param2
- PSPL_Phot_Par_GP_Param2_2


### 2.3 Point source, point lens, photometry and astrometry:

- PSPL_PhotAstrom_noPar_Param1
- PSPL_PhotAstrom_noPar_Param2
- PSPL_PhotAstrom_noPar_Param3
- PSPL_PhotAstrom_noPar_Param4
- PSPL_PhotAstrom_Par_Param4_geoproj
- PSPL_PhotAstrom_Par_Param1
- PSPL_PhotAstrom_Par_Param2
- PSPL_PhotAstrom_Par_Param3
- PSPL_PhotAstrom_Par_Param4
- PSPL_PhotAstrom_Par_Param5
- PSPL_PhotAstrom_LumLens_Par_Param1
- PSPL_PhotAstrom_LumLens_Par_Param2
- PSPL_PhotAstrom_LumLens_Par_Param4
- PSPL_PhotAstrom_noPar_GP_Param1
- PSPL_PhotAstrom_noPar_GP_Param2
- PSPL_PhotAstrom_Par_GP_Param1
- PSPL_PhotAstrom_Par_GP_Param2
- PSPL_PhotAstrom_Par_GP_Param3
- PSPL_PhotAstrom_Par_GP_Param4
- PSPL_PhotAstrom_Par_LumLens_GP_Param1
- PSPL_PhotAstrom_Par_LumLens_GP_Param2
- PSPL_PhotAstrom_Par_LumLens_GP_Param3
- PSPL_PhotAstrom_Par_LumLens_GP_Param4


### 2.4 Point source, point lens, astrometry only

- PSPL_Astrom_Par_Param4
- PSPL_Astrom_Par_Param3


### 2.5 Point soruce, binary lens, photometry only

- PSBL_Phot_noPar_Param1
- PSBL_Phot_Par_Param1
- PSBL_Phot_noPar_GP_Param1
- PSBL_Phot_Par_GP_Param1


### 2.6 Point source, binary lens, photometry and astrometry

- PSBL_PhotAstrom_noPar_Param1
- PSBL_PhotAstrom_noPar_Param2
- PSBL_PhotAstrom_noPar_Param3
- PSBL_PhotAstrom_Par_Param1
- PSBL_PhotAstrom_Par_Param2
- PSBL_PhotAstrom_Par_Param3
- PSBL_PhotAstrom_Par_Param4
- PSBL_PhotAstrom_Par_Param5
- PSBL_PhotAstrom_noPar_GP_Param1
- PSBL_PhotAstrom_noPar_GP_Param2
- PSBL_PhotAstrom_Par_GP_Param1
- PSBL_PhotAstrom_Par_GP_Param2


### 2.7 Binary source, point lens, photometry and only

- BSPL_Phot_noPar_Param1
- BSPL_Phot_Par_Param1
- BSPL_Phot_noPar_GP_Param1
- BSPL_Phot_Par_GP_Param1


### 2.8 Binary source, point lens, photometry and astrometry

- BSPL_PhotAstrom_noPar_Param1
- BSPL_PhotAstrom_noPar_Param2
- BSPL_PhotAstrom_noPar_Param3
- BSPL_PhotAstrom_Par_Param1
- BSPL_PhotAstrom_Par_Param2
- BSPL_PhotAstrom_Par_Param3
- BSPL_PhotAstrom_noPar_GP_Param1
- BSPL_PhotAstrom_noPar_GP_Param2
- BSPL_PhotAstrom_noPar_GP_Param3
- BSPL_PhotAstrom_Par_GP_Param1
- BSPL_PhotAstrom_Par_GP_Param2
- BSPL_PhotAstrom_Par_GP_Param3


### 2.9 Finite source, point lens, photometry and astrometry (broken)

- FSPL_PhotAstrom_Par_Param1


## DEVELOPERS

Each model class i built up from a menu of different features by inheriting from multiple base classes, each from a different 'family' of related classes.

Each microlensing model must contain:

1) A class from the Data Class Family:

- PSPL - base class for all Data classes:
- PSPL_Phot
- PSPL_PhotAstrom
- PSPL_GP_Phot
- PSPL_GP_PhotAstrom

2) A class from the Parallax Class Family:

- ParallaxClassABC - base class for all Parallax classes:
- PSPL_noParallax
- PSPL_Parallax

3) A class from the GP Class Family: (optional)

- PSPL_GP - base class for all GP classes.

4) A class from the Parametrization Class Family:

- PSPL_Param - base class for all Param classes
- PSPL_PhotParam1
- PSPL_PhotParam2
- PSPL_PhotAstromParam1
- PSPL_PhotAstromParam2
- PSPL_PhotAstromParam3
- PSPL_PhotAstromParam4
- PSPL_PhotAstromParam5
- PSPL_GP_PhotParam1
- PSPL_GP_PhotParam2
- PSPL_GP_PhotAstromParam1
- PSPL_GP_PhotAstromParam2
- PSPL_GP_PhotAstromParam3
- PSPL_GP_PhotAstromParam4

There is a similar hierarchy for PSBL, etc.
For example, the PSPL_PhotAstrom_noPar_Param1 model is declared as:

```
class PSPL_PhotAstrom_noPar_Param1(ModelClassABC,
    PSPL_PhotAstrom,
    PSPL_noParallax,
    PSPL_PhotAstromParam1)
```


### 3.1 Class Families

### 3.1.1 Model Class Family

These are the classes that can be instantiated by the user. The base class is ModelClassABC.

### 3.1.2 Data Class Family

These classes inform the model of what type of data will be used by the model. If the model will be for photometry only, then a model with the PSPL_Phot class must be selected. These models have the words Phot in their names. If the model will be using photometry and astrometry data, then a model with the PSPL_PhotAstrom must be selected. These models have the words PhotAstrom in their names.

Data containing astrometry will generate a warning that astrometry data will not be used in the model when run through a model using PSPL_Phot. Data that does not contain astrometry run through a model using PSPL_PhotAstrom will generate a RuntimeError.
The base class is PSPL.

### 3.1.3 Parallax Class Family

These classes set whether the model uses parallax when calculating photometry, calculating astrometry, and fitting data. There are only two options for this class family, PSPL_noParallax and PSPL_Parallax. Models that do not have parallax have the words noPar in their names, while models that do contain parallax have the words Par in their names.
The base class is ParallaxClassABC.

### 3.1.4 Parameterization Class Family

These classes determine which physical parameters define the model. Currently this file supports one parameterization when using only photometry (Phot) and three parametrizations when using photometry and astrometery (PhotAstrom).
The base class is PSPL_Param.
The parameters for each parameterization are:
PhotParam1:
Point source point lens model for microlensing photometry only. This model includes the relative proper motion between the lens and the source. Parameters are reduced with the use of piRel

## Parameters:

t0, u0_amp, tE,
piE_E, piE_N,
b_sff, mag_src,
(ra, dec)

## PhotAstromParam1 :

Point Source Point Lens model for microlensing. This model includes proper motions of both the lens and source.

## Parameters:

$\mathrm{mL}, \mathrm{t} 0$, beta,
dL, dL_dS,
xS0_E, xS0_N,
muL_E, muL_N,
muS_E, muS_N,
b_sff, mag_src,
(ra, dec)

## PhotAstromParam2 :

Point Source Point Lens model for microlensing. This model includes proper motions of the source and the source position on the sky.

## Parameters:

t0, u0_amp, tE, thetaE, piS,
piE_E, piE_N,
xS0_E, xS0_N,
muS_E, muS_N,
b_sff, mag_src,
(ra, dec)

## PhotAstromParam3 :

Point Source Point Lens model for microlensing. This model includes proper motions of the source and the source position on the sky. Note it fits the baseline magnitude rather than the unmagnified source brightness.

## Parameters:

t0, u0_amp, tE, log10_thetaE, piS,
piE_E, piE_N,
xS0_E, xS0_N,
muS_E, muS_N,
b_sff, mag_base,
(ra, dec)
( $r a, d e c$ ) are only required if the model is created with a parallax class. More details about each parameterization can be found in the Parameterization Class docstring.

### 3.1.5 Making a New Model

Each model is, as described above, constructed by combining inheriting from different parent classes that contain the desired features for the model. Each model must have one class from each class family. In addition to this, there are several rules that must be followed when creating a new class.

1) The data class must match the parameterization class. For example, if the chosen data class is PSPL_Phot, then the parameter class must be PSPL_PhotParaml (or a different PhotParam in a future version). If the data class is PSPL_PhotAstrom, then the parameter class must be one of the classes with a PhotAstromParam.
2) Models are built using python's multiple inheritance feature. Therefore the order in which the parent classes are listed in the model class' definition matters. Parent classes to models should always be listed in the order:
a) ModelClassABC
b) Data Class
c) Parallax Class
d) Parameterization Class

If using the optional GP class, then the order is
a) ModelClassABC
b) GP Class
c) Data Class
d) Parallax Class
e) Parameterization Class
3) Each class must be given the @inheritdocstring decorator, and include the following commands in the model's
$\qquad$ __:

- a.super ().__init__(*args, **kwargs)
- startbases(self)
- checkconflicts(self)

Each of these performs the following function:

- super().__init__(*args, **kwargs): Inherits the __init__ from the Parameterization Class.
- startbases (self): Runs a start command on each parent class, giving each parent class a chance to run a set of functions upon instantiation.
- checkconflicts (self): Checks to confirm that the combination of parent classes in the model are valid.

4) Models should be named to reflect the parents classes used to construct it, as outlined in the above sections.

# INSTANTIABLE MODEL CLASSES 

### 4.1 PSPL Model Classes

### 4.1.1 PSPL

class model.PSPL_PhotAstrom_noPar_Param1(*args, **kwargs)
Bases: ModelClassABC, PSPL_PhotAstrom, PSPL_noParallax, PSPL_PhotAstromParam1
Helper class that provides a standard way to create an ABC using inheritance.

## Methods

| animate(tE, time_steps, frame_time, name, ...) | Produces animation of microlensing event. |
| :--- | :--- |
| get_amplification(t) | noParallax: Get the photometric amplification term <br> at a set of times,. |
| get_astrometry(t_obs[, ast_filt_idx]) | noParallax: Position of the observed source position <br> in arcsec. |
| get_astrometry_unlensed(t_obs) | noParallax: Get the astrometry of the source if the <br> lens didn't exist. |
| get_centroid_shift(t) | noParallax: Get the centroid shift (in mas) for a list <br> of observation times (in MJD). |
| get_lens_astrometry(t_obs) | Equation of motion for just the foreground lens. |
| get_resolved_amplification(t) | Get the photometric amplification term at a set of <br> times, t for both the plus and minus images. |
| get_resolved_astrometry(t_obs) | Get the $x, y$ astrometry for each of the two source <br> images, which we label plus and minus. |


| calc_piE_ecliptic |  |
| :--- | :--- |
| get_chi2_astrometry |  |
| get_chi2_photometry |  |
| get_lnL_constant |  |
| get_photometry |  |
| log_likely_astrometry |  |
| log_likely_astrometry_each |  |
| log_likely_photometry |  |
| log_likely_photometry_each |  |

animate( $t E$, time_steps, frame_time, name, size, zoom, astrometry)
Produces animation of microlensing event. This function takes the PSPL and makes an animation, the input variables are as follows

## Parameters

tE:
number of einstein crossings times before/after the peak you want the animation to plot
e.g tE $=2=>$ graph will go from -2 tE to 2 tE

## time_steps:

number of time steps before/after peak, so total number of time steps will be 2 times this value

## frame_time:

times in ms of each frame in the animation
name: string
the animation will be saved as name.html
size: list
[horizontal, vertical] cm's
zoom:
\# of einstein radii plotted in vertical direction

## get_amplification $(t)$

noParallax: Get the photometric amplification term at a set of times, t .

## Parameters

t:
Array of times in MJD.DDD

## get_astrometry (t_obs, ast_filt_idx=0)

noParallax: Position of the observed source position in arcsec.

```
get_astrometry_unlensed(t_obs)
```

noParallax: Get the astrometry of the source if the lens didn't exist.

## Returns

xS_unlensed
[numpy array, dtype=float, shape $=$ len(t_obs) $\times 2$ 2] The unlensed positions of the source in arcseconds.

## get_centroid_shift $(t)$

noParallax: Get the centroid shift (in mas) for a list of observation times (in MJD).

```
get_lens_astrometry(t_obs)
```

Equation of motion for just the foreground lens.

## Parameters

## t_obs

[array_like] Time (in MJD).

```
get_resolved_amplification(t)
```

Get the photometric amplification term at a set of times, $t$ for both the plus and minus images.

## Parameters

t:
Array of times in MJD.DDD
get_resolved_astrometry (t_obs)
Get the $\mathrm{x}, \mathrm{y}$ astrometry for each of the two source images, which we label plus and minus.

## Returns

[xS_plus, xS_minus]
[list of numpy arrays]

- xS_plus is the vector position of the plus image in arcsec
- xS_minus is the vector position of the plus image in arcsec


### 4.1.2 PSPL Parallax

class model.PSPL_PhotAstrom_Par_Param1(*args, **kwargs)
Bases: ModelClassABC, PSPL_PhotAstrom, PSPL_Parallax, PSPL_PhotAstromParam1
Helper class that provides a standard way to create an ABC using inheritance.

## Methods

| animate(tE, time_steps, frame_time, name, ...) | Produces animation of microlensing event. |
| :---: | :---: |
| calc_piE_ecliptic() | Parallax: Get piE_ecliptic |
| get_amplification(t) | Parallax: Get the photometric amplification term at a set of times, $t$. |
| get_astrometry(t_obs[, ast_filt_idx]) | Parallax: Get astrometry |
| get_astrometry_unlensed(t_obs) | Get the astrometry of the source if the lens didn't exist. |
| get_centroid_shift(t[, ast_filt_idx]) | Parallax: Get the centroid shift (in mas) for a list of observation times (in MJD). |
| get_geoproj_ast_params(t0par) |  |
| get_geoproj_params(t0par) |  |
| get_lens_astrometry(t_obs) | Parallax: Get lens astrometry |
| get_resolved_amplification(t) | Parallax: Get the photometric amplification term at a set of times, t for both the plus and minus images. |
| get_resolved_astrometry(t_obs) | Parallax: Get the x, y astrometry for each of the two source images, which we label plus and minus. |


| get_chi2_astrometry |  |
| :--- | :--- |
| get_chi2_photometry |  |
| get_lnL_constant |  |
| get_photometry |  |
| log_likely_astrometry |  |
| log_likely_astrometry_each |  |
| log_likely_photometry |  |
| log_likely_photometry_each |  |
| start |  |

animate ( $t$ E, time_steps, frame_time, name, size, zoom, astrometry)
Produces animation of microlensing event. This function takes the PSPL and makes an animation, the input variables are as follows

## Parameters

tE:
number of einstein crossings times before/after the peak you want the animation to plot
e.g tE $=2=>$ graph will go from -2 tE to 2 tE

## time_steps:

number of time steps before/after peak, so total number of time steps will be 2 times this value
frame_time:
times in ms of each frame in the animation
name: string
the animation will be saved as name.html
size: list
[horizontal, vertical] cm's
zoom:
\# of einstein radii plotted in vertical direction

```
calc_piE_ecliptic()
```

Parallax: Get piE_ecliptic

## get_amplification $(t)$

Parallax: Get the photometric amplification term at a set of times, $t$.

## Parameters

t:
Array of times in MJD.DDD
get_astrometry (t_obs, ast_filt_idx=0)
Parallax: Get astrometry
get_astrometry_unlensed (t_obs)
Get the astrometry of the source if the lens didn't exist.

## Returns

xS_unlensed
[numpy array, dtype=float, shape $=$ len(t_obs) $\times 2$ ] The unlensed positions of the source in arcseconds.

## get_centroid_shift ( $t$, ast_filt_idx=0)

Parallax: Get the centroid shift (in mas) for a list of observation times (in MJD).

## get_lens_astrometry (t_obs)

Parallax: Get lens astrometry
get_resolved_amplification ( $t$ )
Parallax: Get the photometric amplification term at a set of times, t for both the plus and minus images.

## Parameters

t:
Array of times in MJD.DDD
get_resolved_astrometry (t_obs)
Parallax: Get the $\mathrm{x}, \mathrm{y}$ astrometry for each of the two source images, which we label plus and minus.

## Returns

[xS_plus, xS_minus]
[list of numpy arrays]

- xS_plus is the vector position of the plus image.
- xS_minus is the vector position of the plus image.
class model.PSPL_PhotAstrom_LumLens_Par_Param1(*args, **kwargs)
Bases: ModelClassABC, PSPL_PhotAstrom, PSPL_Parallax_LumLens, PSPL_PhotAstromParam1
Helper class that provides a standard way to create an ABC using inheritance.


## Methods

| animate(tE, time_steps, frame_time, name, ...) | Produces animation of microlensing event. |
| :--- | :--- |
| calc_piE_ecliptic() | Parallax: Get piE_ecliptic |
| get_amplification(t) | Parallax: Get the photometric amplification term at a <br> set of times, t. |
| get_astrometry(t_obs[, ast_filt_idx]) | Parallax: Get astrometry |
| get_astrometry_unlensed(t_obs) | Get the astrometry of the source if the lens didn't ex- <br> ist. |
| get_centroid_shift(t[, ast_filt_idx]) | Parallax: Get the centroid shift (in mas) for a list of <br> observation times (in MJD). |
| get_geoproj_ast_params(t0par) |  |
| get_geoproj_params(t0par) | Parallax: Get lens astrometry <br> set of times, t for both the plus and minus images. |
| get_lens_astrometry(t_obs) | Parallax: Get the x, y astrometry for each of the two <br> source images, which we label plus and minus. |
| get_resolved_amplification(t) |  |


| get_chi2_astrometry |  |
| :--- | :--- |
| get_chi2_photometry |  |
| get_lnL_constant |  |
| get_photometry |  |
| log_likely_astrometry |  |
| log_likely_astrometry_each |  |
| log_likely_photometry |  |
| log_likely_photometry_each |  |
| start |  |

animate ( $t E$, time_steps, frame_time, name, size, zoom, astrometry)
Produces animation of microlensing event. This function takes the PSPL and makes an animation, the input variables are as follows

## Parameters

tE:
number of einstein crossings times before/after the peak you want the animation to plot
e.g $\mathrm{tE}=2=>$ graph will go from -2 tE to 2 tE
time_steps:
number of time steps before/after peak, so total number of time steps will be 2 times this value
frame_time:
times in ms of each frame in the animation
name: string
the animation will be saved as name.html
size: list
[horizontal, vertical] cm's
zoom:
\# of einstein radii plotted in vertical direction
calc_piE_ecliptic()
Parallax: Get piE_ecliptic
get_amplification $(t)$
Parallax: Get the photometric amplification term at a set of times, $t$.

## Parameters

t:
Array of times in MJD.DDD

```
get_astrometry(t_obs,ast_filt_idx=0)
```

Parallax: Get astrometry

```
get_astrometry_unlensed(t_obs)
```

Get the astrometry of the source if the lens didn't exist.

## Returns

xS_unlensed
[numpy array, dtype=float, shape $=$ len(t_obs) $\times 2$ ] The unlensed positions of the source in arcseconds.

```
get_centroid_shift(t, ast_filt_idx=0)
```

Parallax: Get the centroid shift (in mas) for a list of observation times (in MJD).

## get_lens_astrometry (t_obs)

Parallax: Get lens astrometry
get_resolved_amplification $(t)$
Parallax: Get the photometric amplification term at a set of times, t for both the plus and minus images.

## Parameters

t:
Array of times in MJD.DDD

## get_resolved_astrometry (t_obs)

Parallax: Get the x , y astrometry for each of the two source images, which we label plus and minus.

## Returns

[xS_plus, xS_minus]
[list of numpy arrays]

- xS_plus is the vector position of the plus image.
- xS_minus is the vector position of the plus image.
class model.PSPL_PhotAstrom_LumLens_Par_Param2 (*args, **kwargs)
Bases: ModelClassABC, PSPL_PhotAstrom, PSPL_Parallax_LumLens, PSPL_PhotAstromParam2
Helper class that provides a standard way to create an ABC using inheritance.

Methods

| animate(tE, time_steps, frame_time, name, ...) | Produces animation of microlensing event. |
| :--- | :--- |
| calc_piE_ecliptic() | Parallax: Get piE_ecliptic |
| get_amplification(t) | Parallax: Get the photometric amplification term at a <br> set of times, t. |
| get_astrometry(t_obs[, ast_filt_idx]) | Parallax: Get astrometry |
| get_astrometry_unlensed(t_obs) | Get the astrometry of the source if the lens didn't ex- <br> ist. |
| get_centroid_shift(t[, ast_filt_idx]) | Parallax: Get the centroid shift (in mas) for a list of <br> observation times (in MJD). |
| get_geoproj_ast_params(t0par) |  |
| get_geoproj_params(t0par) | Parallax: Get lens astrometry <br> Pet of times, Get the photometric amplification term at a a the plus and minus images. |
| get_lens_astrometry(t_obs) | Parallax: Get the x, y astrometry for each of the two <br> source images, which we label plus and minus. |
| get_resolved_amplification(t) |  |


| get_chi2_astrometry |  |
| :--- | :--- |
| get_chi2_photometry |  |
| get_lnL_constant |  |
| get_photometry |  |
| log_likely_astrometry |  |
| log_likely_astrometry_each |  |
| log_likely_photometry |  |
| log_likely_photometry_each |  |
| start |  |

animate ( $t E$, time_steps, frame_time, name, size, zoom, astrometry)
Produces animation of microlensing event. This function takes the PSPL and makes an animation, the input variables are as follows

## Parameters

## tE:

number of einstein crossings times before/after the peak you want the animation to plot
e.g tE $=2=>$ graph will go from -2 tE to 2 tE

## time_steps:

number of time steps before/after peak, so total number of time steps will be 2 times this value

## frame_time:

times in ms of each frame in the animation
name: string
the animation will be saved as name.html
size: list
[horizontal, vertical] cm's
zoom:
\# of einstein radii plotted in vertical direction
calc_piE_ecliptic()
Parallax: Get piE_ecliptic

```
get_amplification(t)
```

Parallax: Get the photometric amplification term at a set of times, t .

## Parameters

t:
Array of times in MJD.DDD
get_astrometry (t_obs, ast_filt_idx=0)
Parallax: Get astrometry
get_astrometry_unlensed (t_obs)
Get the astrometry of the source if the lens didn't exist.

## Returns

xS_unlensed
[numpy array, dtype=float, shape $=$ len(t_obs) $\times 2$ 2] The unlensed positions of the source in arcseconds.

```
get_centroid_shift(t, ast_fil_idx=0)
```

Parallax: Get the centroid shift (in mas) for a list of observation times (in MJD).

## get_lens_astrometry (t_obs)

Parallax: Get lens astrometry
get_resolved_amplification $(t)$
Parallax: Get the photometric amplification term at a set of times, t for both the plus and minus images.

## Parameters

t:
Array of times in MJD.DDD
get_resolved_astrometry (t_obs)
Parallax: Get the $\mathrm{x}, \mathrm{y}$ astrometry for each of the two source images, which we label plus and minus.

## Returns

## [xS_plus, xS_minus]

[list of numpy arrays]

- xS_plus is the vector position of the plus image.
- xS_minus is the vector position of the plus image.
class model.PSPL_PhotAstrom_LumLens_Par_Param4 (*args, **kwargs)
Bases: ModelClassABC, PSPL_PhotAstrom, PSPL_Parallax_LumLens, PSPL_PhotAstromParam4
Helper class that provides a standard way to create an ABC using inheritance.


## Methods

| animate(tE, time_steps, frame_time, name, ...) | Produces animation of microlensing event. |
| :--- | :--- |
| calc_piE_ecliptic() | Parallax: Get piE_ecliptic |
| get_amplification(t) | Parallax: Get the photometric amplification term at a <br> set of times, t. |
| get_astrometry(t_obs[, ast_filt_idx]) | Parallax: Get astrometry |
| get_astrometry_unlensed(t_obs) | Get the astrometry of the source if the lens didn't ex- <br> ist. |
| get_centroid_shift(t[, ast_filt_idx]) | Parallax: Get the centroid shift (in mas) for a list of <br> observation times (in MJD). |
| get_geoproj_ast_params(t0par) | Parallax: Get lens astrometry |
| get_geoproj_params(t0par) | Parallax: Get the photometric amplification term at a <br> set of times, t for both the plus and minus images. |
| get_lens_astrometry(t_obs) | Parallax: Get the x, y astrometry for each of the two <br> source images, which we label plus and minus. |
| get_resolved_amplification(t) |  |


| get_chi2_astrometry |  |
| :--- | :--- |
| get_chi2_photometry |  |
| get_lnL_constant |  |
| get_photometry |  |
| log_likely_astrometry |  |
| log_likely_astrometry_each |  |
| log_likely_photometry |  |
| log_likely_photometry_each |  |
| start |  |

animate ( $t$ E, time_steps, frame_time, name, size, zoom, astrometry)
Produces animation of microlensing event. This function takes the PSPL and makes an animation, the input variables are as follows

## Parameters

tE:
number of einstein crossings times before/after the peak you want the animation to plot
e.g tE $=2=>$ graph will go from -2 tE to 2 tE

## time_steps:

number of time steps before/after peak, so total number of time steps will be 2 times this value

## frame_time:

times in ms of each frame in the animation
name: string
the animation will be saved as name.html
size: list
[horizontal, vertical] cm's

## zoom:

\# of einstein radii plotted in vertical direction
calc_piE_ecliptic()
Parallax: Get piE_ecliptic
get_amplification $(t)$
Parallax: Get the photometric amplification term at a set of times, t .

## Parameters

t:
Array of times in MJD.DDD
get_astrometry (t_obs, ast_filt_idx=0)
Parallax: Get astrometry

## get_astrometry_unlensed (t_obs)

Get the astrometry of the source if the lens didn't exist.

## Returns

xS_unlensed
[numpy array, dtype=float, shape $=$ len(t_obs) x 2] The unlensed positions of the source in arcseconds.

## get_centroid_shift ( $t$, ast_filt_idx=0)

Parallax: Get the centroid shift (in mas) for a list of observation times (in MJD).

## get_lens_astrometry (t_obs)

Parallax: Get lens astrometry

## get_resolved_amplification ( $t$ )

Parallax: Get the photometric amplification term at a set of times, $t$ for both the plus and minus images.

## Parameters

t:
Array of times in MJD.DDD
get_resolved_astrometry (t_obs)
Parallax: Get the x , y astrometry for each of the two source images, which we label plus and minus.

## Returns

[xS_plus, xS_minus]
[list of numpy arrays]

- xS_plus is the vector position of the plus image.
- xS_minus is the vector position of the plus image.


### 4.1.3 PSPL_parallax2 / PSPL_multiphot_parallax

class model.PSPL_PhotAstrom_Par_Param2(*args, **kwargs)
Bases: ModelClassABC, PSPL_PhotAstrom, PSPL_Parallax, PSPL_PhotAstromParam2
Helper class that provides a standard way to create an ABC using inheritance.

## Methods

| animate(tE, time_steps, frame_time, name, ...) | Produces animation of microlensing event. |
| :--- | :--- |
| calc_piE_ecliptic() | Parallax: Get piE_ecliptic |
| get_amplification(t) | Parallax: Get the photometric amplification term at a <br> set of times, t. |
| get_astrometry(t_obs[, ast_filt_idx]) | Parallax: Get astrometry |
| get_astrometry_unlensed(t_obs) | Get the astrometry of the source if the lens didn't ex- <br> ist. |
| get_centroid_shift(t[, ast_filt_idx]) | Parallax: Get the centroid shift (in mas) for a list of <br> observation times (in MJD). |
| get_geoproj_ast_params(t0par) | Parallax: Get lens astrometry |
| get_geoproj_params(t0par) | Parallax: Get the photometric amplification term at a <br> set of times, t for both the plus and minus images. |
| get_lens_astrometry(t_obs) | Parallax: Get the x, y astrometry for each of the two <br> source images, which we label plus and minus. |
| get_resolved_amplification(t) |  |


| get_chi2_astrometry |  |
| :--- | :--- |
| get_chi2_photometry |  |
| get_lnL_constant |  |
| get_photometry |  |
| log_likely_astrometry |  |
| log_likely_astrometry_each |  |
| log_likely_photometry |  |
| log_likely_photometry_each |  |
| start |  |

animate ( $t E$, time_steps, frame_time, name, size, zoom, astrometry)
Produces animation of microlensing event. This function takes the PSPL and makes an animation, the input variables are as follows

## Parameters

tE:
number of einstein crossings times before/after the peak you want the animation to plot
e.g tE $=2$ => graph will go from -2 tE to 2 tE

## time_steps:

number of time steps before/after peak, so total number of time steps will be 2 times this value

## frame_time:

times in ms of each frame in the animation
name: string
the animation will be saved as name.html
size: list
[horizontal, vertical] cm's

## zoom:

\# of einstein radii plotted in vertical direction
calc_piE_ecliptic()
Parallax: Get piE_ecliptic
get_amplification $(t)$
Parallax: Get the photometric amplification term at a set of times, t .

## Parameters

t:
Array of times in MJD.DDD
get_astrometry (t_obs, ast_filt_idx=0)
Parallax: Get astrometry

```
get_astrometry_unlensed(t_obs)
```

Get the astrometry of the source if the lens didn't exist.

## Returns

xS_unlensed
[numpy array, dtype=float, shape $=$ len(t_obs) $\times 2$ ] The unlensed positions of the source in arcseconds.

```
get_centroid_shift(t, ast_filt_idx=0)
```

Parallax: Get the centroid shift (in mas) for a list of observation times (in MJD).

## get_lens_astrometry (t_obs)

Parallax: Get lens astrometry

## get_resolved_amplification ( $t$ )

Parallax: Get the photometric amplification term at a set of times, $t$ for both the plus and minus images.

## Parameters

t:
Array of times in MJD.DDD
get_resolved_astrometry (t_obs)
Parallax: Get the x , y astrometry for each of the two source images, which we label plus and minus.

## Returns

[xS_plus, xS_minus]
[list of numpy arrays]

- xS_plus is the vector position of the plus image.
- xS_minus is the vector position of the plus image.
class model.PSPL_PhotAstrom_noPar_Param2 (*args, **kwargs)
Bases: ModelClassABC, PSPL_PhotAstrom, PSPL_noParallax, PSPL_PhotAstromParam2
Helper class that provides a standard way to create an ABC using inheritance.

Methods

| animate(tE, time_steps, frame_time, name, ...) | Produces animation of microlensing event. |
| :--- | :--- |
| get_amplification(t) | noParallax: Get the photometric amplification term <br> at a set of times,. |
| get_astrometry(t_obs[, ast_filt_idx]) | noParallax: Position of the observed source position <br> in arcsec. |
| get_astrometry_unlensed(t_obs) | noParallax: Get the astrometry of the source if the <br> lens didn't exist. |
| get_centroid_shift(t) | noParallax: Get the centroid shift (in mas) for a list <br> of observation times (in MJD). |
| get_lens_astrometry(t_obs) | Equation of motion for just the foreground lens. |
| get_resolved_amplification(t) | Get the photometric amplification term at a set of <br> times, t for both the plus and minus images. |
| get_resolved_astrometry(t_obs) | Get the $\mathrm{x}, \mathrm{y}$ astrometry for each of the two source <br> images, which we label plus and minus. |


| calc_piE_ecliptic |  |
| :--- | :--- |
| get_chi2_astrometry |  |
| get_chi2_photometry |  |
| get_lnL_constant |  |
| get_photometry |  |
| log_likely_astrometry |  |
| log_likely_astrometry_each |  |
| log_likely_photometry |  |
| log_likely_photometry_each |  |

animate ( $t$ E, time_steps, frame_time, name, size, zoom, astrometry)
Produces animation of microlensing event. This function takes the PSPL and makes an animation, the input variables are as follows

## Parameters

tE:
number of einstein crossings times before/after the peak you want the animation to plot
e.g tE $=2=>$ graph will go from -2 tE to 2 tE
time_steps:
number of time steps before/after peak, so total number of time steps will be 2 times this value

## frame_time:

times in ms of each frame in the animation
name: string
the animation will be saved as name.html
size: list
[horizontal, vertical] cm's
zoom:
\# of einstein radii plotted in vertical direction
get_amplification $(t)$
noParallax: Get the photometric amplification term at a set of times, $t$.

## Parameters

t:
Array of times in MJD.DDD
get_astrometry (t_obs, ast_filt_idx=0)
noParallax: Position of the observed source position in arcsec.
get_astrometry_unlensed (t_obs)
noParallax: Get the astrometry of the source if the lens didn't exist.

## Returns

xS_unlensed
[numpy array, dtype=float, shape $=$ len(t_obs) x 2] The unlensed positions of the source in arcseconds.

## get_centroid_shift ( $t$ )

noParallax: Get the centroid shift (in mas) for a list of observation times (in MJD).

```
get_lens_astrometry(t_obs)
```

Equation of motion for just the foreground lens.

## Parameters

t_obs
[array_like] Time (in MJD).
get_resolved_amplification ( $t$ )
Get the photometric amplification term at a set of times, t for both the plus and minus images.

## Parameters

t:
Array of times in MJD.DDD
get_resolved_astrometry (t_obs)
Get the x , y astrometry for each of the two source images, which we label plus and minus.

## Returns

[xS_plus, xS_minus]
[list of numpy arrays]

- xS_plus is the vector position of the plus image in arcsec
- xS_minus is the vector position of the plus image in arcsec
class model.PSPL_PhotAstrom_Par_Param3(*args, **kwargs)
Bases: ModelClassABC, PSPL_PhotAstrom, PSPL_Parallax, PSPL_PhotAstromParam3
Helper class that provides a standard way to create an ABC using inheritance.


## Methods

| animate(tE, time_steps, frame_time, name, ...) | Produces animation of microlensing event. |
| :--- | :--- |
| calc_piE_ecliptic() | Parallax: Get piE_ecliptic |
| get_amplification(t) | Parallax: Get the photometric amplification term at a <br> set of times, t. |
| get_astrometry(t_obs[, ast_filt_idx]) | Parallax: Get astrometry |
| get_astrometry_unlensed(t_obs) | Get the astrometry of the source if the lens didn't ex- <br> ist. |
| get_centroid_shift(t[, ast_filt_idx]) | Parallax: Get the centroid shift (in mas) for a list of <br> observation times (in MJD). |
| get_geoproj_ast_params(t0par) | Parallax: Get lens astrometry |
| get_geoproj_params(t0par) | Parallax: Get the photometric amplification term at a <br> set of times, t for both the plus and minus images. |
| get_lens_astrometry(t_obs) | Parallax: Get the x, y astrometry for each of the two <br> source images, which we label plus and minus. |
| get_resolved_amplification(t) |  |


| get_chi2_astrometry |  |
| :--- | :--- |
| get_chi2_photometry |  |
| get_lnL_constant |  |
| get_photometry |  |
| log_likely_astrometry |  |
| log_likely_astrometry_each |  |
| log_likely_photometry |  |
| log_likely_photometry_each |  |
| start |  |

animate ( $t E$, time_steps, frame_time, name, size, zoom, astrometry)
Produces animation of microlensing event. This function takes the PSPL and makes an animation, the input variables are as follows

## Parameters

tE:
number of einstein crossings times before/after the peak you want the animation to plot
e.g tE $=2=>$ graph will go from -2 tE to 2 tE
time_steps:
number of time steps before/after peak, so total number of time steps will be 2 times this value

## frame_time:

times in ms of each frame in the animation

## name: string

the animation will be saved as name.html
size: list
[horizontal, vertical] cm's

## zoom:

\# of einstein radii plotted in vertical direction
calc_piE_ecliptic()
Parallax: Get piE_ecliptic
get_amplification $(t)$
Parallax: Get the photometric amplification term at a set of times, $t$.

## Parameters

t:
Array of times in MJD.DDD
get_astrometry (t_obs, ast_filt_idx=0)
Parallax: Get astrometry
get_astrometry_unlensed (t_obs)
Get the astrometry of the source if the lens didn't exist.

## Returns

xS_unlensed
[numpy array, dtype=float, shape $=$ len(t_obs) x 2] The unlensed positions of the source in arcseconds.
get_centroid_shift ( $t$, ast_filt_idx=0)
Parallax: Get the centroid shift (in mas) for a list of observation times (in MJD).

## get_lens_astrometry (t_obs)

Parallax: Get lens astrometry

## get_resolved_amplification ( $t$ )

Parallax: Get the photometric amplification term at a set of times, $t$ for both the plus and minus images.

## Parameters

t:
Array of times in MJD.DDD
get_resolved_astrometry (t_obs)
Parallax: Get the x , y astrometry for each of the two source images, which we label plus and minus.

## Returns

[xS_plus, xS_minus]
[list of numpy arrays]

- xS_plus is the vector position of the plus image.
- xS_minus is the vector position of the plus image.
class model.PSPL_PhotAstrom_Par_Param4 (*args, **kwargs)
Bases: ModelClassABC, PSPL_PhotAstrom, PSPL_Parallax, PSPL_PhotAstromParam4
Helper class that provides a standard way to create an ABC using inheritance.


## Methods

| animate(tE, time_steps, frame_time, name, ...) | Produces animation of microlensing event. |
| :--- | :--- |
| calc_piE_ecliptic() | Parallax: Get piE_ecliptic |
| get_amplification(t) | Parallax: Get the photometric amplification term at a <br> set of times, t. |
| get_astrometry(t_obs[, ast_filt_idx]) | Parallax: Get astrometry |
| get_astrometry_unlensed(t_obs) | Get the astrometry of the source if the lens didn't ex- <br> ist. |
| get_centroid_shift(t[, ast_filt_idx]) | Parallax: Get the centroid shift (in mas) for a list of <br> observation times (in MJD). |
| get_geoproj_ast_params(t0par) | Parallax: Get lens astrometry |
| get_geoproj_params(t0par) | Parallax: Get the photometric amplification term at a <br> set of times, t for both the plus and minus images. |
| get_lens_astrometry(t_obs) | Parallax: Get the x, y astrometry for each of the two <br> source images, which we label plus and minus. |
| get_resolved_amplification(t) |  |


| get_chi2_astrometry |  |
| :--- | :--- |
| get_chi2_photometry |  |
| get_lnL_constant |  |
| get_photometry |  |
| log_likely_astrometry |  |
| log_likely_astrometry_each |  |
| log_likely_photometry |  |
| log_likely_photometry_each |  |
| start |  |

animate ( $t E$, time_steps, frame_time, name, size, zoom, astrometry)
Produces animation of microlensing event. This function takes the PSPL and makes an animation, the input variables are as follows

## Parameters

tE:
number of einstein crossings times before/after the peak you want the animation to plot
e.g tE $=2=>$ graph will go from -2 tE to 2 tE
time_steps:
number of time steps before/after peak, so total number of time steps will be 2 times this value

## frame_time:

times in ms of each frame in the animation

## name: string

the animation will be saved as name.html
size: list
[horizontal, vertical] cm's

## zoom:

\# of einstein radii plotted in vertical direction
calc_piE_ecliptic()
Parallax: Get piE_ecliptic
get_amplification $(t)$
Parallax: Get the photometric amplification term at a set of times, $t$.

## Parameters

t:
Array of times in MJD.DDD
get_astrometry (t_obs, ast_filt_idx=0)
Parallax: Get astrometry
get_astrometry_unlensed (t_obs)
Get the astrometry of the source if the lens didn't exist.

## Returns

xS_unlensed
[numpy array, dtype=float, shape $=$ len(t_obs) x 2] The unlensed positions of the source in arcseconds.

```
get_centroid_shift(t,ast_filt_idx=0)
```

Parallax: Get the centroid shift (in mas) for a list of observation times (in MJD).

```
get_lens_astrometry(t_obs)
```

Parallax: Get lens astrometry

## get_resolved_amplification $(t)$

Parallax: Get the photometric amplification term at a set of times, $t$ for both the plus and minus images.

## Parameters

t:
Array of times in MJD.DDD

## get_resolved_astrometry (t_obs)

Parallax: Get the x , y astrometry for each of the two source images, which we label plus and minus.

## Returns

[xS_plus, xS_minus]
[list of numpy arrays]

- xS_plus is the vector position of the plus image.
- xS_minus is the vector position of the plus image.
class model.PSPL_PhotAstrom_noPar_Param4(*args, **kwargs)
Bases: ModelClassABC, PSPL_PhotAstrom, PSPL_noParallax, PSPL_PhotAstromParam4
Helper class that provides a standard way to create an ABC using inheritance.


## Methods

| animate(tE, time_steps, frame_time, name, ...) | Produces animation of microlensing event. |
| :--- | :--- |
| get_amplification(t) | noParallax: Get the photometric amplification term <br> at a set of times, t. |
| get_astrometry(t_obs[, ast_filt_idx]) | noParallax: Position of the observed source position <br> in arcsec. |
| get_astrometry_unlensed(t_obs) | noParallax: Get the astrometry of the source if the <br> lens didn't exist. |
| get_centroid_shift(t) | noParallax: Get the centroid shift (in mas) for a list <br> of observation times (in MJD). |
| get_lens_astrometry(t_obs) | Equation of motion for just the foreground lens. |
| get_resolved_amplification(t) | Get the photometric amplification term at a set of <br> times, t for both the plus and minus images. |
| get_resolved_astrometry(t_obs) | Get the $\mathrm{x}, \mathrm{y}$ astrometry for each of the two source <br> images, which we label plus and minus. |


| calc_piE_ecliptic |  |
| :--- | :--- |
| get_chi2_astrometry |  |
| get_chi2_photometry |  |
| get_lnL_constant |  |
| get_photometry |  |
| log_likely_astrometry |  |
| log_likely_astrometry_each |  |
| log_likely_photometry |  |
| log_likely_photometry_each |  |

animate ( $t$ E, time_steps, frame_time, name, size, zoom, astrometry)
Produces animation of microlensing event. This function takes the PSPL and makes an animation, the input variables are as follows

## Parameters

## tE:

number of einstein crossings times before/after the peak you want the animation to plot
e.g tE $=2=>$ graph will go from -2 tE to 2 tE
time_steps:
number of time steps before/after peak, so total number of time steps will be 2 times this value
frame_time:
times in ms of each frame in the animation
name: string
the animation will be saved as name.html
size: list
[horizontal, vertical] cm's

## zoom:

\# of einstein radii plotted in vertical direction

```
get_amplification(t)
```

noParallax: Get the photometric amplification term at a set of times, t .

## Parameters

t:
Array of times in MJD.DDD
get_astrometry (t_obs, ast_filt_idx=0)
noParallax: Position of the observed source position in arcsec.
get_astrometry_unlensed (t_obs)
noParallax: Get the astrometry of the source if the lens didn't exist.

## Returns

xS_unlensed
[numpy array, dtype=float, shape $=$ len(t_obs) x 2] The unlensed positions of the source in arcseconds.

## get_centroid_shift $(t)$

noParallax: Get the centroid shift (in mas) for a list of observation times (in MJD).

```
get_lens_astrometry(t_obs)
```

Equation of motion for just the foreground lens.

## Parameters

t_obs
[array_like] Time (in MJD).
get_resolved_amplification $(t)$
Get the photometric amplification term at a set of times, t for both the plus and minus images.

## Parameters

t:
Array of times in MJD.DDD

## get_resolved_astrometry (t_obs)

Get the x , y astrometry for each of the two source images, which we label plus and minus.

## Returns

[xS_plus, xS_minus]
[list of numpy arrays]

- xS_plus is the vector position of the plus image in arcsec
- xS_minus is the vector position of the plus image in arcsec
class model.PSPL_Astrom_Par_Param4 (*args, **kwargs)
Bases: ModelClassABC, PSPL_Astrom, PSPL_Parallax, PSPL_AstromParam4
Helper class that provides a standard way to create an ABC using inheritance.


## Methods

| animate(tE, time_steps, frame_time, name, ...) | Produces animation of microlensing event. |
| :--- | :--- |
| calc_piE_ecliptic() | Parallax: Get piE_ecliptic |
| get_amplification(t) | Parallax: Get the photometric amplification term at a <br> set of times, t. |
| get_astrometry(t_obs[, ast_filt_idx]) | Parallax: Get astrometry |
| get_astrometry_unlensed(t_obs) | Get the astrometry of the source if the lens didn't ex- <br> ist. |
| get_centroid_shift(t[, ast_filt_idx]) | Parallax: Get the centroid shift (in mas) for a list of <br> observation times (in MJD). |
| get_geoproj_ast_params(t0par) | Parallax: Get lens astrometry |
| get_geoproj_params(t0par) | Parallax: Get the photometric amplification term at a <br> set of times, t for both the plus and minus images. |
| get_lens_astrometry(t_obs) | Parallax: Get the x, y astrometry for each of the two <br> source images, which we label plus and minus. |
| get_resolved_amplification(t) |  |


| get_chi2_astrometry |  |
| :--- | :--- |
| get_chi2_photometry |  |
| get_lnL_constant |  |
| get_photometry |  |
| log_likely_astrometry |  |
| log_likely_astrometry_each |  |
| log_likely_photometry |  |
| log_likely_photometry_each |  |
| start |  |

animate ( $t E$, time_steps, frame_time, name, size, zoom, astrometry)
Produces animation of microlensing event. This function takes the PSPL and makes an animation, the input variables are as follows

## Parameters

tE:
number of einstein crossings times before/after the peak you want the animation to plot
e.g tE $=2=>$ graph will go from -2 tE to 2 tE
time_steps:
number of time steps before/after peak, so total number of time steps will be 2 times this value

## frame_time:

times in ms of each frame in the animation

## name: string

the animation will be saved as name.html
size: list
[horizontal, vertical] cm's

## zoom:

\# of einstein radii plotted in vertical direction
calc_piE_ecliptic()
Parallax: Get piE_ecliptic
get_amplification $(t)$
Parallax: Get the photometric amplification term at a set of times, $t$.

## Parameters

t:
Array of times in MJD.DDD
get_astrometry (t_obs, ast_filt_idx=0)
Parallax: Get astrometry
get_astrometry_unlensed (t_obs)
Get the astrometry of the source if the lens didn't exist.

## Returns

xS_unlensed
[numpy array, dtype=float, shape $=$ len(t_obs) x 2] The unlensed positions of the source in arcseconds.

```
get_centroid_shift(t,ast_filt_idx=0)
```

Parallax: Get the centroid shift (in mas) for a list of observation times (in MJD).

```
get_lens_astrometry(t_obs)
```

Parallax: Get lens astrometry

## get_resolved_amplification $(t)$

Parallax: Get the photometric amplification term at a set of times, $t$ for both the plus and minus images.

## Parameters

t:
Array of times in MJD.DDD

## get_resolved_astrometry (t_obs)

Parallax: Get the x , y astrometry for each of the two source images, which we label plus and minus.

## Returns

[xS_plus, xS_minus]
[list of numpy arrays]

- xS_plus is the vector position of the plus image.
- xS_minus is the vector position of the plus image.
class model.PSPL_Astrom_Par_Param3(*args, **kwargs)
Bases: ModelClassABC, PSPL_Astrom, PSPL_Parallax, PSPL_AstromParam3
Helper class that provides a standard way to create an ABC using inheritance.


## Methods

| animate(tE, time_steps, frame_time, name, ...) | Produces animation of microlensing event. |
| :--- | :--- |
| calc_piE_ecliptic() | Parallax: Get piE_ecliptic |
| get_amplification(t) | Parallax: Get the photometric amplification term at a <br> set of times, t. |
| get_astrometry(t_obs[, ast_filt_idx]) | Parallax: Get astrometry |
| get_astrometry_unlensed(t_obs) | Get the astrometry of the source if the lens didn't ex- <br> ist. |
| get_centroid_shift(t[, ast_filt_idx]) | Parallax: Get the centroid shift (in mas) for a list of <br> observation times (in MJD). |
| get_geoproj_ast_params(t0par) | Parallax: Get lens astrometry |
| get_geoproj_params(t0par) | Parallax: Get the photometric amplification term at a <br> set of times, t for both the plus and minus images. |
| get_lens_astrometry(t_obs) | Parallax: Get the x, y astrometry for each of the two <br> source images, which we label plus and minus. |
| get_resolved_amplification(t) |  |


| get_chi2_astrometry |  |
| :--- | :--- |
| get_chi2_photometry |  |
| get_lnL_constant |  |
| get_photometry |  |
| log_likely_astrometry |  |
| log_likely_astrometry_each |  |
| log_likely_photometry |  |
| log_likely_photometry_each |  |
| start |  |

animate ( $t E$, time_steps, frame_time, name, size, zoom, astrometry)
Produces animation of microlensing event. This function takes the PSPL and makes an animation, the input variables are as follows

## Parameters

tE:
number of einstein crossings times before/after the peak you want the animation to plot
e.g tE $=2=>$ graph will go from -2 tE to 2 tE
time_steps:
number of time steps before/after peak, so total number of time steps will be 2 times this value

## frame_time:

times in ms of each frame in the animation

## name: string

the animation will be saved as name.html
size: list
[horizontal, vertical] cm's

## zoom:

\# of einstein radii plotted in vertical direction
calc_piE_ecliptic()
Parallax: Get piE_ecliptic
get_amplification $(t)$
Parallax: Get the photometric amplification term at a set of times, $t$.

## Parameters

t:
Array of times in MJD.DDD
get_astrometry (t_obs, ast_filt_idx=0)
Parallax: Get astrometry
get_astrometry_unlensed (t_obs)
Get the astrometry of the source if the lens didn't exist.

## Returns

xS_unlensed
[numpy array, dtype=float, shape $=$ len(t_obs) x 2] The unlensed positions of the source in arcseconds.

```
get_centroid_shift(t,ast_filt_idx=0)
```

Parallax: Get the centroid shift (in mas) for a list of observation times (in MJD).

```
get_lens_astrometry(t_obs)
```

Parallax: Get lens astrometry

## get_resolved_amplification $(t)$

Parallax: Get the photometric amplification term at a set of times, $t$ for both the plus and minus images.

## Parameters

t:
Array of times in MJD.DDD

## get_resolved_astrometry (t_obs)

Parallax: Get the x , y astrometry for each of the two source images, which we label plus and minus.

## Returns

[xS_plus, xS_minus]
[list of numpy arrays]

- xS_plus is the vector position of the plus image.
- xS_minus is the vector position of the plus image.


### 4.1.4 PSPL_phot

class model.PSPL_Phot_noPar_Param1 (*args, **kwargs)
Bases: ModelClassABC, PSPL_Phot, PSPL_noParallax, PSPL_PhotParam1
Helper class that provides a standard way to create an ABC using inheritance.

Methods

| animate $(\mathrm{tE}$, time_steps, frame_time, name, ...) | Produces animation of microlensing event. |
| :--- | :--- |
| get_amplification $(\mathrm{t})$ | noParallax: Get the photometric amplification term <br> at a set of times, t. |


| calc_piE_ecliptic |  |
| :--- | :--- |
| get_astrometry |  |
| get_astrometry_unlensed |  |
| get_centroid_shift |  |
| get_chi2_photometry |  |
| get_lens_astrometry |  |
| get_lnL_constant |  |
| get_photometry |  |
| get_resolved_amplification |  |
| get_resolved_astrometry |  |
| log_likely_astrometry |  |
| log_likely_photometry |  |
| log_likely_photometry_each |  |

animate ( $t$ E, time_steps, frame_time, name, size, zoom, astrometry)
Produces animation of microlensing event. This function takes the PSPL and makes an animation, the input variables are as follows

## Parameters

tE:
number of einstein crossings times before/after the peak you want the animation to plot
e.g tE $=2=>$ graph will go from -2 tE to 2 tE
time_steps:
number of time steps before/after peak, so total number of time steps will be 2 times this value
frame_time:
times in ms of each frame in the animation
name: string
the animation will be saved as name.html
size: list
[horizontal, vertical] cm's

## zoom:

\# of einstein radii plotted in vertical direction

```
get_amplification(t)
```

noParallax: Get the photometric amplification term at a set of times, t .

## Parameters

t:
Array of times in MJD.DDD
get_astrometry ( $t$, ast_filt_idx=0)
noParallax: Position of the observed source position in arcsec.
get_astrometry_unlensed $(t)$
noParallax: Get the astrometry of the source if the lens didn't exist.

## Returns

## xS_unlensed

[numpy array, dtype=float, shape $=$ len(t_obs) x 2] The unlensed positions of the source in arcseconds.
get_centroid_shift ( $t$, ast_filt_idx=0)
noParallax: Get the centroid shift (in mas) for a list of observation times (in MJD).

```
get_lens_astrometry(t)
```

Equation of motion for just the foreground lens.

## Parameters

t_obs
[array_like] Time (in MJD).
get_resolved_amplification $(t)$
Get the photometric amplification term at a set of times, t for both the plus and minus images.

## Parameters

t:
Array of times in MJD.DDD

## get_resolved_astrometry $(t)$

Get the x , y astrometry for each of the two source images, which we label plus and minus.

## Returns

[xS_plus, xS_minus]
[list of numpy arrays]

- xS_plus is the vector position of the plus image in arcsec
- xS_minus is the vector position of the plus image in arcsec
class model.PSPL_Phot_noPar_Param2 (*args, **kwargs)
Bases: ModelClassABC, PSPL_Phot, PSPL_noParallax, PSPL_PhotParam2
Helper class that provides a standard way to create an ABC using inheritance.


## Methods

| animate $(\mathrm{tE}$, time_steps, frame_time, name, ...) | Produces animation of microlensing event. |
| :--- | :--- |
| get_amplification $(\mathrm{t})$ | noParallax: Get the photometric amplification term <br> at a set of times, t. |


| calc_piE_ecliptic |  |
| :--- | :--- |
| get_astrometry |  |
| get_astrometry_unlensed |  |
| get_centroid_shift |  |
| get_chi2_photometry |  |
| get_lens_astrometry |  |
| get_lnL_constant |  |
| get_photometry |  |
| get_resolved_amplification |  |
| get_resolved_astrometry |  |
| log_likely_astrometry |  |
| log_likely_photometry |  |
| log_likely_photometry_each |  |

animate (tE, time_steps, frame_time, name, size, zoom, astrometry)
Produces animation of microlensing event. This function takes the PSPL and makes an animation, the input variables are as follows

## Parameters

## tE:

number of einstein crossings times before/after the peak you want the animation to plot
e.g tE $=2=>$ graph will go from -2 tE to 2 tE

## time_steps:

number of time steps before/after peak, so total number of time steps will be 2 times this value
frame_time:
times in ms of each frame in the animation
name: string
the animation will be saved as name.html
size: list
[horizontal, vertical] cm's

## zoom:

\# of einstein radii plotted in vertical direction

## get_amplification $(t)$

noParallax: Get the photometric amplification term at a set of times, t .

## Parameters

t:
Array of times in MJD.DDD
get_astrometry (t, ast_fil_idx=0)
noParallax: Position of the observed source position in arcsec.
get_astrometry_unlensed $(t)$
noParallax: Get the astrometry of the source if the lens didn't exist.

## Returns

xS_unlensed
[numpy array, dtype=float, shape $=$ len(t_obs) x 2] The unlensed positions of the source in arcseconds.
get_centroid_shift ( $t$, ast_filt_idx=0)
noParallax: Get the centroid shift (in mas) for a list of observation times (in MJD).
get_lens_astrometry $(t)$
Equation of motion for just the foreground lens.

## Parameters

## t_obs

[array_like] Time (in MJD).

```
get_resolved_amplification(t)
```

Get the photometric amplification term at a set of times, t for both the plus and minus images.

## Parameters

t:
Array of times in MJD.DDD

## get_resolved_astrometry $(t)$

Get the $\mathrm{x}, \mathrm{y}$ astrometry for each of the two source images, which we label plus and minus.

## Returns

[xS_plus, xS_minus]
[list of numpy arrays]

- xS_plus is the vector position of the plus image in arcsec
- xS_minus is the vector position of the plus image in arcsec


### 4.1.5 PSPL_phot_parallax / PSPL_phot_multiphot_parallax

class model.PSPL_Phot_Par_Param1(*args, **kwargs)
Bases: ModelClassABC, PSPL_Phot, PSPL_Parallax, PSPL_PhotParam1
Helper class that provides a standard way to create an ABC using inheritance.

## Methods

| animate(tE, time_steps, frame_time, name, ...) | Produces animation of microlensing event. |
| :--- | :--- |
| calc_piE_ecliptic() | Parallax: Get piE_ecliptic |
| get_amplification(t) | Parallax: Get the photometric amplification term at a <br> set of times, t. |
| get_geoproj_ast_params(t0par) |  |
| get_geoproj_params(t0par) |  |


| get_astrometry |  |
| :--- | :--- |
| get_astrometry_unlensed |  |
| get_centroid_shift |  |
| get_chi2_photometry |  |
| get_lens_astrometry |  |
| get_lnL_constant |  |
| get_photometry |  |
| get_resolved_amplification |  |
| get_resolved_astrometry |  |
| log_likely_astrometry |  |
| log_likely_photometry |  |
| log_likely_photometry_each |  |
| start |  |

animate ( $t$ E, time_steps, frame_time, name, size, zoom, astrometry)
Produces animation of microlensing event. This function takes the PSPL and makes an animation, the input variables are as follows

## Parameters

tE:
number of einstein crossings times before/after the peak you want the animation to plot
e.g $\mathrm{tE}=2=>$ graph will go from -2 tE to 2 tE
time_steps:
number of time steps before/after peak, so total number of time steps will be 2 times this value

## frame_time:

times in ms of each frame in the animation
name: string
the animation will be saved as name.html

## size: list

[horizontal, vertical] cm's
zoom:
\# of einstein radii plotted in vertical direction

```
calc_piE_ecliptic()
```

Parallax: Get piE_ecliptic

## get_amplification $(t)$

Parallax: Get the photometric amplification term at a set of times, t .

## Parameters

t:
Array of times in MJD.DDD
get_astrometry ( $t$, ast_fil_idx=0)
Parallax: Get astrometry
get_astrometry_unlensed $(t)$
Get the astrometry of the source if the lens didn't exist.

## Returns

xS_unlensed
[numpy array, dtype=float, shape $=$ len(t_obs) $\times 2$ 2] The unlensed positions of the source in arcseconds.
get_centroid_shift (t, ast_fil_idx=0)
Parallax: Get the centroid shift (in mas) for a list of observation times (in MJD).

```
get_lens_astrometry(t)
```

Parallax: Get lens astrometry
get_resolved_amplification $(t)$
Parallax: Get the photometric amplification term at a set of times, t for both the plus and minus images.

## Parameters

t:
Array of times in MJD.DDD

## get_resolved_astrometry $(t)$

Parallax: Get the $\mathrm{x}, \mathrm{y}$ astrometry for each of the two source images, which we label plus and minus.

## Returns

[xS_plus, xS_minus]
[list of numpy arrays]

- xS_plus is the vector position of the plus image.
- xS_minus is the vector position of the plus image.
class model.PSPL_Phot_Par_Param2 (*args, **kwargs)
Bases: ModelClassABC, PSPL_Phot, PSPL_Parallax, PSPL_PhotParam2
Helper class that provides a standard way to create an ABC using inheritance.


## Methods

| animate(tE, time_steps, frame_time, name, ...) | Produces animation of microlensing event. |
| :--- | :--- |
| calc_piE_ecliptic() | Parallax: Get piE_ecliptic |
| get_amplification(t) | Parallax: Get the photometric amplification term at a <br> set of times, t. |
| get_geoproj_ast_params(t0par) |  |
| get_geoproj_params(t0par) |  |


| get_astrometry |  |
| :--- | :--- |
| get_astrometry_unlensed |  |
| get_centroid_shift |  |
| get_chi2_photometry |  |
| get_lens_astrometry |  |
| get_lnL_constant |  |
| get_photometry |  |
| get_resolved_amplification |  |
| get_resolved_astrometry |  |
| log_likely_astrometry |  |
| log_likely_photometry |  |
| log_likely_photometry_each |  |
| start |  |

animate ( $t$ E, time_steps, frame_time, name, size, zoom, astrometry)
Produces animation of microlensing event. This function takes the PSPL and makes an animation, the input variables are as follows

## Parameters

tE:
number of einstein crossings times before/after the peak you want the animation to plot
e.g $\mathrm{tE}=2=>$ graph will go from -2 tE to 2 tE
time_steps:
number of time steps before/after peak, so total number of time steps will be 2 times this value

## frame_time:

times in ms of each frame in the animation
name: string
the animation will be saved as name.html

## size: list

[horizontal, vertical] cm's
zoom:
\# of einstein radii plotted in vertical direction

```
calc_piE_ecliptic()
```

Parallax: Get piE_ecliptic

```
get_amplification(t)
```

Parallax: Get the photometric amplification term at a set of times, t .

## Parameters

t:
Array of times in MJD.DDD
get_astrometry ( $t$, ast_fil_idx=0)
Parallax: Get astrometry
get_astrometry_unlensed $(t)$
Get the astrometry of the source if the lens didn't exist.

## Returns

xS_unlensed
[numpy array, dtype=float, shape $=$ len(t_obs) $\times 2$ 2] The unlensed positions of the source in arcseconds.
get_centroid_shift $(t$, ast_filt_idx=0)
Parallax: Get the centroid shift (in mas) for a list of observation times (in MJD).

```
get_lens_astrometry(t)
```

Parallax: Get lens astrometry
get_resolved_amplification $(t)$
Parallax: Get the photometric amplification term at a set of times, t for both the plus and minus images.

## Parameters

t:
Array of times in MJD.DDD
get_resolved_astrometry $(t)$
Parallax: Get the $\mathrm{x}, \mathrm{y}$ astrometry for each of the two source images, which we label plus and minus.

## Returns

[xS_plus, xS_minus]
[list of numpy arrays]

- xS_plus is the vector position of the plus image.
- xS_minus is the vector position of the plus image.


### 4.1.6 PSPL Phot parallax with GP

class model.PSPL_Phot_Par_GP_Param1 (*args, **kwargs)
Bases: ModelClassABC, PSPL_GP, PSPL_Phot, PSPL_Parallax, PSPL_GP_PhotParam1
Helper class that provides a standard way to create an ABC using inheritance.

## Methods

| animate(tE, time_steps, frame_time, name, ...) | Produces animation of microlensing event. |
| :--- | :--- |
| calc_piE_ecliptic() | Parallax: Get piE_ecliptic |
| get_amplification(t) | Parallax: Get the photometric amplification term at a <br> set of times, t. |
| get_geoproj_ast_params(t0par) |  |
| get_geoproj_params(t0par) |  |
| get_log_det_covariance(t_obs, mag_obs, ...) | Returns photometry with GP noise added in. |
| get_photometry_with_gp(t_obs, mag_obs, ...) | Returns photometry with GP noise added in. |
| log_likely_photometry(t_obs, mag_obs, ...[,...]) | Calculate the log-likelihood for the PSPL + GP model <br> and photometric data.. |


| get_astrometry |  |
| :--- | :--- |
| get_astrometry_unlensed |  |
| get_centroid_shift |  |
| get_chi2_photometry |  |
| get_lens_astrometry |  |
| get_lnL_constant |  |
| get_photometry |  |
| get_resolved_amplification |  |
| get_resolved_astrometry |  |
| log_likely_astrometry |  |
| log_likely_photometry_each |  |
| start |  |

animate ( $t$ E, time_steps, frame_time, name, size, zoom, astrometry)
Produces animation of microlensing event. This function takes the PSPL and makes an animation, the input variables are as follows

## Parameters

tE:
number of einstein crossings times before/after the peak you want the animation to plot
e.g tE $=2$ => graph will go from -2 tE to 2 tE
time_steps:
number of time steps before/after peak, so total number of time steps will be 2 times this value
frame_time:
times in ms of each frame in the animation

## name: string

the animation will be saved as name.html

## size: list

[horizontal, vertical] cm's

## zoom:

\# of einstein radii plotted in vertical direction

```
calc_piE_ecliptic()
```

Parallax: Get piE_ecliptic

## get_amplification $(t)$

Parallax: Get the photometric amplification term at a set of times, t .

## Parameters

t:
Array of times in MJD.DDD
get_astrometry ( $t$, ast_filt_idx=0)
Parallax: Get astrometry

## get_astrometry_unlensed $(t)$

Get the astrometry of the source if the lens didn't exist.

## Returns

xS_unlensed
[numpy array, dtype=float, shape $=$ len(t_obs) x 2] The unlensed positions of the source in arcseconds.
get_centroid_shift ( $t$, ast_filt_idx=0)
Parallax: Get the centroid shift (in mas) for a list of observation times (in MJD).

```
get_lens_astrometry(t)
```

Parallax: Get lens astrometry
get_log_det_covariance (t_obs, mag_obs, mag_err_obs, filt_index=0, t_pred=None)
Returns photometry with GP noise added in.

Note: This will throw an error if this is a filter with use_gp_phot[filt_index] = False.
get_photometry_with_gp(t_obs, mag_obs, mag_err_obs, filt_index=0, t_pred=None)
Returns photometry with GP noise added in.

Note: This will throw an error if this is a filter with use_gp_phot[filt_index] = False.

## get_resolved_amplification $(t)$

Parallax: Get the photometric amplification term at a set of times, $t$ for both the plus and minus images.

## Parameters

t:
Array of times in MJD.DDD

## get_resolved_astrometry $(t)$

Parallax: Get the x , y astrometry for each of the two source images, which we label plus and minus.

## Returns

[xS_plus, xS_minus]
[list of numpy arrays]

- xS_plus is the vector position of the plus image.
- xS_minus is the vector position of the plus image.
log_likely_photometry (t_obs, mag_obs, mag_err_obs, fil_index=0)
Calculate the log-likelihood for the PSPL + GP model and photometric data.
Note: The GP will only be used for filters where use_gp_phot[filt_index] = True.
class model.PSPL_Phot_Par_GP_Param2 (*args, **kwargs)
Bases: ModelClassABC, PSPL_GP, PSPL_Phot, PSPL_Parallax, PSPL_GP_PhotParam2
Helper class that provides a standard way to create an ABC using inheritance.


## Methods

| animate(tE, time_steps, frame_time, name, ...) | Produces animation of microlensing event. |
| :--- | :--- |
| calc_piE_ecliptic() | Parallax: Get piE_ecliptic |
| get_amplification(t) | Parallax: Get the photometric amplification term at a <br> set of times, t. |
| get_geoproj_ast_params(t0par) |  |
| get_geoproj_params(t0par) | Returns photometry with GP noise added in. |
| get_log_det_covariance(t_obs, mag_obs, ...) | Returns photometry with GP noise added in. |
| get_photometry_with_gp(t_obs, mag_obs, ...) | Calculate the log-likelihood for the PSPL + GP model <br> log_likely_photometry(t_obs, mag_obs, ..., ...]) |


| get_astrometry |  |
| :--- | :--- |
| get_astrometry_unlensed |  |
| get_centroid_shift |  |
| get_chi2_photometry |  |
| get_lens_astrometry |  |
| get_lnL_constant |  |
| get_photometry |  |
| get_resolved_amplification |  |
| get_resolved_astrometry |  |
| log_likely_astrometry |  |
| log_likely_photometry_each |  |
| start |  |

animate (tE, time_steps, frame_time, name, size, zoom, astrometry)
Produces animation of microlensing event. This function takes the PSPL and makes an animation, the input variables are as follows

## Parameters

tE:
number of einstein crossings times before/after the peak you want the animation to plot
e.g $\mathrm{tE}=2=>$ graph will go from -2 tE to 2 tE

## time_steps:

number of time steps before/after peak, so total number of time steps will be 2 times this value

## frame_time:

times in ms of each frame in the animation
name: string
the animation will be saved as name.html
size: list
[horizontal, vertical] cm's

## zoom:

\# of einstein radii plotted in vertical direction
calc_piE_ecliptic()
Parallax: Get piE_ecliptic
get_amplification $(t)$
Parallax: Get the photometric amplification term at a set of times, t .

## Parameters

t:
Array of times in MJD.DDD
get_astrometry $(t$, ast_fil_id $d x=0)$
Parallax: Get astrometry

## get_astrometry_unlensed $(t)$

Get the astrometry of the source if the lens didn't exist.

## Returns

xS_unlensed
[numpy array, dtype=float, shape $=$ len(t_obs) x 2] The unlensed positions of the source in arcseconds.

```
get_centroid_shift(t, ast_filt_idx=0)
```

Parallax: Get the centroid shift (in mas) for a list of observation times (in MJD).

```
get_lens_astrometry (t)
```

Parallax: Get lens astrometry
get_log_det_covariance (t_obs, mag_obs, mag_err_obs, filt_index=0, t_pred=None)
Returns photometry with GP noise added in.
Note: This will throw an error if this is a filter with $u s e \_g p \_p h o t\left[f i l t \_i n d e x\right]=$ False.
get_photometry_with_gp(t_obs, mag_obs, mag_err_obs, filt_index=0, t_pred=None)
Returns photometry with GP noise added in.
Note: This will throw an error if this is a filter with $u s e_{-}$gp_phot $[$fil_index $]=$False.

## get_resolved_amplification ( $t$ )

Parallax: Get the photometric amplification term at a set of times, $t$ for both the plus and minus images.

## Parameters

t:
Array of times in MJD.DDD

## get_resolved_astrometry $(t)$

Parallax: Get the x , y astrometry for each of the two source images, which we label plus and minus.

## Returns

[xS_plus, xS_minus]
[list of numpy arrays]

- xS_plus is the vector position of the plus image.
- xS_minus is the vector position of the plus image.
log_likely_photometry (t_obs, mag_obs, mag_err_obs, filt_index=0)
Calculate the log-likelihood for the PSPL + GP model and photometric data.

Note: The GP will only be used for filters where $u s e \_g p \_p h o t\left[f i l t \_i n d e x\right]=$ True .
class model.PSPL_Phot_Par_GP_Param1_2 (*args, **kwargs)
Bases: ModelClassABC, PSPL_GP, PSPL_Phot, PSPL_Parallax, PSPL_GP_PhotParam1_2
Helper class that provides a standard way to create an ABC using inheritance.

## Methods

| animate(tE, time_steps, frame_time, name, ...) | Produces animation of microlensing event. |
| :--- | :--- |
| calc_piE_ecliptic() | Parallax: Get piE_ecliptic |
| get_amplification(t) | Parallax: Get the photometric amplification term at a <br> set of times, t. |
| get_geoproj_ast_params(t0par) |  |
| get_geoproj_params(t0par) |  |
| get_log_det_covariance(t_obs, mag_obs, ...) | Returns photometry with GP noise added in. |
| get_photometry_with_gp(t_obs, mag_obs, ...) | Returns photometry with GP noise added in. |
| log_likely_photometry(t_obs, mag_obs, ..., ...]) | Calculate the log-likelihood for the PSPL + GP model <br> and photometric data.. |


| get_astrometry |  |
| :--- | :--- |
| get_astrometry_unlensed |  |
| get_centroid_shift |  |
| get_chi2_photometry |  |
| get_lens_astrometry |  |
| get_lnL_constant |  |
| get_photometry |  |
| get_resolved_amplification |  |
| get_resolved_astrometry |  |
| log_likely_astrometry |  |
| log_likely_photometry_each |  |
| start |  |

animate (tE, time_steps, frame_time, name, size, zoom, astrometry)
Produces animation of microlensing event. This function takes the PSPL and makes an animation, the input variables are as follows

## Parameters

tE:
number of einstein crossings times before/after the peak you want the animation to plot e.g tE $=2=>$ graph will go from -2 tE to 2 tE
time_steps:
number of time steps before/after peak, so total number of time steps will be 2 times this value

## frame_time:

times in ms of each frame in the animation
name: string
the animation will be saved as name.html
size: list
[horizontal, vertical] cm's

## zoom:

\# of einstein radii plotted in vertical direction
calc_piE_ecliptic()
Parallax: Get piE_ecliptic

```
get_amplification(t)
```

Parallax: Get the photometric amplification term at a set of times, $t$.

## Parameters

t:
Array of times in MJD.DDD
get_astrometry ( $t$, ast_filt_idx=0)
Parallax: Get astrometry
get_astrometry_unlensed $(t)$
Get the astrometry of the source if the lens didn't exist.

## Returns

## xS_unlensed

[numpy array, dtype=float, shape $=$ len(t_obs) x 2] The unlensed positions of the source in arcseconds.
get_centroid_shift ( $t$, ast_filt_idx $=0$ )
Parallax: Get the centroid shift (in mas) for a list of observation times (in MJD).
get_lens_astrometry $(t)$
Parallax: Get lens astrometry
get_log_det_covariance (t_obs, mag_obs, mag_err_obs, filt_index=0, t_pred=None)
Returns photometry with GP noise added in.

Note: This will throw an error if this is a filter with use_gp_phot[filt_index] = False.
get_photometry_with_gp(t_obs, mag_obs, mag_err_obs, filt_index=0, t_pred=None)
Returns photometry with GP noise added in.

Note: This will throw an error if this is a filter with use_gp_phot[filt_index] = False.

## get_resolved_amplification( $t$ )

Parallax: Get the photometric amplification term at a set of times, t for both the plus and minus images.

## Parameters

t:
Array of times in MJD.DDD
get_resolved_astrometry $(t)$
Parallax: Get the x , y astrometry for each of the two source images, which we label plus and minus.

## Returns

[xS_plus, xS_minus]
[list of numpy arrays]

- xS_plus is the vector position of the plus image.
- xS_minus is the vector position of the plus image.
log_likely_photometry (t_obs, mag_obs, mag_err_obs, filt_index=0)
Calculate the log-likelihood for the PSPL + GP model and photometric data.

Note: The GP will only be used for filters where use_gp_phot[filt_index] = True.
class model.PSPL_Phot_Par_GP_Param2_2(*args, **kwargs)
Bases: ModelClassABC, PSPL_GP, PSPL_Phot, PSPL_Parallax, PSPL_GP_PhotParam2_2
Helper class that provides a standard way to create an ABC using inheritance.

Methods

| animate(tE, time_steps, frame_time, name, ...) | Produces animation of microlensing event. |
| :--- | :--- |
| calc_piE_ecliptic() | Parallax: Get piE_ecliptic |
| get_amplification(t) | Parallax: Get the photometric amplification term at a <br> set of times, t. |
| get_geoproj_ast_params(t0par) |  |
| get_geoproj_params(t0par) |  |
| get_log_det_covariance(t_obs, mag_obs, ...) | Returns photometry with GP noise added in. |
| get_photometry_with_gp(t_obs, mag_obs, ...) | Returns photometry with GP noise added in. |
| log_likely_photometry(t_obs, mag_obs, ...[,..]) | Calculate the log-likelihood for the PSPL + GP model <br> and photometric data. |


| get_astrometry |  |
| :--- | :--- |
| get_astrometry_unlensed |  |
| get_centroid_shift |  |
| get_chi2_photometry |  |
| get_lens_astrometry |  |
| get_lnL_constant |  |
| get_photometry |  |
| get_resolved_amplification |  |
| get_resolved_astrometry |  |
| log_likely_astrometry |  |
| log_likely_photometry_each |  |
| start |  |

animate ( $t$ E, time_steps, frame_time, name, size, zoom, astrometry)
Produces animation of microlensing event. This function takes the PSPL and makes an animation, the input variables are as follows

## Parameters

tE:
number of einstein crossings times before/after the peak you want the animation to plot
e.g tE $=2=>$ graph will go from -2 tE to 2 tE
time_steps:
number of time steps before/after peak, so total number of time steps will be 2 times this value
frame_time:
times in ms of each frame in the animation

## name: string

the animation will be saved as name.html
size: list
[horizontal, vertical] cm's

## zoom:

\# of einstein radii plotted in vertical direction

```
calc_piE_ecliptic()
```

Parallax: Get piE_ecliptic

## get_amplification $(t)$

Parallax: Get the photometric amplification term at a set of times, t .

## Parameters

t:
Array of times in MJD.DDD
get_astrometry ( $t$, ast_filt_idx=0)
Parallax: Get astrometry

## get_astrometry_unlensed $(t)$

Get the astrometry of the source if the lens didn't exist.

## Returns

xS_unlensed
[numpy array, dtype=float, shape $=$ len(t_obs) x 2] The unlensed positions of the source in arcseconds.
get_centroid_shift ( $t$, ast_filt_idx=0)
Parallax: Get the centroid shift (in mas) for a list of observation times (in MJD).

```
get_lens_astrometry(t)
```

Parallax: Get lens astrometry
get_log_det_covariance (t_obs, mag_obs, mag_err_obs, filt_index=0, t_pred=None)
Returns photometry with GP noise added in.

Note: This will throw an error if this is a filter with use_gp_phot[filt_index] = False.
get_photometry_with_gp(t_obs, mag_obs, mag_err_obs, filt_index=0, t_pred=None)
Returns photometry with GP noise added in.

Note: This will throw an error if this is a filter with use_gp_phot[filt_index] = False.

## get_resolved_amplification( $t$ )

Parallax: Get the photometric amplification term at a set of times, $t$ for both the plus and minus images.

## Parameters

t:
Array of times in MJD.DDD

## get_resolved_astrometry ( $t$ )

Parallax: Get the x , y astrometry for each of the two source images, which we label plus and minus.

## Returns

[xS_plus, xS_minus]
[list of numpy arrays]

- xS_plus is the vector position of the plus image.
- xS_minus is the vector position of the plus image.
log_likely_photometry (t_obs, mag_obs, mag_err_obs, filt_index=0)
Calculate the log-likelihood for the PSPL + GP model and photometric data.

Note: The GP will only be used for filters where $u s e \_g p \_p h o t\left[f i l t \_i n d e x\right]=$ True.

### 4.1.7 PSPL Phot, no parallax with GP

class model.PSPL_Phot_noPar_GP_Param1 (*args, **kwargs)
Bases: ModelClassABC, PSPL_GP, PSPL_Phot, PSPL_noParallax, PSPL_GP_PhotParam1
Helper class that provides a standard way to create an ABC using inheritance.

Methods

| animate(tE, time_steps, frame_time, name, ...) | Produces animation of microlensing event. |
| :--- | :--- |
| get_amplification(t) | noParallax: Get the photometric amplification term <br> at a set of times, t. |
| get_log_det_covariance(t_obs, mag_obs, ...) | Returns photometry with GP noise added in. |
| get_photometry_with_gp(t_obs, mag_obs, ...) | Returns photometry with GP noise added in. |
| log_likely_photometry(t_obs, mag_obs, ..., ...]) | Calculate the log-likelihood for the PSPL + GP model <br> and photometric data.. |


| calc_piE_ecliptic |  |
| :--- | :--- |
| get_astrometry |  |
| get_astrometry_unlensed |  |
| get_centroid_shift |  |
| get_chi2_photometry |  |
| get_lens_astrometry |  |
| get_lnL_constant |  |
| get_photometry |  |
| get_resolved_amplification |  |
| get_resolved_astrometry |  |
| log_likely_astrometry |  |
| log_likely_photometry_each |  |

animate ( $t E$, time_steps, frame_time, name, size, zoom, astrometry)
Produces animation of microlensing event. This function takes the PSPL and makes an animation, the input variables are as follows

## Parameters

## tE:

number of einstein crossings times before/after the peak you want the animation to plot
e.g tE $=2=>$ graph will go from -2 tE to 2 tE

## time_steps:

number of time steps before/after peak, so total number of time steps will be 2 times this value

## frame_time:

times in ms of each frame in the animation

## name: string

the animation will be saved as name.html
size: list
[horizontal, vertical] cm's
zoom:
\# of einstein radii plotted in vertical direction
get_amplification $(t)$
noParallax: Get the photometric amplification term at a set of times, t .

## Parameters

t:
Array of times in MJD.DDD
get_astrometry ( $t$, ast_filt_idx=0)
noParallax: Position of the observed source position in arcsec.
get_astrometry_unlensed $(t)$
noParallax: Get the astrometry of the source if the lens didn't exist.

## Returns

xS_unlensed
[numpy array, dtype=float, shape $=$ len(t_obs) x 2] The unlensed positions of the source in arcseconds.

```
get_centroid_shift(t, ast_filt_idx=0)
```

noParallax: Get the centroid shift (in mas) for a list of observation times (in MJD).

```
get_lens_astrometry(t)
```

Equation of motion for just the foreground lens.

## Parameters

## t_obs

[array_like] Time (in MJD).
get_log_det_covariance (t_obs, mag_obs, mag_err_obs, filt_index=0, t_pred=None)
Returns photometry with GP noise added in.

Note: This will throw an error if this is a filter with use_gp_phot[filt_index] = False.
get_photometry_with_gp(t_obs, mag_obs, mag_err_obs, filt_index=0, t_pred=None)
Returns photometry with GP noise added in.

Note: This will throw an error if this is a filter with use_gp_phot[filt_index] = False.
get_resolved_amplification ( $t$ )
Get the photometric amplification term at a set of times, $t$ for both the plus and minus images.

## Parameters

t:
Array of times in MJD.DDD
get_resolved_astrometry $(t)$
Get the $\mathrm{x}, \mathrm{y}$ astrometry for each of the two source images, which we label plus and minus.

## Returns

[xS_plus, xS_minus]
[list of numpy arrays]

- xS_plus is the vector position of the plus image in arcsec
- xS_minus is the vector position of the plus image in arcsec
log_likely_photometry (t_obs, mag_obs, mag_err_obs, filt_index=0)
Calculate the log-likelihood for the PSPL + GP model and photometric data.
Note: The GP will only be used for filters where use_gp_phot[filt_index] = True.
class model.PSPL_Phot_noPar_GP_Param2 (*args, **kwargs)
Bases: ModelClassABC, PSPL_GP, PSPL_Phot, PSPL_noParallax, PSPL_GP_PhotParam2
Helper class that provides a standard way to create an ABC using inheritance.


## Methods

| animate(tE, time_steps, frame_time, name, ...) | Produces animation of microlensing event. |
| :--- | :--- |
| get_amplification(t) | noParallax: Get the photometric amplification term <br> at a set of times, t. |
| get_log_det_covariance(t_obs, mag_obs, ...) | Returns photometry with GP noise added in. |
| get_photometry_with_gp(t_obs, mag_obs, ...) | Returns photometry with GP noise added in. |
| log_likely_photometry $\left(\mathrm{t} \_\right.$obs, mag_obs, ...[,...]) | Calculate the log-likelihood for the PSPL + GP model <br> and photometric data. |


| calc_piE_ecliptic |  |
| :--- | :--- |
| get_astrometry |  |
| get_astrometry_unlensed |  |
| get_centroid_shift |  |
| get_chi2_photometry |  |
| get_lens_astrometry |  |
| get_lnL_constant |  |
| get_photometry |  |
| get_resolved_amplification |  |
| get_resolved_astrometry |  |
| log_likely_astrometry |  |
| log_likely_photometry_each |  |

animate ( $t E$, time_steps, frame_time, name, size, zoom, astrometry)
Produces animation of microlensing event. This function takes the PSPL and makes an animation, the input variables are as follows

## Parameters

## tE:

number of einstein crossings times before/after the peak you want the animation to plot
e.g tE $=2=>$ graph will go from -2 tE to 2 tE

## time_steps:

number of time steps before/after peak, so total number of time steps will be 2 times this value

## frame_time:

times in ms of each frame in the animation
name: string
the animation will be saved as name.html
size: list
[horizontal, vertical] cm's
zoom:
\# of einstein radii plotted in vertical direction

## get_amplification $(t)$

noParallax: Get the photometric amplification term at a set of times, t .

## Parameters

t:
Array of times in MJD.DDD

```
get_astrometry(t, ast_filt_idx=0)
```

noParallax: Position of the observed source position in arcsec.

## get_astrometry_unlensed $(t)$

noParallax: Get the astrometry of the source if the lens didn't exist.

## Returns

## xS_unlensed

[numpy array, dtype=float, shape $=$ len(t_obs) x 2] The unlensed positions of the source in arcseconds.
get_centroid_shift ( $t$, ast_filt_idx=0)
noParallax: Get the centroid shift (in mas) for a list of observation times (in MJD).

```
get_lens_astrometry(t)
```

Equation of motion for just the foreground lens.

## Parameters

## t_obs

[array_like] Time (in MJD).

```
get_log_det_covariance(t_obs, mag_obs, mag_err_obs, filt_index=0, t_pred=None)
```

Returns photometry with GP noise added in.

Note: This will throw an error if this is a filter with use_gp_phot[filt_index] = False.
get_photometry_with_gp(t_obs, mag_obs, mag_err_obs, filt_index=0, t_pred=None)
Returns photometry with GP noise added in.

Note: This will throw an error if this is a filter with use_gp_phot[filt_index] = False.
get_resolved_amplification ( $t$ )
Get the photometric amplification term at a set of times, t for both the plus and minus images.

## Parameters

t:
Array of times in MJD.DDD
get_resolved_astrometry $(t)$
Get the x , y astrometry for each of the two source images, which we label plus and minus.

## Returns

[xS_plus, xS_minus]
[list of numpy arrays]

- xS_plus is the vector position of the plus image in arcsec
- xS_minus is the vector position of the plus image in arcsec
log_likely_photometry (t_obs, mag_obs, mag_err_obs, filt_index=0)
Calculate the log-likelihood for the PSPL + GP model and photometric data.

Note: The GP will only be used for filters where use_gp_phot[filt_index] = True.

### 4.1.8 PSPL PhotAstrom, parallax with GP

class model.PSPL_PhotAstrom_Par_GP_Param1 (*args, **kwargs)
Bases: ModelClassABC, PSPL_GP, PSPL_PhotAstrom, PSPL_Parallax, PSPL_GP_PhotAstromParam1
Helper class that provides a standard way to create an ABC using inheritance.

## Methods

| animate(tE, time_steps, frame_time, name, ...) | Produces animation of microlensing event. |
| :--- | :--- |
| calc_piE_ecliptic() | Parallax: Get piE_ecliptic |
| get_amplification(t) | Parallax: Get the photometric amplification term at a <br> set of times, t. |
| get_astrometry(t_obs[, ast_filt_idx]) | Parallax: Get astrometry |
| get_astrometry_unlensed(t_obs) | Get the astrometry of the source if the lens didn't ex- <br> ist. |
| get_centroid_shift(t[, ast_filt_idx]) | Parallax: Get the centroid shift (in mas) for a list of <br> observation times (in MJD). |
| get_geoproj_ast_params(t0par) | Parallax: Get lens astrometry |
| get_geoproj_params(t0par) | Returns photometry with GP noise added in. |
| get_lens_astrometry(t_obs) | Returns photometry with GP noise added in. <br> set of times, t for both the plus and minus images. |
| get_log_det_covariance(t_obs, mag_obs, ...) | Parallax: Get the x, y astrometry for each of the two <br> source images, which we label plus and minus. |
| get_photometry_with_gp(t_obs, mag_obs, ...) | Calculate the log-likelihood for the PSPL + GP model <br> and photometric data. |
| get_resolved_amplification(t) |  |


| get_chi2_astrometry |  |
| :--- | :--- |
| get_chi2_photometry |  |
| get_lnL_constant |  |
| get_photometry |  |
| log_likely_astrometry |  |
| log_likely_astrometry_each |  |
| log_likely_photometry_each |  |
| start |  |

animate ( $t E$, time_steps, frame_time, name, size, zoom, astrometry)
Produces animation of microlensing event. This function takes the PSPL and makes an animation, the input variables are as follows

## Parameters

tE:
number of einstein crossings times before/after the peak you want the animation to plot
e.g tE $=2$ => graph will go from -2 tE to 2 tE
time_steps:
number of time steps before/after peak, so total number of time steps will be 2 times this value

## frame_time:

times in ms of each frame in the animation

## name: string

the animation will be saved as name.html
size: list
[horizontal, vertical] cm's
zoom:
\# of einstein radii plotted in vertical direction

## calc_piE_ecliptic()

Parallax: Get piE_ecliptic

## get_amplification $(t)$

Parallax: Get the photometric amplification term at a set of times, t .

## Parameters

t:
Array of times in MJD.DDD
get_astrometry (t_obs, ast_filt_idx=0)
Parallax: Get astrometry
get_astrometry_unlensed (t_obs)
Get the astrometry of the source if the lens didn't exist.

## Returns

xS_unlensed
[numpy array, dtype=float, shape $=$ len(t_obs) x 2] The unlensed positions of the source in arcseconds.

## get_centroid_shift ( $t$, ast_filt_idx=0)

Parallax: Get the centroid shift (in mas) for a list of observation times (in MJD).

```
get_lens_astrometry(t_obs)
```

Parallax: Get lens astrometry
get_log_det_covariance (t_obs, mag_obs, mag_err_obs, filt_index=0, t_pred=None)
Returns photometry with GP noise added in.

Note: This will throw an error if this is a filter with use_gp_phot[filt_index] = False.
get_photometry_with_gp(t_obs, mag_obs, mag_err_obs, filt_index=0, t_pred=None)
Returns photometry with GP noise added in.

Note: This will throw an error if this is a filter with use_gp_phot[filt_index] = False.

## get_resolved_amplification( $t$ )

Parallax: Get the photometric amplification term at a set of times, $t$ for both the plus and minus images.

## Parameters

t:
Array of times in MJD.DDD

## get_resolved_astrometry (t_obs)

Parallax: Get the x , y astrometry for each of the two source images, which we label plus and minus.

## Returns

[xS_plus, xS_minus]
[list of numpy arrays]

- xS_plus is the vector position of the plus image.
- xS_minus is the vector position of the plus image.
log_likely_photometry (t_obs, mag_obs, mag_err_obs, filt_index=0)
Calculate the log-likelihood for the PSPL + GP model and photometric data.

Note: The GP will only be used for filters where use_gp_phot[filt_index] = True.
class model.PSPL_PhotAstrom_Par_GP_Param2 (*args, **kwargs)
Bases: ModelClassABC, PSPL_GP, PSPL_PhotAstrom, PSPL_Parallax, PSPL_GP_PhotAstromParam2
Helper class that provides a standard way to create an ABC using inheritance.

Methods

| animate(tE, time_steps, frame_time, name, ...) | Produces animation of microlensing event. |
| :--- | :--- |
| calc_piE_ecliptic() | Parallax: Get piE_ecliptic |
| get_amplification(t) | Parallax: Get the photometric amplification term at a <br> set of times, t. |
| get_astrometry(t_obs[, ast_filt_idx]) | Parallax: Get astrometry |
| get_astrometry_unlensed(t_obs) | Get the astrometry of the source if the lens didn't ex- <br> ist. |
| get_centroid_shift(t[, ast_filt_idx]) | Parallax: Get the centroid shift (in mas) for a list of <br> observation times (in MJD). |
| get_geoproj_ast_params(t0par) | Parallax: Get lens astrometry |
| get_geoproj_params(t0par) | Returns photometry with GP noise added in. |
| get_lens_astrometry(t_obs) | Peturns photometry with GP noise added in. <br> set of times, t for both the plus and minus images. |
| get_log_det_covariance(t_obs, mag_obs, ...) | Parallax: Get the x, y astrometry for each of the two <br> source images, which we label plus and minus. |
| get_photometry_with_gp(t_obs, mag_obs, ...) | Calculate the log-likelihood for the PSPL + GP model <br> and photometric data. |
| get_resolved_amplification(t) |  |
| get_resolved_astrometry(t_obs) |  |
| log_likely_photometry(t_obs, mag_obs, ...[, ...]) |  |


| get_chi2_astrometry |  |
| :--- | :--- |
| get_chi2_photometry |  |
| get_lnL_constant |  |
| get_photometry |  |
| log_likely_astrometry |  |
| log_likely_astrometry_each |  |
| log_likely_photometry_each |  |
| start |  |

animate ( $t$ E, time_steps, frame_time, name, size, zoom, astrometry)
Produces animation of microlensing event. This function takes the PSPL and makes an animation, the input variables are as follows

## Parameters

tE:
number of einstein crossings times before/after the peak you want the animation to plot
e.g tE $=2=>$ graph will go from -2 tE to 2 tE
time_steps:
number of time steps before/after peak, so total number of time steps will be 2 times this value
frame_time:
times in ms of each frame in the animation
name: string
the animation will be saved as name.html
size: list
[horizontal, vertical] cm's

## zoom:

\# of einstein radii plotted in vertical direction
calc_piE_ecliptic()
Parallax: Get piE_ecliptic
get_amplification $(t)$
Parallax: Get the photometric amplification term at a set of times, $t$.

## Parameters

t:
Array of times in MJD.DDD

```
get_astrometry(t_obs,ast_fil_idx=0)
```

Parallax: Get astrometry

```
get_astrometry_unlensed(t_obs)
```

Get the astrometry of the source if the lens didn't exist.

## Returns

xS_unlensed
[numpy array, dtype=float, shape $=$ len(t_obs) x 2] The unlensed positions of the source in arcseconds.
get_centroid_shift $(t$, ast_fil_idx $=0)$
Parallax: Get the centroid shift (in mas) for a list of observation times (in MJD).
get_lens_astrometry (t_obs)
Parallax: Get lens astrometry
get_log_det_covariance (t_obs, mag_obs, mag_err_obs, filt_index=0, t_pred=None)
Returns photometry with GP noise added in.
Note: This will throw an error if this is a filter with use_gp_phot[filt_index] = False.
get_photometry_with_gp $\left(t_{-} o b s\right.$, mag_obs, mag_err_obs, fil_index $=0, t_{-}$pred=None)
Returns photometry with GP noise added in.

Note: This will throw an error if this is a filter with use_gp_phot[filt_index] = False.

```
get_resolved_amplification \((t)\)
```

Parallax: Get the photometric amplification term at a set of times, t for both the plus and minus images.

## Parameters

t:
Array of times in MJD.DDD

```
get_resolved_astrometry(t_obs)
```

Parallax: Get the $\mathrm{x}, \mathrm{y}$ astrometry for each of the two source images, which we label plus and minus.

## Returns

[xS_plus, xS_minus]
[list of numpy arrays]

- xS_plus is the vector position of the plus image.
- xS_minus is the vector position of the plus image.
log_likely_photometry (t_obs, mag_obs, mag_err_obs, fil_index=0)
Calculate the log-likelihood for the PSPL + GP model and photometric data.

Note: The GP will only be used for filters where $u s e \_g p \_p h o t\left[f i l t \_i n d e x\right]=$ True.
class model.PSPL_PhotAstrom_Par_GP_Param3(*args, **kwargs)
Bases: ModelClassABC, PSPL_GP, PSPL_PhotAstrom, PSPL_Parallax, PSPL_GP_PhotAstromParam3
Helper class that provides a standard way to create an ABC using inheritance.

## Methods

| animate(tE, time_steps, frame_time, name, ...) | Produces animation of microlensing event. |
| :--- | :--- |
| calc_piE_ecliptic() | Parallax: Get piE_ecliptic |
| get_amplification(t) | Parallax: Get the photometric amplification term at a <br> set of times, t. |
| get_astrometry(t_obs[, ast_filt_idx]) | Parallax: Get astrometry |
| get_astrometry_unlensed(t_obs) | Get the astrometry of the source if the lens didn't ex- <br> ist. |
| get_centroid_shift(t[, ast_filt_idx]) | Parallax: Get the centroid shift (in mas) for a list of <br> observation times (in MJD). |
| get_geoproj_ast_params(t0par) | Parallax: Get lens astrometry |
| get_geoproj_params(t0par) | Returns photometry with GP noise added in. |
| get_lens_astrometry(t_obs) | Peturns photometry with GP noise added in. <br> set of times, t for both the plus and minus images. |
| get_log_det_covariance(t_obs, mag_obs, ...) | Parallax: Get the x, y astrometry for each of the two <br> source images, which we label plus and minus. |
| get_photometry_with_gp(t_obs, mag_obs, ...) | Calculate the log-likelihood for the PSPL + GP model <br> and photometric data. |
| get_resolved_amplification(t) |  |


| get_chi2_astrometry |  |
| :--- | :--- |
| get_chi2_photometry |  |
| get_lnL_constant |  |
| get_photometry |  |
| log_likely_astrometry |  |
| log_likely_astrometry_each |  |
| log_likely_photometry_each |  |
| start |  |

animate ( $t E$, time_steps, frame_time, name, size, zoom, astrometry)
Produces animation of microlensing event. This function takes the PSPL and makes an animation, the input variables are as follows

## Parameters

tE:
number of einstein crossings times before/after the peak you want the animation to plot
e.g tE $=2$ => graph will go from -2 tE to 2 tE
time_steps:
number of time steps before/after peak, so total number of time steps will be 2 times this value

## frame_time:

times in ms of each frame in the animation

## name: string

the animation will be saved as name.html
size: list
[horizontal, vertical] cm's
zoom:
\# of einstein radii plotted in vertical direction

## calc_piE_ecliptic()

Parallax: Get piE_ecliptic

## get_amplification $(t)$

Parallax: Get the photometric amplification term at a set of times, t .

## Parameters

t:
Array of times in MJD.DDD
get_astrometry (t_obs, ast_filt_idx=0)
Parallax: Get astrometry
get_astrometry_unlensed (t_obs)
Get the astrometry of the source if the lens didn't exist.

## Returns

xS_unlensed
[numpy array, dtype=float, shape $=$ len(t_obs) x 2] The unlensed positions of the source in arcseconds.
get_centroid_shift ( $t$, ast_filt_idx $=0$ )
Parallax: Get the centroid shift (in mas) for a list of observation times (in MJD).

```
get_lens_astrometry(t_obs)
```

Parallax: Get lens astrometry
get_log_det_covariance (t_obs, mag_obs, mag_err_obs, filt_index=0, t_pred=None)
Returns photometry with GP noise added in.

Note: This will throw an error if this is a filter with use_gp_phot[filt_index] = False.
get_photometry_with_gp(t_obs, mag_obs, mag_err_obs, filt_index=0, t_pred=None)
Returns photometry with GP noise added in.

Note: This will throw an error if this is a filter with use_gp_phot[filt_index] = False.

## get_resolved_amplification( $t$ )

Parallax: Get the photometric amplification term at a set of times, t for both the plus and minus images.

## Parameters

t:
Array of times in MJD.DDD

## get_resolved_astrometry (t_obs)

Parallax: Get the x , y astrometry for each of the two source images, which we label plus and minus.

## Returns

[xS_plus, xS_minus]
[list of numpy arrays]

- xS_plus is the vector position of the plus image.
- xS_minus is the vector position of the plus image.
log_likely_photometry (t_obs, mag_obs, mag_err_obs, filt_index=0)
Calculate the log-likelihood for the PSPL + GP model and photometric data.

Note: The GP will only be used for filters where $u s e \_g p \_p h o t\left[f i l t \_i n d e x\right]=$ True.
class model.PSPL_PhotAstrom_Par_GP_Param4 (*args, **kwargs)
Bases: ModelClassABC, PSPL_GP, PSPL_PhotAstrom, PSPL_Parallax, PSPL_GP_PhotAstromParam4
Helper class that provides a standard way to create an ABC using inheritance.

Methods

| animate(tE, time_steps, frame_time, name, ...) | Produces animation of microlensing event. |
| :--- | :--- |
| calc_piE_ecliptic() | Parallax: Get piE_ecliptic |
| get_amplification(t) | Parallax: Get the photometric amplification term at a <br> set of times, t. |
| get_astrometry(t_obs[, ast_filt_idx]) | Parallax: Get astrometry |
| get_astrometry_unlensed(t_obs) | Get the astrometry of the source if the lens didn't ex- <br> ist. |
| get_centroid_shift(t[, ast_filt_idx]) | Parallax: Get the centroid shift (in mas) for a list of <br> observation times (in MJD). |
| get_geoproj_ast_params(t0par) | Parallax: Get lens astrometry |
| get_geoproj_params(t0par) | Returns photometry with GP noise added in. |
| get_lens_astrometry(t_obs) | Peturns photometry with GP noise added in. <br> set of times, t for both the plus and minus images. |
| get_log_det_covariance(t_obs, mag_obs, ...) | Parallax: Get the x, y astrometry for each of the two <br> source images, which we label plus and minus. |
| get_photometry_with_gp(t_obs, mag_obs, ...) | Calculate the log-likelihood for the PSPL + GP model <br> and photometric data. |
| get_resolved_amplification(t) |  |


| get_chi2_astrometry |  |
| :--- | :--- |
| get_chi2_photometry |  |
| get_lnL_constant |  |
| get_photometry |  |
| log_likely_astrometry |  |
| log_likely_astrometry_each |  |
| log_likely_photometry_each |  |
| start |  |

animate ( $t$ E, time_steps, frame_time, name, size, zoom, astrometry)
Produces animation of microlensing event. This function takes the PSPL and makes an animation, the input variables are as follows

## Parameters

tE:
number of einstein crossings times before/after the peak you want the animation to plot
e.g tE $=2=>$ graph will go from -2 tE to 2 tE
time_steps:
number of time steps before/after peak, so total number of time steps will be 2 times this value
frame_time:
times in ms of each frame in the animation
name: string
the animation will be saved as name.html
size: list
[horizontal, vertical] cm's

## zoom:

\# of einstein radii plotted in vertical direction
calc_piE_ecliptic()
Parallax: Get piE_ecliptic
get_amplification $(t)$
Parallax: Get the photometric amplification term at a set of times, $t$.

## Parameters

t:
Array of times in MJD.DDD

```
get_astrometry(t_obs,ast_fil_idx=0)
```

Parallax: Get astrometry

```
get_astrometry_unlensed(t_obs)
```

Get the astrometry of the source if the lens didn't exist.

## Returns

xS_unlensed
[numpy array, dtype=float, shape $=$ len(t_obs) x 2] The unlensed positions of the source in arcseconds.
get_centroid_shift $(t$, ast_fil_idx $=0)$
Parallax: Get the centroid shift (in mas) for a list of observation times (in MJD).
get_lens_astrometry (t_obs)
Parallax: Get lens astrometry
get_log_det_covariance (t_obs, mag_obs, mag_err_obs, filt_index=0, t_pred=None)
Returns photometry with GP noise added in.
Note: This will throw an error if this is a filter with use_gp_phot[filt_index] = False.
get_photometry_with_gp(t_obs, mag_obs, mag_err_obs, filt_index=0, t_pred=None)
Returns photometry with GP noise added in.

Note: This will throw an error if this is a filter with $u s e \_$gp_phot $[$filt_index $]=$False.

```
get_resolved_amplification \((t)\)
```

Parallax: Get the photometric amplification term at a set of times, t for both the plus and minus images.

## Parameters

t:
Array of times in MJD.DDD

```
get_resolved_astrometry(t_obs)
```

Parallax: Get the $\mathrm{x}, \mathrm{y}$ astrometry for each of the two source images, which we label plus and minus.

## Returns

[xS_plus, xS_minus]
[list of numpy arrays]

- xS_plus is the vector position of the plus image.
- xS_minus is the vector position of the plus image.
log_likely_photometry (t_obs, mag_obs, mag_err_obs, fil_index=0)
Calculate the log-likelihood for the PSPL + GP model and photometric data.

Note: The GP will only be used for filters where $u s e \_g p \_p h o t\left[f i l t \_i n d e x\right]=$ True.
class model.PSPL_PhotAstrom_Par_LumLens_GP_Param1 (*args, **kwargs)
Bases: ModelClassABC, PSPL_GP, PSPL_PhotAstrom, PSPL_Parallax_LumLens, PSPL_GP_PhotAstromParam1

Helper class that provides a standard way to create an ABC using inheritance.

## Methods

| animate(tE, time_steps, frame_time, name, ...) | Produces animation of microlensing event. |
| :--- | :--- |
| calc_piE_ecliptic() | Parallax: Get piE_ecliptic |
| get_amplification(t) | Parallax: Get the photometric amplification term at a <br> set of times, t. |
| get_astrometry(t_obs[, ast_filt_idx]) | Parallax: Get astrometry |
| get_astrometry_unlensed(t_obs) | Get the astrometry of the source if the lens didn't ex- <br> ist. |
| get_centroid_shift(t[, ast_filt_idx]) | Parallax: Get the centroid shift (in mas) for a list of <br> observation times (in MJD). |
| get_geoproj_ast_params(t0par) | Parallax: Get lens astrometry |
| get_geoproj_params(t0par) | Returns photometry with GP noise added in. |
| get_lens_astrometry(t_obs) | Peturns photometry with GP noise added in. <br> set of times, t for both the plus and minus images. |
| get_log_det_covariance(t_obs, mag_obs, ...) | Parallax: Get the x, y astrometry for each of the two <br> source images, which we label plus and minus. |
| get_photometry_with_gp(t_obs, mag_obs, ...) | Calculate the log-likelihood for the PSPL + GP model <br> and photometric data. |
| get_resolved_amplification(t) |  |


| get_chi2_astrometry |  |
| :--- | :--- |
| get_chi2_photometry |  |
| get_lnL_constant |  |
| get_photometry |  |
| log_likely_astrometry |  |
| log_likely_astrometry_each |  |
| log_likely_photometry_each |  |
| start |  |

animate ( $t E$, time_steps, frame_time, name, size, zoom, astrometry)
Produces animation of microlensing event. This function takes the PSPL and makes an animation, the input variables are as follows

## Parameters

tE:
number of einstein crossings times before/after the peak you want the animation to plot
e.g tE $=2$ => graph will go from -2 tE to 2 tE
time_steps:
number of time steps before/after peak, so total number of time steps will be 2 times this value

## frame_time:

times in ms of each frame in the animation

## name: string

the animation will be saved as name.html
size: list
[horizontal, vertical] cm's
zoom:
\# of einstein radii plotted in vertical direction

## calc_piE_ecliptic()

Parallax: Get piE_ecliptic

## get_amplification $(t)$

Parallax: Get the photometric amplification term at a set of times, t .

## Parameters

t:
Array of times in MJD.DDD
get_astrometry (t_obs, ast_filt_idx=0)
Parallax: Get astrometry
get_astrometry_unlensed (t_obs)
Get the astrometry of the source if the lens didn't exist.

## Returns

xS_unlensed
[numpy array, dtype=float, shape $=$ len(t_obs) x 2] The unlensed positions of the source in arcseconds.
get_centroid_shift ( $t$, ast_filt_idx $=0$ )
Parallax: Get the centroid shift (in mas) for a list of observation times (in MJD).

```
get_lens_astrometry(t_obs)
```

Parallax: Get lens astrometry
get_log_det_covariance (t_obs, mag_obs, mag_err_obs, filt_index=0, t_pred=None)
Returns photometry with GP noise added in.

Note: This will throw an error if this is a filter with use_gp_phot[filt_index] = False.
get_photometry_with_gp(t_obs, mag_obs, mag_err_obs, filt_index=0, t_pred=None)
Returns photometry with GP noise added in.

Note: This will throw an error if this is a filter with use_gp_phot[filt_index] = False.

## get_resolved_amplification( $t$ )

Parallax: Get the photometric amplification term at a set of times, $t$ for both the plus and minus images.

## Parameters

t:
Array of times in MJD.DDD

## get_resolved_astrometry (t_obs)

Parallax: Get the x , y astrometry for each of the two source images, which we label plus and minus.

## Returns

[xS_plus, xS_minus]
[list of numpy arrays]

- xS_plus is the vector position of the plus image.
- xS_minus is the vector position of the plus image.
log_likely_photometry (t_obs, mag_obs, mag_err_obs, filt_index=0)
Calculate the log-likelihood for the PSPL + GP model and photometric data.

Note: The GP will only be used for filters where use_gp_phot[filt_index] = True.
class model.PSPL_PhotAstrom_Par_LumLens_GP_Param2 (*args, **kwargs)
Bases: ModelClassABC, PSPL_GP, PSPL_PhotAstrom, PSPL_Parallax_LumLens, PSPL_GP_PhotAstromParam2

Helper class that provides a standard way to create an ABC using inheritance.

## Methods

| animate(tE, time_steps, frame_time, name, ...) | Produces animation of microlensing event. |
| :--- | :--- |
| calc_piE_ecliptic() | Parallax: Get piE_ecliptic |
| get_amplification(t) | Parallax: Get the photometric amplification term at a <br> set of times, t. |
| get_astrometry(t_obs[, ast_filt_idx]) | Parallax: Get astrometry |
| get_astrometry_unlensed(t_obs) | Get the astrometry of the source if the lens didn't ex- <br> ist. |
| get_centroid_shift(t[, ast_filt_idx]) | Parallax: Get the centroid shift (in mas) for a list of <br> observation times (in MJD). |
| get_geoproj_ast_params(t0par) | Parallax: Get lens astrometry |
| get_geoproj_params(t0par) | Returns photometry with GP noise added in. |
| get_lens_astrometry(t_obs) | Returns photometry with GP noise added in. <br> set of times, t for both the plus and minus images. |
| get_log_det_covariance(t_obs, mag_obs, ...) | Parallax: Get the x, y astrometry for each of the two <br> source images, which we label plus and minus. |
| get_photometry_with_gp(t_obs, mag_obs, ...) | Calculate the log-likelihood for the PSPL + GP model <br> and photometric data. |
| get_resolved_amplification(t) |  |


| get_chi2_astrometry |  |
| :--- | :--- |
| get_chi2_photometry |  |
| get_lnL_constant |  |
| get_photometry |  |
| log_likely_astrometry |  |
| log_likely_astrometry_each |  |
| log_likely_photometry_each |  |
| start |  |

animate ( $t$ E, time_steps, frame_time, name, size, zoom, astrometry)
Produces animation of microlensing event. This function takes the PSPL and makes an animation, the input variables are as follows

## Parameters

tE:
number of einstein crossings times before/after the peak you want the animation to plot
e.g tE $=2=>$ graph will go from -2 tE to 2 tE
time_steps:
number of time steps before/after peak, so total number of time steps will be 2 times this value
frame_time:
times in ms of each frame in the animation
name: string
the animation will be saved as name.html
size: list
[horizontal, vertical] cm's

## zoom:

\# of einstein radii plotted in vertical direction
calc_piE_ecliptic()
Parallax: Get piE_ecliptic
get_amplification $(t)$
Parallax: Get the photometric amplification term at a set of times, $t$.

## Parameters

t:
Array of times in MJD.DDD

```
get_astrometry(t_obs,ast_fil_idx=0)
```

Parallax: Get astrometry

```
get_astrometry_unlensed(t_obs)
```

Get the astrometry of the source if the lens didn't exist.

## Returns

xS_unlensed
[numpy array, dtype=float, shape $=$ len(t_obs) x 2] The unlensed positions of the source in arcseconds.
get_centroid_shift $(t$, ast_fil_idx $=0)$
Parallax: Get the centroid shift (in mas) for a list of observation times (in MJD).
get_lens_astrometry (t_obs)
Parallax: Get lens astrometry
get_log_det_covariance (t_obs, mag_obs, mag_err_obs, filt_index=0, t_pred=None)
Returns photometry with GP noise added in.
Note: This will throw an error if this is a filter with use_gp_phot[filt_index] = False.
get_photometry_with_gp $\left(t_{-} o b s\right.$, mag_obs, mag_err_obs, fil_index $=0, t_{-}$pred=None)
Returns photometry with GP noise added in.

Note: This will throw an error if this is a filter with use_gp_phot[filt_index] = False.

```
get_resolved_amplification \((t)\)
```

Parallax: Get the photometric amplification term at a set of times, t for both the plus and minus images.

## Parameters

t:
Array of times in MJD.DDD

```
get_resolved_astrometry(t_obs)
```

Parallax: Get the $\mathrm{x}, \mathrm{y}$ astrometry for each of the two source images, which we label plus and minus.

## Returns

[xS_plus, xS_minus]
[list of numpy arrays]

- xS_plus is the vector position of the plus image.
- xS_minus is the vector position of the plus image.
log_likely_photometry (t_obs, mag_obs, mag_err_obs, fil_index=0)
Calculate the log-likelihood for the PSPL + GP model and photometric data.

Note: The GP will only be used for filters where use_gp_phot[filt_index] = True.
class model.PSPL_PhotAstrom_Par_LumLens_GP_Param3(*args, **kwargs)
Bases: ModelClassABC, PSPL_GP, PSPL_PhotAstrom, PSPL_Parallax_LumLens, PSPL_GP_PhotAstromParam3

Helper class that provides a standard way to create an ABC using inheritance.

## Methods

| animate(tE, time_steps, frame_time, name, ...) | Produces animation of microlensing event. |
| :--- | :--- |
| calc_piE_ecliptic() | Parallax: Get piE_ecliptic |
| get_amplification(t) | Parallax: Get the photometric amplification term at a <br> set of times, t. |
| get_astrometry(t_obs[, ast_filt_idx]) | Parallax: Get astrometry |
| get_astrometry_unlensed(t_obs) | Get the astrometry of the source if the lens didn't ex- <br> ist. |
| get_centroid_shift(t[, ast_filt_idx]) | Parallax: Get the centroid shift (in mas) for a list of <br> observation times (in MJD). |
| get_geoproj_ast_params(t0par) | Parallax: Get lens astrometry |
| get_geoproj_params(t0par) | Returns photometry with GP noise added in. |
| get_lens_astrometry(t_obs) | Peturns photometry with GP noise added in. <br> set of times, t for both the plus and minus images. |
| get_log_det_covariance(t_obs, mag_obs, ...) | Parallax: Get the x, y astrometry for each of the two <br> source images, which we label plus and minus. |
| get_photometry_with_gp(t_obs, mag_obs, ...) | Calculate the log-likelihood for the PSPL + GP model <br> and photometric data. |
| get_resolved_amplification(t) |  |


| get_chi2_astrometry |  |
| :--- | :--- |
| get_chi2_photometry |  |
| get_lnL_constant |  |
| get_photometry |  |
| log_likely_astrometry |  |
| log_likely_astrometry_each |  |
| log_likely_photometry_each |  |
| start |  |

animate ( $t E$, time_steps, frame_time, name, size, zoom, astrometry)
Produces animation of microlensing event. This function takes the PSPL and makes an animation, the input variables are as follows

## Parameters

tE:
number of einstein crossings times before/after the peak you want the animation to plot
e.g tE $=2$ => graph will go from -2 tE to 2 tE
time_steps:
number of time steps before/after peak, so total number of time steps will be 2 times this value

## frame_time:

times in ms of each frame in the animation

## name: string

the animation will be saved as name.html
size: list
[horizontal, vertical] cm's
zoom:
\# of einstein radii plotted in vertical direction

## calc_piE_ecliptic()

Parallax: Get piE_ecliptic

## get_amplification $(t)$

Parallax: Get the photometric amplification term at a set of times, t .

## Parameters

t:
Array of times in MJD.DDD
get_astrometry (t_obs, ast_filt_idx=0)
Parallax: Get astrometry
get_astrometry_unlensed (t_obs)
Get the astrometry of the source if the lens didn't exist.

## Returns

xS_unlensed
[numpy array, dtype=float, shape $=$ len(t_obs) x 2] The unlensed positions of the source in arcseconds.
get_centroid_shift ( $t$, ast_filt_idx $=0$ )
Parallax: Get the centroid shift (in mas) for a list of observation times (in MJD).

```
get_lens_astrometry(t_obs)
```

Parallax: Get lens astrometry
get_log_det_covariance (t_obs, mag_obs, mag_err_obs, filt_index=0, t_pred=None)
Returns photometry with GP noise added in.

Note: This will throw an error if this is a filter with use_gp_phot[filt_index] = False.
get_photometry_with_gp(t_obs, mag_obs, mag_err_obs, filt_index=0, t_pred=None)
Returns photometry with GP noise added in.

Note: This will throw an error if this is a filter with use_gp_phot[filt_index] = False.

## get_resolved_amplification( $t$ )

Parallax: Get the photometric amplification term at a set of times, t for both the plus and minus images.

## Parameters

t:
Array of times in MJD.DDD

## get_resolved_astrometry (t_obs)

Parallax: Get the x , y astrometry for each of the two source images, which we label plus and minus.

## Returns

[xS_plus, xS_minus]
[list of numpy arrays]

- xS_plus is the vector position of the plus image.
- xS_minus is the vector position of the plus image.
log_likely_photometry (t_obs, mag_obs, mag_err_obs, filt_index=0)
Calculate the log-likelihood for the PSPL + GP model and photometric data.

Note: The GP will only be used for filters where $u s e \_g p \_p h o t\left[f i l t \_i n d e x\right]=$ True .
class model.PSPL_PhotAstrom_Par_LumLens_GP_Param4 (*args, **kwargs)
Bases: ModelClassABC, PSPL_GP, PSPL_PhotAstrom, PSPL_Parallax_LumLens, PSPL_GP_PhotAstromParam4

Helper class that provides a standard way to create an ABC using inheritance.

## Methods

| animate(tE, time_steps, frame_time, name, ...) | Produces animation of microlensing event. |
| :--- | :--- |
| calc_piE_ecliptic() | Parallax: Get piE_ecliptic |
| get_amplification(t) | Parallax: Get the photometric amplification term at a <br> set of times, t. |
| get_astrometry(t_obs[, ast_filt_idx]) | Parallax: Get astrometry |
| get_astrometry_unlensed(t_obs) | Get the astrometry of the source if the lens didn't ex- <br> ist. |
| get_centroid_shift(t[, ast_filt_idx]) | Parallax: Get the centroid shift (in mas) for a list of <br> observation times (in MJD). |
| get_geoproj_ast_params(t0par) | Parallax: Get lens astrometry |
| get_geoproj_params(t0par) | Returns photometry with GP noise added in. |
| get_lens_astrometry(t_obs) | Returns photometry with GP noise added in. <br> set of times, t for both the plus and minus images. |
| get_log_det_covariance(t_obs, mag_obs, ...) | Parallax: Get the x, y astrometry for each of the two <br> source images, which we label plus and minus. |
| get_photometry_with_gp(t_obs, mag_obs, ...) | Calculate the log-likelihood for the PSPL + GP model <br> and photometric data. |
| get_resolved_astrometry(t_obs) |  |


| get_chi2_astrometry |  |
| :--- | :--- |
| get_chi2_photometry |  |
| get_lnL_constant |  |
| get_photometry |  |
| log_likely_astrometry |  |
| log_likely_astrometry_each |  |
| log_likely_photometry_each |  |
| start |  |

animate ( $t$ E, time_steps, frame_time, name, size, zoom, astrometry)
Produces animation of microlensing event. This function takes the PSPL and makes an animation, the input variables are as follows

## Parameters

tE:
number of einstein crossings times before/after the peak you want the animation to plot
e.g tE $=2=>$ graph will go from -2 tE to 2 tE
time_steps:
number of time steps before/after peak, so total number of time steps will be 2 times this value
frame_time:
times in ms of each frame in the animation
name: string
the animation will be saved as name.html
size: list
[horizontal, vertical] cm's

## zoom:

\# of einstein radii plotted in vertical direction
calc_piE_ecliptic()
Parallax: Get piE_ecliptic
get_amplification $(t)$
Parallax: Get the photometric amplification term at a set of times, $t$.

## Parameters

t:
Array of times in MJD.DDD

```
get_astrometry(t_obs,ast_fil_idx=0)
```

Parallax: Get astrometry

```
get_astrometry_unlensed(t_obs)
```

Get the astrometry of the source if the lens didn't exist.

## Returns

xS_unlensed
[numpy array, dtype=float, shape $=$ len(t_obs) x 2] The unlensed positions of the source in arcseconds.
get_centroid_shift ( $t$, ast_filt_idx=0)
Parallax: Get the centroid shift (in mas) for a list of observation times (in MJD).
get_lens_astrometry (t_obs)
Parallax: Get lens astrometry
get_log_det_covariance (t_obs, mag_obs, mag_err_obs, filt_index=0, t_pred=None)
Returns photometry with GP noise added in.
Note: This will throw an error if this is a filter with use_gp_phot[filt_index] = False.
get_photometry_with_gp(t_obs, mag_obs, mag_err_obs, filt_index=0, t_pred=None)
Returns photometry with GP noise added in.

Note: This will throw an error if this is a filter with use_gp_phot[filt_index] = False.

```
get_resolved_amplification \((t)\)
```

Parallax: Get the photometric amplification term at a set of times, t for both the plus and minus images.

## Parameters

t:
Array of times in MJD.DDD

```
get_resolved_astrometry(t_obs)
```

Parallax: Get the $\mathrm{x}, \mathrm{y}$ astrometry for each of the two source images, which we label plus and minus.

## Returns

[xS_plus, xS_minus]
[list of numpy arrays]

- xS_plus is the vector position of the plus image.
- xS_minus is the vector position of the plus image.
log_likely_photometry (t_obs, mag_obs, mag_err_obs, filt_index=0)
Calculate the log-likelihood for the PSPL + GP model and photometric data.

Note: The GP will only be used for filters where use_gp_phot[filt_index] = True.

### 4.1.9 PSPL PhotAstrom, no parallax with GP

class model.PSPL_PhotAstrom_noPar_GP_Param1(*args, **kwargs)
Bases: ModelClassABC, PSPL_GP, PSPL_PhotAstrom, PSPL_noParallax, PSPL_GP_PhotAstromParam1
Helper class that provides a standard way to create an ABC using inheritance.

## Methods

| animate(tE, time_steps, frame_time, name, ...) | Produces animation of microlensing event. |
| :--- | :--- |
| get_amplification(t) | noParallax: Get the photometric amplification term <br> at a set of times,. |
| get_astrometry(t_obs[, ast_filt_idx]) | noParallax: Position of the observed source position <br> in arcsec. |
| get_astrometry_unlensed(t_obs) | noParallax: Get the astrometry of the source if the <br> lens didn't exist. |
| get_centroid_shift(t) | noParallax: Get the centroid shift (in mas) for a list <br> of observation times (in MJD). |
| get_lens_astrometry(t_obs) | Equation of motion for just the foreground lens. |
| get_log_det_covariance(t_obs, mag_obs, ...) | Returns photometry with GP noise added in. |
| get_photometry_with_gp(t_obs, mag_obs, ...) | Returns photometry with GP noise added in. |
| get_resolved_amplification(t) | Get the photometric amplification term at a set of <br> times, t for both the plus and minus images. |
| get_resolved_astrometry(t_obs) | Get the x, y astrometry for each of the two source <br> images, which we label plus and minus. |
| log_likely_photometry(t_obs, mag_obs, ...[, ...]) | Calculate the log-likelihood for the PSPL + GP model <br> and photometric data. |


| calc_piE_ecliptic |  |
| :--- | :--- |
| get_chi2_astrometry |  |
| get_chi2_photometry |  |
| get_lnL_constant |  |
| get_photometry |  |
| log_likely_astrometry |  |
| log_likely_astrometry_each |  |
| log_likely_photometry_each |  |

animate ( $t E$, time_steps, frame_time, name, size, zoom, astrometry)
Produces animation of microlensing event. This function takes the PSPL and makes an animation, the input variables are as follows

## Parameters

tE:
number of einstein crossings times before/after the peak you want the animation to plot
e.g $\mathrm{tE}=2$ => graph will go from -2 tE to 2 tE
time_steps:
number of time steps before/after peak, so total number of time steps will be 2 times this value

## frame_time:

times in ms of each frame in the animation

## name: string

the animation will be saved as name.html
size: list
[horizontal, vertical] cm's

## zoom:

\# of einstein radii plotted in vertical direction

## get_amplification $(t)$

noParallax: Get the photometric amplification term at a set of times, t .

## Parameters

t:
Array of times in MJD.DDD
get_astrometry (t_obs, ast_filt_idx=0)
noParallax: Position of the observed source position in arcsec.
get_astrometry_unlensed(t_obs)
noParallax: Get the astrometry of the source if the lens didn't exist.

## Returns

xS_unlensed
[numpy array, dtype=float, shape $=$ len(t_obs) x 2] The unlensed positions of the source in arcseconds.

## get_centroid_shift ( $t$ )

noParallax: Get the centroid shift (in mas) for a list of observation times (in MJD).

## get_lens_astrometry (t_obs)

Equation of motion for just the foreground lens.

## Parameters

t_obs
[array_like] Time (in MJD).
get_log_det_covariance (t_obs, mag_obs, mag_err_obs, filt_index=0, t_pred=None)
Returns photometry with GP noise added in.

Note: This will throw an error if this is a filter with use_gp_phot[filt_index] = False.
get_photometry_with_gp(t_obs, mag_obs, mag_err_obs, filt_index=0, t_pred=None)
Returns photometry with GP noise added in.

Note: This will throw an error if this is a filter with $u s e \_g p \_p h o t\left[f i l t \_i n d e x\right]=$ False .

```
get_resolved_amplification(t)
```

Get the photometric amplification term at a set of times, $t$ for both the plus and minus images.

## Parameters

t:
Array of times in MJD.DDD

```
get_resolved_astrometry(t_obs)
```

Get the $\mathrm{x}, \mathrm{y}$ astrometry for each of the two source images, which we label plus and minus.

## Returns

[xS_plus, xS_minus]
[list of numpy arrays]

- xS_plus is the vector position of the plus image in arcsec
- xS_minus is the vector position of the plus image in arcsec
log_likely_photometry (t_obs, mag_obs, mag_err_obs, filt_index=0)
Calculate the log-likelihood for the PSPL + GP model and photometric data.

Note: The GP will only be used for filters where use_gp_phot[filt_index] = True.
class model.PSPL_PhotAstrom_noPar_GP_Param2 (*args, **kwargs)
Bases: ModelClassABC, PSPL_GP, PSPL_PhotAstrom, PSPL_noParallax, PSPL_GP_PhotAstromParam2
Helper class that provides a standard way to create an ABC using inheritance.

Methods

| animate(tE, time_steps, frame_time, name, ...) | Produces animation of microlensing event. |
| :--- | :--- |
| get_amplification(t) | noParallax: Get the photometric amplification term <br> at a set of times,. |
| get_astrometry(t_obs[, ast_filt_idx]) | noParallax: Position of the observed source position <br> in arcsec. |
| get_astrometry_unlensed(t_obs) | noParallax: Get the astrometry of the source if the <br> lens didn't exist. |
| get_centroid_shift(t) | noParallax: Get the centroid shift (in mas) for a list <br> of observation times (in MJD). |
| get_lens_astrometry(t_obs) | Equation of motion for just the foreground lens. |
| get_log_det_covariance(t_obs, mag_obs, ...) | Returns photometry with GP noise added in. |
| get_photometry_with_gp(t_obs, mag_obs, ...) | Returns photometry with GP noise added in. <br> get_resolved_amplification(t) |
| Get the photometric amplification term at a set of |  |
| times, t for both the plus and minus images. |  |


| calc_piE_ecliptic |  |
| :--- | :--- |
| get_chi2_astrometry |  |
| get_chi2_photometry |  |
| get_lnL_constant |  |
| get_photometry |  |
| log_likely_astrometry |  |
| log_likely_astrometry_each |  |
| log_likely_photometry_each |  |

animate ( $t$ E, time_steps, frame_time, name, size, zoom, astrometry)
Produces animation of microlensing event. This function takes the PSPL and makes an animation, the input variables are as follows

## Parameters

tE:
number of einstein crossings times before/after the peak you want the animation to plot
e.g tE $=2=>$ graph will go from -2 tE to 2 tE

## time_steps:

number of time steps before/after peak, so total number of time steps will be 2 times this value

## frame_time:

times in ms of each frame in the animation

## name: string

the animation will be saved as name.html
size: list
[horizontal, vertical] cm's

## zoom:

\# of einstein radii plotted in vertical direction

## get_amplification $(t)$

noParallax: Get the photometric amplification term at a set of times, t .

## Parameters

t:
Array of times in MJD.DDD
get_astrometry (t_obs, ast_filt_idx=0)
noParallax: Position of the observed source position in arcsec.
get_astrometry_unlensed(t_obs)
noParallax: Get the astrometry of the source if the lens didn't exist.

## Returns

xS_unlensed
[numpy array, dtype=float, shape $=$ len(t_obs) $x$ 2] The unlensed positions of the source in arcseconds.

## get_centroid_shift ( $t$ )

noParallax: Get the centroid shift (in mas) for a list of observation times (in MJD).

```
get_lens_astrometry(t_obs)
```

Equation of motion for just the foreground lens.

## Parameters

t_obs
[array_like] Time (in MJD).
get_log_det_covariance (t_obs, mag_obs, mag_err_obs, filt_index=0, t_pred=None)
Returns photometry with GP noise added in.

Note: This will throw an error if this is a filter with use_gp_phot[filt_index] = False.
get_photometry_with_gp(t_obs, mag_obs, mag_err_obs, filt_index=0, t_pred=None)
Returns photometry with GP noise added in.

Note: This will throw an error if this is a filter with use_gp_phot[fil_index] = False.

## get_resolved_amplification $(t)$

Get the photometric amplification term at a set of times, t for both the plus and minus images.

## Parameters

t:
Array of times in MJD.DDD
get_resolved_astrometry (t_obs)
Get the $\mathrm{x}, \mathrm{y}$ astrometry for each of the two source images, which we label plus and minus.

## Returns

[xS_plus, xS_minus]
[list of numpy arrays]

- xS_plus is the vector position of the plus image in arcsec
- xS_minus is the vector position of the plus image in arcsec
log_likely_photometry (t_obs, mag_obs, mag_err_obs, fil_index=0)
Calculate the log-likelihood for the PSPL + GP model and photometric data.

Note: The GP will only be used for filters where use_gp_phot[filt_index] = True.

### 4.2 PSBL User Classes

### 4.2.1 PSBL

class model.PSBL_PhotAstrom_noPar_Param1(*args, **kwargs)
Bases: ModelClassABC, PSBL_PhotAstrom, PSBL_noParallax, PSBL_PhotAstromParam1
Helper class that provides a standard way to create an ABC using inheritance.

## Methods

| animate(tE, time_steps, frame_time, name, ...) | Produces animation of microlensing event. |
| :--- | :--- |
| get_all_arrays(t_obs[, check_sols, rescale]) | Obtain the image and amplitude arrays for each t_obs. |
| get_amp_arr(z_arr, z1, z2) | Calculations amplification array |
| get_amplification(t_obs[, amp_arr]) | noParallax: Get the photometric amplification term <br> at a set of times, t. |
| get_astrometry(t_obs[, image_arr, amp_arr, ...]) | Position of the observed (unresolved) source position <br> in arcsec. |
| get_astrometry_unlensed(t_obs) | Get the astrometry of the source if the lens didn't ex- <br> ist. |
| get_centroid(t_obs[, ast_filt_idx, ...]) | PSPL: Get the centroid shift (in mas) for a list of ob- <br> servation times (in MJD). |
| get_centroid_shift(t) | noParallax: Get the centroid shift (in mas) for a list <br> of observation times (in MJD). |
| get_complex_pos(t_obs) | Get the positions of the lenses and source as complex <br> numbers. |
| get_image_pos_arr(w, z1, z2, m1, m2[,..]) | Gets image positions. |
| get_image_pos_arr_old(w, z1, z2[, check_sols]) | Gets image positions. |
| get_lens_astrometry(t_obs) | Equation of motion for just the foreground lens sys- <br> tem. |
| get_photometry(t_obs[, filt_idx, amp_arr, ...]) | Get the photometry for each of the lensed source im- <br> ages. |
| get_resolved_amplification(t) | Get the photometric amplification term at a set of <br> times, t for both the plus and minus images. |
| get_resolved_astrometry(t_obs[, image_arr, | Position of the observed source position in arcsec. <br> ...]) |
| get_resolved_lens_astrometry(t_obs) | Equation of motion for just the foreground lenses, in- <br> dividually. |
| get_resolved_photometry(t_obs[, filt_idx, ...]) | Get the photometry for each of the lensed source im- <br> ages. |
| Mescale_complex_pos(w, z1, z2) | Make sure everything is roughly centered on the ori- <br> gin in a 1 x 1 box. |


| calc_piE_ecliptic |  |
| :--- | :--- |
| get_chi2_astrometry |  |
| get_chi2_photometry |  |
| get_lnL_constant |  |
| log_likely_astrometry |  |
| log_likely_astrometry_each |  |
| log_likely_photometry |  |
| log_likely_photometry_each |  |

animate(tE, time_steps, frame_time, name, size, zoom, astrometry)
Produces animation of microlensing event. This function takes the PSPL and makes an animation, the input variables are as follows

## Parameters

tE:
number of einstein crossings times before/after the peak you want the animation to plot

```
            e.g tE = 2 => graph will go from -2 tE to 2 tE
time_steps:
    number of time steps before/after peak, so total number of time steps will be 2 times this
    value
frame_time:
    times in ms of each frame in the animation
name: string
    the animation will be saved as name.html
size: list
    [horizontal, vertical] cm's
zoom:
    # of einstein radii plotted in vertical direction
get_all_arrays(t_obs, check_sols=True, rescale=True)
Obtain the image and amplitude arrays for each t_obs.
```


## Parameters

## t_obs

```
[array_like] Array of times to model.
```


## Returns

## images

```
[array_like] Array/tuple of complex positions of each images at each t_obs.
amp_arr
[array_like] Array/tuple of amplification of each images at each t_obs.
get_amp_arr (z_arr, z1, z2)
Calculations amplification array
Calculates the amplification A from the Jacobian \(\mathrm{J}, A=1 /|J|\)
```


## Parameters

```
z_arr
[array_like]
Complex position of images. Shape \(=\) [N_times, N_solutions, 1]
- note this could be jagged.
z1
[array_like] Complex position(s) of lens 1 (primary). Shape = [N_times, 1]
z2
[array_like] Complex position(s) of lens 2 (secondary). Shape = [N_times, 1]
```


## Returns

```
amp_arr
[array_like] BLEH
get_amplification(t_obs, amp_arr=None)
noParallax: Get the photometric amplification term at a set of times, t .
```


## Parameters

```
t:
Array of times in MJD.DDD
```

get_astrometry (t_obs, image_arr=None, amp_arr=None, ast_filt_idx=0)
Position of the observed (unresolved) source position in arcsec.

## Parameters

## t_obs

[array_like] Array of times to model.

## Returns

model_pos
[array_like] Array of vector positions of the centroid at each t_obs.
get_astrometry_unlensed (t_obs)
Get the astrometry of the source if the lens didn't exist.

## Returns

xS_unlensed
[numpy array, dtype=float, shape $=$ len(t_obs) $\times 2$ ] The unlensed positions of the source in arcseconds.
get_centroid(t_obs, ast_filt_idx=0, image_arr=None, amp_arr=None)
PSPL: Get the centroid shift (in mas) for a list of observation times (in MJD).

## Parameters

## t_obs

[array or float]

## Returns

## Centroid offset on the plane of the sky in milli-arcseoncds.

## Other Parameters

ast_filt_idx
[int] Index into the photometry parameter lists for the photometry that corresponds to this astrometry data set.
image_arr
[list] List returned from PSPL get_all_arrays() used to improve efficiency.
amp_arr
[list] List returned from PSPL get_all_arrays() used to improve efficiency.

## get_centroid_shift $(t)$

noParallax: Get the centroid shift (in mas) for a list of observation times (in MJD).
get_complex_pos(t_obs)
Get the positions of the lenses and source as complex numbers. This is needed for further calculations.
Note that all units are still the same as before, this is just rewriting vectors $z=(x, y)$ as $z=x+i y$.

## Returns

w
[complex array] Source position as an array of complex numbers with real = east component, imaginary $=$ north component
z1
[complex array] Lens primary component position as an array of complex numbers with real $=$ east component, imaginary $=$ north component

## z2

[complex array] Lens secondary component position as an array of complex numbers with real $=$ east component, imaginary $=$ north component
get_image_pos_arr $(w, z 1, z 2, m 1, m 2$, check_sols=True)
Gets image positions.

Solve the fifth-order polynomial and get the image positions.
See PSBL writeup for full equations.
All angular distances are in arcsec.

## Parameters

w
[array_like] Complex position(s) of the source. Shape $=\left[\mathrm{N} \_\right.$times, 1]
z1
[array_like] Complex position(s) of lens 1 (primary). Shape $=[$ N_times, 1]
z2
[array_like] Complex position(s) of lens 2 (secondary). Shape $=\left[\mathrm{N} \_\right.$times, 1]

## check_sols

[bool, optional] If True, calculated roots are checked against the lens equation, and output will only contain those within self.root_tol. If False, all calculated roots are returned.

## Returns

z_arr
[array_like] Rank-1 array of polynomial roots, possibly complex. If check_sols = True, only roots solving the lens equation are returned.
get_image_pos_arr_old ( $w, z 1, z 2$, check_sols=True)
Gets image positions. | Solve the fifth-order polynomial and get the image positions. | See PSBL writeup for full equations. | All angular distances are in arcsec.

## Parameters

w
[array_like] Complex position(s) of the source. Shape $=\left[\mathrm{N} \_\right.$times, 1]

## z1

[array_like] Complex position(s) of lens 1 (primary). Shape $=[$ N_times, 1]
z2
[array_like] Complex position(s) of lens 2 (secondary). Shape $=\left[\mathrm{N} \_\right.$times, 1]
check_sols
[bool, optional] If True, calculated roots are checked against the lens equation, and output will only contain those within self.root_tol. If False, all calculated roots are returned.

## Returns

z_arr
[array_like] Position of the lensed source images. Rank-1 array of polynomial roots, possibly complex. If check_sols $=$ True, only roots solving the lens equation are returned.
get_lens_astrometry (t_obs)
Equation of motion for just the foreground lens system.

## Parameters

t_obs
[array_like] Time (in MJD).
get_photometry (t_obs, filt_idx=0, amp_arr=None, print_warning=True)
Get the photometry for each of the lensed source images.

## Parameters

t_obs
[array_like] Array of times to model.

## Returns

mag_model
[array_like] Magnitude of the centroid at t_obs.

```
get_resolved_amplification(t)
```

Get the photometric amplification term at a set of times, t for both the plus and minus images.

## Parameters

t:
Array of times in MJD.DDD

```
get_resolved_astrometry(t_obs, image_arr=None, amp_arr=None)
```

Position of the observed source position in arcsec.

## Parameters

t_obs
[array_like, shape $=\left[\mathrm{N} \_\right.$times $\left.]\right]$Array of times to model.

## Returns

model_pos
[array_like. shape $=[$ N_times, N_images, 2] Array of vector positions of the centroid at each t_obs.
get_resolved_lens_astrometry (t_obs)
Equation of motion for just the foreground lenses, individually.

## Parameters

## t_obs

[array_like] Time (in MJD).

```
get_resolved_photometry(t_obs, filt_idx=0, amp_arr=None, print_warning=True)
```

Get the photometry for each of the lensed source images. Implement with no blending (since we don't support different blendings for the different images).

## Parameters

t_obs
[array_like] Array of times to model.

## Returns

mag_model
[array_like] Magnitude of each lensed image centroid at t_obs. Shape = [5, len(t_obs)]
rescale_complex_pos $(w, z 1, z 2)$
Make sure everything is roughly centered on the origin in a $1 \times 1$ box.
class model.PSBL_PhotAstrom_noPar_Param2 (*args, **kwargs)
Bases: ModelClassABC, PSBL_PhotAstrom, PSBL_noParallax, PSBL_PhotAstromParam2
Helper class that provides a standard way to create an ABC using inheritance.

## Methods

| animate(tE, time_steps, frame_time, name, ...) | Produces animation of microlensing event. |
| :---: | :---: |
| get_all_arrays(t_obs[, check_sols, rescale]) | Obtain the image and amplitude arrays for each t_obs. |
| get_amp_arr(z_arr, z1, z2) | Calculations amplification array |
| get_amplification(t_obs[, amp_arr]) | noParallax: Get the photometric amplification term at a set of times, $t$. |
| get_astrometry(t_obs[, image_arr, amp_arr, ...]) | Position of the observed (unresolved) source position in arcsec. |
| get_astrometry_unlensed(t_obs) | Get the astrometry of the source if the lens didn't exist. |
| get_centroid(t_obs[, ast_filt_idx, ...]) | PSPL: Get the centroid shift (in mas) for a list of observation times (in MJD). |
| get_centroid_shift(t) | noParallax: Get the centroid shift (in mas) for a list of observation times (in MJD). |
| get_complex_pos(t_obs) | Get the positions of the lenses and source as complex numbers. |
| get_image_pos_arr(w, z1, z2, m1, m2[,...]) | Gets image positions. |
| get_image_pos_arr_old(w, z1, z2[, check_sols]) | Gets image positions. |
| get_lens_astrometry(t_obs) | Equation of motion for just the foreground lens system. |
| get_photometry(t_obs[, filt_idx, amp_arr, ...]) | Get the photometry for each of the lensed source images. |
| get_resolved_amplification(t) | Get the photometric amplification term at a set of times, t for both the plus and minus images. |
| get_resolved_astrometry(t_obs[, image_arr, ...]) | Position of the observed source position in arcsec. |
| get_resolved_lens_astrometry(t_obs) | Equation of motion for just the foreground lenses, individually. |
| get_resolved_photometry(t_obs[, filt_idx, ...]) | Get the photometry for each of the lensed source images. |
| rescale_complex_pos(w, z1, z2) | Make sure everything is roughly centered on the origin in a $1 \times 1$ box. |


| calc_piE_ecliptic |  |
| :--- | :--- |
| get_chi2_astrometry |  |
| get_chi2_photometry |  |
| get_lnL_constant |  |
| log_likely_astrometry |  |
| log_likely_astrometry_each |  |
| log_likely_photometry |  |
| log_likely_photometry_each |  |

animate ( $t$ E, time_steps, frame_time, name, size, zoom, astrometry)
Produces animation of microlensing event. This function takes the PSPL and makes an animation, the input variables are as follows

## Parameters

tE:
number of einstein crossings times before/after the peak you want the animation to plot
e.g tE $=2=>$ graph will go from -2 tE to 2 tE

## time_steps:

number of time steps before/after peak, so total number of time steps will be 2 times this value

## frame_time:

times in ms of each frame in the animation
name: string
the animation will be saved as name.html
size: list
[horizontal, vertical] cm's

## zoom:

\# of einstein radii plotted in vertical direction
get_all_arrays(t_obs, check_sols=True, rescale=True)
Obtain the image and amplitude arrays for each t_obs.

## Parameters

t_obs
[array_like] Array of times to model.

## Returns

images
[array_like] Array/tuple of complex positions of each images at each t_obs.
amp_arr
[array_like] Array/tuple of amplification of each images at each t_obs.
get_amp_arr (z_arr, z1, z2)
Calculations amplification array
Calculates the amplification A from the Jacobian $\mathrm{J}, A=1 /|J|$

## Parameters

z_arr
[array_like]
Complex position of images. Shape $=$ [N_times, N_solutions, 1]

- note this could be jagged.
z1
[array_like] Complex position(s) of lens 1 (primary). Shape = [N_times, 1]
z2
[array_like] Complex position(s) of lens 2 (secondary). Shape = [N_times, 1]


## Returns

amp_arr
[array_like] BLEH
get_amplification(t_obs,amp_arr=None)
noParallax: Get the photometric amplification term at a set of times, t .

## Parameters

t:
Array of times in MJD.DDD
get_astrometry (t_obs, image_arr=None, amp_arr=None, ast_filt_idx=0)
Position of the observed (unresolved) source position in arcsec.

## Parameters

t_obs
[array_like] Array of times to model.

## Returns

model_pos
[array_like] Array of vector positions of the centroid at each t_obs.

## get_astrometry_unlensed (t_obs)

Get the astrometry of the source if the lens didn't exist.

## Returns

xS_unlensed
[numpy array, dtype=float, shape $=$ len(t_obs) x 2] The unlensed positions of the source in arcseconds.
get_centroid(t_obs, ast_filt_idx=0, image_arr=None, amp_arr=None)
PSPL: Get the centroid shift (in mas) for a list of observation times (in MJD).

## Parameters

t_obs
[array or float]

## Returns

Centroid offset on the plane of the sky in milli-arcseoncds.

## Other Parameters

ast_filt_idx
[int] Index into the photometry parameter lists for the photometry that corresponds to this astrometry data set.
image_arr
[list] List returned from PSPL get_all_arrays() used to improve efficiency.
amp_arr
[list] List returned from PSPL get_all_arrays() used to improve efficiency.

```
get_centroid_shift(t)
```

noParallax: Get the centroid shift (in mas) for a list of observation times (in MJD).

```
get_complex_pos(t_obs)
```

Get the positions of the lenses and source as complex numbers. This is needed for further calculations.
Note that all units are still the same as before, this is just rewriting vectors $z=(x, y)$ as $z=x+i y$.

## Returns

w
[complex array] Source position as an array of complex numbers with real = east component, imaginary $=$ north component
z1
[complex array] Lens primary component position as an array of complex numbers with real $=$ east component, imaginary $=$ north component
z2
[complex array] Lens secondary component position as an array of complex numbers with real = east component, imaginary = north component

```
get_image_pos_arr(w,zl,z2,ml,m2,check_sols=True)
```

Gets image positions.

Solve the fifth-order polynomial and get the image positions.
See PSBL writeup for full equations.
All angular distances are in arcsec.

## Parameters

w
[array_like] Complex position(s) of the source. Shape = [N_times, 1]
z1
[array_like] Complex position(s) of lens 1 (primary). Shape $=[$ N_times, 1]
z2
[array_like] Complex position(s) of lens 2 (secondary). Shape $=[$ N_times, 1]
check_sols
[bool, optional] If True, calculated roots are checked against the lens equation, and output will only contain those within self.root_tol. If False, all calculated roots are returned.

## Returns

z_arr
[array_like] Rank-1 array of polynomial roots, possibly complex. If check_sols = True, only roots solving the lens equation are returned.
get_image_pos_arr_old ( $w, z 1, z 2$, check_sols=True)
Gets image positions. | Solve the fifth-order polynomial and get the image positions. | See PSBL writeup for full equations. | All angular distances are in arcsec.

## Parameters

w
[array_like] Complex position(s) of the source. Shape $=[\mathrm{N}$ _times, 1]
z1
[array_like] Complex position(s) of lens 1 (primary). Shape $=[$ N_times, 1]
z2
[array_like] Complex position(s) of lens 2 (secondary). Shape = [N_times, 1]
check_sols
[bool, optional] If True, calculated roots are checked against the lens equation, and output will only contain those within self.root_tol. If False, all calculated roots are returned.

## Returns

z_arr
[array_like] Position of the lensed source images. Rank-1 array of polynomial roots, possibly complex. If check_sols $=$ True, only roots solving the lens equation are returned.

## get_lens_astrometry (t_obs)

Equation of motion for just the foreground lens system.

## Parameters

t_obs
[array_like] Time (in MJD).
get_photometry (t_obs, filt_idx=0, amp_arr=None, print_warning=True)
Get the photometry for each of the lensed source images.

## Parameters

```
t_obs
                    [array_like] Array of times to model.
```


## Returns

mag_model
[array_like] Magnitude of the centroid at t_obs.
get_resolved_amplification ( $t$ )
Get the photometric amplification term at a set of times, t for both the plus and minus images.

## Parameters

t:
Array of times in MJD.DDD
get_resolved_astrometry (t_obs, image_arr=None, amp_arr=None)
Position of the observed source position in arcsec.

## Parameters

t_obs
[array_like, shape $=[$ N_times $]]$ Array of times to model.

## Returns

model_pos
[array_like. shape $=\left[\mathrm{N} \_\right.$times, $\left.\left.\mathrm{N} \_i m a g e s, 2\right]\right]$ Array of vector positions of the centroid at each t_obs.

## get_resolved_lens_astrometry (t_obs)

Equation of motion for just the foreground lenses, individually.

## Parameters

t_obs
[array_like] Time (in MJD).
get_resolved_photometry(t_obs, filt_idx=0, amp_arr=None, print_warning=True)
Get the photometry for each of the lensed source images. Implement with no blending (since we don't support different blendings for the different images).

## Parameters

t_obs
[array_like] Array of times to model.

## Returns

mag_model
[array_like] Magnitude of each lensed image centroid at t_obs. Shape $=[5$, len(t_obs)]
rescale_complex_pos $(w, z l, z 2)$
Make sure everything is roughly centered on the origin in a $1 \times 1$ box.

### 4.2.2 PSBL Parallax

class model.PSBL_PhotAstrom_Par_Param1(*args, **kwargs)
Bases: ModelClassABC, PSBL_PhotAstrom, PSBL_Parallax, PSBL_PhotAstromParam1
Helper class that provides a standard way to create an ABC using inheritance.

## Methods

| animate(tE, time_steps, frame_time, name, ...) | Produces animation of microlensing event. |
| :---: | :---: |
| calc_piE_ecliptic() | Parallax: Get piE_ecliptic |
| get_all_arrays(t_obs[, check_sols, rescale]) | Obtain the image and amplitude arrays for each t_obs. |
| get_amp_arr(z_arr, z1, z2) | Calculations amplification array |
| get_amplification(t_obs[, amp_arr]) | noParallax: Get the photometric amplification term at a set of times, $t$. |
| get_astrometry(t_obs[, image_arr, amp_arr, ...]) | Position of the observed (unresolved) source position in arcsec. |
| get_astrometry_unlensed(t_obs) | Get the astrometry of the source if the lens didn't exist. |
| get_centroid(t_obs[, ast_filt_idx, ...]) | PSPL: Get the centroid shift (in mas) for a list of observation times (in MJD). |
| get_centroid_shift(t[, ast_filt_idx]) | Parallax: Get the centroid shift (in mas) for a list of observation times (in MJD). |
| get_complex_pos(t_obs) | Get the positions of the lenses and source as complex numbers. |
| get_geoproj_ast_params(t0par) |  |
| get_geoproj_params(t0par) |  |
| get_image_pos_arr(w, z1, z2, m1, m2[,...]) | Gets image positions. |
| get_image_pos_arr_old(w, z1, z2[, check_sols]) | Gets image positions. |
| get_lens_astrometry(t_obs) | Equation of motion for just the foreground lens system. |
| get_photometry(t_obs[, filt_idx, amp_arr, ...]) | Get the photometry for each of the lensed source images. |
| get_resolved_amplification(t) | Parallax: Get the photometric amplification term at a set of times, t for both the plus and minus images. |
| get_resolved_astrometry(t_obs[, image_arr, ...]) | Position of the observed source position in arcsec. |
| get_resolved_lens_astrometry(t_obs) | Equation of motion for just the foreground lenses, individually. |
| get_resolved_photometry(t_obs[, filt_idx, ...]) | Get the photometry for each of the lensed source images. |
| rescale_complex_pos(w, z1, z2) | Make sure everything is roughly centered on the origin in a $1 \times 1$ box. |


| get_chi2_astrometry |  |
| :--- | :--- |
| get_chi2_photometry |  |
| get_lnL_constant |  |
| log_likely_astrometry |  |
| log_likely_astrometry_each |  |
| log_likely_photometry |  |
| log_likely_photometry_each |  |
| start |  |

animate ( $t$ E, time_steps, frame_time, name, size, zoom, astrometry)
Produces animation of microlensing event. This function takes the PSPL and makes an animation, the input variables are as follows

## Parameters

tE:
number of einstein crossings times before/after the peak you want the animation to plot
e.g tE $=2=>$ graph will go from -2 tE to 2 tE

## time_steps:

number of time steps before/after peak, so total number of time steps will be 2 times this value
frame_time:
times in ms of each frame in the animation
name: string
the animation will be saved as name.html
size: list
[horizontal, vertical] cm's
zoom:
\# of einstein radii plotted in vertical direction
calc_piE_ecliptic()
Parallax: Get piE_ecliptic
get_all_arrays(t_obs, check_sols=True, rescale=True)
Obtain the image and amplitude arrays for each t_obs.

## Parameters

## t_obs

[array_like] Array of times to model.

## Returns

images
[array_like] Array/tuple of complex positions of each images at each t_obs.
amp_arr
[array_like] Array/tuple of amplification of each images at each t_obs.
get_amp_arr (z_arr, z1, z2)
Calculations amplification array
Calculates the amplification A from the Jacobian J, $A=1 /|J|$

## Parameters

z_arr
[array_like]
Complex position of images. Shape $=$ [N_times, N_solutions, 1]

- note this could be jagged.
z1
[array_like] Complex position(s) of lens 1 (primary). Shape = [N_times, 1]
z2
[array_like] Complex position(s) of lens 2 (secondary). Shape = [N_times, 1]


## Returns

amp_arr
[array_like] BLEH
get_amplification(t_obs, amp_arr=None)
noParallax: Get the photometric amplification term at a set of times, t .

## Parameters

t:
Array of times in MJD.DDD
get_astrometry (t_obs, image_arr=None, amp_arr=None, ast_filt_idx=0)
Position of the observed (unresolved) source position in arcsec.

## Parameters

t_obs
[array_like] Array of times to model.

## Returns

model_pos
[array_like] Array of vector positions of the centroid at each t_obs.

## get_astrometry_unlensed (t_obs)

Get the astrometry of the source if the lens didn't exist.

## Returns

xS_unlensed
[numpy array, dtype=float, shape $=$ len(t_obs) x 2] The unlensed positions of the source in arcseconds.
get_centroid(t_obs, ast_filt_idx=0, image_arr=None, amp_arr=None)
PSPL: Get the centroid shift (in mas) for a list of observation times (in MJD).

## Parameters

t_obs
[array or float]

## Returns

Centroid offset on the plane of the sky in milli-arcseoncds.

## Other Parameters

ast_filt_idx
[int] Index into the photometry parameter lists for the photometry that corresponds to this astrometry data set.

## image_arr

[list] List returned from PSPL get_all_arrays() used to improve efficiency.
amp_arr
[list] List returned from PSPL get_all_arrays() used to improve efficiency.

```
get_centroid_shift (t, ast_filt_idx=0)
```

Parallax: Get the centroid shift (in mas) for a list of observation times (in MJD).

```
get_complex_pos(t_obs)
```

Get the positions of the lenses and source as complex numbers. This is needed for further calculations.
Note that all units are still the same as before, this is just rewriting vectors $z=(x, y)$ as $z=x+i y$.

## Returns

w
[complex array] Source position as an array of complex numbers with real = east component, imaginary $=$ north component
z1
[complex array] Lens primary component position as an array of complex numbers with real $=$ east component, imaginary $=$ north component
z2
[complex array] Lens secondary component position as an array of complex numbers with real = east component, imaginary = north component
get_image_pos_arr( $w, z 1, z 2, m 1, m 2$, check_sols=True)
Gets image positions.

Solve the fifth-order polynomial and get the image positions.
See PSBL writeup for full equations.
All angular distances are in arcsec.

## Parameters

w
[array_like] Complex position(s) of the source. Shape = [N_times, 1]
z1
[array_like] Complex position(s) of lens 1 (primary). Shape $=[$ N_times, 1]
z2
[array_like] Complex position(s) of lens 2 (secondary). Shape $=[$ N_times, 1]
check_sols
[bool, optional] If True, calculated roots are checked against the lens equation, and output will only contain those within self.root_tol. If False, all calculated roots are returned.

## Returns

z_arr
[array_like] Rank-1 array of polynomial roots, possibly complex. If check_sols = True, only roots solving the lens equation are returned.
get_image_pos_arr_old $(w, z 1, z 2$, check_sols=True)
Gets image positions. | Solve the fifth-order polynomial and get the image positions. | See PSBL writeup for full equations. | All angular distances are in arcsec.

## Parameters

w
[array_like] Complex position(s) of the source. Shape $=[\mathrm{N}$ _times, 1]
z1
[array_like] Complex position(s) of lens 1 (primary). Shape $=[$ N_times, 1]
z2
[array_like] Complex position(s) of lens 2 (secondary). Shape = [N_times, 1]
check_sols
[bool, optional] If True, calculated roots are checked against the lens equation, and output will only contain those within self.root_tol. If False, all calculated roots are returned.

## Returns

z_arr
[array_like] Position of the lensed source images. Rank-1 array of polynomial roots, possibly complex. If check_sols $=$ True, only roots solving the lens equation are returned.

## get_lens_astrometry (t_obs)

Equation of motion for just the foreground lens system.

## Parameters

t_obs
[array_like] Time (in MJD).
get_photometry (t_obs, filt_idx=0, amp_arr=None, print_warning=True)
Get the photometry for each of the lensed source images.

## Parameters

t_obs
[array_like] Array of times to model.

## Returns

mag_model
[array_like] Magnitude of the centroid at t_obs.
get_resolved_amplification ( $t$ )
Parallax: Get the photometric amplification term at a set of times, t for both the plus and minus images.

## Parameters

t:
Array of times in MJD.DDD
get_resolved_astrometry (t_obs, image_arr=None, amp_arr=None)
Position of the observed source position in arcsec.

## Parameters

t_obs
[array_like, shape $=[$ N_times $]$ ] Array of times to model.

## Returns

model_pos
[array_like. shape $=\left[\mathrm{N} \_\right.$times, $\left.\left.\mathrm{N} \_i m a g e s, 2\right]\right]$ Array of vector positions of the centroid at each t_obs.

## get_resolved_lens_astrometry (t_obs)

Equation of motion for just the foreground lenses, individually.

## Parameters

t_obs
[array_like] Time (in MJD).
get_resolved_photometry (t_obs, filt_idx=0, amp_arr=None, print_warning=True)
Get the photometry for each of the lensed source images. Implement with no blending (since we don't support different blendings for the different images).

## Parameters

t_obs
[array_like] Array of times to model.

## Returns

mag_model
[array_like] Magnitude of each lensed image centroid at t_obs. Shape $=[5$, len(t_obs)]
rescale_complex_pos $(w, z l, z 2)$
Make sure everything is roughly centered on the origin in a $1 \times 1$ box.
class model.PSBL_PhotAstrom_Par_Param2 (*args, **kwargs)
Bases: ModelClassABC, PSBL_PhotAstrom, PSBL_Parallax, PSBL_PhotAstromParam2
Helper class that provides a standard way to create an ABC using inheritance.

## Methods

| animate(tE, time_steps, frame_time, name, ...) | Produces animation of microlensing event. |
| :---: | :---: |
| calc_piE_ecliptic() | Parallax: Get piE_ecliptic |
| get_all_arrays(t_obs[, check_sols, rescale]) | Obtain the image and amplitude arrays for each t_obs. |
| get_amp_arr(z_arr, z1, z2) | Calculations amplification array |
| get_amplification(t_obs[, amp_arr]) | noParallax: Get the photometric amplification term at a set of times, $t$. |
| get_astrometry(t_obs[, image_arr, amp_arr, ...]) | Position of the observed (unresolved) source position in arcsec. |
| get_astrometry_unlensed(t_obs) | Get the astrometry of the source if the lens didn't exist. |
| get_centroid(t_obs[, ast_filt_idx, ...]) | PSPL: Get the centroid shift (in mas) for a list of observation times (in MJD). |
| get_centroid_shift(t[, ast_filt_idx]) | Parallax: Get the centroid shift (in mas) for a list of observation times (in MJD). |
| get_complex_pos(t_obs) | Get the positions of the lenses and source as complex numbers. |
| get_geoproj_ast_params(t0par) |  |
| get_geoproj_params(t0par) |  |
| get_image_pos_arr(w, z1, z2, m1, m2[, ...]) | Gets image positions. |
| get_image_pos_arr_old(w, z1, z2[, check_sols]) | Gets image positions. |
| get_lens_astrometry(t_obs) | Equation of motion for just the foreground lens system. |
| get_photometry(t_obs[, filt_idx, amp_arr, ...]) | Get the photometry for each of the lensed source images. |
| get_resolved_amplification(t) | Parallax: Get the photometric amplification term at a set of times, t for both the plus and minus images. |
| get_resolved_astrometry(t_obs[, image_arr, ...]) | Position of the observed source position in arcsec. |
| get_resolved_lens_astrometry(t_obs) | Equation of motion for just the foreground lenses, individually. |
| get_resolved_photometry(t_obs[, filt_idx, ...]) | Get the photometry for each of the lensed source images. |
| rescale_complex_pos(w, z1, z2) | Make sure everything is roughly centered on the origin in a $1 \times 1$ box. |


| get_chi2_astrometry |  |
| :--- | :--- |
| get_chi2_photometry |  |
| get_lnL_constant |  |
| log_likely_astrometry |  |
| log_likely_astrometry_each |  |
| log_likely_photometry |  |
| log_likely_photometry_each |  |
| start |  |

animate ( $t$ E, time_steps, frame_time, name, size, zoom, astrometry)
Produces animation of microlensing event. This function takes the PSPL and makes an animation, the input variables are as follows

## Parameters

tE:
number of einstein crossings times before/after the peak you want the animation to plot
e.g tE $=2=>$ graph will go from -2 tE to 2 tE
time_steps:
number of time steps before/after peak, so total number of time steps will be 2 times this value
frame_time:
times in ms of each frame in the animation
name: string
the animation will be saved as name.html
size: list
[horizontal, vertical] cm's
zoom:
\# of einstein radii plotted in vertical direction
calc_piE_ecliptic()
Parallax: Get piE_ecliptic
get_all_arrays(t_obs, check_sols=True, rescale=True)
Obtain the image and amplitude arrays for each $t$ _obs.

## Parameters

## t_obs

[array_like] Array of times to model.

## Returns

images
[array_like] Array/tuple of complex positions of each images at each t_obs.
amp_arr
[array_like] Array/tuple of amplification of each images at each t_obs.
get_amp_arr (z_arr, z1, z2)
Calculations amplification array
Calculates the amplification A from the Jacobian $\mathrm{J}, A=1 /|J|$

## Parameters

z_arr
[array_like]
Complex position of images. Shape $=$ [N_times, N_solutions, 1]

- note this could be jagged.
z1
[array_like] Complex position(s) of lens 1 (primary). Shape = [N_times, 1]
z2
[array_like] Complex position(s) of lens 2 (secondary). Shape = [N_times, 1]


## Returns

amp_arr
[array_like] BLEH
get_amplification(t_obs, amp_arr=None)
noParallax: Get the photometric amplification term at a set of times, t .

## Parameters

t:
Array of times in MJD.DDD
get_astrometry (t_obs, image_arr=None, amp_arr=None, ast_filt_idx=0)
Position of the observed (unresolved) source position in arcsec.

## Parameters

t_obs
[array_like] Array of times to model.

## Returns

model_pos
[array_like] Array of vector positions of the centroid at each t_obs.

## get_astrometry_unlensed (t_obs)

Get the astrometry of the source if the lens didn't exist.

## Returns

xS_unlensed
[numpy array, dtype=float, shape $=$ len(t_obs) x 2] The unlensed positions of the source in arcseconds.
get_centroid(t_obs, ast_filt_idx=0, image_arr=None, amp_arr=None)
PSPL: Get the centroid shift (in mas) for a list of observation times (in MJD).

## Parameters

t_obs
[array or float]

## Returns

Centroid offset on the plane of the sky in milli-arcseoncds.

## Other Parameters

ast_filt_idx
[int] Index into the photometry parameter lists for the photometry that corresponds to this astrometry data set.

## image_arr

[list] List returned from PSPL get_all_arrays() used to improve efficiency.
amp_arr
[list] List returned from PSPL get_all_arrays() used to improve efficiency.
get_centroid_shift ( $t$, ast_filt_idx=0)
Parallax: Get the centroid shift (in mas) for a list of observation times (in MJD).

```
get_complex_pos(t_obs)
```

Get the positions of the lenses and source as complex numbers. This is needed for further calculations.
Note that all units are still the same as before, this is just rewriting vectors $z=(x, y)$ as $z=x+i y$.

## Returns

w
[complex array] Source position as an array of complex numbers with real = east component, imaginary $=$ north component
z1
[complex array] Lens primary component position as an array of complex numbers with real $=$ east component, imaginary $=$ north component
z2
[complex array] Lens secondary component position as an array of complex numbers with real = east component, imaginary = north component

```
get_image_pos_arr(w,zl,z2,ml,m2,check_sols=True)
```

Gets image positions.

Solve the fifth-order polynomial and get the image positions.
See PSBL writeup for full equations.
All angular distances are in arcsec.

## Parameters

w
[array_like] Complex position(s) of the source. Shape = [N_times, 1]
z1
[array_like] Complex position(s) of lens 1 (primary). Shape $=[$ N_times, 1]
z2
[array_like] Complex position(s) of lens 2 (secondary). Shape $=[$ N_times, 1]
check_sols
[bool, optional] If True, calculated roots are checked against the lens equation, and output will only contain those within self.root_tol. If False, all calculated roots are returned.

## Returns

z_arr
[array_like] Rank-1 array of polynomial roots, possibly complex. If check_sols = True, only roots solving the lens equation are returned.
get_image_pos_arr_old ( $w, z 1, z 2$, check_sols=True)
Gets image positions. | Solve the fifth-order polynomial and get the image positions. | See PSBL writeup for full equations. | All angular distances are in arcsec.

## Parameters

w
[array_like] Complex position(s) of the source. Shape $=[\mathrm{N}$ _times, 1]
z1
[array_like] Complex position(s) of lens 1 (primary). Shape $=[$ N_times, 1]
z2
[array_like] Complex position(s) of lens 2 (secondary). Shape = [N_times, 1]
check_sols
[bool, optional] If True, calculated roots are checked against the lens equation, and output will only contain those within self.root_tol. If False, all calculated roots are returned.

## Returns

z_arr
[array_like] Position of the lensed source images. Rank-1 array of polynomial roots, possibly complex. If check_sols $=$ True, only roots solving the lens equation are returned.

## get_lens_astrometry (t_obs)

Equation of motion for just the foreground lens system.

## Parameters

t_obs
[array_like] Time (in MJD).
get_photometry (t_obs, filt_idx=0, amp_arr=None, print_warning=True)
Get the photometry for each of the lensed source images.

## Parameters

t_obs
[array_like] Array of times to model.

## Returns

mag_model
[array_like] Magnitude of the centroid at t_obs.
get_resolved_amplification ( $t$ )
Parallax: Get the photometric amplification term at a set of times, t for both the plus and minus images.

## Parameters

t:
Array of times in MJD.DDD
get_resolved_astrometry (t_obs, image_arr=None, amp_arr=None)
Position of the observed source position in arcsec.

## Parameters

t_obs
[array_like, shape $=[$ N_times $]$ ] Array of times to model.

## Returns

model_pos
[array_like. shape $=\left[\mathrm{N} \_\right.$times, $\left.\left.\mathrm{N} \_i m a g e s, 2\right]\right]$ Array of vector positions of the centroid at each t_obs.

## get_resolved_lens_astrometry (t_obs)

Equation of motion for just the foreground lenses, individually.

## Parameters

t_obs
[array_like] Time (in MJD).
get_resolved_photometry (t_obs, filt_idx=0, amp_arr=None, print_warning=True)
Get the photometry for each of the lensed source images. Implement with no blending (since we don't support different blendings for the different images).

## Parameters

t_obs
[array_like] Array of times to model.

## Returns

mag_model
[array_like] Magnitude of each lensed image centroid at t_obs. Shape $=[5$, len(t_obs)]
rescale_complex_pos $(w, z l, z 2)$
Make sure everything is roughly centered on the origin in a $1 \times 1$ box.
class model.PSBL_PhotAstrom_Par_Param3(*args, **kwargs)
Bases: ModelClassABC, PSBL_PhotAstrom, PSBL_Parallax, PSBL_PhotAstromParam3
Helper class that provides a standard way to create an ABC using inheritance.

## Methods

| animate(tE, time_steps, frame_time, name, ...) | Produces animation of microlensing event. |
| :---: | :---: |
| calc_piE_ecliptic() | Parallax: Get piE_ecliptic |
| get_all_arrays(t_obs[, check_sols, rescale]) | Obtain the image and amplitude arrays for each t_obs. |
| get_amp_arr(z_arr, z1, z2) | Calculations amplification array |
| get_amplification(t_obs[, amp_arr]) | noParallax: Get the photometric amplification term at a set of times, $t$. |
| get_astrometry(t_obs[, image_arr, amp_arr, ...]) | Position of the observed (unresolved) source position in arcsec. |
| get_astrometry_unlensed(t_obs) | Get the astrometry of the source if the lens didn't exist. |
| get_centroid(t_obs[, ast_filt_idx, ...]) | PSPL: Get the centroid shift (in mas) for a list of observation times (in MJD). |
| get_centroid_shift(t[, ast_filt_idx]) | Parallax: Get the centroid shift (in mas) for a list of observation times (in MJD). |
| get_complex_pos(t_obs) | Get the positions of the lenses and source as complex numbers. |
| get_geoproj_ast_params(t0par) |  |
| get_geoproj_params(t0par) |  |
| get_image_pos_arr(w, z1, z2, m1, m2[, ...]) | Gets image positions. |
| get_image_pos_arr_old(w, z1, z2[, check_sols]) | Gets image positions. |
| get_lens_astrometry(t_obs) | Equation of motion for just the foreground lens system. |
| get_photometry(t_obs[, filt_idx, amp_arr, ...]) | Get the photometry for each of the lensed source images. |
| get_resolved_amplification(t) | Parallax: Get the photometric amplification term at a set of times, t for both the plus and minus images. |
| get_resolved_astrometry(t_obs[, image_arr, ...]) | Position of the observed source position in arcsec. |
| get_resolved_lens_astrometry(t_obs) | Equation of motion for just the foreground lenses, individually. |
| get_resolved_photometry(t_obs[, filt_idx, ...]) | Get the photometry for each of the lensed source images. |
| rescale_complex_pos(w, z1, z2) | Make sure everything is roughly centered on the origin in a $1 \times 1$ box. |


| get_chi2_astrometry |  |
| :--- | :--- |
| get_chi2_photometry |  |
| get_lnL_constant |  |
| log_likely_astrometry |  |
| log_likely_astrometry_each |  |
| log_likely_photometry |  |
| log_likely_photometry_each |  |
| start |  |

animate ( $t$ E, time_steps, frame_time, name, size, zoom, astrometry)
Produces animation of microlensing event. This function takes the PSPL and makes an animation, the input variables are as follows

## Parameters

tE:
number of einstein crossings times before/after the peak you want the animation to plot
e.g tE $=2=>$ graph will go from -2 tE to 2 tE
time_steps:
number of time steps before/after peak, so total number of time steps will be 2 times this value
frame_time:
times in ms of each frame in the animation
name: string
the animation will be saved as name.html
size: list
[horizontal, vertical] cm's
zoom:
\# of einstein radii plotted in vertical direction
calc_piE_ecliptic()
Parallax: Get piE_ecliptic
get_all_arrays(t_obs, check_sols=True, rescale=True)
Obtain the image and amplitude arrays for each t_obs.

## Parameters

## t_obs

[array_like] Array of times to model.

## Returns

images
[array_like] Array/tuple of complex positions of each images at each t_obs.
amp_arr
[array_like] Array/tuple of amplification of each images at each t_obs.
get_amp_arr (z_arr, z1, z2)
Calculations amplification array
Calculates the amplification A from the Jacobian $\mathrm{J}, A=1 /|J|$

## Parameters

z_arr
[array_like]
Complex position of images. Shape $=$ [N_times, N_solutions, 1]

- note this could be jagged.
z1
[array_like] Complex position(s) of lens 1 (primary). Shape = [N_times, 1]
z2
[array_like] Complex position(s) of lens 2 (secondary). Shape = [N_times, 1]


## Returns

amp_arr
[array_like] BLEH
get_amplification(t_obs,amp_arr=None)
noParallax: Get the photometric amplification term at a set of times, t .

## Parameters

t:
Array of times in MJD.DDD
get_astrometry (t_obs, image_arr=None, amp_arr=None, ast_filt_idx=0)
Position of the observed (unresolved) source position in arcsec.

## Parameters

t_obs
[array_like] Array of times to model.

## Returns

model_pos
[array_like] Array of vector positions of the centroid at each t_obs.

## get_astrometry_unlensed (t_obs)

Get the astrometry of the source if the lens didn't exist.

## Returns

xS_unlensed
[numpy array, dtype=float, shape $=$ len(t_obs) x 2] The unlensed positions of the source in arcseconds.
get_centroid(t_obs, ast_filt_idx=0, image_arr=None, amp_arr=None)
PSPL: Get the centroid shift (in mas) for a list of observation times (in MJD).

## Parameters

t_obs
[array or float]

## Returns

Centroid offset on the plane of the sky in milli-arcseoncds.

## Other Parameters

ast_filt_idx
[int] Index into the photometry parameter lists for the photometry that corresponds to this astrometry data set.

## image_arr

[list] List returned from PSPL get_all_arrays() used to improve efficiency.
amp_arr
[list] List returned from PSPL get_all_arrays() used to improve efficiency.
get_centroid_shift ( $t$, ast_filt_idx=0)
Parallax: Get the centroid shift (in mas) for a list of observation times (in MJD).

```
get_complex_pos(t_obs)
```

Get the positions of the lenses and source as complex numbers. This is needed for further calculations.
Note that all units are still the same as before, this is just rewriting vectors $z=(x, y)$ as $z=x+i y$.

## Returns

w
[complex array] Source position as an array of complex numbers with real = east component, imaginary $=$ north component
z1
[complex array] Lens primary component position as an array of complex numbers with real $=$ east component, imaginary $=$ north component
z2
[complex array] Lens secondary component position as an array of complex numbers with real = east component, imaginary = north component

```
get_image_pos_arr(w,zl,z2,ml,m2,check_sols=True)
```

Gets image positions.

Solve the fifth-order polynomial and get the image positions.
See PSBL writeup for full equations.
All angular distances are in arcsec.

## Parameters

w
[array_like] Complex position(s) of the source. Shape = [N_times, 1]
z1
[array_like] Complex position(s) of lens 1 (primary). Shape $=[$ N_times, 1]
z2
[array_like] Complex position(s) of lens 2 (secondary). Shape $=[$ N_times, 1]
check_sols
[bool, optional] If True, calculated roots are checked against the lens equation, and output will only contain those within self.root_tol. If False, all calculated roots are returned.

## Returns

z_arr
[array_like] Rank-1 array of polynomial roots, possibly complex. If check_sols = True, only roots solving the lens equation are returned.
get_image_pos_arr_old ( $w, z 1, z 2$, check_sols=True)
Gets image positions. | Solve the fifth-order polynomial and get the image positions. | See PSBL writeup for full equations. | All angular distances are in arcsec.

## Parameters

w
[array_like] Complex position(s) of the source. Shape $=[\mathrm{N}$ _times, 1]
z1
[array_like] Complex position(s) of lens 1 (primary). Shape $=[$ N_times, 1]
z2
[array_like] Complex position(s) of lens 2 (secondary). Shape = [N_times, 1]
check_sols
[bool, optional] If True, calculated roots are checked against the lens equation, and output will only contain those within self.root_tol. If False, all calculated roots are returned.

## Returns

z_arr
[array_like] Position of the lensed source images. Rank-1 array of polynomial roots, possibly complex. If check_sols $=$ True, only roots solving the lens equation are returned.

## get_lens_astrometry (t_obs)

Equation of motion for just the foreground lens system.

## Parameters

t_obs
[array_like] Time (in MJD).
get_photometry (t_obs, filt_idx=0, amp_arr=None, print_warning=True)
Get the photometry for each of the lensed source images.

## Parameters

t_obs
[array_like] Array of times to model.

## Returns

mag_model
[array_like] Magnitude of the centroid at t_obs.
get_resolved_amplification ( $t$ )
Parallax: Get the photometric amplification term at a set of times, t for both the plus and minus images.

## Parameters

t:
Array of times in MJD.DDD
get_resolved_astrometry (t_obs, image_arr=None, amp_arr=None)
Position of the observed source position in arcsec.

## Parameters

t_obs
[array_like, shape $=[$ N_times $]$ ] Array of times to model.

## Returns

model_pos
[array_like. shape $=\left[\mathrm{N} \_\right.$times, $\left.\left.\mathrm{N} \_i m a g e s, 2\right]\right]$ Array of vector positions of the centroid at each t_obs.

## get_resolved_lens_astrometry (t_obs)

Equation of motion for just the foreground lenses, individually.

## Parameters

t_obs
[array_like] Time (in MJD).
get_resolved_photometry (t_obs, filt_idx=0, amp_arr=None, print_warning=True)
Get the photometry for each of the lensed source images. Implement with no blending (since we don't support different blendings for the different images).

## Parameters

t_obs
[array_like] Array of times to model.

## Returns

mag_model
[array_like] Magnitude of each lensed image centroid at t_obs. Shape $=[5$, len(t_obs)]
rescale_complex_pos $(w, z l, z 2)$
Make sure everything is roughly centered on the origin in a $1 \times 1$ box.
class model.PSBL_PhotAstrom_Par_Param4(*args, **kwargs)
Bases: ModelClassABC, PSBL_PhotAstrom, PSBL_Parallax, PSBL_PhotAstromParam4
Helper class that provides a standard way to create an ABC using inheritance.

## Methods

| animate(tE, time_steps, frame_time, name, ...) | Produces animation of microlensing event. |
| :---: | :---: |
| calc_piE_ecliptic() | Parallax: Get piE_ecliptic |
| get_all_arrays(t_obs[, check_sols, rescale]) | Obtain the image and amplitude arrays for each t_obs. |
| get_amp_arr(z_arr, z1, z2) | Calculations amplification array |
| get_amplification(t_obs[, amp_arr]) | noParallax: Get the photometric amplification term at a set of times, $t$. |
| get_astrometry(t_obs[, image_arr, amp_arr, ...]) | Position of the observed (unresolved) source position in arcsec. |
| get_astrometry_unlensed(t_obs) | Get the astrometry of the source if the lens didn't exist. |
| get_centroid(t_obs[, ast_filt_idx, ...]) | PSPL: Get the centroid shift (in mas) for a list of observation times (in MJD). |
| get_centroid_shift(t[, ast_filt_idx]) | Parallax: Get the centroid shift (in mas) for a list of observation times (in MJD). |
| get_complex_pos(t_obs) | Get the positions of the lenses and source as complex numbers. |
| get_geoproj_ast_params(t0par) |  |
| get_geoproj_params(t0par) |  |
| get_image_pos_arr(w, z1, z2, m1, m2[, ...]) | Gets image positions. |
| get_image_pos_arr_old(w, z1, z2[, check_sols]) | Gets image positions. |
| get_lens_astrometry(t_obs) | Equation of motion for just the foreground lens system. |
| get_photometry(t_obs[, filt_idx, amp_arr, ...]) | Get the photometry for each of the lensed source images. |
| get_resolved_amplification(t) | Parallax: Get the photometric amplification term at a set of times, t for both the plus and minus images. |
| get_resolved_astrometry(t_obs[, image_arr, ...]) | Position of the observed source position in arcsec. |
| get_resolved_lens_astrometry(t_obs) | Equation of motion for just the foreground lenses, individually. |
| get_resolved_photometry(t_obs[, filt_idx, ...]) | Get the photometry for each of the lensed source images. |
| rescale_complex_pos(w, z1, z2) | Make sure everything is roughly centered on the origin in a $1 \times 1$ box. |


| get_chi2_astrometry |  |
| :--- | :--- |
| get_chi2_photometry |  |
| get_lnL_constant |  |
| log_likely_astrometry |  |
| log_likely_astrometry_each |  |
| log_likely_photometry |  |
| log_likely_photometry_each |  |
| start |  |

animate ( $t$ E, time_steps, frame_time, name, size, zoom, astrometry)
Produces animation of microlensing event. This function takes the PSPL and makes an animation, the input variables are as follows

## Parameters

tE:
number of einstein crossings times before/after the peak you want the animation to plot
e.g tE $=2=>$ graph will go from -2 tE to 2 tE
time_steps:
number of time steps before/after peak, so total number of time steps will be 2 times this value
frame_time:
times in ms of each frame in the animation
name: string
the animation will be saved as name.html
size: list
[horizontal, vertical] cm's
zoom:
\# of einstein radii plotted in vertical direction
calc_piE_ecliptic()
Parallax: Get piE_ecliptic
get_all_arrays(t_obs, check_sols=True, rescale=True)
Obtain the image and amplitude arrays for each t_obs.

## Parameters

## t_obs

[array_like] Array of times to model.

## Returns

images
[array_like] Array/tuple of complex positions of each images at each t_obs.
amp_arr
[array_like] Array/tuple of amplification of each images at each t_obs.
get_amp_arr (z_arr, z1, z2)
Calculations amplification array
Calculates the amplification A from the Jacobian $\mathrm{J}, A=1 /|J|$

## Parameters

z_arr
[array_like]
Complex position of images. Shape $=$ [N_times, N_solutions, 1]

- note this could be jagged.
z1
[array_like] Complex position(s) of lens 1 (primary). Shape = [N_times, 1]
z2
[array_like] Complex position(s) of lens 2 (secondary). Shape = [N_times, 1]


## Returns

amp_arr
[array_like] BLEH
get_amplification(t_obs, amp_arr=None)
noParallax: Get the photometric amplification term at a set of times, t .

## Parameters

t:
Array of times in MJD.DDD
get_astrometry (t_obs, image_arr=None, amp_arr=None, ast_filt_idx=0)
Position of the observed (unresolved) source position in arcsec.

## Parameters

t_obs
[array_like] Array of times to model.

## Returns

model_pos
[array_like] Array of vector positions of the centroid at each t_obs.

## get_astrometry_unlensed (t_obs)

Get the astrometry of the source if the lens didn't exist.

## Returns

xS_unlensed
[numpy array, dtype=float, shape $=$ len(t_obs) x 2] The unlensed positions of the source in arcseconds.
get_centroid(t_obs, ast_filt_idx=0, image_arr=None, amp_arr=None)
PSPL: Get the centroid shift (in mas) for a list of observation times (in MJD).

## Parameters

t_obs
[array or float]

## Returns

Centroid offset on the plane of the sky in milli-arcseoncds.

## Other Parameters

ast_filt_idx
[int] Index into the photometry parameter lists for the photometry that corresponds to this astrometry data set.

## image_arr

[list] List returned from PSPL get_all_arrays() used to improve efficiency.
amp_arr
[list] List returned from PSPL get_all_arrays() used to improve efficiency.

```
get_centroid_shift (t, ast_filt_idx=0)
```

Parallax: Get the centroid shift (in mas) for a list of observation times (in MJD).

```
get_complex_pos(t_obs)
```

Get the positions of the lenses and source as complex numbers. This is needed for further calculations.
Note that all units are still the same as before, this is just rewriting vectors $z=(x, y)$ as $z=x+i y$.

## Returns

w
[complex array] Source position as an array of complex numbers with real = east component, imaginary $=$ north component
z1
[complex array] Lens primary component position as an array of complex numbers with real $=$ east component, imaginary $=$ north component
z2
[complex array] Lens secondary component position as an array of complex numbers with real = east component, imaginary = north component

```
get_image_pos_arr(w,zl,z2,ml,m2,check_sols=True)
```

Gets image positions.

Solve the fifth-order polynomial and get the image positions.
See PSBL writeup for full equations.
All angular distances are in arcsec.

## Parameters

w
[array_like] Complex position(s) of the source. Shape = [N_times, 1]
z1
[array_like] Complex position(s) of lens 1 (primary). Shape $=[$ N_times, 1]
z2
[array_like] Complex position(s) of lens 2 (secondary). Shape $=[$ N_times, 1]
check_sols
[bool, optional] If True, calculated roots are checked against the lens equation, and output will only contain those within self.root_tol. If False, all calculated roots are returned.

## Returns

z_arr
[array_like] Rank-1 array of polynomial roots, possibly complex. If check_sols = True, only roots solving the lens equation are returned.
get_image_pos_arr_old ( $w, z 1, z 2$, check_sols=True)
Gets image positions. | Solve the fifth-order polynomial and get the image positions. | See PSBL writeup for full equations. | All angular distances are in arcsec.

## Parameters

w
[array_like] Complex position(s) of the source. Shape $=[\mathrm{N}$ _times, 1]
z1
[array_like] Complex position(s) of lens 1 (primary). Shape $=[$ N_times, 1]
z2
[array_like] Complex position(s) of lens 2 (secondary). Shape = [N_times, 1]
check_sols
[bool, optional] If True, calculated roots are checked against the lens equation, and output will only contain those within self.root_tol. If False, all calculated roots are returned.

## Returns

z_arr
[array_like] Position of the lensed source images. Rank-1 array of polynomial roots, possibly complex. If check_sols $=$ True, only roots solving the lens equation are returned.

## get_lens_astrometry (t_obs)

Equation of motion for just the foreground lens system.

## Parameters

t_obs
[array_like] Time (in MJD).
get_photometry (t_obs, filt_idx=0, amp_arr=None, print_warning=True)
Get the photometry for each of the lensed source images.

## Parameters

t_obs
[array_like] Array of times to model.

## Returns

mag_model
[array_like] Magnitude of the centroid at t_obs.
get_resolved_amplification ( $t$ )
Parallax: Get the photometric amplification term at a set of times, t for both the plus and minus images.

## Parameters

t:
Array of times in MJD.DDD
get_resolved_astrometry (t_obs, image_arr=None, amp_arr=None)
Position of the observed source position in arcsec.

## Parameters

t_obs
[array_like, shape $=[$ N_times $]$ ] Array of times to model.

## Returns

model_pos
[array_like. shape $=\left[\mathrm{N} \_\right.$times, $\left.\left.\mathrm{N} \_i m a g e s, 2\right]\right]$ Array of vector positions of the centroid at each t_obs.

## get_resolved_lens_astrometry (t_obs)

Equation of motion for just the foreground lenses, individually.

## Parameters

t_obs
[array_like] Time (in MJD).
get_resolved_photometry (t_obs, filt_idx=0, amp_arr=None, print_warning=True)
Get the photometry for each of the lensed source images. Implement with no blending (since we don't support different blendings for the different images).

## Parameters

t_obs
[array_like] Array of times to model.

## Returns

## mag_model

[array_like] Magnitude of each lensed image centroid at t_obs. Shape = [5, len(t_obs)]
rescale_complex_pos $(w, z 1, z 2)$
Make sure everything is roughly centered on the origin in a $1 \times 1$ box.

### 4.2.3 PSBL_Phot

class model.PSBL_Phot_noPar_Param1 (*args, **kwargs)
Bases: ModelClassABC, PSBL_Phot, PSBL_noParallax, PSBL_PhotParam1
Helper class that provides a standard way to create an ABC using inheritance.

## Methods

| animate(tE, time_steps, frame_time, name, ...) | Produces animation of microlensing event. |
| :--- | :--- |
| get_all_arrays(t_obs[, check_sols, rescale]) | Obtain the image and amplitude arrays for each t_obs. |
| get_amp_arr(z_arr, z1, z2) | Calculations amplification array |
| get_amplification(t_obs[, amp_arr]) | noParallax: Get the photometric amplification term <br> at a set of times, t. |
| get_astrometry(t_obs[, image_arr, amp_arr, ...]) | Position of the observed (unresolved) source position <br> in Einstein radii. |
| get_astrometry_unlensed(t_obs) | Get the astrometry of the source if the lens didn't ex- <br> ist. |
| get_complex_pos(t_obs) | Get the positions of the lenses and source as complex <br> numbers. |
| get_image_pos_arr(w, z1, z2, m1, m2[, ...]) | Gets image positions. |
| get_image_pos_arr_old(w, z1, z2[, check_sols]) | Gets image positions. |
| get_photometry(t_obs[, filt_idx, amp_arr, ...]) | Get the photometry for each of the lensed source im- <br> ages. |
| get_resolved_astrometry(t_obs[, image_arr, | Position of the observed source position in Einstein <br> radii. |
| ..$])$ | Equation of motion for just the foreground lenses, in- <br> dividually. |
| get_resolved_lens_astrometry(t_obs) | Get the photometry for each of the lensed source im- <br> ages. |
| get_resolved_photometry(t_obs[, filt_idx, ...]) |  |
| rescale_complex_pos(w, z1, z2) | Make sure everything is roughly centered on the ori- <br> gin in a 1 x 1 box. |


| calc_piE_ecliptic |  |
| :--- | :--- |
| get_centroid_shift |  |
| get_chi2_photometry |  |
| get_lens_astrometry |  |
| get_lnL_constant |  |
| get_resolved_amplification |  |
| log_likely_astrometry |  |
| log_likely_photometry |  |
| log_likely_photometry_each |  |

animate ( $t$ E, time_steps, frame_time, name, size, zoom, astrometry)
Produces animation of microlensing event. This function takes the PSPL and makes an animation, the input variables are as follows

## Parameters

tE:
number of einstein crossings times before/after the peak you want the animation to plot
e.g tE $=2$ => graph will go from -2 tE to 2 tE
time_steps:
number of time steps before/after peak, so total number of time steps will be 2 times this value

## frame_time:

times in ms of each frame in the animation

## name: string

the animation will be saved as name.html
size: list
[horizontal, vertical] cm's

## zoom:

\# of einstein radii plotted in vertical direction
get_all_arrays(t_obs, check_sols=True, rescale=True)
Obtain the image and amplitude arrays for each t_obs.

## Parameters

t_obs
[array_like] Array of times to model.

## Returns

images
[array_like] Array/tuple of complex positions of each images at each $t$ _obs.
amp_arr
[array_like] Array/tuple of amplification of each images at each t_obs.

```
get_amp_arr(z_arr,z1,z2)
```

Calculations amplification array
Calculates the amplification A from the Jacobian $\mathrm{J}, A=1 /|J|$

## Parameters

z_arr
[array_like]
Complex position of images. Shape $=$ [N_times, N_solutions, 1]

- note this could be jagged.
z1
[array_like] Complex position(s) of lens 1 (primary). Shape = [N_times, 1]
z2
[array_like] Complex position(s) of lens 2 (secondary). Shape = [N_times, 1]


## Returns

amp_arr
[array_like] BLEH
get_amplification(t_obs,amp_arr=None)
noParallax: Get the photometric amplification term at a set of times, t .

## Parameters

t:
Array of times in MJD.DDD
get_astrometry (t_obs, image_arr=None, amp_arr=None, ast_fil_idx=0)
Position of the observed (unresolved) source position in Einstein radii.

## Parameters

## t_obs

[array_like] Array of times to model.

## Returns

model_pos
[array_like] Array of vector positions of the centroid at each t_obs.

```
get_astrometry_unlensed(t_obs)
```

Get the astrometry of the source if the lens didn't exist. Note, this is a photometry only model, so units are in Einstein radii.

## Returns

xS_unlensed
[numpy array, dtype=float, shape $=$ len(t_obs) x 2] The unlensed positions of the source in Einstein radii.

## Notes

Note: Note, this is a photometry only model, so units are in Einstein radii.
get_centroid_shift ( $t$, ast_fil__idx=0)
noParallax: Get the centroid shift (in mas) for a list of observation times (in MJD).
get_complex_pos(t_obs)
Get the positions of the lenses and source as complex numbers.
This is needed for further calculations. Note that all units are still the same as before, this is just rewriting vectors $z=(x, y)$ as $z=x+i y$.

## Returns

w
[complex array] Source position as an array of complex numbers with real = east component, imaginary $=$ north component
z1
[complex array] Lens primary component position as an array of complex numbers with real $=$ east component, imaginary $=$ north component
z2
[complex array] Lens secondary component position as an array of complex numbers with real $=$ east component, imaginary $=$ north component
get_image_pos_arr $(w, z 1, z 2, m 1, m 2$, check_sols=True $)$
Gets image positions.

Solve the fifth-order polynomial and get the image positions.
See PSBL writeup for full equations.
All angular distances are in arcsec.

## Parameters

w
[array_like] Complex position(s) of the source. Shape = [N_times, 1]
z1
[array_like] Complex position(s) of lens 1 (primary). Shape $=[$ N_times, 1]
z2
[array_like] Complex position(s) of lens 2 (secondary). Shape $=\left[\mathrm{N} \_\right.$times, 1]
check_sols
[bool, optional] If True, calculated roots are checked against the lens equation, and output will only contain those within self.root_tol. If False, all calculated roots are returned.

## Returns

z_arr
[array_like] Rank-1 array of polynomial roots, possibly complex. If check_sols = True, only roots solving the lens equation are returned.
get_image_pos_arr_old ( $w, z 1, z 2$, check_sols=True)
Gets image positions. | Solve the fifth-order polynomial and get the image positions. | See PSBL writeup for full equations. | All angular distances are in arcsec.

## Parameters

w
[array_like] Complex position(s) of the source. Shape $=[$ N_times, 1]
z1
[array_like] Complex position(s) of lens 1 (primary). Shape $=[$ N_times, 1]
z2
[array_like] Complex position(s) of lens 2 (secondary). Shape $=[$ N_times, 1]
check_sols
[bool, optional] If True, calculated roots are checked against the lens equation, and output will only contain those within self.root_tol. If False, all calculated roots are returned.

## Returns

z_arr
[array_like] Position of the lensed source images. Rank-1 array of polynomial roots, possibly complex. If check_sols $=$ True, only roots solving the lens equation are returned.

```
get_lens_astrometry (t)
```

Equation of motion for just the foreground lens.

## Parameters

## t_obs

[array_like] Time (in MJD).
get_photometry (t_obs, filt_idx=0, amp_arr=None, print_warning=True)
Get the photometry for each of the lensed source images.

## Parameters

```
t_obs
                    [array_like] Array of times to model.
```


## Returns

mag_model
[array_like] Magnitude of the centroid at t _obs.
get_resolved_amplification $(t)$
Get the photometric amplification term at a set of times, t for both the plus and minus images.

## Parameters

t:
Array of times in MJD.DDD
get_resolved_astrometry (t_obs, image_arr=None, amp_arr=None)
Position of the observed source position in Einstein radii.

## Parameters

t_obs
[array_like, shape $=$ [N_times]] Array of times to model.

## Returns

model_pos
[array_like. shape $=\left[\mathrm{N} \_\right.$times, $\left.\left.\mathrm{N} \_i m a g e s, 2\right]\right]$ Array of vector positions of the centroid at each t_obs.

## get_resolved_lens_astrometry (t_obs)

Equation of motion for just the foreground lenses, individually.

## Parameters

t_obs
[array_like] Time (in MJD).

## Notes

Note: Note, this is a photometry only model, so units are in Einstein radii.
get_resolved_photometry (t_obs, filt_idx=0, amp_arr=None, print_warning=True)
Get the photometry for each of the lensed source images. Implement with no blending (since we don't support different blendings for the different images).

## Parameters

## t_obs

[array_like] Array of times to model.

## Returns

mag_model
[array_like] Magnitude of each lensed image centroid at t_obs. Shape $=[5$, len(t_obs)]
rescale_complex_pos $(w, z 1, z 2)$
Make sure everything is roughly centered on the origin in a $1 \times 1$ box.

### 4.2.4 PSBL_phot_parallax

class model.PSBL_Phot_Par_Param1 (*args, **kwargs)
Bases: ModelClassABC, PSBL_Phot, PSBL_Parallax, PSBL_PhotParam1
Helper class that provides a standard way to create an ABC using inheritance.

## Methods

| animate(tE, time_steps, frame_time, name, ...) | Produces animation of microlensing event. |
| :--- | :--- |
| calc_piE_ecliptic() | Parallax: Get piE_ecliptic |
| get_all_arrays(t_obs[, check_sols, rescale]) | Obtain the image and amplitude arrays for each t_obs. |
| get_amp_arr(z_arr, z1, z2) | Calculations amplification array <br> get_amplification(t_obs[, amp_arr]) <br> noParallax: Get the photometric amplification term <br> at a set of times, t. |
| get_astrometry(t_obs[, image_arr, amp_arr, ...]) | Position of the observed (unresolved) source position <br> in Einstein radii. |
| get_astrometry_unlensed(t_obs) | Get the astrometry of the source if the lens didn't ex- <br> ist. |
| get_geoproj_ast_params(t0par) | Get the positions of the lenses and source as complex <br> numbers. |
| get_geoproj_params(t0par) | Gets image positions. <br> get_image_pos_arr(w, z1, z2, m1, m2[, ...]) |
| get_image_pos_arr_old(w, z1, z2[, check_sols]) | Gets image positions. |
| get_photometry(t_obs[, filt_idx, amp_arr, ...]) | Get the photometry for each of the lensed source im- <br> ages. |
| get_resolved_astrometry(t_obs[, $\quad$ image_arr, | Position of the observed source position in Einstein <br> radii. |
| get_resolved_lens_astrometry(t_obs) | Equation of motion for just the foreground lenses, in- <br> dividually. |
| get_resolved_photometry(t_obs[, filt_idx, ...]) | Get the photometry for each of the lensed source im- <br> ages. |
| rescale_complex_pos(w, z1, z2) | Make sure everything is roughly centered on the ori- <br> gin in a 1 x 1 box. |


| get_centroid_shift |  |
| :--- | :--- |
| get_chi2_photometry |  |
| get_lens_astrometry |  |
| get_lnL_constant |  |
| get_resolved_amplification |  |
| log_likely_astrometry |  |
| log_likely_photometry |  |
| log_likely_photometry_each |  |
| start |  |

animate (tE, time_steps, frame_time, name, size, zoom, astrometry)
Produces animation of microlensing event. This function takes the PSPL and makes an animation, the input variables are as follows

## Parameters

tE:
number of einstein crossings times before/after the peak you want the animation to plot
e.g $\mathrm{tE}=2=>$ graph will go from -2 tE to 2 tE

## time_steps:

number of time steps before/after peak, so total number of time steps will be 2 times this value

## frame_time:

times in ms of each frame in the animation
name: string
the animation will be saved as name.html
size: list
[horizontal, vertical] cm's

## zoom:

\# of einstein radii plotted in vertical direction
calc_piE_ecliptic()
Parallax: Get piE_ecliptic
get_all_arrays(t_obs, check_sols=True, rescale=True)
Obtain the image and amplitude arrays for each t _obs.

## Parameters

t_obs
[array_like] Array of times to model.

## Returns

images
[array_like] Array/tuple of complex positions of each images at each t_obs.
amp_arr
[array_like] Array/tuple of amplification of each images at each t_obs.
get_amp_arr (z_arr, z1,z2)
Calculations amplification array
Calculates the amplification A from the Jacobian $\mathrm{J}, A=1 /|J|$

## Parameters

z_arr
[array_like]
Complex position of images. Shape $=$ [N_times, N_solutions, 1]

- note this could be jagged.
z1
[array_like] Complex position(s) of lens 1 (primary). Shape = [N_times, 1]
z2
[array_like] Complex position(s) of lens 2 (secondary). Shape = [N_times, 1]


## Returns

amp_arr
[array_like] BLEH
get_amplification(t_obs, amp_arr=None)
noParallax: Get the photometric amplification term at a set of times, $t$.

## Parameters

t:
Array of times in MJD.DDD
get_astrometry (t_obs, image_arr=None, amp_arr=None, ast_filt_idx=0)
Position of the observed (unresolved) source position in Einstein radii.

## Parameters

t_obs
[array_like] Array of times to model.

## Returns

model_pos
[array_like] Array of vector positions of the centroid at each t_obs.

```
get_astrometry_unlensed(t_obs)
```

Get the astrometry of the source if the lens didn't exist. Note, this is a photometry only model, so units are in Einstein radii.

## Returns

xS_unlensed
[numpy array, dtype=float, shape $=$ len(t_obs) x 2] The unlensed positions of the source in Einstein radii.

## Notes

Note: Note, this is a photometry only model, so units are in Einstein radii.

## get_centroid_shift ( $t$, ast_fil_idx=0)

Parallax: Get the centroid shift (in mas) for a list of observation times (in MJD).

```
get_complex_pos(t_obs)
```

Get the positions of the lenses and source as complex numbers.
This is needed for further calculations. Note that all units are still the same as before, this is just rewriting vectors $z=(x, y)$ as $z=x+i y$.

## Returns

w
[complex array] Source position as an array of complex numbers with real = east component, imaginary $=$ north component
z1
[complex array] Lens primary component position as an array of complex numbers with real $=$ east component, imaginary $=$ north component
z2
[complex array] Lens secondary component position as an array of complex numbers with real $=$ east component, imaginary $=$ north component
get_image_pos_arr( $w, z 1, z 2, m 1, m 2$, check_sols=True)
Gets image positions.

Solve the fifth-order polynomial and get the image positions.

See PSBL writeup for full equations.
All angular distances are in arcsec.

## Parameters

w
[array_like] Complex position(s) of the source. Shape $=[$ N_times, 1]
z1
[array_like] Complex position(s) of lens 1 (primary). Shape $=[$ N_times, 1]
z2
[array_like] Complex position(s) of lens 2 (secondary). Shape $=\left[\mathrm{N} \_\right.$times, 1]

## check_sols

[bool, optional] If True, calculated roots are checked against the lens equation, and output will only contain those within self.root_tol. If False, all calculated roots are returned.

## Returns

## z_arr

[array_like] Rank-1 array of polynomial roots, possibly complex. If check_sols = True, only roots solving the lens equation are returned.

```
get_image_pos_arr_old( }w,z1,z2,\mathrm{ check_sols=True)
```

Gets image positions. | Solve the fifth-order polynomial and get the image positions. | See PSBL writeup for full equations. | All angular distances are in arcsec.

## Parameters

w
[array_like] Complex position(s) of the source. Shape $=\left[\mathrm{N} \_\right.$times, 1]
z1
[array_like] Complex position(s) of lens 1 (primary). Shape $=[$ N_times, 1]
z2
[array_like] Complex position(s) of lens 2 (secondary). Shape $=\left[\mathrm{N} \_\right.$times, 1]
check_sols
[bool, optional] If True, calculated roots are checked against the lens equation, and output will only contain those within self.root_tol. If False, all calculated roots are returned.

## Returns

z_arr
[array_like] Position of the lensed source images. Rank-1 array of polynomial roots, possibly complex. If check_sols = True, only roots solving the lens equation are returned.

## get_lens_astrometry $(t)$

Parallax: Get lens astrometry
get_photometry (t_obs, filt_idx=0, amp_arr=None, print_warning=True)
Get the photometry for each of the lensed source images.

## Parameters

## t_obs

[array_like] Array of times to model.

## Returns

mag_model
[array_like] Magnitude of the centroid at t_obs.
get_resolved_amplification $(t)$
Parallax: Get the photometric amplification term at a set of times, t for both the plus and minus images.

## Parameters

t:
Array of times in MJD.DDD
get_resolved_astrometry (t_obs, image_arr=None, amp_arr=None)
Position of the observed source position in Einstein radii.

## Parameters

t_obs
[array_like, shape $=$ [N_times]] Array of times to model.

## Returns

model_pos
[array_like. shape $=\left[\mathrm{N} \_\right.$times, $\left.\left.\mathrm{N} \_i m a g e s, 2\right]\right]$ Array of vector positions of the centroid at each t_obs.

## get_resolved_lens_astrometry (t_obs)

Equation of motion for just the foreground lenses, individually.

## Parameters

t_obs
[array_like] Time (in MJD).

## Notes

Note: Note, this is a photometry only model, so units are in Einstein radii.
get_resolved_photometry (t_obs, filt_idx=0, amp_arr=None, print_warning=True)
Get the photometry for each of the lensed source images. Implement with no blending (since we don't support different blendings for the different images).

## Parameters

t_obs
[array_like] Array of times to model.

## Returns

mag_model
[array_like] Magnitude of each lensed image centroid at t_obs. Shape $=[5$, len(t_obs)]
rescale_complex_pos $(w, z l, z 2)$
Make sure everything is roughly centered on the origin in a $1 \times 1$ box.

### 4.2.5 PSBL PhotAstrom, no parallax with GP

## class model.PSBL_PhotAstrom_noPar_GP_Param1(*args, **kwargs)

Bases: ModelClassABC, PSPL_GP, PSBL_PhotAstrom, PSBL_noParallax, PSBL_PhotAstromParam1
Helper class that provides a standard way to create an ABC using inheritance.

## Methods

| animate(tE, time_steps, frame_time, name, ...) | Produces animation of microlensing event. |
| :---: | :---: |
| get_all_arrays(t_obs[, check_sols, rescale]) | Obtain the image and amplitude arrays for each t_obs. |
| get_amp_arr(z_arr, z1, z2) | Calculations amplification array |
| get_amplification(t_obs[, amp_arr]) | noParallax: Get the photometric amplification term at a set of times, $t$. |
| get_astrometry(t_obs[, image_arr, amp_arr, ...]) | Position of the observed (unresolved) source position in arcsec. |
| get_astrometry_unlensed(t_obs) | Get the astrometry of the source if the lens didn't exist. |
| get_centroid(t_obs[, ast_filt_idx, ...]) | PSPL: Get the centroid shift (in mas) for a list of observation times (in MJD). |
| get_centroid_shift(t) | noParallax: Get the centroid shift (in mas) for a list of observation times (in MJD). |
| get_complex_pos(t_obs) | Get the positions of the lenses and source as complex numbers. |
| get_image_pos_arr(w, z1, z2, m1, m2[, ...]) | Gets image positions. |
| get_image_pos_arr_old(w, z1, z2[, check_sols]) | Gets image positions. |
| get_lens_astrometry(t_obs) | Equation of motion for just the foreground lens system. |
| get_log_det_covariance(t_obs, mag_obs, ...) | Returns photometry with GP noise added in. |
| get_photometry(t_obs[, filt_idx, amp_arr, ...]) | Get the photometry for each of the lensed source images. |
| get_photometry_with_gp(t_obs, mag_obs, ...) | Returns photometry with GP noise added in. |
| get_resolved_amplification(t) | Get the photometric amplification term at a set of times, t for both the plus and minus images. |
| get_resolved_astrometry(t_obs[, image_arr, ...]) | Position of the observed source position in arcsec. |
| get_resolved_lens_astrometry(t_obs) | Equation of motion for just the foreground lenses, individually. |
| get_resolved_photometry(t_obs[, filt_idx, ...]) | Get the photometry for each of the lensed source images. |
| log_likely_photometry(t_obs, mag_obs, ...[, ...]) | Calculate the log-likelihood for the PSPL + GP model and photometric data. |
| rescale_complex_pos(w, z1, z2) | Make sure everything is roughly centered on the origin in a $1 \times 1$ box. |


| calc_piE_ecliptic |  |
| :--- | :--- |
| get_chi2_astrometry |  |
| get_chi2_photometry |  |
| get_lnL_constant |  |
| log_likely_astrometry |  |
| log_likely_astrometry_each |  |
| log_likely_photometry_each |  |

animate ( $t E$, time_steps, frame_time, name, size, zoom, astrometry)
Produces animation of microlensing event. This function takes the PSPL and makes an animation, the input variables are as follows

## Parameters

tE:
number of einstein crossings times before/after the peak you want the animation to plot
e.g tE $=2=>$ graph will go from -2 tE to 2 tE
time_steps:
number of time steps before/after peak, so total number of time steps will be 2 times this value
frame_time:
times in ms of each frame in the animation
name: string
the animation will be saved as name.html
size: list
[horizontal, vertical] cm's

## zoom:

\# of einstein radii plotted in vertical direction
get_all_arrays(t_obs, check_sols=True, rescale=True)
Obtain the image and amplitude arrays for each t_obs.

## Parameters

t_obs
[array_like] Array of times to model.

## Returns

images
[array_like] Array/tuple of complex positions of each images at each t_obs.
amp_arr
[array_like] Array/tuple of amplification of each images at each t_obs.

```
get_amp_arr(z_arr, z1,z2)
```

Calculations amplification array
Calculates the amplification A from the Jacobian $\mathbf{J}, A=1 /|J|$

## Parameters

z_arr
[array_like]

Complex position of images. Shape $=$ [N_times, N_solutions, 1]

- note this could be jagged.
z1
[array_like] Complex position(s) of lens 1 (primary). Shape $=$ [N_times, 1]
z2
[array_like] Complex position(s) of lens 2 (secondary). Shape = [N_times, 1]


## Returns

amp_arr
[array_like] BLEH
get_amplification(t_obs, amp_arr=None)
noParallax: Get the photometric amplification term at a set of times, $t$.

## Parameters

t:
Array of times in MJD.DDD
get_astrometry (t_obs, image_arr=None, amp_arr=None, ast_filt_idx=0)
Position of the observed (unresolved) source position in arcsec.

## Parameters

## t_obs

[array_like] Array of times to model.

## Returns

model_pos
[array_like] Array of vector positions of the centroid at each t_obs.
get_astrometry_unlensed(t_obs)
Get the astrometry of the source if the lens didn't exist.

## Returns

xS_unlensed
[numpy array, dtype=float, shape $=$ len(t_obs) $\times 2$ 2] The unlensed positions of the source in arcseconds.
get_centroid (t_obs, ast_filt_idx=0, image_arr=None, amp_arr=None)
PSPL: Get the centroid shift (in mas) for a list of observation times (in MJD).

## Parameters

t_obs
[array or float]

## Returns

Centroid offset on the plane of the sky in milli-arcseoncds.

## Other Parameters

ast_filt_idx
[int] Index into the photometry parameter lists for the photometry that corresponds to this astrometry data set.
image_arr
[list] List returned from PSPL get_all_arrays() used to improve efficiency.

## amp_arr

[list] List returned from PSPL get_all_arrays() used to improve efficiency.

## get_centroid_shift $(t)$

noParallax: Get the centroid shift (in mas) for a list of observation times (in MJD).
get_complex_pos(t_obs)
Get the positions of the lenses and source as complex numbers. This is needed for further calculations. Note that all units are still the same as before, this is just rewriting vectors $z=(x, y)$ as $z=x+i y$.

## Returns

w
[complex array] Source position as an array of complex numbers with real = east component, imaginary $=$ north component
z1
[complex array] Lens primary component position as an array of complex numbers with real $=$ east component, imaginary $=$ north component
z2
[complex array] Lens secondary component position as an array of complex numbers with real $=$ east component, imaginary $=$ north component

```
get_image_pos_arr(w,zl,z2,m1,m2,check_sols=True)
```

Gets image positions.

Solve the fifth-order polynomial and get the image positions.
See PSBL writeup for full equations.
All angular distances are in arcsec.

## Parameters

w
[array_like] Complex position(s) of the source. Shape $=[\mathrm{N}$ _times, 1]
z1
[array_like] Complex position(s) of lens 1 (primary). Shape $=[$ N_times, 1]
z2
[array_like] Complex position(s) of lens 2 (secondary). Shape $=[$ N_times, 1]
check_sols
[bool, optional] If True, calculated roots are checked against the lens equation, and output will only contain those within self.root_tol. If False, all calculated roots are returned.

## Returns

z_arr
[array_like] Rank-1 array of polynomial roots, possibly complex. If check_sols = True, only roots solving the lens equation are returned.

```
get_image_pos_arr_old(w,z1,z2, check_sols=True)
```

Gets image positions. | Solve the fifth-order polynomial and get the image positions. | See PSBL writeup for full equations. | All angular distances are in arcsec.

## Parameters

w
[array_like] Complex position(s) of the source. Shape $=\left[\mathrm{N} \_\right.$times, 1]
z1
[array_like] Complex position(s) of lens 1 (primary). Shape $=[$ N_times, 1]
z2
[array_like] Complex position(s) of lens 2 (secondary). Shape $=\left[\mathrm{N} \_\right.$times, 1]

## check_sols

[bool, optional] If True, calculated roots are checked against the lens equation, and output will only contain those within self.root_tol. If False, all calculated roots are returned.

## Returns

z_arr
[array_like] Position of the lensed source images. Rank-1 array of polynomial roots, possibly complex. If check_sols = True, only roots solving the lens equation are returned.

## get_lens_astrometry (t_obs)

Equation of motion for just the foreground lens system.

## Parameters

t_obs
[array_like] Time (in MJD).
get_log_det_covariance (t_obs, mag_obs, mag_err_obs, filt_index=0, t_pred=None)
Returns photometry with GP noise added in.

Note: This will throw an error if this is a filter with use_gp_phot[filt_index] = False.
get_photometry (t_obs, filt_idx=0, amp_arr=None, print_warning=True)
Get the photometry for each of the lensed source images.

## Parameters

## t_obs

[array_like] Array of times to model.

## Returns

mag_model
[array_like] Magnitude of the centroid at t_obs.
get_photometry_with_gp(t_obs, mag_obs, mag_err_obs, filt_index=0, t_pred=None)
Returns photometry with GP noise added in.

Note: This will throw an error if this is a filter with use_gp_phot[filt_index] = False.

## get_resolved_amplification ( $t$ )

Get the photometric amplification term at a set of times, t for both the plus and minus images.

## Parameters

t:
Array of times in MJD.DDD
get_resolved_astrometry (t_obs, image_arr=None, amp_arr=None)
Position of the observed source position in arcsec.

## Parameters

t_obs
[array_like, shape $=[$ N_times] $]$ Array of times to model.

## Returns

model_pos
[array_like. shape $=\left[\mathrm{N} \_\right.$times, $\mathrm{N} \_$images, 2] $]$Array of vector positions of the centroid at each t_obs.
get_resolved_lens_astrometry (t_obs)
Equation of motion for just the foreground lenses, individually.

## Parameters

## t_obs

[array_like] Time (in MJD).
get_resolved_photometry (t_obs, filt_idx=0, amp_arr=None, print_warning=True)
Get the photometry for each of the lensed source images. Implement with no blending (since we don't support different blendings for the different images).

## Parameters

## t_obs

[array_like] Array of times to model.

## Returns

mag_model
[array_like] Magnitude of each lensed image centroid at t_obs. Shape = [5, len(t_obs)]
log_likely_photometry (t_obs, mag_obs, mag_err_obs, filt_index=0)
Calculate the log-likelihood for the PSPL + GP model and photometric data.

Note: The GP will only be used for filters where use_gp_phot[filt_index] = True.
rescale_complex_pos $(w, z 1, z 2)$
Make sure everything is roughly centered on the origin in a $1 \times 1$ box.
class model.PSBL_PhotAstrom_noPar_GP_Param2(*args, **kwargs)
Bases: ModelClassABC, PSPL_GP, PSBL_PhotAstrom, PSBL_noParallax, PSBL_PhotAstromParam2
Helper class that provides a standard way to create an ABC using inheritance.

## Methods

| animate(tE, time_steps, frame_time, name, ...) | Produces animation of microlensing event. |
| :--- | :--- |
| get_all_arrays(t_obs[, check_sols, rescale]) | Obtain the image and amplitude arrays for each t_obs. |
| get_amp_arr(z_arr, zl, z2) | Calculations amplification array |
| get_amplification(t_obs[, amp_arr]) | noParallax: Get the photometric amplification term <br> at a set of times, t. |
| get_astrometry(t_obs[, image_arr, amp_arr, ...]) | Position of the observed (unresolved) source position <br> in arcsec. |
| get_astrometry_unlensed(t_obs) | Get the astrometry of the source if the lens didn't ex- <br> ist. |
| get_centroid(t_obs[, ast_filt_idx, ...]) | PSPL: Get the centroid shift (in mas) for a list of ob- <br> servation times (in MJD). |
| get_centroid_shift(t) | noParallax: Get the centroid shift (in mas) for a list <br> of observation times (in MJD). |
| get_complex_pos(t_obs) | Get the positions of the lenses and source as complex <br> numbers. |
| get_image_pos_arr(w, z1, z2, m1, m2[, ...]) | Gets image positions. |
| get_image_pos_arr_old(w, z1, z2[, check_sols]) | Gets image positions. |
| get_lens_astrometry(t_obs) | Equation of motion for just the foreground lens sys- <br> tem. |
| get_log_det_covariance(t_obs, mag_obs, ...) | Returns photometry with GP noise added in. |
| get_photometry(t_obs[, filt_idx, amp_arr, ...]) | Get the photometry for each of the lensed source im- <br> ages. |
| get_photometry_with_gp(t_obs, mag_obs, ...) | Returns photometry with GP noise added in. |
| get_resolved_amplification(t) | Get the photometric amplification term at a set of <br> times, t for both the plus and minus images. |
| get_resolved_astrometry(t_obs[, $\quad$ image_arr, | Position of the observed source position in arcsec. |
| $\ldots .])$. | Equation of motion for just the foreground lenses, in- <br> dividually. |
| get_resolved_lens_astrometry(t_obs) | Get the photometry for each of the lensed source im- <br> ages. |
| get_resolved_photometry(t_obs[, filt_idx, ...]) |  |
| and photometric data. |  |


| calc_piE_ecliptic |  |
| :--- | :--- |
| get_chi2_astrometry |  |
| get_chi2_photometry |  |
| get_lnL_constant |  |
| log_likely_astrometry |  |
| log_likely_astrometry_each |  |
| log_likely_photometry_each |  |

animate (tE, time_steps, frame_time, name, size, zoom, astrometry)
Produces animation of microlensing event. This function takes the PSPL and makes an animation, the input variables are as follows

## Parameters

## tE:

number of einstein crossings times before/after the peak you want the animation to plot
e.g tE $=2=>$ graph will go from -2 tE to 2 tE

## time_steps:

number of time steps before/after peak, so total number of time steps will be 2 times this value

## frame_time:

times in ms of each frame in the animation
name: string
the animation will be saved as name.html
size: list
[horizontal, vertical] cm's
zoom:
\# of einstein radii plotted in vertical direction
get_all_arrays(t_obs, check_sols=True, rescale=True)
Obtain the image and amplitude arrays for each t _obs.

## Parameters

t_obs
[array_like] Array of times to model.

## Returns

images
[array_like] Array/tuple of complex positions of each images at each t_obs.
amp_arr
[array_like] Array/tuple of amplification of each images at each t_obs.
get_amp_arr (z_arr, z1, z2)
Calculations amplification array
Calculates the amplification A from the Jacobian $\mathrm{J}, A=1 /|J|$

## Parameters

z_arr
[array_like]
Complex position of images. Shape = [N_times, N_solutions, 1]

- note this could be jagged.
z1
[array_like] Complex position(s) of lens 1 (primary). Shape = [N_times, 1]
z2
[array_like] Complex position(s) of lens 2 (secondary). Shape = [N_times, 1]


## Returns

amp_arr
[array_like] BLEH
get_amplification(t_obs,amp_arr=None)
noParallax: Get the photometric amplification term at a set of times, t .

## Parameters

t:
Array of times in MJD.DDD
get_astrometry(t_obs, image_arr=None, amp_arr=None, ast_filt_idx=0)
Position of the observed (unresolved) source position in arcsec.

## Parameters

## t_obs

[array_like] Array of times to model.

## Returns

model_pos
[array_like] Array of vector positions of the centroid at each t_obs.
get_astrometry_unlensed (t_obs)
Get the astrometry of the source if the lens didn't exist.

## Returns

xS_unlensed
[numpy array, dtype=float, shape $=$ len(t_obs) $\times 2$ 2] The unlensed positions of the source in arcseconds.

```
get_centroid(t_obs,ast_filt_idx=0, image_arr=None, amp_arr=None)
```

PSPL: Get the centroid shift (in mas) for a list of observation times (in MJD).

## Parameters

## t_obs

[array or float]

## Returns

Centroid offset on the plane of the sky in milli-arcseoncds.

## Other Parameters

ast_filt_idx
[int] Index into the photometry parameter lists for the photometry that corresponds to this astrometry data set.
image_arr
[list] List returned from PSPL get_all_arrays() used to improve efficiency.
amp_arr
[list] List returned from PSPL get_all_arrays() used to improve efficiency.

## get_centroid_shift ( $t$ )

noParallax: Get the centroid shift (in mas) for a list of observation times (in MJD).

```
get_complex_pos(t_obs)
```

Get the positions of the lenses and source as complex numbers. This is needed for further calculations.
Note that all units are still the same as before, this is just rewriting vectors $z=(x, y)$ as $z=x+i y$.

## Returns

w
[complex array] Source position as an array of complex numbers with real = east component, imaginary $=$ north component
z1
[complex array] Lens primary component position as an array of complex numbers with real $=$ east component, imaginary $=$ north component
z2
[complex array] Lens secondary component position as an array of complex numbers with real $=$ east component, imaginary $=$ north component

```
get_image_pos_arr(w,zl,z2,ml,m2,check_sols=True)
```

Gets image positions.

Solve the fifth-order polynomial and get the image positions.
See PSBL writeup for full equations.
All angular distances are in arcsec.

## Parameters

w
[array_like] Complex position(s) of the source. Shape = [N_times, 1]
z1
[array_like] Complex position(s) of lens 1 (primary). Shape $=[$ N_times, 1]
z2
[array_like] Complex position(s) of lens 2 (secondary). Shape $=[$ N_times, 1]
check_sols
[bool, optional] If True, calculated roots are checked against the lens equation, and output will only contain those within self.root_tol. If False, all calculated roots are returned.

## Returns

## z_arr

[array_like] Rank-1 array of polynomial roots, possibly complex. If check_sols = True, only roots solving the lens equation are returned.

```
get_image_pos_arr_old(w,z1,z2, check_sols=True)
```

Gets image positions. | Solve the fifth-order polynomial and get the image positions. | See PSBL writeup for full equations. | All angular distances are in arcsec.

## Parameters

w
[array_like] Complex position(s) of the source. Shape $=[\mathrm{N}$ _times, 1]
z1
[array_like] Complex position(s) of lens 1 (primary). Shape $=[$ N_times, 1]
z2
[array_like] Complex position(s) of lens 2 (secondary). Shape $=[$ N_times, 1]

## check_sols

[bool, optional] If True, calculated roots are checked against the lens equation, and output will only contain those within self.root_tol. If False, all calculated roots are returned.

## Returns

z_arr
[array_like] Position of the lensed source images. Rank-1 array of polynomial roots, possibly complex. If check_sols $=$ True, only roots solving the lens equation are returned.
get_lens_astrometry (t_obs)
Equation of motion for just the foreground lens system.

## Parameters

t_obs
[array_like] Time (in MJD).
get_log_det_covariance (t_obs, mag_obs, mag_err_obs, filt_index=0, t_pred=None)
Returns photometry with GP noise added in.

Note: This will throw an error if this is a filter with use_gp_phot[filt_index] = False.
get_photometry (t_obs, filt_idx=0, amp_arr=None, print_warning=True)
Get the photometry for each of the lensed source images.

## Parameters

## t_obs

[array_like] Array of times to model.

## Returns

mag_model
[array_like] Magnitude of the centroid at t_obs.
get_photometry_with_gp(t_obs, mag_obs, mag_err_obs, filt_index=0, t_pred=None)
Returns photometry with GP noise added in.

Note: This will throw an error if this is a filter with use_gp_phot[filt_index] = False.

## get_resolved_amplification( $t$ )

Get the photometric amplification term at a set of times, t for both the plus and minus images.

## Parameters

t:
Array of times in MJD.DDD
get_resolved_astrometry (t_obs, image_arr=None, amp_arr=None)
Position of the observed source position in arcsec.

## Parameters

t_obs
[array_like, shape $=[$ N_times] $]$ Array of times to model.

## Returns

model_pos
[array_like. shape $=\left[\mathrm{N} \_\right.$times, $\mathrm{N} \_$images, 2] Array of vector positions of the centroid at each t_obs.
get_resolved_lens_astrometry (t_obs)
Equation of motion for just the foreground lenses, individually.

## Parameters

t_obs
[array_like] Time (in MJD).
get_resolved_photometry (t_obs, filt_idx=0, amp_arr=None, print_warning=True)
Get the photometry for each of the lensed source images. Implement with no blending (since we don't support different blendings for the different images).

## Parameters

t_obs
[array_like] Array of times to model.

## Returns

mag_model
[array_like] Magnitude of each lensed image centroid at t_obs. Shape $=[5$, len(t_obs)]
$\log$ _likely_photometry $\left(t \_o b s, m a g \_o b s, m a g \_e r r \_o b s, f i l t \_i n d e x=0\right)$
Calculate the log-likelihood for the PSPL + GP model and photometric data.

Note: The GP will only be used for filters where $u s e \_g p \_p h o t\left[f i l t \_i n d e x\right]=$ True .
rescale_complex_pos $(w, z 1, z 2)$
Make sure everything is roughly centered on the origin in a $1 \times 1$ box.

### 4.2.6 PSBL PhotAstrom, parallax with GP

class model.PSBL_PhotAstrom_Par_GP_Param1(*args, **kwargs)
Bases: ModelClassABC, PSPL_GP, PSBL_PhotAstrom, PSBL_Parallax, PSBL_PhotAstromParam1
Helper class that provides a standard way to create an ABC using inheritance.

## Methods

| animate(tE, time_steps, frame_time, name, ...) | Produces animation of microlensing event. |
| :---: | :---: |
| calc_piE_ecliptic() | Parallax: Get piE_ecliptic |
| get_all_arrays(t_obs[, check_sols, rescale]) | Obtain the image and amplitude arrays for each t_obs. |
| get_amp_arr(z_arr, z1, z2) | Calculations amplification array |
| get_amplification(t_obs[, amp_arr]) | noParallax: Get the photometric amplification term at a set of times, $t$. |
| get_astrometry(t_obs[, image_arr, amp_arr, ...]) | Position of the observed (unresolved) source position in arcsec. |
| get_astrometry_unlensed(t_obs) | Get the astrometry of the source if the lens didn't exist. |
| get_centroid(t_obs[, ast_filt_idx, ...]) | PSPL: Get the centroid shift (in mas) for a list of observation times (in MJD). |
| get_centroid_shift(t[, ast_filt_idx]) | Parallax: Get the centroid shift (in mas) for a list of observation times (in MJD). |
| get_complex_pos(t_obs) | Get the positions of the lenses and source as complex numbers. |
| get_geoproj_ast_params(t0par) |  |
| get_geoproj_params(t0par) |  |
| get_image_pos_arr(w, z1, z2, m1, m2[, ...]) | Gets image positions. |
| get_image_pos_arr_old(w, z1, z2[, check_sols]) | Gets image positions. |
| get_lens_astrometry(t_obs) | Equation of motion for just the foreground lens system. |
| get_log_det_covariance(t_obs, mag_obs, ...) | Returns photometry with GP noise added in. |
| get_photometry(t_obs[, filt_idx, amp_arr, ...]) | Get the photometry for each of the lensed source images. |
| get_photometry_with_gp(t_obs, mag_obs, ...) | Returns photometry with GP noise added in. |
| get_resolved_amplification(t) | Parallax: Get the photometric amplification term at a set of times, $t$ for both the plus and minus images. |
| get_resolved_astrometry(t_obs[, image_arr, ...]) | Position of the observed source position in arcsec. |
| get_resolved_lens_astrometry(t_obs) | Equation of motion for just the foreground lenses, individually. |
| get_resolved_photometry(t_obs[, filt_idx, ...]) | Get the photometry for each of the lensed source images. |
| log_likely_photometry(t_obs, mag_obs, ...[, ...]) | Calculate the log-likelihood for the PSPL + GP model and photometric data. |
| rescale_complex_pos(w, z1, z2) | Make sure everything is roughly centered on the origin in a $1 \times 1$ box. |


| get_chi2_astrometry |  |
| :--- | :--- |
| get_chi2_photometry |  |
| get_lnL_constant |  |
| log_likely_astrometry |  |
| log_likely_astrometry_each |  |
| log_likely_photometry_each |  |
| start |  |

animate ( $t$ E, time_steps, frame_time, name, size, zoom, astrometry)
Produces animation of microlensing event. This function takes the PSPL and makes an animation, the input variables are as follows

## Parameters

tE:
number of einstein crossings times before/after the peak you want the animation to plot
e.g tE $=2=>$ graph will go from -2 tE to 2 tE

## time_steps:

number of time steps before/after peak, so total number of time steps will be 2 times this value

## frame_time:

times in ms of each frame in the animation
name: string
the animation will be saved as name.html
size: list
[horizontal, vertical] cm's

## zoom:

\# of einstein radii plotted in vertical direction

```
calc_piE_ecliptic()
```

Parallax: Get piE_ecliptic
get_all_arrays(t_obs, check_sols=True, rescale=True)
Obtain the image and amplitude arrays for each $t$ _obs.

## Parameters

## t_obs

[array_like] Array of times to model.

## Returns

## images

[array_like] Array/tuple of complex positions of each images at each $t$ _obs.
amp_arr
[array_like] Array/tuple of amplification of each images at each t_obs.
get_amp_arr (z_arr, z1, z2)
Calculations amplification array
Calculates the amplification A from the Jacobian J, $A=1 /|J|$

## Parameters

z_arr
[array_like]
Complex position of images. Shape $=$ [N_times, N_solutions, 1]

- note this could be jagged.
z1
[array_like] Complex position(s) of lens 1 (primary). Shape = [N_times, 1]
z2
[array_like] Complex position(s) of lens 2 (secondary). Shape = [N_times, 1]


## Returns

amp_arr
[array_like] BLEH
get_amplification (t_obs, amp_arr=None)
noParallax: Get the photometric amplification term at a set of times, $t$.

## Parameters

t:
Array of times in MJD.DDD
get_astrometry(t_obs, image_arr=None, amp_arr=None, ast_filt_idx=0)
Position of the observed (unresolved) source position in arcsec.

## Parameters

## t_obs

[array_like] Array of times to model.

## Returns

model_pos
[array_like] Array of vector positions of the centroid at each t_obs.
get_astrometry_unlensed(t_obs)
Get the astrometry of the source if the lens didn't exist.

## Returns

xS_unlensed
[numpy array, dtype=float, shape $=$ len(t_obs) x 2] The unlensed positions of the source in arcseconds.
get_centroid(t_obs, ast_fil_idx=0, image_arr=None, amp_arr=None)
PSPL: Get the centroid shift (in mas) for a list of observation times (in MJD).

## Parameters

t_obs
[array or float]

## Returns

Centroid offset on the plane of the sky in milli-arcseoncds.

## Other Parameters

ast_filt_idx
[int] Index into the photometry parameter lists for the photometry that corresponds to this astrometry data set.
image_arr
[list] List returned from PSPL get_all_arrays() used to improve efficiency.
amp_arr
[list] List returned from PSPL get_all_arrays() used to improve efficiency.
get_centroid_shift ( $t$, ast_fil_id $x=0$ )
Parallax: Get the centroid shift (in mas) for a list of observation times (in MJD).

```
get_complex_pos(t_obs)
```

Get the positions of the lenses and source as complex numbers. This is needed for further calculations. Note that all units are still the same as before, this is just rewriting vectors $z=(x, y)$ as $z=x+i y$.

## Returns

w
[complex array] Source position as an array of complex numbers with real = east component, imaginary $=$ north component

## z1

[complex array] Lens primary component position as an array of complex numbers with real $=$ east component, imaginary $=$ north component
z2
[complex array] Lens secondary component position as an array of complex numbers with real $=$ east component, imaginary $=$ north component

```
get_image_pos_arr(w,zl,z2,m1,m2,check_sols=True)
```

Gets image positions.

Solve the fifth-order polynomial and get the image positions.
See PSBL writeup for full equations.
All angular distances are in arcsec.

## Parameters

w
[array_like] Complex position(s) of the source. Shape $=[$ N_times, 1]
z1
[array_like] Complex position(s) of lens 1 (primary). Shape = [N_times, 1]
z2
[array_like] Complex position(s) of lens 2 (secondary). Shape $=[$ N_times, 1]

## check_sols

[bool, optional] If True, calculated roots are checked against the lens equation, and output will only contain those within self.root_tol. If False, all calculated roots are returned.

## Returns

z_arr
[array_like] Rank-1 array of polynomial roots, possibly complex. If check_sols = True, only roots solving the lens equation are returned.

```
get_image_pos_arr_old(w,z1,z2, check_sols=True)
```

Gets image positions. | Solve the fifth-order polynomial and get the image positions. | See PSBL writeup for full equations. | All angular distances are in arcsec.

## Parameters

w
[array_like] Complex position(s) of the source. Shape $=\left[\mathrm{N} \_\right.$times, 1]
z1
[array_like] Complex position(s) of lens 1 (primary). Shape $=[$ N_times, 1]

## z2

[array_like] Complex position(s) of lens 2 (secondary). Shape $=[$ N_times, 1]
check_sols
[bool, optional] If True, calculated roots are checked against the lens equation, and output will only contain those within self.root_tol. If False, all calculated roots are returned.

## Returns

z_arr
[array_like] Position of the lensed source images. Rank-1 array of polynomial roots, possibly complex. If check_sols $=$ True, only roots solving the lens equation are returned.
get_lens_astrometry (t_obs)
Equation of motion for just the foreground lens system.

## Parameters

t_obs
[array_like] Time (in MJD).
get_log_det_covariance (t_obs, mag_obs, mag_err_obs, filt_index=0, t_pred=None)
Returns photometry with GP noise added in.

Note: This will throw an error if this is a filter with use_gp_phot[filt_index] = False.
get_photometry (t_obs, filt_idx=0, amp_arr=None, print_warning=True)
Get the photometry for each of the lensed source images.

## Parameters

## t_obs

[array_like] Array of times to model.

## Returns

mag_model
[array_like] Magnitude of the centroid at t_obs.
get_photometry_with_gp(t_obs, mag_obs, mag_err_obs, filt_index=0, t_pred=None)
Returns photometry with GP noise added in.

Note: This will throw an error if this is a filter with use_gp_phot[filt_index] = False.

```
get_resolved_amplification(t)
```

Parallax: Get the photometric amplification term at a set of times, $t$ for both the plus and minus images.

## Parameters

t:
Array of times in MJD.DDD
get_resolved_astrometry (t_obs, image_arr=None, amp_arr=None)
Position of the observed source position in arcsec.

## Parameters

t_obs
[array_like, shape $=[$ N_times $]$ ] Array of times to model.

## Returns

model_pos
[array_like. shape $=\left[\mathrm{N} \_\right.$times, $\left.\left.\mathrm{N} \_i m a g e s, 2\right]\right]$ Array of vector positions of the centroid at each t_obs.
get_resolved_lens_astrometry (t_obs)
Equation of motion for just the foreground lenses, individually.

## Parameters

t_obs
[array_like] Time (in MJD).
get_resolved_photometry (t_obs, filt_idx=0, amp_arr=None, print_warning=True)
Get the photometry for each of the lensed source images. Implement with no blending (since we don't support different blendings for the different images).

## Parameters

## t_obs

[array_like] Array of times to model.

## Returns

mag_model
[array_like] Magnitude of each lensed image centroid at t_obs. Shape $=[5$, len(t_obs)]
log_likely_photometry (t_obs, mag_obs, mag_err_obs, filt_index=0)
Calculate the log-likelihood for the PSPL + GP model and photometric data.

Note: The GP will only be used for filters where $u s e \_g p \_p h o t\left[f i l t \_i n d e x\right]=$ True.
rescale_complex_pos $(w, z 1, z 2)$
Make sure everything is roughly centered on the origin in a $1 \times 1$ box.
class model.PSBL_PhotAstrom_Par_GP_Param2(*args, **kwargs)
Bases: ModelClassABC, PSPL_GP, PSBL_PhotAstrom, PSBL_Parallax, PSBL_PhotAstromParam2
Helper class that provides a standard way to create an ABC using inheritance.

## Methods

| animate(tE, time_steps, frame_time, name, ...) | Produces animation of microlensing event. |
| :---: | :---: |
| calc_piE_ecliptic() | Parallax: Get piE_ecliptic |
| get_all_arrays(t_obs[, check_sols, rescale]) | Obtain the image and amplitude arrays for each t_obs. |
| get_amp_arr(z_arr, z1, z2) | Calculations amplification array |
| get_amplification(t_obs[, amp_arr]) | noParallax: Get the photometric amplification term at a set of times, $t$. |
| get_astrometry(t_obs[, image_arr, amp_arr, ...]) | Position of the observed (unresolved) source position in arcsec. |
| get_astrometry_unlensed(t_obs) | Get the astrometry of the source if the lens didn't exist. |
| get_centroid(t_obs[, ast_filt_idx, ...]) | PSPL: Get the centroid shift (in mas) for a list of observation times (in MJD). |
| get_centroid_shift(t[, ast_filt_idx]) | Parallax: Get the centroid shift (in mas) for a list of observation times (in MJD). |
| get_complex_pos(t_obs) | Get the positions of the lenses and source as complex numbers. |
| get_geoproj_ast_params(t0par) |  |
| get_geoproj_params(t0par) |  |
| get_image_pos_arr(w, z1, z2, m1, m2[, ...]) | Gets image positions. |
| get_image_pos_arr_old(w, z1, z2[, check_sols]) | Gets image positions. |
| get_lens_astrometry(t_obs) | Equation of motion for just the foreground lens system. |
| get_log_det_covariance(t_obs, mag_obs, ...) | Returns photometry with GP noise added in. |
| get_photometry(t_obs[, filt_idx, amp_arr, ...]) | Get the photometry for each of the lensed source images. |
| get_photometry_with_gp(t_obs, mag_obs, ...) | Returns photometry with GP noise added in. |
| get_resolved_amplification(t) | Parallax: Get the photometric amplification term at a set of times, $t$ for both the plus and minus images. |
| get_resolved_astrometry(t_obs[, image_arr, ...]) | Position of the observed source position in arcsec. |
| get_resolved_lens_astrometry(t_obs) | Equation of motion for just the foreground lenses, individually. |
| get_resolved_photometry(t_obs[, filt_idx, ...]) | Get the photometry for each of the lensed source images. |
| log_likely_photometry(t_obs, mag_obs, ...[, ...]) | Calculate the log-likelihood for the PSPL + GP model and photometric data. |
| rescale_complex_pos(w, z1, z2) | Make sure everything is roughly centered on the origin in a $1 \times 1$ box. |


| get_chi2_astrometry |  |
| :--- | :--- |
| get_chi2_photometry |  |
| get_lnL_constant |  |
| log_likely_astrometry |  |
| log_likely_astrometry_each |  |
| log_likely_photometry_each |  |
| start |  |

animate ( $t$ E, time_steps, frame_time, name, size, zoom, astrometry)
Produces animation of microlensing event. This function takes the PSPL and makes an animation, the input variables are as follows

## Parameters

tE:
number of einstein crossings times before/after the peak you want the animation to plot
e.g tE $=2=>$ graph will go from -2 tE to 2 tE

## time_steps:

number of time steps before/after peak, so total number of time steps will be 2 times this value

## frame_time:

times in ms of each frame in the animation
name: string
the animation will be saved as name.html
size: list
[horizontal, vertical] cm's

## zoom:

\# of einstein radii plotted in vertical direction

```
calc_piE_ecliptic()
```

Parallax: Get piE_ecliptic
get_all_arrays(t_obs, check_sols=True, rescale=True)
Obtain the image and amplitude arrays for each $t$ _obs.

## Parameters

## t_obs

[array_like] Array of times to model.

## Returns

## images

[array_like] Array/tuple of complex positions of each images at each $t$ _obs.
amp_arr
[array_like] Array/tuple of amplification of each images at each t_obs.
get_amp_arr (z_arr, z1, z2)
Calculations amplification array
Calculates the amplification A from the Jacobian J, $A=1 /|J|$

## Parameters

z_arr
[array_like]
Complex position of images. Shape $=$ [N_times, N_solutions, 1]

- note this could be jagged.
z1
[array_like] Complex position(s) of lens 1 (primary). Shape = [N_times, 1]
z2
[array_like] Complex position(s) of lens 2 (secondary). Shape = [N_times, 1]


## Returns

amp_arr
[array_like] BLEH
get_amplification(t_obs, amp_arr=None)
noParallax: Get the photometric amplification term at a set of times, $t$.

## Parameters

t:
Array of times in MJD.DDD
get_astrometry(t_obs, image_arr=None, amp_arr=None, ast_filt_idx=0)
Position of the observed (unresolved) source position in arcsec.

## Parameters

## t_obs

[array_like] Array of times to model.

## Returns

model_pos
[array_like] Array of vector positions of the centroid at each t_obs.
get_astrometry_unlensed(t_obs)
Get the astrometry of the source if the lens didn't exist.

## Returns

xS_unlensed
[numpy array, dtype=float, shape $=$ len(t_obs) x 2] The unlensed positions of the source in arcseconds.
get_centroid(t_obs, ast_fil_idx=0, image_arr=None, amp_arr=None)
PSPL: Get the centroid shift (in mas) for a list of observation times (in MJD).

## Parameters

t_obs
[array or float]

## Returns

Centroid offset on the plane of the sky in milli-arcseoncds.

## Other Parameters

ast_filt_idx
[int] Index into the photometry parameter lists for the photometry that corresponds to this astrometry data set.
image_arr
[list] List returned from PSPL get_all_arrays() used to improve efficiency.
amp_arr
[list] List returned from PSPL get_all_arrays() used to improve efficiency.
get_centroid_shift ( $t$, ast_fil_id $x=0$ )
Parallax: Get the centroid shift (in mas) for a list of observation times (in MJD).

```
get_complex_pos(t_obs)
```

Get the positions of the lenses and source as complex numbers. This is needed for further calculations. Note that all units are still the same as before, this is just rewriting vectors $z=(x, y)$ as $z=x+i y$.

## Returns

w
[complex array] Source position as an array of complex numbers with real = east component, imaginary $=$ north component

## z1

[complex array] Lens primary component position as an array of complex numbers with real $=$ east component, imaginary $=$ north component
z2
[complex array] Lens secondary component position as an array of complex numbers with real $=$ east component, imaginary $=$ north component

```
get_image_pos_arr(w,zl,z2,m1,m2,check_sols=True)
```

Gets image positions.

Solve the fifth-order polynomial and get the image positions.
See PSBL writeup for full equations.
All angular distances are in arcsec.

## Parameters

w
[array_like] Complex position(s) of the source. Shape $=[$ N_times, 1]
z1
[array_like] Complex position(s) of lens 1 (primary). Shape = [N_times, 1]
z2
[array_like] Complex position(s) of lens 2 (secondary). Shape $=[$ N_times, 1]

## check_sols

[bool, optional] If True, calculated roots are checked against the lens equation, and output will only contain those within self.root_tol. If False, all calculated roots are returned.

## Returns

z_arr
[array_like] Rank-1 array of polynomial roots, possibly complex. If check_sols = True, only roots solving the lens equation are returned.

```
get_image_pos_arr_old(w,z1,z2, check_sols=True)
```

Gets image positions. | Solve the fifth-order polynomial and get the image positions. | See PSBL writeup for full equations. | All angular distances are in arcsec.

## Parameters

w
[array_like] Complex position(s) of the source. Shape $=\left[\mathrm{N} \_\right.$times, 1]
z1
[array_like] Complex position(s) of lens 1 (primary). Shape $=[$ N_times, 1]

## z2

[array_like] Complex position(s) of lens 2 (secondary). Shape $=[$ N_times, 1]
check_sols
[bool, optional] If True, calculated roots are checked against the lens equation, and output will only contain those within self.root_tol. If False, all calculated roots are returned.

## Returns

z_arr
[array_like] Position of the lensed source images. Rank-1 array of polynomial roots, possibly complex. If check_sols $=$ True, only roots solving the lens equation are returned.
get_lens_astrometry (t_obs)
Equation of motion for just the foreground lens system.

## Parameters

t_obs
[array_like] Time (in MJD).
get_log_det_covariance (t_obs, mag_obs, mag_err_obs, filt_index=0, t_pred=None)
Returns photometry with GP noise added in.

Note: This will throw an error if this is a filter with use_gp_phot[filt_index] = False.
get_photometry (t_obs, filt_idx=0, amp_arr=None, print_warning=True)
Get the photometry for each of the lensed source images.

## Parameters

## t_obs

[array_like] Array of times to model.

## Returns

mag_model
[array_like] Magnitude of the centroid at t_obs.
get_photometry_with_gp(t_obs, mag_obs, mag_err_obs, filt_index=0, t_pred=None)
Returns photometry with GP noise added in.

Note: This will throw an error if this is a filter with use_gp_phot[filt_index] = False.

```
get_resolved_amplification(t)
```

Parallax: Get the photometric amplification term at a set of times, $t$ for both the plus and minus images.

## Parameters

t:
Array of times in MJD.DDD
get_resolved_astrometry (t_obs, image_arr=None, amp_arr=None)
Position of the observed source position in arcsec.

## Parameters

t_obs
[array_like, shape $=[$ N_times $]$ ] Array of times to model.

## Returns

model_pos
[array_like. shape $=\left[\mathrm{N} \_\right.$times, $\left.\left.\mathrm{N} \_i m a g e s, 2\right]\right]$ Array of vector positions of the centroid at each t_obs.
get_resolved_lens_astrometry (t_obs)
Equation of motion for just the foreground lenses, individually.

## Parameters

t_obs
[array_like] Time (in MJD).
get_resolved_photometry (t_obs, filt_idx=0, amp_arr=None, print_warning=True)
Get the photometry for each of the lensed source images. Implement with no blending (since we don't support different blendings for the different images).

## Parameters

## t_obs

[array_like] Array of times to model.

## Returns

mag_model
[array_like] Magnitude of each lensed image centroid at t_obs. Shape $=[5$, len(t_obs)]
log_likely_photometry (t_obs, mag_obs, mag_err_obs, filt_index=0)
Calculate the log-likelihood for the PSPL + GP model and photometric data.

Note: The GP will only be used for filters where $u s e \_g p \_p h o t\left[f i l t \_i n d e x\right]=$ True.
rescale_complex_pos $(w, z 1, z 2)$
Make sure everything is roughly centered on the origin in a $1 \times 1$ box.

### 4.2.7 PSBL Phot, no parallax with GP

class model.PSBL_Phot_noPar_GP_Param1 (*args, **kwargs)
Bases: ModelClassABC, PSPL_GP, PSBL_Phot, PSBL_noParallax, PSBL_PhotParam1
Helper class that provides a standard way to create an ABC using inheritance.

## Methods

| animate(tE, time_steps, frame_time, name, ...) | Produces animation of microlensing event. |
| :--- | :--- |
| get_all_arrays(t_obs[, check_sols, rescale]) | Obtain the image and amplitude arrays for each t_obs. |
| get_amp_arr(z_arr, z1, z2) | Calculations amplification array |
| get_amplification(t_obs[, amp_arr]) | noParallax: Get the photometric amplification term <br> at a set of times, t. |
| get_astrometry(t_obs[, image_arr, amp_arr, ...]) | Position of the observed (unresolved) source position <br> in Einstein radii. |
| get_astrometry_unlensed(t_obs) | Get the astrometry of the source if the lens didn't ex- <br> ist. |
| get_complex_pos(t_obs) | Get the positions of the lenses and source as complex <br> numbers. |
| get_image_pos_arr(w, z1, z2, m1, m2[, ...]) | Gets image positions. |
| get_image_pos_arr_old(w, z1, z2[, check_sols]) | Gets image positions. |
| get_log_det_covariance(t_obs, mag_obs, ...) | Returns photometry with GP noise added in. |
| get_photometry(t_obs[, filt_idx, amp_arr, ...]) | Get the photometry for each of the lensed source im- <br> ages. |
| get_photometry_with_gp(t_obs, mag_obs, ...) | Returns photometry with GP noise added in. |
| get_resolved_astrometry(t_obs[, image_arr, | Position of the observed source position in Einstein <br> radii. |
| get_resolved_lens_astrometry(t_obs) | Equation of motion for just the foreground lenses, in- <br> dividually. |
| get_resolved_photometry(t_obs[, filt_idx, ...]) | Get the photometry for each of the lensed source im- <br> ages. |
| log_likely_photometry(t_obs, mag_obs, ...[, ...]) | Calculate the log-likelihood for the PSPL + GP model <br> and photometric data. |
| rescale_complex_pos(w, z1, z2) | Make sure everything is roughly centered on the ori- <br> gin in a 1 x 1 box. |


| calc_piE_ecliptic |  |
| :--- | :--- |
| get_centroid_shift |  |
| get_chi2_photometry |  |
| get_lens_astrometry |  |
| get_lnL_constant |  |
| get_resolved_amplification |  |
| log_likely_astrometry |  |
| log_likely_photometry_each |  |

animate ( $t E$, time_steps, frame_time, name, size, zoom, astrometry)
Produces animation of microlensing event. This function takes the PSPL and makes an animation, the input variables are as follows

## Parameters

tE:
number of einstein crossings times before/after the peak you want the animation to plot
e.g $\mathrm{tE}=2$ => graph will go from -2 tE to 2 tE

## time_steps:

number of time steps before/after peak, so total number of time steps will be 2 times this
value

## frame_time:

times in ms of each frame in the animation
name: string
the animation will be saved as name.html
size: list
[horizontal, vertical] cm's
zoom:
\# of einstein radii plotted in vertical direction
get_all_arrays (t_obs, check_sols=True, rescale=True)
Obtain the image and amplitude arrays for each t_obs.

## Parameters

t_obs
[array_like] Array of times to model.

## Returns

images
[array_like] Array/tuple of complex positions of each images at each $t$ _obs.
amp_arr
[array_like] Array/tuple of amplification of each images at each t_obs.
get_amp_arr (z_arr, z1, z2)
Calculations amplification array
Calculates the amplification A from the Jacobian J, $A=1 /|J|$

## Parameters

z_arr
[array_like]
Complex position of images. Shape $=$ [N_times, N_solutions, 1]

- note this could be jagged.
z1
[array_like] Complex position(s) of lens 1 (primary). Shape = [N_times, 1]
z2
[array_like] Complex position(s) of lens 2 (secondary). Shape = [N_times, 1]


## Returns

amp_arr
[array_like] BLEH
get_amplification (t_obs, amp_arr=None)
noParallax: Get the photometric amplification term at a set of times, $t$.

## Parameters

t:
Array of times in MJD.DDD
get_astrometry (t_obs, image_arr=None, amp_arr=None, ast_filt_idx=0)
Position of the observed (unresolved) source position in Einstein radii.

## Parameters

## t_obs

[array_like] Array of times to model.

## Returns

model_pos
[array_like] Array of vector positions of the centroid at each t_obs.
get_astrometry_unlensed (t_obs)
Get the astrometry of the source if the lens didn't exist. Note, this is a photometry only model, so units are in Einstein radii.

## Returns

## xS_unlensed

[numpy array, dtype=float, shape = len(t_obs) x 2] The unlensed positions of the source in Einstein radii.

## Notes

Note: Note, this is a photometry only model, so units are in Einstein radii.

```
get_centroid_shift(t, ast_filt_idx=0)
```

noParallax: Get the centroid shift (in mas) for a list of observation times (in MJD).

```
get_complex_pos(t_obs)
```

Get the positions of the lenses and source as complex numbers.
This is needed for further calculations. Note that all units are still the same as before, this is just rewriting vectors $z=(x, y)$ as $z=x+i y$.

## Returns

w
[complex array] Source position as an array of complex numbers with real = east component, imaginary $=$ north component
z1
[complex array] Lens primary component position as an array of complex numbers with real $=$ east component, imaginary $=$ north component
z2
[complex array] Lens secondary component position as an array of complex numbers with real $=$ east component, imaginary $=$ north component
get_image_pos_arr $(w, z 1, z 2, m 1, m 2$, check_sols=True)
Gets image positions.

Solve the fifth-order polynomial and get the image positions.
See PSBL writeup for full equations.
All angular distances are in arcsec.

## Parameters

w
[array_like] Complex position(s) of the source. Shape $=[$ N_times, 1]
z1
[array_like] Complex position(s) of lens 1 (primary). Shape $=\left[\mathrm{N} \_\right.$times, 1$]$
z2
[array_like] Complex position(s) of lens 2 (secondary). Shape = [N_times, 1]

## check_sols

[bool, optional] If True, calculated roots are checked against the lens equation, and output will only contain those within self.root_tol. If False, all calculated roots are returned.

## Returns

z_arr
[array_like] Rank-1 array of polynomial roots, possibly complex. If check_sols = True, only roots solving the lens equation are returned.

```
get_image_pos_arr_old(w,z1,z2, check_sols=True)
```

Gets image positions. | Solve the fifth-order polynomial and get the image positions. | See PSBL writeup for full equations. | All angular distances are in arcsec.

## Parameters

w
[array_like] Complex position(s) of the source. Shape $=\left[\mathrm{N} \_\right.$times, 1]

## z1

[array_like] Complex position(s) of lens 1 (primary). Shape = [N_times, 1]
z2
[array_like] Complex position(s) of lens 2 (secondary). Shape = [N_times, 1]
check_sols
[bool, optional] If True, calculated roots are checked against the lens equation, and output will only contain those within self.root_tol. If False, all calculated roots are returned.

## Returns

## z_arr

[array_like] Position of the lensed source images. Rank-1 array of polynomial roots, possibly complex. If check_sols = True, only roots solving the lens equation are returned.

```
get_lens_astrometry (t)
```

Equation of motion for just the foreground lens.

## Parameters

t_obs
[array_like] Time (in MJD).
get_log_det_covariance (t_obs, mag_obs, mag_err_obs, filt_index=0, t_pred=None)
Returns photometry with GP noise added in.

Note: This will throw an error if this is a filter with use_gp_phot[filt_index] = False.
get_photometry (t_obs, filt_idx=0, amp_arr=None, print_warning=True)
Get the photometry for each of the lensed source images.

## Parameters

t_obs
[array_like] Array of times to model.

## Returns

mag_model
[array_like] Magnitude of the centroid at t_obs.
get_photometry_with_gp(t_obs, mag_obs, mag_err_obs, filt_index=0, t_pred=None)
Returns photometry with GP noise added in.

Note: This will throw an error if this is a filter with use_gp_phot[filt_index] = False.
get_resolved_amplification $(t)$
Get the photometric amplification term at a set of times, $t$ for both the plus and minus images.

## Parameters

t:
Array of times in MJD.DDD
get_resolved_astrometry (t_obs, image_arr=None, amp_arr=None)
Position of the observed source position in Einstein radii.

## Parameters

## t_obs

[array_like, shape $=[$ N_times $]]$ Array of times to model.

## Returns

model_pos
[array_like. shape $=[$ N_times, N_images, 2] $]$ Array of vector positions of the centroid at each t_obs.
get_resolved_lens_astrometry (t_obs)
Equation of motion for just the foreground lenses, individually.

## Parameters

t_obs
[array_like] Time (in MJD).

## Notes

Note: Note, this is a photometry only model, so units are in Einstein radii.
get_resolved_photometry (t_obs, filt_idx=0, amp_arr=None, print_warning=True)
Get the photometry for each of the lensed source images. Implement with no blending (since we don't support different blendings for the different images).

## Parameters

## t_obs

[array_like] Array of times to model.

## Returns

mag_model
[array_like] Magnitude of each lensed image centroid at t_obs. Shape $=[5$, len(t_obs)]
log_likely_photometry (t_obs, mag_obs, mag_err_obs, filt_index=0)
Calculate the log-likelihood for the PSPL + GP model and photometric data.

Note: The GP will only be used for filters where $u s e \_g p \_p h o t\left[f i l t \_i n d e x\right]=$ True .
rescale_complex_pos $(w, z 1, z 2)$
Make sure everything is roughly centered on the origin in a $1 \times 1$ box.

### 4.2.8 PSBL Phot, parallax with GP

class model.PSBL_Phot_Par_GP_Param1 (*args, **kwargs)
Bases: ModelClassABC, PSPL_GP, PSBL_Phot, PSBL_Parallax, PSBL_PhotParam1
Helper class that provides a standard way to create an ABC using inheritance.

## Methods

| animate(tE, time_steps, frame_time, name, ...) | Produces animation of microlensing event. |
| :---: | :---: |
| calc_piE_ecliptic() | Parallax: Get piE_ecliptic |
| get_all_arrays(t_obs[, check_sols, rescale]) | Obtain the image and amplitude arrays for each t_obs. |
| get_amp_arr(z_arr, z1, z2) | Calculations amplification array |
| get_amplification(t_obs[, amp_arr]) | noParallax: Get the photometric amplification term at a set of times, $t$. |
| get_astrometry(t_obs[, image_arr, amp_arr, ...]) | Position of the observed (unresolved) source position in Einstein radii. |
| get_astrometry_unlensed(t_obs) | Get the astrometry of the source if the lens didn't exist. |
| get_complex_pos(t_obs) | Get the positions of the lenses and source as complex numbers. |
| get_geoproj_ast_params(t0par) |  |
| get_geoproj_params(t0par) |  |
| get_image_pos_arr(w, z1, z2, m1, m2[,...]) | Gets image positions. |
| get_image_pos_arr_old(w, z1, z2[, check_sols]) | Gets image positions. |
| get_log_det_covariance(t_obs, mag_obs, ...) | Returns photometry with GP noise added in. |
| get_photometry(t_obs[, filt_idx, amp_arr, ...]) | Get the photometry for each of the lensed source images. |
| get_photometry_with_gp(t_obs, mag_obs, ...) | Returns photometry with GP noise added in. |
| get_resolved_astrometry(t_obs[, image_arr, ...]) | Position of the observed source position in Einstein radii. |
| get_resolved_lens_astrometry(t_obs) | Equation of motion for just the foreground lenses, individually. |
| get_resolved_photometry(t_obs[, filt_idx, ...]) | Get the photometry for each of the lensed source images. |
| log_likely_photometry(t_obs, mag_obs, ...[, ...]) | Calculate the log-likelihood for the PSPL + GP model and photometric data. |
| rescale_complex_pos(w, z1, z2) | Make sure everything is roughly centered on the origin in a $1 \times 1$ box. |


| get_centroid_shift |  |
| :--- | :--- |
| get_chi2_photometry |  |
| get_lens_astrometry |  |
| get_lnL_constant |  |
| get_resolved_amplification |  |
| log_likely_astrometry |  |
| log_likely_photometry_each |  |
| start |  |

animate ( $t$ E, time_steps, frame_time, name, size, zoom, astrometry)
Produces animation of microlensing event. This function takes the PSPL and makes an animation, the input variables are as follows

## Parameters

tE:
number of einstein crossings times before/after the peak you want the animation to plot
e.g tE $=2=>$ graph will go from -2 tE to 2 tE

## time_steps:

number of time steps before/after peak, so total number of time steps will be 2 times this value

## frame_time:

times in ms of each frame in the animation

## name: string

the animation will be saved as name.html
size: list
[horizontal, vertical] cm's

## zoom:

\# of einstein radii plotted in vertical direction

## calc_piE_ecliptic()

Parallax: Get piE_ecliptic
get_all_arrays(t_obs, check_sols=True, rescale=True)
Obtain the image and amplitude arrays for each t _obs.

## Parameters

## t_obs

[array_like] Array of times to model.

## Returns

images
[array_like] Array/tuple of complex positions of each images at each $t$ _obs.
amp_arr
[array_like] Array/tuple of amplification of each images at each t_obs.
get_amp_arr (z_arr, z1, z2)
Calculations amplification array
Calculates the amplification A from the Jacobian $\mathbf{J}, A=1 /|J|$

## Parameters

z_arr
[array_like]
Complex position of images. Shape $=$ [N_times, N_solutions, 1]

- note this could be jagged.
z1
[array_like] Complex position(s) of lens 1 (primary). Shape = [N_times, 1]
z2
[array_like] Complex position(s) of lens 2 (secondary). Shape = [N_times, 1]


## Returns

amp_arr
[array_like] BLEH
get_amplification(t_obs, amp_arr=None)
noParallax: Get the photometric amplification term at a set of times, t .

## Parameters

t:
Array of times in MJD.DDD
get_astrometry (t_obs, image_arr=None, amp_arr=None, ast_filt_idx=0)
Position of the observed (unresolved) source position in Einstein radii.

## Parameters

## t_obs

[array_like] Array of times to model.

## Returns

model_pos
[array_like] Array of vector positions of the centroid at each t_obs.
get_astrometry_unlensed (t_obs)
Get the astrometry of the source if the lens didn't exist. Note, this is a photometry only model, so units are in Einstein radii.

## Returns

xS_unlensed
[numpy array, dtype=float, shape $=$ len(t_obs) x 2] The unlensed positions of the source in Einstein radii.

## Notes

Note: Note, this is a photometry only model, so units are in Einstein radii.
get_centroid_shift ( $t$, ast_filt_idx=0)
Parallax: Get the centroid shift (in mas) for a list of observation times (in MJD).

```
get_complex_pos(t_obs)
```

Get the positions of the lenses and source as complex numbers.
This is needed for further calculations. Note that all units are still the same as before, this is just rewriting vectors $z=(x, y)$ as $z=x+i y$.

## Returns

w
[complex array] Source position as an array of complex numbers with real = east component, imaginary $=$ north component
z1
[complex array] Lens primary component position as an array of complex numbers with real $=$ east component, imaginary $=$ north component
z2
[complex array] Lens secondary component position as an array of complex numbers with real $=$ east component, imaginary $=$ north component
get_image_pos_arr $(w, z 1, z 2, m 1, m 2$, check_sols=True $)$
Gets image positions.

Solve the fifth-order polynomial and get the image positions.
See PSBL writeup for full equations.
All angular distances are in arcsec.

## Parameters

w
[array_like] Complex position(s) of the source. Shape $=\left[\mathrm{N} \_\right.$times, 1]
z1
[array_like] Complex position(s) of lens 1 (primary). Shape $=[$ N_times, 1]
z2
[array_like] Complex position(s) of lens 2 (secondary). Shape $=\left[\mathrm{N} \_\right.$times, 1]
check_sols
[bool, optional] If True, calculated roots are checked against the lens equation, and output will only contain those within self.root_tol. If False, all calculated roots are returned.

## Returns

z_arr
[array_like] Rank-1 array of polynomial roots, possibly complex. If check_sols = True, only roots solving the lens equation are returned.
get_image_pos_arr_old $(w, z 1, z 2$, check_sols=True)
Gets image positions. | Solve the fifth-order polynomial and get the image positions. | See PSBL writeup for full equations. | All angular distances are in arcsec.

## Parameters

w
[array_like] Complex position(s) of the source. Shape $=\left[\mathrm{N} \_\right.$times, 1]
z1
[array_like] Complex position(s) of lens 1 (primary). Shape $=[$ N_times, 1]
z2
[array_like] Complex position(s) of lens 2 (secondary). Shape $=\left[\mathrm{N} \_\right.$times, 1]

## check_sols

[bool, optional] If True, calculated roots are checked against the lens equation, and output will only contain those within self.root_tol. If False, all calculated roots are returned.

## Returns

z_arr
[array_like] Position of the lensed source images. Rank-1 array of polynomial roots, possibly complex. If check_sols = True, only roots solving the lens equation are returned.
get_lens_astrometry $(t)$
Parallax: Get lens astrometry
get_log_det_covariance (t_obs, mag_obs, mag_err_obs, filt_index=0, t_pred=None)
Returns photometry with GP noise added in.

Note: This will throw an error if this is a filter with use_gp_phot[filt_index] = False.
get_photometry (t_obs, filt_idx=0, amp_arr=None, print_warning=True)
Get the photometry for each of the lensed source images.

## Parameters

## t_obs

[array_like] Array of times to model.

## Returns

mag_model
[array_like] Magnitude of the centroid at t_obs.
get_photometry_with_gp(t_obs, mag_obs, mag_err_obs, filt_index=0, t_pred=None)
Returns photometry with GP noise added in.

Note: This will throw an error if this is a filter with use_gp_phot[filt_index] = False.
get_resolved_amplification $(t)$
Parallax: Get the photometric amplification term at a set of times, $t$ for both the plus and minus images.

## Parameters

t:
Array of times in MJD.DDD
get_resolved_astrometry (t_obs, image_arr=None, amp_arr=None)
Position of the observed source position in Einstein radii.

## Parameters

## t_obs

[array_like, shape $=[$ N_times $]]$ Array of times to model.

## Returns

model_pos
[array_like. shape $=[$ N_times, N_images, 2]] Array of vector positions of the centroid at each t_obs.
get_resolved_lens_astrometry (t_obs)
Equation of motion for just the foreground lenses, individually.

## Parameters

t_obs
[array_like] Time (in MJD).

## Notes

Note: Note, this is a photometry only model, so units are in Einstein radii.
get_resolved_photometry (t_obs, fil__idx=0, amp_arr=None, print_warning=True)
Get the photometry for each of the lensed source images. Implement with no blending (since we don't support different blendings for the different images).

## Parameters

## t_obs

[array_like] Array of times to model.

## Returns

mag_model
[array_like] Magnitude of each lensed image centroid at t_obs. Shape $=[5$, len(t_obs)]
log_likely_photometry (t_obs, mag_obs, mag_err_obs, filt_index=0)
Calculate the log-likelihood for the PSPL + GP model and photometric data.

Note: The GP will only be used for filters where use_gp_phot[filt_index] = True.

```
rescale_complex_pos(w,z1,z2)
```

Make sure everything is roughly centered on the origin in a $1 \times 1$ box.

### 4.3 FSPL User Class

### 4.3.1 FSPL_parallax

class model.FSPL_PhotAstrom_Par_Param1 (*args, **kwargs)
Bases: ModelClassABC, FSPL_PhotAstrom, FSPL_Parallax, FSPL_PhotAstromParam1
Helper class that provides a standard way to create an ABC using inheritance.

## Methods

| animate(tE, time_steps, frame_time, name, ...) | Produces animation of microlensing event. |
| :--- | :--- |
| calc_piE_ecliptic() | Parallax: Get piE_ecliptic |
| get_amplification(t) | Parallax: Get the photometric amplification term at a <br> set of times, t. |
| get_astrometry(t_obs[, ast_filt_idx]) | Parallax: Get astrometry |
| get_astrometry_unlensed(t) | Outputs position of source unlensed. |
| get_centroid_shift(t[, ast_filt_idx]) | Parallax: Get the centroid shift (in mas) for a list of <br> observation times (in MJD). |
| get_geoproj_ast_params(t0par) | Parallax: Get lens astrometry |
| get_geoproj_params(t0par) | Get the photometry for each of the lensed source im- <br> ages. |
| get_lens_astrometry(t_obs) | Parallax: Get the photometric amplification term at a <br> set of times, t for both the plus and minus images. |
| get_photometry(t_obs[, filt_idx, amp_arr, ...]) | Return astrometric points that outline the outer cir- <br> cumference of the source star. |
| get_resolved_amplification(t) |  |


| get_chi2_astrometry |  |
| :--- | :--- |
| get_chi2_photometry |  |
| get_lnL_constant |  |
| get_resolved_astrometry |  |
| log_likely_astrometry |  |
| log_likely_astrometry_each |  |
| log_likely_photometry |  |
| log_likely_photometry_each |  |
| start |  |

animate ( $t E$, time_steps, frame_time, name, size, zoom, astrometry)
Produces animation of microlensing event. This function takes the PSPL and makes an animation, the input variables are as follows

## Parameters

tE:
number of einstein crossings times before/after the peak you want the animation to plot
e.g tE $=2$ => graph will go from -2 tE to 2 tE
time_steps:
number of time steps before/after peak, so total number of time steps will be 2 times this value

## frame_time:

times in ms of each frame in the animation
name: string
the animation will be saved as name.html

## size: list

[horizontal, vertical] cm's
zoom:
\# of einstein radii plotted in vertical direction
calc_piE_ecliptic()
Parallax: Get piE_ecliptic
get_amplification $(t)$
Parallax: Get the photometric amplification term at a set of times, t .

## Parameters

t:
Array of times in MJD.DDD
get_astrometry (t_obs, ast_fil_idx=0)
Parallax: Get astrometry
get_astrometry_unlensed $(t)$
Outputs position of source unlensed.
Input a list of times and it will output the position of the source had it not been lensed at each of the times in the list
e.g if $\mathrm{n}=4$, and say $\mathrm{v}=[1,0] \&$ the times are $[0,1,2]$ in years.

This will return
(( ( $(1,0),(0,1),(-1,0),(0,-1)),((2,0),(1,1),(0,0),(1,-1)),($ $(3,0),(2,1),(1,0),(2,-1))) \ldots$
$=($ positions at $\mathrm{t}=0)$, (positions at $\mathrm{t}=1)$, (positions at $\mathrm{t}=2)$
so np.array (positions) is an array which contains an array for each time step with the positions of all the points on the boundary of the source.

## get_centroid_shift (t, ast_filt_idx=0)

Parallax: Get the centroid shift (in mas) for a list of observation times (in MJD).

## get_lens_astrometry (t_obs)

Parallax: Get lens astrometry
get_photometry (t_obs, filt_idx=0, amp_arr=None, print_warning=True)
Get the photometry for each of the lensed source images.

## Parameters

## t_obs

[array_like] Array of times to model.

## Other Parameters

## amp_arr

[array_like] Amplifications of each individual image at each time, i.e. amp_arr.shape = (len(t_obs), number of images at each t_obs).
This will over-ride t_obs; but is more efficient when calculating both photometry and astrometry. If None, then just use t_obs.

## Returns

mag_model
[array_like] Magnitude of the centroid at t_obs.
get_resolved_amplification $(t)$
Parallax: Get the photometric amplification term at a set of times, $t$ for both the plus and minus images.

## Parameters

t:
Array of times in MJD.DDD
get_resolved_astrometry $(t)$
Parallax: Get the x , y astrometry for each of the two source images, which we label plus and minus.

## Returns

[xS_plus, xS_minus]
[list of numpy arrays]

- xS_plus is the vector position of the plus image.
- xS_minus is the vector position of the plus image.


## get_source_outline_astrometry ( $r, n$, center)

Return astrometric points that outline the outer circumference of the source star.

The outline is described as a circle of radius self.radius and is evaluated at self.n_outline number of points.
takes in the radius of the circle, centre position and number of points we are approximating the circle by and returns a numpy array of positions
e.g: $(((1,0),(0,1),(-1,0),(0,-1)))$ if $\mathrm{n}=4$ and radius $=1$

## Returns

source_points
[numpy array] Returns an array of shape = [2, self.n_outline, len(time)]

## POINT SOURCE POINT LENS (PSPL) CLASSES

### 5.1 Parameterization Class Family

Here is a subsection of the inheritance diagram for the PSPL Parametrization classes showing the AstromParam, PhotParam, and PhotAstromParam structures.


### 5.1.1 PSPL Models

class model.PSPL_Param(*args, **kwargs)
Bases: ABC
An abstract class that all Param classes should sub-class. This serves as a reminder for the class variables that MUST be set.
class model.PSPL_AstromParam4 ( $t 0, u 0_{0} a m p, t E$, thetaE, piS, piE_E, piE_N, $x S 0 \_E, x S 0 \_N, m u S \_E, m u S \_N$, $r a L=$ None, decL=None)
Bases: PSPL_Param
Point Source Point Lens model for microlensing. This model includes proper motions of the source and the source position on the sky. It is the same as PSPL_PhotAstromParam2 except it fits for baseline instead of source magnitude.

## Notes

Note: Required parameters if calculating with parallax

- raL: Right ascension of the lens in decimal degrees.
- decL: Declination of the lens in decimal degrees.


## Attributes

t0: float
Time of photometric peak, as seen from Earth (MJD.DDD)
u0_amp: float
Angular distance between the lens and source on the plane of the sky at closest approach in units of thetaE. Can be

- positive (u0_amp > 0 when u0_hat[0] >0) or
- negative (u0_amp < 0 when u0_hat[0] < 0).


## tE: float

Einstein crossing time (days).

## thetaE: float

The size of the Einstein radius in (mas).
piS: float
Amplitude of the parallax ( $1 \mathrm{AU} / \mathrm{dS}$ ) of the source. (mas)

## piE_E: float

The microlensing parallax in the East direction in units of thetaE

## piE_N: float

The microlensing parallax in the North direction in units of thetaE

## xS0_E: float

RA Source position on sky at $\mathrm{t}=\mathrm{t} 0(\operatorname{arcsec})$ in an arbitrary ref. frame.
xS0_N: float
Dec source position on sky at $\mathrm{t}=\mathrm{t} 0(\operatorname{arcsec})$ in an arbitrary ref. frame.
muS_E: float
RA Source proper motion (mas/yr)
muS_N: float
Dec Source proper motion (mas/yr)
class model.PSPL_AstromParam3 (tO, u0_amp, $t E$, $\log 10 \_t h e t a E, p i S, p i E \_E, p i E \_N, x S O \_E, x S O \_N, m u S \_E$, $\left.m u S \_N, r a L=N o n e, \operatorname{dec} L=N o n e\right)$

Bases: PSPL_Param
DESCRIPTION: Point Source Point Lens model for microlensing. This model includes proper motions of the source and the source position on the sky. It is the same as PSPL_PhotAstromParam3 except it fits only astrometry, no photometry.

## Notes

Note: Required parameters if calculating with parallax

- raL: Right ascension of the lens in decimal degrees.
- decL: Declination of the lens in decimal degrees.


## Attributes

## t0: float

Time of photometric peak, as seen from Earth (MJD.DDD)
u0_amp: float
Angular distance between the lens and source on the plane of the sky at closest approach in units of thetaE. Can be

- positive (u0_amp > 0 when $u 0 \_h a t[0]>0$ ) or
- negative (u0_amp < 0 when u0_hat[0] < 0).


## tE: float

Einstein crossing time (days).

## log10_thetaE: float

The $\log$ of the Einstein radius $\log 10$ (thetaE/mas).

## piS: float

Amplitude of the parallax ( $1 \mathrm{AU} / \mathrm{dS}$ ) of the source. (mas)

## piE_E: float

The microlensing parallax in the East direction in units of thetaE

## piE_N: float

The microlensing parallax in the North direction in units of thetaE

## xS0_E: float

RA Source position on sky at $\mathrm{t}=\mathrm{t} 0(\operatorname{arcsec})$ in an arbitrary ref. frame.

## xS0_N: float

Dec source position on sky at $\mathrm{t}=\mathrm{t} 0(\operatorname{arcsec})$ in an arbitrary ref. frame.
muS_E: float
RA Source proper motion (mas/yr)
muS_N: float
Dec Source proper motion (mas/yr)
class model.PSPL_PhotParam1 ( $\left.t 0, u 0 \_a m p, t E, p i E \_E, p i E_{-} N, b \_s f f, m a g \_s r c, r a L=N o n e, d e c L=N o n e\right)$
Bases: PSPL_Param
PSPL model for photometry only.
Point source point lens model for microlensing photometry only. This model includes the relative proper motion between the lens and the source. Parameters are reduced with the use of piRel (rather than dL and dS) and muRel (rather than muL and muS).

Note the attributes, RA (raL) and Dec (decL) are required if you are calculating a model with parallax.

## Attributes

## t0: float

Time of photometric peak, as seen from Earth (MJD.DDD)

## u0_amp: float

Angular distance between the lens and source on the plane of the sky at closest approach in units of thetaE. It can be

- positive ( $u 0$ _amp $>0$ when $u 0 \_h a t[0]>0$ ) or
- negative (u0_amp < 0 when u0_hat[0] < 0).


## tE: float

Einstein crossing time in days.

## piE_E: float

The microlensing parallax in the East direction in units of thetaE.

## piE_N: float

The microlensing parallax in the North direction in units of thetaE
b_sff: numpy array or list
The ratio of the source flux to the total (source + neighbors + lens) $b_{s} f f=f_{S} /\left(f_{S}+f_{L}+\right.$
$\left.f_{N}\right)$. This must be passed in as a list or array, with one entry for each photometric filter.

## mag_src: numpy array or list

Photometric magnitude of the source. This must be passed in as a list or array, with one entry for each photometric filter.

## raL: float, optional

Right ascension of the lens in decimal degrees.

## decL: float, optional

Declination of the lens in decimal degrees.
class model.PSPL_PhotParam2 (t0, u0_amp, $t E$, piE_E, piE_N, b_sff, mag_base, raL=None, decL=None)
Bases: PSPL_Param
DESCRIPTION: Point source point lens model for microlensing photometry only. This model includes the relative proper motion between the lens and the source. Parameters are reduced with the use of piRel (rather than dL and dS) and muRel (rather than muL and muS). Same as PSPL_PhotParam1, except fits for mag_base instead of mag_src.

## Notes

Note: Required parameters if calculating with parallax

- raL: Right ascension of the lens in decimal degrees.
- decL: Declination of the lens in decimal degrees.


## Attributes

## t0: float

Time of photometric peak, as seen from Earth (MJD.DDD)

## u0_amp: float

Angular distance between the lens and source on the plane of the sky at closest approach in units of thetaE. It can be positive (u0_amp > 0 when u0_hat[0] > 0 ) or negative (u0_amp < 0 when u0_hat[0] < 0).

## tE: float

Einstein crossing time in days.

## piE_E: float

The microlensing parallax in the East direction in units of thetaE.

## piE_N: float

The microlensing parallax in the North direction in units of thetaE

## b_sff: numpy array or list

The ratio of the source flux to the total (source + neighbors + lens) $b_{s} f f=f_{S} /\left(f_{S}+f_{L}+\right.$ $\left.f_{N}\right)$. This must be passed in as a list or array, with one entry for each photometric filter.

## mag_base: numpy array or list

Photometric magnitude of the base. This must be passed in as a list or array, with one entry for each photometric filter.
class model.PSPL_PhotAstromParam1 $\left(m L, t 0, \operatorname{beta}, d L, d L_{-} d S, x S O_{-} E, x S O_{-} N, m u L_{-} E, m u L_{-} N, m u S \_E\right.$, muS_N, b_sff, mag_src, raL=None, $\operatorname{dec} L=N o n e)$
Bases: PSPL_Param
PSPL model for astrometry and photometry - physical parameterization.
A Point Source Point Lens model for microlensing. This model uses a parameterization that depends on only physical quantities such as the lens mass and positions and proper motions of both the lens and source.
Note the attributes, RA (raL) and Dec (decL) are required if you are calculating a model with parallax.

## Attributes

## mL: float

Mass of the lens (Msun)

## t0: float

Time of photometric peak, as seen from Earth (MJD.DDD)

## beta: float

Angular distance between the lens and source on the plane of the sky (mas). Can be

- positive (u0_amp > 0 when u0_hat[0] < 0) or
- negative ( $u 0$ _amp $<0$ when $u 0 \_h a t[0]>0$ ).


## dL: float

Distance from the observer to the lens (pc)

## dL_dS: float

Ratio of Distance from the obersver to the lens to Distance from the observer to the source

## xS0_E: float

RA Source position on sky at $\mathrm{t}=\mathrm{t} 0(\operatorname{arcsec})$ in an arbitrary ref. frame.

## xS0_N: float

Dec source position on sky at $\mathrm{t}=\mathrm{t} 0(\operatorname{arcsec})$ in an arbitrary ref. frame.

## muL_E: float

RA Lens proper motion (mas/yr)
muL_N: float
Dec Lens proper motion (mas/yr)
muS_E: float
RA Source proper motion (mas/yr)

## muS_N: float

Dec Source proper motion (mas/yr)
b_sff: numpy array or list
The ratio of the source flux to the total (source + neighbors + lens) $b_{s} f f=f_{S} /\left(f_{S}+f_{L}+\right.$
$\left.f_{N}\right)$. This must be passed in as a list or array, with one entry for each photometric filter.

## mag_src: numpy array or list

Photometric magnitude of the source. This must be passed in as a list or array, with one entry for each photometric filter.

## raL: float, optional

Right ascension of the lens in decimal degrees.

## decL: float, optional

Declination of the lens in decimal degrees.
class model.PSPL_PhotAstromParam2 ( $t 0, u 0 \_a m p, t E$, thetaE, $p i S, p i E \_E, p i E \_N, x S O \_E, x S O \_N, m u S \_E$, $\left.m u S \_N, b \_s f f, m a g \_s r c, r a L=N o n e, \operatorname{dec} L=N o n e\right)$
Bases: PSPL_Param
PSPL model for photometry and astrometry - photom-like parameterization
Point Source Point Lens model for microlensing. This model includes proper motions of the source and the source position on the sky.

## Attributes

## t0: float

Time of photometric peak, as seen from Earth (MJD.DDD)

## u0_amp: float

Angular distance between the lens and source on the plane of the sky at closest approach in units of thetaE. Can be

- positive (u0_amp > 0 when $u 0 \_h a t[0]>0$ ) or
- negative (u0_amp < 0 when u0_hat[0] < 0).


## tE: float

Einstein crossing time (days).

## thetaE: float

The size of the Einstein radius in (mas).

## piS: float

Amplitude of the parallax (1AU/dS) of the source. (mas)
piE_E: float
The microlensing parallax in the East direction in units of thetaE
piE_N: float
The microlensing parallax in the North direction in units of thetaE

## xS0_E: float

RA Source position on sky at $\mathrm{t}=\mathrm{t} 0(\operatorname{arcsec})$ in an arbitrary ref. frame.

## xS0_N: float

Dec source position on sky at $\mathrm{t}=\mathrm{t} 0(\operatorname{arcsec})$ in an arbitrary ref. frame.
muS_E: float
RA Source proper motion (mas/yr)

## muS_N: float

Source proper motion (mas/yr)
b_sff: numpy array or list
The ratio of the source flux to the total (source + neighbors + lens) $b_{s} f f=f_{S} /\left(f_{S}+f_{L}+\right.$
$\left.f_{N}\right)$. This must be passed in as a list or array, with one entry for each photometric filter.

## mag_src: numpy array or list

Photometric magnitude of the source. This must be passed in as a list or array, with one entry for each photometric filter.

## raL: float, optional

Right ascension of the lens in decimal degrees.

## decL: float, optional

Declination of the lens in decimal degrees.
class model.PSPL_PhotAstromParam3 $\left(t 0, u 0 \_a m p, t E, \log 10 \_t h e t a E, p i S, p i E \_E, p i E \_N, x S O \_E, x S O \_N\right.$, $\left.m u S \_E, m u S \_N, b \_s f f, m a g \_b a s e, r a L=N o n e, \operatorname{dec} L=N o n e\right)$
Bases: PSPL_Param
Point Source Point Lens model for microlensing. This model includes proper motions of the source and the source position on the sky. It is the same as PSPL_PhotAstromParam4 except it fits for $\log 10$ (thetaE) instead of thetaE.

## Notes

Note: Required parameters if calculating with parallax

- raL: Right ascension of the lens in decimal degrees.
- decL: Declination of the lens in decimal degrees.


## Attributes

t0
[float] Time of photometric peak, as seen from Earth (MJD.DDD)
u0_amp
[float] Angular distance between the source and the GEOMETRIC center of the lenses on the plane of the sky at closest approach in units of thetaE. Can be

- positive ( $u 0$ _amp $>0$ when $u 0 \_$hat $[0]>0$ ) or
- negative (u0_amp < 0 when u0_hat[0] < 0).
tE
[float] Einstein crossing time (days).
$\log 10 \_$thetaE
[float] The size of the Einstein radius in (mas).
piS
[float] Amplitude of the parallax (1AU/dS) of the source. (mas)
piE_E
[float] The microlensing parallax in the East direction in units of thetaE


## piE_N

[float] The microlensing parallax in the North direction in units of thetaE

## xSO_E

[float] R.A. of source position on sky at $\mathrm{t}=\mathrm{t} 0(\mathrm{arcsec})$ in an arbitrary ref. frame.

## xSO_N

[float] Dec. of source position on sky at $\mathrm{t}=\mathrm{t} 0(\mathrm{arcsec})$ in an arbitrary ref. frame.

## muS_E

[float] RA Source proper motion (mas/yr)
muS_N
[float] Dec Source proper motion (mas/yr)
b_sff
[numpy array or list] The ratio of the source flux to the total (source + neighbors + lenses). One for each filter.

$$
b_{s} f f=f_{S} /\left(f_{S}+f_{L}+f_{N}\right) .
$$

This must be passed in as a list or array, with one entry for each photometric filter.

## mag_base

[numpy array or list] Photometric magnitude of the base. This must be passed in as a list or array, with one entry for each photometric filter.
class model.PSPL_PhotAstromParam4 (to, uO_amp, tE, thetaE, piS, piE_E, piE_N, $x S O \_E, x S O \_N, m u S \_E$, muS_N, b_sff, mag_base, raL=None, $\operatorname{dec} L=$ None)
Bases: PSPL_Param
DESCRIPTION: Point Source Point Lens model for microlensing. This model includes proper motions of the source and the source position on the sky. It is the same as PSPL_PhotAstromParam2 except it fits for baseline instead of source magnitude.

## Parameters

to
[float] Time of photometric peak, as seen from Earth (MJD.DDD)
u0_amp
[float] Angular distance between the source and the GEOMETRIC center of the lenses on the plane of the sky at closest approach in units of thetaE. Can be

- positive (u0_amp >0 when u0_hat[0] >0) or
- negative (u0_amp < 0 when u0_hat[0] < 0 ).
tE
[float] Einstein crossing time (days).


## thetaE:

The size of the Einstein radius in (mas).

## piS

[float] Amplitude of the parallax ( $1 \mathrm{AU} / \mathrm{dS}$ ) of the source. (mas)
piE_E
[float] The microlensing parallax in the East direction in units of thetaE
piE_N
[float] The microlensing parallax in the North direction in units of thetaE

## xS0_E

[float] R.A. of source position on sky at $t=t 0(\operatorname{arcsec})$ in an arbitrary ref. frame.

## xSO_N

[float] Dec. of source position on sky at $t=t 0(\operatorname{arcsec})$ in an arbitrary ref. frame.
muS_E
[float] RA Source proper motion (mas/yr)
muS_N
[float] Dec Source proper motion (mas/yr)
b_sff
[numpy array or list] The ratio of the source flux to the total (source + neighbors + lenses). One for each filter.

$$
b_{s} f f=f_{S} /\left(f_{S}+f_{L}+f_{N}\right)
$$

This must be passed in as a list or array, with one entry for each photometric filter.

## mag_base

[numpy array or list] Photometric magnitude of the base. This must be passed in as a list or array, with one entry for each photometric filter.

## Notes

Note: Required parameters if calculating with parallax

- raL: Right ascension of the lens in decimal degrees.
- decL: Declination of the lens in decimal degrees.
class model.PSPL_PhotAstromParam5 ( $t 0$, u0_amp, $t E$, log10_thetaE, $p i S, p i E \_E, p i E N \_p i E E, x S 0 \_E, x S 0 \_N$, $\left.m u S \_E, m u S \_N, b \_s f f, m a g \_b a s e, r a L=N o n e, d e c L=N o n e\right)$
Bases: PSPL_Param
DESCRIPTION: Point Source Point Lens model for microlensing. This model includes proper motions of the source and the source position on the sky. It fits for piEN/piEE and piEE, instead of piEE and piEN.


## Notes

Note: Required parameters if calculating with parallax

- raL: Right ascension of the lens in decimal degrees.
- decL: Declination of the lens in decimal degrees.


## Attributes

t0
[float] Time of photometric peak, as seen from Earth (MJD.DDD)
u0_amp
[float] Angular distance between the source and the lens on the plane of the sky at closest approach in units of thetaE. Can

- positive (u0_amp > 0 when $u 0 \_$hat $[0]>0$ ) or
- negative (u0_amp < 0 when u0_hat[0] < 0).
tE
[float] Einstein crossing time (days).


## $\log 10$ _theta $E$

[float] The size of the Einstein radius in (mas).

## piS

[float] Amplitude of the parallax (1AU/dS) of the source. (mas)

```
piEN_piEE
```

[float] Ratio of piE_N to piE_E.

## piE_E

[float] The microlensing parallax in the East direction in units of thetaE

## xS0_E

[float] R.A. of source position on sky at $\mathrm{t}=\mathrm{t} 0(\operatorname{arcsec})$ in an arbitrary ref. frame.

## xS0_N

[float] Dec. of source position on sky at $\mathrm{t}=\mathrm{t} 0(\operatorname{arcsec})$ in an arbitrary ref. frame.
muS_E
[float] RA Source proper motion (mas/yr)
muS_N
[float] Dec Source proper motion (mas/yr)
b_sff
[numpy array or list] The ratio of the source flux to the total (source + neighbors + lenses). One for each filter.

$$
b_{s} f f=f_{S} /\left(f_{S}+f_{L}+f_{N}\right)
$$

This must be passed in as a list or array, with one entry for each photometric filter.

## mag_base

[numpy array or list] Photometric magnitude of the base. This must be passed in as a list or array, with one entry for each photometric filter.

### 5.1.2 GP Models

class model.PSPL_GP_PhotParam1 (t0, u0_amp, $t E$, $p i E \_E, p i E \_N, b_{-} s f f, m a g \_s r c, g p \_l o g \_s i g m a, g p \_l o g \_r h o$, gp_log_SO, gp_log_omega0, raL=None, decL=None)
Bases: PSPL_PhotParam1
class model.PSPL_GP_PhotParam1_2 $\left(t 0, u 0 \_a m p, t E, p i E \_E, p i E \_N, b \_s f, m a g \_s r c, g p \_l o g \_s i g m a, g p \_r h o\right.$, gp_log_omega04_SO, gp_log_omega0, raL=None, decL=None)
Bases: PSPL_PhotParam1
Figuring out the new prior parametrization.
class model.PSPL_GP_PhotParam2 (t0, u0_amp, tE, piE_E, piE_N, b_sff, mag_base, gp_log_sigma, gp_log_rho, gp_log_SO, gp_log_omega0, raL=None, decL=None)
Bases: PSPL_PhotParam2

```
class model.PSPL_GP_PhotParam2_2 (t0, u0_amp, tE, piE_E, piE_N, b_sff, mag_base, gp_log_sigma, gp_rho,
                gp_log_omega04_SO, gp_log_omega0, raL=None, \(\operatorname{dec} L=\) None \()\)
    Bases: PSPL_PhotParam2
class model.PSPL_GP_PhotAstromParam1 \(\left(m L, t 0\right.\), beta, \(d L, d L_{-} d S, x S O_{-} E, x S O_{-} N, m u L \_E, m u L_{-} N\), \(m u S \_E\),
                        muS_N, b_sff, mag_src, gp_log_sigma, gp_rho,
                        \(\left.g p \_l o g \_o m e g a 04 \_S 0, g p \_l o g \_o m e g a 0, r a L=N o n e, d e c L=N o n e\right)\)
```

Bases: PSPL_PhotAstromParam1
class model.PSPL_GP_PhotAstromParam2 ( $t 0, u 0_{-} a m p, t E$, thetaE, $p i S, p i E \_E, p i E \_N, x S O_{-} E, x S O \_N, m u S_{-} E$, $m u S \_N, b \_s f f, m a g \_s r c, g p \_l o g \_s i g m a, g p \_r h o$, gp_log_omega04_SO, gp_log_omega0, raL=None, decL=None)

Bases: PSPL_PhotAstromParam2
class model.PSPL_GP_PhotAstromParam2 ( $t 0, u 0_{-} a m p, t E$, thetaE, $p i S, p i E \_E, p i E \_N, x S O \_E, x S O \_N, m u S \_E$, muS_N, b_sff, mag_src, gp_log_sigma, gp_rho, $\left.g p \_l o g \_o m e g a 04 \_S 0, g p \_l o g \_o m e g a 0, r a L=N o n e, d e c L=N o n e\right)$
Bases: PSPL_PhotAstromParam2
class model.PSPL_GP_PhotAstromParam3 $\left(t 0, u 0 \_a m p, t E, \log 10 \_t h e t a E, p i S, p i E \_E, p i E \_N, x S O \_E, x S O \_N\right.$, $m u S \_E, m u S \_N, b_{-} s f f, m a g \_b a s e$, gp_log_sigma, gp_rho, gp_log_omega04_S0, gp_log_omega0, raL=None, $\operatorname{dec} L=$ None)
Bases: PSPL_PhotAstromParam3
Point Source Point Lens with GP model for microlensing. This model includes proper motions of the source and the source position on the sky. It is the same as PSPL_PhotAstromParam4 except it fits for $\log 10$ (thetaE) instead of thetaE.

## Notes

Note: raL and $\operatorname{dec} L$ are required parameters if calculating with parallax

## Attributes

## t0: float

Time of photometric peak, as seen from Earth (MJD.DDD)

## u0_amp: float

Angular distance between the lens and source on the plane of the sky at closest approach in units of thetaE. Can be

- positive (u0_amp >0 when u0_hat[0] >0) or
- negative (u0_amp < 0 when u0_hat[0] < 0).
tE: float
Einstein crossing time (days).
$\log 10 \_$thetaE: float
$\log 10$ of the size of the Einstein radius in (mas).
piS: float
Amplitude of the parallax ( $1 \mathrm{AU} / \mathrm{dS}$ ) of the source. (mas)
piE_E: float
The microlensing parallax in the East direction in units of thetaE


## piE_N: float

The microlensing parallax in the North direction in units of thetaE

## xS0_E: float

RA Source position on sky at $\mathrm{t}=\mathrm{t} 0(\operatorname{arcsec})$ in an arbitrary ref. frame.

## xS0_N: float

Dec source position on sky at $\mathrm{t}=\mathrm{t} 0(\operatorname{arcsec})$ in an arbitrary ref. frame.

## muS_E: float

RA Source proper motion (mas/yr)

## muS_N: float

Dec Source proper motion (mas/yr)

## b_sff: numpy array or list of floats

The ratio of the source flux to the total (source + neighbors + lens) $b_{s} f f=f_{S} /\left(f_{S}+f_{L}+\right.$ $\left.f_{N}\right)$. This must be passed in as a list or array, with one entry for each photometric filter.

## mag_base: numpy array or list of floats

Photometric magnitude of the base. This must be passed in as a list or array, with one entry for each photometric filter.

## gp_log_sigma: float

Guassian process $\log (\sigma)$ for the Matern 3/2 kernel.

## gp_rho: float

Guassian process $\rho$ for the Matern 3/2 kernel.

## gp_log_omega04_S0: float

Guassian process $\log \left(\omega_{0}^{4} * S_{0}\right)$ from the SHO kernel.

## gp_log_omega0: float

Guassian process $\log \left(\omega_{0}\right)$ from the SHO kernel.

## raL: float, optional

Right ascension of the lens in decimal degrees.

## decL: float, optional

Declination of the lens in decimal degrees.
class model.PSPL_GP_PhotAstromParam4 $\left(t 0, u 0 \_a m p, t E\right.$, theta $E, p i S, p i E_{-} E, p i E \_N, x S 0 \_E, x S 0 \_N, m u S \_E$, muS_N, b_sff,mag_base, gp_log_sigma, gp_rho, gp_log_omega04_SO, gp_log_omega0, raL=None, $\operatorname{dec} L=N o n e)$
Bases: PSPL_PhotAstromParam4
Point Source Point Lens with GP model for microlensing. This model includes proper motions of the source and the source position on the sky. It is the same as PSPL_PhotAstromParam2 except it fits for baseline instead of source magnitude.

## Notes

## Note:

$r a L$ and $d e c L$ are required parameters if calculating with parallax
For an explanation of the Guassian process parameters, see Golovich et al. 2019()

## Attributes

## t0: float

Time of photometric peak, as seen from Earth (MJD.DDD)

## u0_amp: float

Angular distance between the lens and source on the plane of the sky at closest approach in units of thetaE. Can be

- positive (u0_amp > 0 when u0_hat[0] >0) or
- negative (u0_amp < 0 when u0_hat[0] < 0).


## tE: float

Einstein crossing time (days).

## thetaE: float

The size of the Einstein radius in (mas).

## piS: float

Amplitude of the parallax (1AU/dS) of the source. (mas)
piE_E: float
The microlensing parallax in the East direction in units of thetaE

## piE_N: float

The microlensing parallax in the North direction in units of thetaE

## xS0_E: float

RA Source position on sky at $\mathrm{t}=\mathrm{t} 0(\operatorname{arcsec})$ in an arbitrary ref. frame.

## xS0_N: float

Dec source position on sky at $\mathrm{t}=\mathrm{t} 0(\operatorname{arcsec})$ in an arbitrary ref. frame.

## muS_E: float

RA Source proper motion (mas/yr)
muS_N: float
Dec Source proper motion (mas/yr)
b_sff: float
The ratio of the source flux to the total (source + neighbors + lens) $b_{s} f f=f_{S} /\left(f_{S}+f_{L}+\right.$ $\left.f_{N}\right)$. This must be passed in as a list or array, with one entry for each photometric filter.
mag_base: float
Photometric magnitude of the base. This must be passed in as a list or array, with one entry for each photometric filter.

## gp_log_sigma: float

Guassian process $\log (\sigma)$ for the Matern 3/2 kernel.

## gp_rho: float

Guassian process $\rho$ for the Matern 3/2 kernel.

# gp_log_omega04_S0: float 

Guassian process $\log \left(\omega_{0}^{4} * S_{0}\right)$ from the SHO kernel.
gp_log_omega0: float
Guassian process $\log \left(\omega_{0}\right)$ from the SHO kernel.
raL: float, optional
Right ascension of the lens in decimal degrees.
decL: float, optional
Declination of the lens in decimal degrees.

### 5.2 Data Class Family

```
class model.PSPL
```

Bases: ABC

Methods
animate(tE, time_steps, frame_time, name, ...) Produces animation of microlensing event.

| get_chi2_photometry |  |
| :--- | :--- |
| get_lnL_constant |  |
| get_photometry |  |
| log_likely_photometry |  |
| log_likely_photometry_each |  |

animate (tE, time_steps, frame_time, name, size, zoom, astrometry)
Produces animation of microlensing event. This function takes the PSPL and makes an animation, the input variables are as follows

## Parameters

tE:
number of einstein crossings times before/after the peak you want the animation to plot
e.g tE $=2=>$ graph will go from -2 tE to 2 tE
time_steps:
number of time steps before/after peak, so total number of time steps will be 2 times this value

## frame_time:

times in ms of each frame in the animation

## name: string

the animation will be saved as name.html

## size: list

[horizontal, vertical] cm's

## zoom:

\# of einstein radii plotted in vertical direction

## class model.PSPL_Phot

Bases: PSPL
Contains methods for model a PSPL photometry only. This is a Data-type class in our hierarchy. It is abstract and should not be instantiated.

Methods
animate(tE, time_steps, frame_time, name, ...) Produces animation of microlensing event.

| get_astrometry |  |
| :--- | :--- |
| get_astrometry_unlensed |  |
| get_centroid_shift |  |
| get_chi2_photometry |  |
| get_lens_astrometry |  |
| get_lnL_constant |  |
| get_photometry |  |
| get_resolved_amplification |  |
| get_resolved_astrometry |  |
| log_likely_astrometry |  |
| log_likely_photometry |  |
| log_likely_photometry_each |  |

animate ( $t$ E, time_steps, frame_time, name, size, zoom, astrometry)
Produces animation of microlensing event. This function takes the PSPL and makes an animation, the input variables are as follows

## Parameters

tE:
number of einstein crossings times before/after the peak you want the animation to plot
e.g tE $=2=>$ graph will go from -2 tE to 2 tE
time_steps:
number of time steps before/after peak, so total number of time steps will be 2 times this value

## frame_time:

times in ms of each frame in the animation
name: string
the animation will be saved as name.html
size: list
[horizontal, vertical] cm's

## zoom:

\# of einstein radii plotted in vertical direction
class model.PSPL_PhotAstrom
Bases: PSPL
Contains methods for model a PSPL photometry + astrometry. This is a Data-type class in our hierarchy. It is abstract and should not be instantiated.

## Methods

animate(tE, time_steps, frame_time, name, ...) Produces animation of microlensing event.

| get_chi2_astrometry |  |
| :--- | :--- |
| get_chi2_photometry |  |
| get_lnL_constant |  |
| get_photometry |  |
| log_likely_astrometry |  |
| log_likely_astrometry_each |  |
| log_likely_photometry |  |
| log_likely_photometry_each |  |

animate ( $t$ E, time_steps, frame_time, name, size, zoom, astrometry)
Produces animation of microlensing event. This function takes the PSPL and makes an animation, the input variables are as follows

## Parameters

tE:
number of einstein crossings times before/after the peak you want the animation to plot
e.g tE $=2$ => graph will go from -2 tE to 2 tE

## time_steps:

number of time steps before/after peak, so total number of time steps will be 2 times this value

## frame_time:

times in ms of each frame in the animation
name: string
the animation will be saved as name.html
size: list
[horizontal, vertical] cm's

## zoom:

\# of einstein radii plotted in vertical direction
class model.PSPL_Astrom
Bases: PSPL
Contains methods for model a PSPL photometry + astrometry. This is a Data-type class in our hierarchy. It is abstract and should not be instantiated.

## Methods

animate(tE, time_steps, frame_time, name, ...) Produces animation of microlensing event.

| get_chi2_astrometry |  |
| :--- | :--- |
| get_chi2_photometry |  |
| get_lnL_constant |  |
| get_photometry |  |
| log_likely_astrometry |  |
| log_likely_astrometry_each |  |
| log_likely_photometry |  |
| log_likely_photometry_each |  |

animate ( $t$ E, time_steps, frame_time, name, size, zoom, astrometry)
Produces animation of microlensing event. This function takes the PSPL and makes an animation, the input variables are as follows

## Parameters

tE:
number of einstein crossings times before/after the peak you want the animation to plot
e.g tE $=2$ => graph will go from -2 tE to 2 tE

## time_steps:

number of time steps before/after peak, so total number of time steps will be 2 times this value

## frame_time:

times in ms of each frame in the animation

## name: string

the animation will be saved as name.html

## size: list

[horizontal, vertical] cm's

## zoom:

\# of einstein radii plotted in vertical direction

### 5.3 GP Class Family

```
class model.Celerite_GP_Model(pspl_model,filter_index)
```

Bases: Model
This is nedeed for the GP. Just a wrapper over our model so it is a celerite model.

## Attributes

full_size
The total number of parameters (including frozen parameters)
parameter_vector
An array of all parameters (including frozen parameters)
vector_size
The number of active (or unfrozen) parameters

## Methods

| compute_gradient(*args, **kwargs) | Compute the "gradient" of the model for the current <br> parameters |
| :--- | :--- |
| freeze_all_parameters() | Freeze all parameters of the model |
| freeze_parameter(name) | Freeze a parameter by name |
| get_parameter(name) | Get a parameter value by name |
| get_parameter_bounds([include_frozen]) | Get a list of the parameter bounds |
| get_parameter_dict([include_frozen]) | Get an ordered dictionary of the parameters |
| get_parameter_names([include_frozen]) | Get a list of the parameter names |
| get_parameter_vector([include_frozen]) | Get an array of the parameter values in the correct <br> order |
| get_value(t_obs) | Compute the "value" of the model for the current pa- <br> rameters |
| log_prior() | Compute the log prior probability of the current pa- <br> rameters |
| set_parameter(name, value) | Set a parameter value by name |
| set_parameter_vector(vector[, include_frozen]) | Set the parameter values to the given vector |
| thaw_all_parameters() | Thaw all parameters of the model |
| thaw_parameter(name) | Thaw a parameter by name |

## get_gradient

get_value (t_obs)
Compute the "value" of the model for the current parameters
This method should be overloaded by subclasses to implement the actual functionality of the model.
compute_gradient (*args, **kwargs)
Compute the "gradient" of the model for the current parameters
This method should be overloaded by subclasses to implement the actual functionality of the model. The output of this function should be an array where the first dimension is full_size.

## freeze_all_parameters()

Freeze all parameters of the model

## freeze_parameter (name)

Freeze a parameter by name
Args:
name: The name of the parameter

## property full_size

The total number of parameters (including frozen parameters)

## get_parameter (name)

Get a parameter value by name
Args:
name: The name of the parameter

```
get_parameter_bounds(include_frozen=False)
```

Get a list of the parameter bounds
Args:
include_frozen (Optional[bool]): Should the frozen parameters be included in the returned value? (default: False)
get_parameter_dict(include_frozen=False)
Get an ordered dictionary of the parameters
Args:
include_frozen (Optional[bool]): Should the frozen parameters be included in the returned value? (default: False)
get_parameter_names(include_frozen=False)
Get a list of the parameter names
Args:
include_frozen (Optional[bool]): Should the frozen parameters be included in the returned value? (default: False)
get_parameter_vector(include_frozen=False)
Get an array of the parameter values in the correct order
Args:
include_frozen (Optional[bool]): Should the frozen parameters be included in the returned value? (default: False)

## log_prior ()

Compute the $\log$ prior probability of the current parameters
property parameter_vector
An array of all parameters (including frozen parameters)

```
set_parameter(name, value)
```

Set a parameter value by name
Args:
name: The name of the parameter value (float): The new value for the parameter

```
set_parameter_vector(vector, include_frozen=False)
```

Set the parameter values to the given vector
Args:
vector (array[vector_size] or array[full_size]): The target
parameter vector. This must be in the same order as parameter_names and it should only include frozen parameters if include_frozen is True.
include_frozen (Optional[bool]): Should the frozen parameters be included in the returned value? (default: False)
thaw_all_parameters()
Thaw all parameters of the model
thaw_parameter (name)
Thaw a parameter by name

## Args:

name: The name of the parameter
property vector_size
The number of active (or unfrozen) parameters
class model.PSPL_GP
Bases: ABC
PSPL object that has optional support for gaussian process on each photometric filter.

## Methods

get_log_det_covariance(t_obs, mag_obs, ...) Returns photometry with GP noise added in.
get_photometry_with_gp(t_obs, mag_obs, ...) Returns photometry with GP noise added in.
log_likely_photometry(t_obs, mag_obs, ...[, ...]) Calculate the log-likelihood for the PSPL + GP model and photometric data.
get_photometry_with_gp(t_obs, mag_obs, mag_err_obs, filt_index=0, t_pred=None)
Returns photometry with GP noise added in.

Note: This will throw an error if this is a filter with use_gp_phot[filt_index] = False.
get_log_det_covariance (t_obs, mag_obs, mag_err_obs, filt_index=0, t_pred=None)
Returns photometry with GP noise added in.

Note: This will throw an error if this is a filter with use_gp_phot[filt_index] = False.
log_likely_photometry (t_obs, mag_obs, mag_err_obs, filt_index=0)
Calculate the log-likelihood for the PSPL + GP model and photometric data.

Note: The GP will only be used for filters where use_gp_phot[filt_index]=True.
class model.PSPL_GP
Bases: ABC
PSPL object that has optional support for gaussian process on each photometric filter.

## Methods

get_log_det_covariance(t_obs, mag_obs, ...) Returns photometry with GP noise added in.
get_photometry_with_gp(t_obs, mag_obs, ...) Returns photometry with GP noise added in.
log_likely_photometry(t_obs, mag_obs, ...[, ...]) Calculate the log-likelihood for the PSPL + GP model and photometric data.
get_photometry_with_gp(t_obs, mag_obs, mag_err_obs, filt_index=0, t_pred=None)
Returns photometry with GP noise added in.

Note: This will throw an error if this is a filter with use_gp_phot[filt_index] = False.
get_log_det_covariance (t_obs, mag_obs, mag_err_obs, filt_index=0, t_pred=None)
Returns photometry with GP noise added in.

Note: This will throw an error if this is a filter with use_gp_phot $[$ fil_index $]=$ False.
log_likely_photometry (t_obs, mag_obs, mag_err_obs, filt_index=0)
Calculate the log-likelihood for the PSPL + GP model and photometric data.
Note: The GP will only be used for filters where use_gp_phot[fil__index] = True.

### 5.4 Parallax Class Family

class model.PSPL_noParallax

Bases: ParallaxClassABC

## Methods

| get_amplification(t) | noParallax: Get the photometric amplification term <br> at a set of times, t. |
| :--- | :--- |
| get_astrometry(t_obs[, ast_filt_idx]) | noParallax: Position of the observed source position <br> in arcsec. |
| get_astrometry_unlensed(t_obs) | noParallax: Get the astrometry of the source if the <br> lens didn't exist. |
| get_centroid_shift(t) | noParallax: Get the centroid shift (in mas) for a list <br> of observation times (in MJD). |
| get_lens_astrometry(t_obs) | Equation of motion for just the foreground lens. |
| get_resolved_amplification(t) | Get the photometric amplification term at a set of <br> times, t for both the plus and minus images. |
| get_resolved_astrometry(t_obs) | Get the $x, y$ astrometry for each of the two source <br> images, which we label plus and minus. |

## calc_piE_ecliptic

## get_amplification $(t)$

noParallax: Get the photometric amplification term at a set of times, $t$.

## Parameters

t:
Array of times in MJD.DDD
get_lens_astrometry (t_obs)
Equation of motion for just the foreground lens.

## Parameters

t_obs
[array_like] Time (in MJD).
get_astrometry (t_obs, ast_filt_idx=0)
noParallax: Position of the observed source position in arcsec.
get_centroid_shift ( $t$ )
noParallax: Get the centroid shift (in mas) for a list of observation times (in MJD).
get_astrometry_unlensed (t_obs)
noParallax: Get the astrometry of the source if the lens didn't exist.

## Returns

xS_unlensed
[numpy array, dtype=float, shape $=$ len(t_obs) $\times 2$ ] The unlensed positions of the source in arcseconds.

```
get_resolved_amplification(t)
```

Get the photometric amplification term at a set of times, t for both the plus and minus images.

## Parameters

t:
Array of times in MJD.DDD

## get_resolved_astrometry (t_obs)

Get the x , y astrometry for each of the two source images, which we label plus and minus.

## Returns

[xS_plus, xS_minus]
[list of numpy arrays]

- xS_plus is the vector position of the plus image in arcsec
- xS_minus is the vector position of the plus image in arcsec
class model.PSPL_Parallax
Bases: ParallaxClassABC


## Methods

| calc_piE_ecliptic() | Parallax: Get piE_ecliptic |
| :--- | :--- |
| get_amplification(t) | Parallax: Get the photometric amplification term at a <br> set of times, t. |
| get_astrometry(t_obs[, ast_filt_idx]) | Parallax: Get astrometry <br> get_astrometry_unlensed(t_obs) <br> ist. |
| get_centroid_shift(t[, ast_filt_idx]) | Parallax: Get the centroid shift (in mas) for a list of <br> observation times (in MJD). |
| get_geoproj_ast_params(t0par) | Parallax: Get lens astrometry |
| get_geoproj_params(t0par) | Parallax: Get the photometric amplification term at a <br> set of times, $t$ for both the plus and minus images. |
| get_lens_astrometry(t_obs) | Parallax: Get the x, y astrometry for each of the two <br> source images, which we label plus and minus. |
| get_resolved_amplification(t) |  |

## start

## get_amplification $(t)$

Parallax: Get the photometric amplification term at a set of times, t .

## Parameters

t:
Array of times in MJD.DDD

## get_lens_astrometry (t_obs)

Parallax: Get lens astrometry
get_astrometry (t_obs, ast_filt_idx=0)
Parallax: Get astrometry
get_centroid_shift ( $t$, ast_filt_idx=0)
Parallax: Get the centroid shift (in mas) for a list of observation times (in MJD).
get_astrometry_unlensed(t_obs)
Get the astrometry of the source if the lens didn't exist.

## Returns

## xS_unlensed

[numpy array, dtype=float, shape $=$ len(t_obs) $\times 2$ 2] The unlensed positions of the source in arcseconds.

```
get_resolved_amplification(t)
```

Parallax: Get the photometric amplification term at a set of times, $t$ for both the plus and minus images.

## Parameters

t:
Array of times in MJD.DDD

## get_resolved_astrometry(t_obs)

Parallax: Get the x , y astrometry for each of the two source images, which we label plus and minus.

## Returns

[xS_plus, xS_minus]
[list of numpy arrays]

- xS_plus is the vector position of the plus image.
- xS_minus is the vector position of the plus image.
calc_piE_ecliptic()
Parallax: Get piE_ecliptic
class model.PSPL_Parallax
Bases: ParallaxClassABC


## Methods

| calc_piE_ecliptic() | Parallax: Get piE_ecliptic |
| :--- | :--- |
| get_amplification(t) | Parallax: Get the photometric amplification term at a <br> set of times, t. |
| get_astrometry(t_obs[, ast_filt_idx]) | Parallax: Get astrometry |
| get_astrometry_unlensed(t_obs) | Get the astrometry of the source if the lens didn't ex- <br> ist. |
| get_centroid_shift(t[, ast_filt_idx]) | Parallax: Get the centroid shift (in mas) for a list of <br> observation times (in MJD). |
| get_geoproj_ast_params(t0par) | Parallax: Get lens astrometry <br> Parallax: Get the photometric amplification term at a <br> set of times, tor both the plus and minus images. |
| get_lens_astrometry(t_obs) | Parallax: Get the x, y astrometry for each of the two <br> source images, which we label plus and minus. |
| get_resolved_amplification(t) |  |
| get_resolved_astrometry(t_obs) |  |

## start

## get_amplification $(t)$

Parallax: Get the photometric amplification term at a set of times, t.

## Parameters

t:
Array of times in MJD.DDD
get_lens_astrometry(t_obs)
Parallax: Get lens astrometry
get_astrometry (t_obs, ast_fil_idx=0)
Parallax: Get astrometry

```
get_centroid_shift (t, ast_fil_idx=0)
```

Parallax: Get the centroid shift (in mas) for a list of observation times (in MJD).

```
get_astrometry_unlensed(t_obs)
```

Get the astrometry of the source if the lens didn't exist.

## Returns

## xS_unlensed

[numpy array, dtype=float, shape $=$ len(t_obs) x 2] The unlensed positions of the source in arcseconds.

## get_resolved_amplification $(t)$

Parallax: Get the photometric amplification term at a set of times, t for both the plus and minus images.

## Parameters

t:
Array of times in MJD.DDD

## get_resolved_astrometry (t_obs)

Parallax: Get the x , y astrometry for each of the two source images, which we label plus and minus.

## Returns

## [xS_plus, xS_minus]

[list of numpy arrays]

- xS_plus is the vector position of the plus image.
- xS_minus is the vector position of the plus image.

```
calc_piE_ecliptic()
```

Parallax: Get piE_ecliptic

```
class model.PSPL_noParallax_LumLens
```

Bases: PSPL_noParallax

## Methods

| get_amplification(t) | noParallax: Get the photometric amplification term <br> at a set of times, t. |
| :--- | :--- |
| get_astrometry(t_obs[, ast_filt_idx]) | noParallax: Position of the observed source position <br> in arcsec. |
| get_astrometry_unlensed(t_obs) | noParallax: Get the astrometry of the source if the <br> lens didn't exist. |
| get_centroid_shift(t[, ast_filt_idx]) | noParallax: Get the centroid shift (in mas) for a list <br> of observation times (in MJD). |
| get_lens_astrometry(t_obs) | Equation of motion for just the foreground lens. <br> get_resolved_amplification(t) <br> times, photometric amplification term at a set of the plus and minus images. |
| get_resolved_astrometry(t_obs) | Get the x, y astrometry for each of the two source <br> images, which we label plus and minus. |

## calc_piE_ecliptic

get_centroid_shift ( $t$, ast_filt_idx=0)
noParallax: Get the centroid shift (in mas) for a list of observation times (in MJD).
get_amplification $(t)$
noParallax: Get the photometric amplification term at a set of times, t .

## Parameters

t:
Array of times in MJD.DDD
get_astrometry (t_obs, ast_filt_idx=0)
noParallax: Position of the observed source position in arcsec.
get_astrometry_unlensed(t_obs)
noParallax: Get the astrometry of the source if the lens didn't exist.

## Returns

xS_unlensed
[numpy array, dtype=float, shape $=$ len(t_obs) $\times 2$ 2] The unlensed positions of the source in arcseconds.

```
get_lens_astrometry(t_obs)
```

Equation of motion for just the foreground lens.

## Parameters

t_obs
[array_like] Time (in MJD).
get_resolved_amplification $(t)$
Get the photometric amplification term at a set of times, $t$ for both the plus and minus images.

## Parameters

t:
Array of times in MJD.DDD

## get_resolved_astrometry (t_obs)

Get the x , y astrometry for each of the two source images, which we label plus and minus.

## Returns

[xS_plus, xS_minus]
[list of numpy arrays]

- xS_plus is the vector position of the plus image in arcsec
- xS_minus is the vector position of the plus image in arcsec
class model.PSPL_Parallax_LumLens
Bases: PSPL_Parallax


## Methods

| calc_piE_ecliptic() | Parallax: Get piE_ecliptic |
| :--- | :--- |
| get_amplification(t) | Parallax: Get the photometric amplification term at a <br> set of times, t. |
| get_astrometry(t_obs[, ast_filt_idx]) | Parallax: Get astrometry <br> get_astrometry_unlensed(t_obs) <br> ist. |
| get_centroid_shift(t[, ast_filt_idx]) | Parallax: Get the centroid shift (in mas) for a list of <br> observation times (in MJD). |
| get_geoproj_ast_params(t0par) |  |
| get_geoproj_params(t0par) | Parallax: Get lens astrometry <br> set of times, the for both the plus and minus images. |
| get_lens_astrometry(t_obs) | Parallax: Get the x, y astrometry for each of the two <br> source images, which we label plus and minus. |
| get_resolved_amplification(t) |  |

## start

get_astrometry (t_obs, ast_filt_idx=0)
Parallax: Get astrometry

```
get_centroid_shift(t,ast_filt_idx=0)
```

Parallax: Get the centroid shift (in mas) for a list of observation times (in MJD).

## calc_piE_ecliptic()

Parallax: Get piE_ecliptic
get_amplification $(t)$
Parallax: Get the photometric amplification term at a set of times, t .

## Parameters

t:
Array of times in MJD.DDD
get_astrometry_unlensed(t_obs)
Get the astrometry of the source if the lens didn't exist.

## Returns

## xS_unlensed

[numpy array, dtype=float, shape $=$ len(t_obs) $\times 2$ 2] The unlensed positions of the source in arcseconds.

## get_lens_astrometry (t_obs)

Parallax: Get lens astrometry

## get_resolved_amplification $(t)$

Parallax: Get the photometric amplification term at a set of times, $t$ for both the plus and minus images.

## Parameters

t:
Array of times in MJD.DDD
get_resolved_astrometry (t_obs)
Parallax: Get the $\mathrm{x}, \mathrm{y}$ astrometry for each of the two source images, which we label plus and minus.

## Returns

[xS_plus, xS_minus]
[list of numpy arrays]

- xS_plus is the vector position of the plus image.
- xS_minus is the vector position of the plus image.

```
class model.PSPL_Parallax
```

Bases: ParallaxClassABC

Methods

| calc_piE_ecliptic() | Parallax: Get piE_ecliptic |
| :--- | :--- |
| get_amplification(t) | Parallax: Get the photometric amplification term at a <br> set of times, t. |
| get_astrometry(t_obs[, ast_filt_idx]) | Parallax: Get astrometry <br> get_astrometry_unlensed(t_obs) <br> ist. |
| get_centroid_shift(t[, ast_filt_idx]) | Parallax: Get the centroid shift (in mas) for a list of <br> observation times (in MJD). |
| get_geoproj_ast_params(t0par) |  |
| get_geoproj_params(t0par) | Parallax: Get lens astrometry <br> set of times, t for both the plus and minus images. |
| get_lens_astrometry(t_obs) | Parallax: Get the x, y astrometry for each of the two <br> source images, which we label plus and minus. |
| get_resolved_amplification(t) |  |

## start

## get_amplification $(t)$

Parallax: Get the photometric amplification term at a set of times, t.

## Parameters

t:
Array of times in MJD.DDD
get_lens_astrometry (t_obs)
Parallax: Get lens astrometry
get_astrometry (t_obs, ast_fil_idx=0)
Parallax: Get astrometry
get_centroid_shift ( $t$, ast_filt_idx=0)
Parallax: Get the centroid shift (in mas) for a list of observation times (in MJD).

## get_astrometry_unlensed (t_obs)

Get the astrometry of the source if the lens didn't exist.

## Returns

xS_unlensed
[numpy array, dtype=float, shape $=$ len(t_obs) x 2] The unlensed positions of the source in arcseconds.

## get_resolved_amplification ( $t$ )

Parallax: Get the photometric amplification term at a set of times, t for both the plus and minus images.

## Parameters

t:
Array of times in MJD.DDD

## get_resolved_astrometry (t_obs)

Parallax: Get the x , y astrometry for each of the two source images, which we label plus and minus.

## Returns

[xS_plus, xS_minus]
[list of numpy arrays]

- xS_plus is the vector position of the plus image.
- xS_minus is the vector position of the plus image.
calc_piE_ecliptic()
Parallax: Get piE_ecliptic


## POINT SOURCE BINARY LENS (PSBL) CLASSES

### 6.1 Data Class Family

## class model. PSBL

Bases: PSPL
Contains methods for model a PSBL photometry + astrometry. This is a Data-type class in our hierarchy. It is abstract and should not be instantiated.

## Methods

| animate(tE, time_steps, frame_time, name, ...) | Produces animation of microlensing event. |
| :--- | :--- |
| get_all_arrays(t_obs[, check_sols, rescale]) | Obtain the image and amplitude arrays for each t_obs. |
| get_amp_arr(z_arr, z1, z2) | Calculations amplification array |
| get_image_pos_arr(w, z1, z2, m1, m2[, ...]) | Gets image positions. |
| get_image_pos_arr_old(w, z1, z2[, check_sols]) | Gets image positions. |
| get_photometry(t_obs[, filt_idx, amp_arr, ...]) | Get the photometry for each of the lensed source im- <br> ages. |
| get_resolved_photometry(t_obs[, filt_idx, ...]) | Get the photometry for each of the lensed source im- <br> ages. |
| rescale_complex_pos(w, z1, z2) | Make sure everything is roughly centered on the ori- <br> gin in a 1 x 1 box. |


| get_chi2_photometry |  |
| :--- | :--- |
| get_lnL_constant |  |
| log_likely_photometry |  |
| log_likely_photometry_each |  |

get_amp_arr (z_arr, z1, z2)
Calculations amplification array
Calculates the amplification A from the Jacobian J, $A=1 /|J|$

## Parameters

## z_arr

[array_like]
Complex position of images. Shape $=$ [N_times, N_solutions, 1]

- note this could be jagged.
z1
[array_like] Complex position(s) of lens 1 (primary). Shape = [N_times, 1]
z2
[array_like] Complex position(s) of lens 2 (secondary). Shape = [N_times, 1]


## Returns

amp_arr
[array_like] BLEH
rescale_complex_pos $(w, z 1, z 2)$
Make sure everything is roughly centered on the origin in a $1 \times 1$ box.
get_image_pos_arr_old ( $w, z 1, z 2$, check_sols=True)
Gets image positions. | Solve the fifth-order polynomial and get the image positions. | See PSBL writeup for full equations. | All angular distances are in arcsec.

## Parameters

w
[array_like] Complex position(s) of the source. Shape $=\left[\mathrm{N} \_\right.$times, 1]
z1
[array_like] Complex position(s) of lens 1 (primary). Shape $=[$ N_times, 1]
z2
[array_like] Complex position(s) of lens 2 (secondary). Shape $=\left[\mathrm{N} \_\right.$times, 1]
check_sols
[bool, optional] If True, calculated roots are checked against the lens equation, and output will only contain those within self.root_tol. If False, all calculated roots are returned.

## Returns

z_arr
[array_like] Position of the lensed source images. Rank-1 array of polynomial roots, possibly complex. If check_sols $=$ True, only roots solving the lens equation are returned.

```
get_image_pos_arr(w,z1,z2,m1,m2,check_sols=True)
```

Gets image positions.

Solve the fifth-order polynomial and get the image positions.
See PSBL writeup for full equations.
All angular distances are in arcsec.

## Parameters

w
[array_like] Complex position(s) of the source. Shape $=\left[\mathrm{N} \_\right.$times, 1]
z1
[array_like] Complex position(s) of lens 1 (primary). Shape $=[$ N_times, 1]
z2
[array_like] Complex position(s) of lens 2 (secondary). Shape = [N_times, 1]
check_sols
[bool, optional] If True, calculated roots are checked against the lens equation, and output will only contain those within self.root_tol. If False, all calculated roots are returned.

## Returns

z_arr
[array_like] Rank-1 array of polynomial roots, possibly complex. If check_sols = True, only roots solving the lens equation are returned.
get_all_arrays(t_obs, check_sols=True, rescale=True)
Obtain the image and amplitude arrays for each t _obs.

## Parameters

## t_obs

[array_like] Array of times to model.

## Returns

images
[array_like] Array/tuple of complex positions of each images at each t_obs.
amp_arr
[array_like] Array/tuple of amplification of each images at each t_obs.
get_resolved_photometry (t_obs, filt_idx=0, amp_arr=None, print_warning=True)
Get the photometry for each of the lensed source images. Implement with no blending (since we don't support different blendings for the different images).

## Parameters

## t_obs

[array_like] Array of times to model.

## Returns

mag_model
[array_like] Magnitude of each lensed image centroid at t_obs. Shape $=[5$, len(t_obs)]
get_photometry (t_obs, filt_idx=0, amp_arr=None, print_warning=True)
Get the photometry for each of the lensed source images.

## Parameters

## t_obs

[array_like] Array of times to model.

## Returns

mag_model
[array_like] Magnitude of the centroid at t_obs.
animate ( $t$ E, time_steps, frame_time, name, size, zoom, astrometry)
Produces animation of microlensing event. This function takes the PSPL and makes an animation, the input variables are as follows

## Parameters

## tE:

number of einstein crossings times before/after the peak you want the animation to plot
e.g tE $=2=>$ graph will go from -2 tE to 2 tE

## time_steps:

number of time steps before/after peak, so total number of time steps will be 2 times this value

## frame_time:

times in ms of each frame in the animation
name: string
the animation will be saved as name.html

## size: list

[horizontal, vertical] cm's

## zoom:

\# of einstein radii plotted in vertical direction
class model.PSBL_Phot
Bases: PSBL, PSPL_Phot

## Methods

| animate(tE, time_steps, frame_time, name, ...) | Produces animation of microlensing event. |
| :--- | :--- |
| get_all_arrays(t_obs[, check_sols, rescale]) | Obtain the image and amplitude arrays for each t_obs. |
| get_amp_arr(z_arr, z1, z2) | Calculations amplification array |
| get_astrometry(t_obs[, image_arr, amp_arr, ...]) | Position of the observed (unresolved) source position <br> in Einstein radii. |
| get_astrometry_unlensed(t_obs) | Get the astrometry of the source if the lens didn't ex- <br> ist. |
| get_complex_pos(t_obs) | Get the positions of the lenses and source as complex <br> numbers. |
| get_image_pos_arr(w, z1, z2, m1, m2[, ...]) | Gets image positions. |
| get_image_pos_arr_old(w, z1, z2[, check_sols]) | Gets image positions. |
| get_photometry(t_obs[, filt_idx, amp_arr, ...]) | Get the photometry for each of the lensed source im- <br> ages. |
| get_resolved_astrometry(t_obs[, image_arr, | Position of the observed source position in Einstein <br> radii. |
| get_resolved_lens_astrometry(t_obs) | Equation of motion for just the foreground lenses, in- <br> dividually. |
| get_resolved_photometry(t_obs[, filt_idx, ...]) | Get the photometry for each of the lensed source im- <br> ages. |
| rescale_complex_pos(w, z1, z2) | Make sure everything is roughly centered on the ori- <br> gin in a 1 x 1 box. |


| get_centroid_shift |  |
| :--- | :--- |
| get_chi2_photometry |  |
| get_lens_astrometry |  |
| get_lnL_constant |  |
| get_resolved_amplification |  |
| log_likely_astrometry |  |
| log_likely_photometry |  |
| log_likely_photometry_each |  |

## get_complex_pos(t_obs)

Get the positions of the lenses and source as complex numbers.
This is needed for further calculations. Note that all units are still the same as before, this is just rewriting vectors $z=(x, y)$ as $z=x+i y$.

## Returns

w
[complex array] Source position as an array of complex numbers with real = east component, imaginary $=$ north component
z1
[complex array] Lens primary component position as an array of complex numbers with real $=$ east component, imaginary $=$ north component
z2
[complex array] Lens secondary component position as an array of complex numbers with real $=$ east component, imaginary $=$ north component
get_resolved_lens_astrometry (t_obs)
Equation of motion for just the foreground lenses, individually.

## Parameters

t_obs
[array_like] Time (in MJD).

## Notes

Note: Note, this is a photometry only model, so units are in Einstein radii.

## get_astrometry_unlensed (t_obs)

Get the astrometry of the source if the lens didn't exist. Note, this is a photometry only model, so units are in Einstein radii.

## Returns

xS_unlensed
[numpy array, dtype=float, shape $=$ len(t_obs) x 2] The unlensed positions of the source in Einstein radii.

## Notes

Note: Note, this is a photometry only model, so units are in Einstein radii.

```
get_resolved_astrometry(t_obs, image_arr=None, amp_arr=None)
```

Position of the observed source position in Einstein radii.

## Parameters

t_obs
[array_like, shape $=$ [N_times] $]$ Array of times to model.

## Returns

model_pos
[array_like. shape $=\left[\mathrm{N} \_\right.$times, $\left.\left.\mathrm{N} \_i m a g e s, 2\right]\right]$ Array of vector positions of the centroid at each t_obs.
get_astrometry(t_obs, image_arr=None, amp_arr=None, ast_filt_idx=0)
Position of the observed (unresolved) source position in Einstein radii.

## Parameters

t_obs
[array_like] Array of times to model.

## Returns

model_pos
[array_like] Array of vector positions of the centroid at each t_obs.
animate (tE, time_steps, frame_time, name, size, zoom, astrometry)
Produces animation of microlensing event. This function takes the PSPL and makes an animation, the input variables are as follows

## Parameters

## tE:

number of einstein crossings times before/after the peak you want the animation to plot
e.g $\mathrm{tE}=2$ => graph will go from -2 tE to 2 tE
time_steps:
number of time steps before/after peak, so total number of time steps will be 2 times this value
frame_time:
times in ms of each frame in the animation
name: string
the animation will be saved as name.html
size: list
[horizontal, vertical] cm's

## zoom:

\# of einstein radii plotted in vertical direction
get_all_arrays(t_obs, check_sols=True, rescale=True)
Obtain the image and amplitude arrays for each t_obs.

## Parameters

t_obs
[array_like] Array of times to model.

## Returns

images
[array_like] Array/tuple of complex positions of each images at each t_obs.
amp_arr
[array_like] Array/tuple of amplification of each images at each $t$ _obs.
get_amp_arr (z_arr, z1, z2)
Calculations amplification array
Calculates the amplification A from the Jacobian $\mathbf{J}, A=1 /|J|$

## Parameters

## z_arr

[array_like]
Complex position of images. Shape $=$ [N_times, N_solutions, 1]

- note this could be jagged.
z1
[array_like] Complex position(s) of lens 1 (primary). Shape = [N_times, 1]
z2
[array_like] Complex position(s) of lens 2 (secondary). Shape $=$ [N_times, 1]


## Returns

amp_arr
[array_like] BLEH
get_image_pos_arr $(w, z 1, z 2, m 1, m 2$, check_sols=True)
Gets image positions.

Solve the fifth-order polynomial and get the image positions.
See PSBL writeup for full equations.
All angular distances are in arcsec.

## Parameters

w
[array_like] Complex position(s) of the source. Shape = [N_times, 1]
z1
[array_like] Complex position(s) of lens 1 (primary). Shape $=[$ N_times, 1]
z2
[array_like] Complex position(s) of lens 2 (secondary). Shape $=\left[\mathrm{N} \_\right.$times, 1]
check_sols
[bool, optional] If True, calculated roots are checked against the lens equation, and output will only contain those within self.root_tol. If False, all calculated roots are returned.

## Returns

z_arr
[array_like] Rank-1 array of polynomial roots, possibly complex. If check_sols = True, only roots solving the lens equation are returned.
get_image_pos_arr_old( $w, z 1, z 2$, check_sols=True)
Gets image positions. | Solve the fifth-order polynomial and get the image positions. | See PSBL writeup for full equations. | All angular distances are in arcsec.

## Parameters

w
[array_like] Complex position(s) of the source. Shape = [N_times, 1]
z1
[array_like] Complex position(s) of lens 1 (primary). Shape $=[$ N_times, 1]
z2
[array_like] Complex position(s) of lens 2 (secondary). Shape $=\left[\mathrm{N} \_\right.$times, 1]

## check_sols

[bool, optional] If True, calculated roots are checked against the lens equation, and output will only contain those within self.root_tol. If False, all calculated roots are returned.

## Returns

z_arr
[array_like] Position of the lensed source images. Rank-1 array of polynomial roots, possibly complex. If check_sols $=$ True, only roots solving the lens equation are returned.

```
get_photometry(t_obs, filt_idx=0, amp_arr=None, print_warning=True)
```

Get the photometry for each of the lensed source images.

## Parameters

## t_obs

[array_like] Array of times to model.

## Returns

mag_model
[array_like] Magnitude of the centroid at t_obs.
get_resolved_photometry (t_obs, filt_idx=0, amp_arr=None, print_warning=True)
Get the photometry for each of the lensed source images. Implement with no blending (since we don't support different blendings for the different images).

## Parameters

t_obs
[array_like] Array of times to model.

## Returns

mag_model
[array_like] Magnitude of each lensed image centroid at t_obs. Shape $=[5$, len(t_obs)]
rescale_complex_pos $(w, z 1, z 2)$
Make sure everything is roughly centered on the origin in a $1 \times 1$ box.
class model.PSBL_PhotAstrom
Bases: PSBL, PSPL_PhotAstrom
Contains methods for model a PSPL photometry + astrometry. This is a Data-type class in our hierarchy. It is abstract and should not be instantiated.

## Methods

| animate(tE, time_steps, frame_time, name, ...) | Produces animation of microlensing event. |
| :--- | :--- |
| get_all_arrays(t_obs[, check_sols, rescale]) | Obtain the image and amplitude arrays for each t_obs. |
| get_amp_arr(z_arr, z1, z2) | Calculations amplification array |
| get_astrometry(t_obs[, image_arr, amp_arr, ...]) | Position of the observed (unresolved) source position <br> in arcsec. |
| get_astrometry_unlensed(t_obs) | Get the astrometry of the source if the lens didn't ex- <br> ist. |
| get_centroid(t_obs[, ast_filt_idx, ...]) | PSPL: Get the centroid shift (in mas) for a list of ob- <br> servation times (in MJD). |
| get_complex_pos(t_obs) | Get the positions of the lenses and source as complex <br> numbers. |
| get_image_pos_arr(w, z1, z2, m1, m2[, ...]) | Gets image positions. |
| get_image_pos_arr_old(w, z1, z2[, check_sols]) | Gets image positions. |
| get_lens_astrometry(t_obs) | Equation of motion for just the foreground lens sys- <br> tem. |
| get_photometry(t_obs[, filt_idx, amp_arr, ...]) | Get the photometry for each of the lensed source im- <br> ages. |
| get_resolved_astrometry(t_obs[, image_arr, Position of the observed source position in arcsec. <br> $\ldots .])$. Equation of motion for just the foreground lenses, in- <br> dividually. <br> get_resolved_lens_astrometry(t_obs) Get the photometry for each of the lensed source im- <br> ages. <br> get_resolved_photometry(t_obs[, filt_idx, ...])  <br> rescale_complex_pos(w, z1, z2) Make sure everything is roughly centered on the ori- <br> gin in a 1 x 1 box. |  |


| get_chi2_astrometry |  |
| :--- | :--- |
| get_chi2_photometry |  |
| get_lnL_constant |  |
| log_likely_astrometry |  |
| log_likely_astrometry_each |  |
| log_likely_photometry |  |
| log_likely_photometry_each |  |

## get_complex_pos(t_obs)

Get the positions of the lenses and source as complex numbers. This is needed for further calculations. Note that all units are still the same as before, this is just rewriting vectors $z=(x, y)$ as $z=x+i y$.

## Returns

w
[complex array] Source position as an array of complex numbers with real = east component, imaginary $=$ north component
z1
[complex array] Lens primary component position as an array of complex numbers with real $=$ east component, imaginary $=$ north component
z2
[complex array] Lens secondary component position as an array of complex numbers with real $=$ east component, imaginary $=$ north component
get_resolved_astrometry (t_obs, image_arr=None, amp_arr=None)
Position of the observed source position in arcsec.

## Parameters

## t_obs

[array_like, shape $=[$ N_times] $]$ Array of times to model.

## Returns

model_pos
[array_like. shape $=\left[\mathrm{N} \_\right.$times, $\left.\left.\mathrm{N} \_i m a g e s, 2\right]\right]$ Array of vector positions of the centroid at each t_obs.
get_astrometry(t_obs, image_arr=None, amp_arr=None, ast_filt_idx=0)
Position of the observed (unresolved) source position in arcsec.

## Parameters

## t_obs

[array_like] Array of times to model.

## Returns

model_pos
[array_like] Array of vector positions of the centroid at each t_obs.
get_astrometry_unlensed (t_obs)
Get the astrometry of the source if the lens didn't exist.

## Returns

xS_unlensed
[numpy array, dtype=float, shape $=$ len(t_obs) x 2] The unlensed positions of the source in arcseconds.
get_lens_astrometry (t_obs)
Equation of motion for just the foreground lens system.

## Parameters

t_obs
[array_like] Time (in MJD).
get_resolved_lens_astrometry (t_obs)
Equation of motion for just the foreground lenses, individually.

## Parameters

## t_obs

[array_like] Time (in MJD).
get_centroid(t_obs, ast_fil_idx=0, image_arr=None, amp_arr=None)
PSPL: Get the centroid shift (in mas) for a list of observation times (in MJD).

## Parameters

## t_obs

[array or float]

## Returns

Centroid offset on the plane of the sky in milli-arcseoncds.

## Other Parameters

ast_filt_idx
[int] Index into the photometry parameter lists for the photometry that corresponds to this astrometry data set.
image_arr
[list] List returned from PSPL get_all_arrays() used to improve efficiency.
amp_arr
[list] List returned from PSPL get_all_arrays() used to improve efficiency.
animate (tE, time_steps, frame_time, name, size, zoom, astrometry)
Produces animation of microlensing event. This function takes the PSPL and makes an animation, the input variables are as follows

## Parameters

## tE:

number of einstein crossings times before/after the peak you want the animation to plot
e.g $\mathrm{tE}=2$ => graph will go from -2 tE to 2 tE

## time_steps:

number of time steps before/after peak, so total number of time steps will be 2 times this value
frame_time:
times in ms of each frame in the animation

## name: string

the animation will be saved as name.html

## size: list

[horizontal, vertical] cm's

## zoom:

\# of einstein radii plotted in vertical direction

```
get_all_arrays(t_obs,check_sols=True, rescale=True)
```

Obtain the image and amplitude arrays for each t_obs.

## Parameters

## t_obs

[array_like] Array of times to model.

## Returns

images
[array_like] Array/tuple of complex positions of each images at each $t$ _obs.
amp_arr
[array_like] Array/tuple of amplification of each images at each t_obs.
get_amp_arr (z_arr, z1, z2)
Calculations amplification array
Calculates the amplification A from the Jacobian $\mathrm{J}, A=1 /|J|$

## Parameters

z_arr
[array_like]
Complex position of images. Shape $=$ [N_times, $N_{-}$solutions, 1]

- note this could be jagged.
z1
[array_like] Complex position(s) of lens 1 (primary). Shape = [N_times, 1]
z2
[array_like] Complex position(s) of lens 2 (secondary). Shape $=$ [N_times, 1]


## Returns

amp_arr
[array_like] BLEH
get_image_pos_arr $(w, z 1, z 2, m 1, m 2$, check_sols=True)
Gets image positions.

Solve the fifth-order polynomial and get the image positions.
See PSBL writeup for full equations.
All angular distances are in arcsec.

## Parameters

w
[array_like] Complex position(s) of the source. Shape = [N_times, 1]
z1
[array_like] Complex position(s) of lens 1 (primary). Shape $=[$ N_times, 1]
z2
[array_like] Complex position(s) of lens 2 (secondary). Shape = [N_times, 1]
check_sols
[bool, optional] If True, calculated roots are checked against the lens equation, and output will only contain those within self.root_tol. If False, all calculated roots are returned.

## Returns

z_arr
[array_like] Rank-1 array of polynomial roots, possibly complex. If check_sols = True, only roots solving the lens equation are returned.
get_image_pos_arr_old( $w, z 1, z 2$, check_sols=True)
Gets image positions. | Solve the fifth-order polynomial and get the image positions. | See PSBL writeup for full equations. | All angular distances are in arcsec.

## Parameters

w
[array_like] Complex position(s) of the source. Shape $=\left[\mathrm{N} \_\right.$times, 1]
z1
[array_like] Complex position(s) of lens 1 (primary). Shape $=[$ N_times, 1]
z2
[array_like] Complex position(s) of lens 2 (secondary). Shape $=\left[\mathrm{N} \_\right.$times, 1]

## check_sols

[bool, optional] If True, calculated roots are checked against the lens equation, and output will only contain those within self.root_tol. If False, all calculated roots are returned.

## Returns

z_arr
[array_like] Position of the lensed source images. Rank-1 array of polynomial roots, possibly complex. If check_sols = True, only roots solving the lens equation are returned.

```
get_photometry(t_obs, filtidx=0, amp_arr=None, print_warning=True)
```

Get the photometry for each of the lensed source images.

## Parameters

t_obs
[array_like] Array of times to model.

## Returns

mag_model
[array_like] Magnitude of the centroid at t_obs.
get_resolved_photometry (t_obs, filt_idx=0, amp_arr=None, print_warning=True)
Get the photometry for each of the lensed source images. Implement with no blending (since we don't support different blendings for the different images).

## Parameters

## t_obs

[array_like] Array of times to model.

## Returns

mag_model
[array_like] Magnitude of each lensed image centroid at t_obs. Shape $=[5$, len(t_obs)]
rescale_complex_pos $(w, z l, z 2)$
Make sure everything is roughly centered on the origin in a $1 \times 1$ box.

### 6.2 Parallax Class Family - PSBL

```
class model.PSPL_Parallax
```

Bases: ParallaxClassABC

## Methods

| calc_piE_ecliptic() | Parallax: Get piE_ecliptic |
| :--- | :--- |
| get_amplification(t) | Parallax: Get the photometric amplification term at a <br> set of times, t. |
| get_astrometry(t_obs[, ast_filt_idx]) | Parallax: Get astrometry <br> get_astrometry_unlensed(t_obs) <br> ist. |
| get_centroid_shift(t[, ast_filt_idx]) | Parallax: Get the centroid shift (in mas) for a list of <br> observation times (in MJD). |
| get_geoproj_ast_params(t0par) |  |
| get_geoproj_params(t0par) | Parallax: Get lens astrometry <br> set of times, the for both the plus and minus images. |
| get_lens_astrometry(t_obs) | Parallax: Get the x, y astrometry for each of the two <br> source images, which we label plus and minus. |
| get_resolved_amplification(t) |  |

## start

## get_amplification $(t)$

Parallax: Get the photometric amplification term at a set of times, t .

## Parameters

t:
Array of times in MJD.DDD

## get_lens_astrometry (t_obs)

Parallax: Get lens astrometry

```
get_astrometry(t_obs,ast_filt_idx=0)
```

Parallax: Get astrometry

```
get_centroid_shift(t,ast_filt_idx=0)
```

Parallax: Get the centroid shift (in mas) for a list of observation times (in MJD).
get_astrometry_unlensed(t_obs)
Get the astrometry of the source if the lens didn't exist.

## Returns

## xS_unlensed

[numpy array, dtype=float, shape $=$ len(t_obs) $\times 2$ 2] The unlensed positions of the source in arcseconds.

```
get_resolved_amplification(t)
```

Parallax: Get the photometric amplification term at a set of times, $t$ for both the plus and minus images.

## Parameters

t:
Array of times in MJD.DDD

## get_resolved_astrometry(t_obs)

Parallax: Get the x , y astrometry for each of the two source images, which we label plus and minus.

## Returns

[xS_plus, xS_minus]
[list of numpy arrays]

- xS_plus is the vector position of the plus image.
- xS_minus is the vector position of the plus image.
calc_piE_ecliptic()
Parallax: Get piE_ecliptic

```
class model.PSPL_noParallax
```

Bases: ParallaxClassABC

## Methods

| get_amplification(t) | noParallax: Get the photometric amplification term <br> at a set of times,. |
| :--- | :--- |
| get_astrometry(t_obs[, ast_filt_idx]) | noParallax: Position of the observed source position <br> in arcsec. |
| get_astrometry_unlensed(t_obs) | noParallax: Get the astrometry of the source if the <br> lens didn't exist. |
| get_centroid_shift(t) | noParallax: Get the centroid shift (in mas) for a list <br> of observation times (in MJD). |
| get_lens_astrometry(t_obs) | Equation of motion for just the foreground lens. |
| get_resolved_amplification(t) | Get the photometric amplification term at a set of <br> times, $t$ for both the plus and minus images. |
| get_resolved_astrometry(t_obs) | Get the x, y astrometry for each of the two source <br> images, which we label plus and minus. |

## calc_piE_ecliptic

## get_amplification $(t)$

noParallax: Get the photometric amplification term at a set of times, t .

## Parameters

t:
Array of times in MJD.DDD

```
get_lens_astrometry(t_obs)
```

Equation of motion for just the foreground lens.

## Parameters

t_obs
[array_like] Time (in MJD).
get_astrometry (t_obs, ast_filt_idx=0)
noParallax: Position of the observed source position in arcsec.
get_centroid_shift $(t)$
noParallax: Get the centroid shift (in mas) for a list of observation times (in MJD).
get_astrometry_unlensed (t_obs)
noParallax: Get the astrometry of the source if the lens didn't exist.

## Returns

xS_unlensed
[numpy array, dtype=float, shape $=$ len(t_obs) $\times 2$ 2] The unlensed positions of the source in arcseconds.
get_resolved_amplification $(t)$
Get the photometric amplification term at a set of times, t for both the plus and minus images.

## Parameters

t:
Array of times in MJD.DDD
get_resolved_astrometry (t_obs)
Get the $\mathrm{x}, \mathrm{y}$ astrometry for each of the two source images, which we label plus and minus.

## Returns

[xS_plus, xS_minus]
[list of numpy arrays]

- xS_plus is the vector position of the plus image in arcsec
- xS_minus is the vector position of the plus image in arcsec


### 6.3 Parameterization Class Family - PSBL

### 6.3.1 PSBL Models

class model.PSBL_PhotAstromParam1 $\left(m L p, m L s, t 0, x S O_{-} E, x S O_{-} N, b e t a, m u L_{-} E, m u L \_N, m u S \_E, m u S_{-} N, d L\right.$, $d S$, sep, alpha, b_sff, mag_src, raL=None, $\left.d e c L=N o n e, ~ r o o t \_t o l=1 e-08\right) ~$
Bases: PSPL_Param
Point source binary lens. It has 3 more parameters than PSPL (additional mass term, separation, and angle of approach). Note that this is a STATIC binary lens, i.e. there is no orbital motion.

```
Attributes
    mLp,mLs
        [float] Masses of the lenses (Msun)
    t0
    [float] Time of photometric peak, as seen from Earth (MJD.DDD)
xS0_E
    [float] R.A. of source position on sky at t = t0 (arcsec) in an arbitrary ref. frame.
xS0_N
    [float] Dec. of source position on sky at t=t0 (arcsec) in an arbitrary ref. frame.
beta
        [float] Angular distance between the source and the GEOMETRIC center of the lenses on
        the plane of the sky (mas). Can be
```

- positive (u0_amp > 0 when $\left.u 0 \_h a t[0]>0\right)$ or
- negative ( $u 0$ _amp $<0$ when $u 0 \_h a t[0]<0$ ).
muL_E
[float] Lens system proper motion in the RA direction (mas/yr)
muL_N
[float] Lens system proper motion in the Dec. direction (mas/yr)
muS_E
[float] Source proper motion in the RA direction (mas/yr)
muS_N
[float] Source proper motion in the Dec. direction (mas/yr)
dL
[float] Distance from the observer to the lens system (pc)
dS
[float] Distance from the observer to the source (pc)
sep
[float] Angular separation of the two lenses (mas)


## alpha

[float] Angle made between the binary axis and North; measured in degrees East of North.
b_sff
[numpy array or list] The ratio of the source flux to the total (source + neighbors + lenses).
One for each filter.

## mag_src

[numpy array or list] Source magnitude, unlensed. One in each filter.
root_tol
[float] Tolerance in comparing the polynomial roots to the physical solutions. Default $=1 \mathrm{e}-8$
class model.PSBL_PhotAstromParam2 ( $t 0, u 0 \_a m p, t E$, thetaE, $p i S, p i E \_E, p i E \_N, x S 0 \_E, x S 0 \_N, m u S \_E$, $m u S \_N, q, s e p, a l p h a, b \_s f f, m a g \_s r c, r a L=N o n e, \operatorname{dec} L=N o n e$, root_tol=1e-08)
Bases: PSPL_Param
Point source binary lens. It has 3 more parameters than PSPL (additional mass term, separation, and angle of approach). Note that this is a STATIC binary lens, i.e. there is no orbital motion.

## Attributes

t0
[float] Time of photometric peak, as seen from Earth (MJD.DDD)
u0_amp
[float] Angular distance between the source and the GEOMETRIC center of the lenses on the plane of the sky at closest approach in units of thetaE. Can be

- positive ( $u 0$ _amp $>0$ when $u 0 \_$hat $[0]>0$ ) or
- negative (u0_amp < 0 when u0_hat[0] < 0).
tE
[float] Einstein crossing time (days).


## thetaE

[float] The size of the Einstein radius in (mas).

## piS

[float] Amplitude of the parallax ( $1 \mathrm{AU} / \mathrm{dS}$ ) of the source. (mas)

## piE_E

[float] The microlensing parallax in the East direction in units of thetaE
piE_N
[float] The microlensing parallax in the North direction in units of thetaE
xSO_E
[float] R.A. of source position on sky at $\mathrm{t}=\mathrm{t} 0(\mathrm{arcsec})$ in an arbitrary ref. frame.
xS0_N
[float] Dec. of source position on sky at $\mathrm{t}=\mathrm{t} 0(\mathrm{arcsec})$ in an arbitrary ref. frame.
muS_E
[float] RA Source proper motion (mas/yr)
muS_N
[float] Dec Source proper motion (mas/yr)
q
[float] Mass ratio (M2 / M1)
sep
[float] Angular separation of the two lenses (mas)
alpha
[float] Angle made between the binary axis and North; measured in degrees East of North.
b_sff
[numpy array or list] The ratio of the source flux to the total (source + neighbors + lenses). One for each filter.

## mag_src

[numpy array or list] Source magnitude, unlensed. One in each filter.
root_tol
[float] Tolerance in comparing the polynomial roots to the physical solutions. Default $=1 \mathrm{e}-8$
class model.PSBL_PhotAstromParam3 $\left(t 0, u 0 \_a m p, t E, \log 10 \_t h e t a E, p i S, p i E \_E, p i E \_N, x S O \_E, x S O \_N\right.$, muS_E, muS_N, q, sep, alpha, b_sff, mag_base, raL=None, decL=None, root_tol=le-08)
Bases: PSPL_Param
Point source binary lens. It has 3 more parameters than PSPL (additional mass term, separation, and angle of approach). Note that this is a STATIC binary lens, i.e. there is no orbital motion.

## Attributes

t0
[float] Time of photometric peak, as seen from Earth (MJD.DDD) FIXME: THIS IS NOT RIGHT

## u0_amp

[float] Angular distance between the source and the GEOMETRIC center of the lenses on the plane of the sky at closest approach in units of thetaE. Can

- positive ( $\mathrm{u} 0 \_$_amp > 0 when u0_hat $[0]>0$ ) or
- negative (u0_amp < 0 when u0_hat[0] < 0 ).
tE
[float] Einstein crossing time (days).


## $\log 10 \_$thetaE

[float] The size of the Einstein radius in (mas).

## piS

[float] Amplitude of the parallax (1AU/dS) of the source. (mas)
piE_E
[float] The microlensing parallax in the East direction in units of thetaE
piE_N
[float] The microlensing parallax in the North direction in units of thetaE
xS0_E
[float] R.A. of source position on sky at $\mathrm{t}=\mathrm{t} 0(\operatorname{arcsec})$ in an arbitrary ref. frame.
xS0_N
[float] Dec. of source position on sky at $t=t 0(\operatorname{arcsec})$ in an arbitrary ref. frame.
muS_E
[float] RA Source proper motion (mas/yr)
muS_N
[float] Dec Source proper motion (mas/yr)
q
[float] Mass ratio (M2 / M1)
sep
[float] Angular separation of the two lenses (mas)

## alpha

[float] Angle made between the binary axis and North; measured in degrees East of North.
b_sff
[numpy array or list] The ratio of the source flux to the total (source + neighbors + lenses). One for each filter.

$$
b_{s} f f=f_{S} /\left(f_{S}+f_{L}+f_{N}\right)
$$

This must be passed in as a list or array, with one entry for each photometric filter.

## mag_base

[numpy array or list] Photometric magnitude of the base. This must be passed in as a list or array, with one entry for each photometric filter.

## root_tol

[float] Tolerance in comparing the polynomial roots to the physical solutions. Default $=1 \mathrm{e}-8$
class model.PSBL_PhotAstromParam4(t0_com, u0_amp_com, $t E$, thetaE, piS, piE_E, piE_N, xSO_E, xSO_N, $m u S \_E, m u S \_N, q$, sep, alpha, b_sff, mag_src, $r a L=N o n e, d e c L=N o n e$, root_tol=1e-08)

Bases: PSPL_Param
Point source binary lens. It has 3 more parameters than PSPL (additional mass term, separation, and angle of approach). Note that this is a STATIC binary lens, i.e. there is no orbital motion.

## Attributes

## t0_com

[float] Time of photometric peak, as seen from Earth (MJD.DDD) FIXME: THIS IS NOT RIGHT

```
    u0_amp_com
        [float] Angular distance between the source and the binary lens COM on the plane of the sky
        at closest approach in units of thetaE. Can be
    - positive (u0_amp > 0 when u0_hat[0] > 0 ) or
    - negative (u0_amp < 0 when u0_hat[0] < 0).
tE
    [float] Einstein crossing time (days).
thetaE
    [float] The size of the Einstein radius in (mas).
piS
    [float] Amplitude of the parallax (1AU/dS) of the source. (mas)
piE_E
    [float] The microlensing parallax in the East direction in units of thetaE
piE_N
    [float] The microlensing parallax in the North direction in units of thetaE
xS0_E
    [float] R.A. of source position on sky at \(\mathrm{t}=\mathrm{t} 0(\operatorname{arcsec})\) in an arbitrary ref. frame.
xS0_N
    [float] Dec. of source position on sky at \(\mathrm{t}=\mathrm{t} 0(\operatorname{arcsec})\) in an arbitrary ref. frame.
    muS_E
    [float] RA Source proper motion (mas/yr)
    muS_N
    [float] Dec Source proper motion (mas/yr)
    beta
        [float] Angular distance between the source and the GEOMETRIC center of the lenses on
    the plane of the sky (mas).
q
    [float] Mass ratio (M2 / M1)
sep
    [float] Angular separation of the two lenses (mas)
alpha
    [float] Angle made between the binary axis and North; measured in degrees East of North.
    b_sff
    [numpy array or list] The ratio of the source flux to the total (source + neighbors + lenses).
    One for each filter.
    mag_src
    [numpy array or list] Source magnitude, unlensed. One in each filter.
root_tol
[float] Tolerance in comparing the polynomial roots to the physical solutions. Default \(=1 \mathrm{e}-8\)
class model.PSBL_PhotAstromParam5 (t0_prim, u0_amp_prim, \(t E\), thetaE, piS, piE_E, piEN_piEE, xSO_E,
                                    \(x S 0 \_N, m u S \_E, m u S \_N, q\), sep, alpha, \(b \_s f f\), mag_base, \(r a L=N o n e\),
    \(\operatorname{dec} L=N o n e\), root_tol=le-08)
```

Bases: PSPL_Param

Point source binary lens. It has 3 more parameters than PSPL (additional mass term, separation, and angle of approach). Note that this is a STATIC binary lens, i.e. there is no orbital motion.

## Attributes

## t0_prim

[float] Time of photometric peak, as seen from Earth (MJD.DDD) FIXME: THIS IS NOT RIGHT

## u0_amp_prim

[float] Angular distance between the source and the PRIMARY lens on the plane of the sky at closest approach in units of thetaE. Can be

- positive (u0_amp > 0 when u0_hat[0] > 0 ) or
- negative (u0_amp < 0 when u0_hat[0] < 0).


## tE

[float] Einstein crossing time (days).

## thetaE

[float] The size of the Einstein radius in (mas).

## piS

[float] Amplitude of the parallax (1AU/dS) of the source. (mas)

## piE_E

[float] The microlensing parallax in the East direction in units of thetaE
piE_N
[float] The microlensing parallax in the North direction in units of thetaE

## xS0_E

[float] R.A. of source position on sky at $\mathrm{t}=\mathrm{t} 0(\operatorname{arcsec})$ in an arbitrary ref. frame.

## xS0_N

[float] Dec. of source position on sky at $\mathrm{t}=\mathrm{t} 0(\operatorname{arcsec})$ in an arbitrary ref. frame.
muS_E
[float] RA Source proper motion (mas/yr)
muS_N
[float] Dec Source proper motion (mas/yr)

## beta

[float] Angular distance between the source and the GEOMETRIC center of the lenses on the plane of the sky (mas).
q
[float] Mass ratio (M2 / M1)

## sep

[float] Angular separation of the two lenses (mas)
alpha
[float] Angle made between the binary axis and North; measured in degrees East of North.
b_sff
[numpy array or list] The ratio of the source flux to the total (source + neighbors + lenses).
One for each filter.

## mag_src

[numpy array or list] Source magnitude, unlensed. One in each filter.

## root_tol

[float] Tolerance in comparing the polynomial roots to the physical solutions. Default $=1 \mathrm{e}-8$
class model.PSBL_PhotAstromParam6(t0_prim, u0_amp_prim, $t E$, thetaE, piS, piE_E, piE_N, xSO_E, xSO_N, $m u S \_E, m u S \_N, q$, sep, alpha, $b \_s f f, m a g \_s r c, r a L=N o n e, d e c L=N o n e$, root_tol=1e-08)
Bases: PSPL_Param
Point source binary lens. It has 3 more parameters than PSPL (additional mass term, separation, and angle of approach). Note that this is a STATIC binary lens, i.e. there is no orbital motion.

```
Attributes
    t0_prim
    [float] Time of photometric peak, as seen from Earth (MJD.DDD) FIXME: THIS IS NOT
    RIGHT
    u0_amp_prim
    [float] Angular distance between the source and the PRIMARY lens on the plane of the sky
    at closest approach in units of thetaE. Can be
    - positive (u0_amp > 0 when u0_hat[0] > 0) or
    - negative (u0_amp < 0 when u0_hat[0] < 0).
tE
    [float] Einstein crossing time (days).
thetaE
    [float] The size of the Einstein radius in (mas).
piS
    [float] Amplitude of the parallax (1AU/dS) of the source. (mas)
piE_E
    [float] The microlensing parallax in the East direction in units of thetaE
piE_N
    [float] The microlensing parallax in the North direction in units of thetaE
xSO_E
    [float] R.A. of source position on sky at t = t0 (arcsec) in an arbitrary ref. frame.
xS0_N
    [float] Dec. of source position on sky at t=t0(arcsec) in an arbitrary ref. frame.
    muS_E
    [float] RA Source proper motion (mas/yr)
    muS_N
    [float] Dec Source proper motion (mas/yr)
    beta
        [float] Angular distance between the source and the GEOMETRIC center of the lenses on
        the plane of the sky (mas).
    q
    [float] Mass ratio (M2 / M1)
sep
    [float] Angular separation of the two lenses (mas)
alpha
    [float] Angle made between the binary axis and North; measured in degrees East of North.
```

```
        b_sff
            [numpy array or list] The ratio of the source flux to the total (source + neighbors + lenses).
            One for each filter.
```


## mag_src

```
[numpy array or list] Source magnitude, unlensed. One in each filter.
```


## root_tol

```
[float] Tolerance in comparing the polynomial roots to the physical solutions. Default \(=1 \mathrm{e}-8\)
class model.PSBL_PhotParam1 \(\left(t 0, u 0 \_a m p, t E, p i E \_E, p i E \_N, q, s e p, p h i, b_{-} s f f, m a g \_s r c, r a L=N o n e\right.\), decL=None, root_tol=1e-08)
Bases: PSPL_Param
```

Point source binary lens, photometry only.

## It has $\mathbf{3}$ more parameters than PSPL_PhotParam1:

- mass ratio
- separation - in units of thetaE
- angle of approach

Note that this is a STATIC binary lens, i.e. there is no orbital motion.

## Attributes

## t0: float

Time of photometric peak, as seen from Earth [MJD]

## u0_amp: float

Angular distance between the lens and source on the plane of the sky at closest approach in units of thetaE. It can be

- positive (u0_amp >0 when u0_hat[0] >0) or
- negative (u0_amp < 0 when u0_hat[0] < 0).


## tE: float

Einstein crossing time. [MJD]

## piE_E: float

The microlensing parallax in the East direction in units of thetaE

## piE_N: float

The microlensing parallax in the North direction in units of thetaE

## q: float

Mass ratio (low-mass / high-mass)
sep: float
Angular separation of the two lenses in units of thetaE where thetaE is defined with the total binary mass.

## phi: float

Angle made between the binary axis and the relative proper motion vector, measured in degrees.
b_sff: array or list
The ratio of the source flux to the total (source + neighbors + lens) $b_{s} f f=f_{S} /\left(f_{S}+f_{L}+\right.$ $\left.f_{N}\right)$. This must be passed in as a list or array, with one entry for each photometric filter.

## mag_src: array or list

Photometric magnitude of the source. This must be passed in as a list or array, with one entry for each photometric filter.

## Other Parameters

## raL: float

Right ascension of the lens in decimal degrees. Required if calculating with parallax decL: float

Declination of the lens in decimal degrees. Required if calculating with parallax
root_tol: float

Tolerance in comparing the polynomial roots to the physical solutions.
Default $=0.0$

### 6.3.2 GP Models

class model.PSBL_GP_PhotParam1 ( $t 0, u 0 \_a m p, t E, p i E \_E, p i E_{-} N, q, s e p, p h i, b \_s f f, m a g \_s r c, g p \_l o g \_s i g m a$, gp_log_rho, gp_log_SO, gp_log_omega0, raL=None, decL=None, root_tol=1e-08)
Bases: PSBL_PhotParam1
class model.PSBL_GP_PhotAstromParam1 $m L p, m L s, t 0, x S O \_E, x S O \_N$, beta, $m u L_{-} E, m u L \_N, m u S \_E, m u S \_N$, $d L, d S$, sep, alpha, b_sff, mag_src, gp_log_sigma, gp_log_rho, gp_log_S0, gp_log_omega0, raL=None, decL=None, root_tol=1e-08)

Bases: PSBL_PhotAstromParam1
class model.PSBL_GP_PhotAstromParam2 $\left(t 0, u 0_{-} a m p, t E\right.$, thetaE, $p i S, p i E_{-} E, p i E \_N, x S 0 \_E, x S 0 \_N, m u S \_E$, $m u S \_N, q$, sep, alpha, b_sff, mag_src, gp_log_sigma, gp_log_rho, gp_log_SO, gp_log_omega0, raL=None, decL=None, root_tol=1e-08)
Bases: PSBL_PhotAstromParam2

## BINARY SOURCE POINT LENS (BSPL) CLASSES

### 7.1 Parameterization Class Family - BSPL

### 7.1.1 BSPL Models

class model.BSPL_PhotParam1 ( $t 0, u 0 \_a m p, t E, p i E \_E, p i E_{-} N$, sep, phi, mag_src_pri, mag_src_sec, $b_{-} s f f$, $r a L=N o n e, \operatorname{dec} L=$ None $)$

Bases: PSPL_Param
BSPL model for photometry only
A Binary point Source Point Lens model for microlensing.
Note the attributes, RA (raL) and Dec (decL) are required if you are calculating a model with parallax.

## Attributes

## t0: float

Time of photometric peak, as seen from Earth (MJD.DDD)

## u0_amp: float

Angular distance between the lens and source on the plane of the sky at closest approach in units of thetaE. It can be

- positive (u0_amp > 0 when u0_hat[0] > 0 ) or
- negative (u0_amp < 0 when u0_hat[0] < 0).

Note, since this is a binary source, we are expressing the nominal source position as that of the primary star in the source binary system.

## tE: float

Einstein crossing time. [MJD]
piE_E: float
The microlensing parallax in the East direction in units of thetaE
piE_N: float
The microlensing parallax in the North direction in units of thetaE
sep: float
Angular separation of the source scondary from the source primary (in units of thetaE).
phi: float
Angle made between the binary axis and the relative proper motion vector, measured in degrees.

## mag_src_pri: array or list

Photometric magnitude of the first (primary) source. This must be passed in as a list or array, with one entry for each photometric filter.

## mag_src_sec: array or list

Photometric magnitude of the second (secondary) source. This must be passed in as a list or array, with one entry for each photometric filter.

## b_sff: array or list

The ratio of the source flux to the total (source + neighbors + lens) $b_{s} f f=\left(f_{S} 1+\right.$ $\left.f_{S} 2\right) /\left(f_{S} 1+f_{s} 2+f_{L}+f_{N}\right)$. This must be passed in as a list or array, with one entry for each photometric filter.

## raL: float, optional

Right ascension of the lens in decimal degrees.

## decL: float, optional

Declination of the lens in decimal degrees.
class model.BSPL_PhotAstromParam1 $\left(m L, t 0\right.$, beta, $d L, d L_{-} d S, x S O_{-} E, x S O_{-} N, m u L_{-} E, m u L_{-} N, m u S \_E, m u S \_N$, sep, alpha, mag_src_pri, mag_src_sec, b_sff, raL=None, decL=None)

## Bases: PSPL_Param

BSPL model for astrometry and photometry - physical parameterization.
A Binary point Source Point Lens model for microlensing. This model uses a parameterization that depends on only physical quantities such as the lens mass and positions and proper motions of both the lens and source.
Note the attributes, RA (raL) and Dec ( decL ) are required if you are calculating a model with parallax.

## Attributes

mL: float
Mass of the lens (Msun)

## t0: float

Time of photometric peak, as seen from Earth (MJD.DDD)

## beta: float

Angular distance between the lens and primary source on the plane of the sky (mas). Can be

- positive ( $u 0$ _amp $>0$ when $u 0 \_$hat $[0]<0$ ) or
- negative (u0_amp < 0 when u0_hat[0] > 0 ).

Note, since this is a binary source, we are expressing the nominal source position as that of the primary star in the source binary system.

## dL: float

Distance from the observer to the lens (pc)

## dL_dS: float

Ratio of Distance from the obersver to the lens to Distance from the observer to the source

## xS0_E: float

RA Source position on sky at $t=t 0(\operatorname{arcsec})$ in an arbitrary ref. frame. This should be the position of the source primary.

## xS0_N: float

Dec source position on sky at $\mathrm{t}=\mathrm{t} 0(\operatorname{arcsec})$ in an arbitrary ref. frame. This should be the position of the source primary.

## muL_E: float

RA Lens proper motion (mas/yr)

## muL_N: float

Dec Lens proper motion (mas/yr)

## muS_E: float

RA Source proper motion (mas/yr) Identical proper motions are assumed for the source primary and secondary.

## muS_N: float

Dec Source proper motion (mas/yr) Identical proper motions are assumed for the source primary and secondary.
sep: float
Angular separation of the source scondary from the source primary (mas).

## alpha: float

Angle made between the binary source axis and North; measured in degrees East of North.

## mag_src_pri: array or list

Photometric magnitude of the first (primary) source. This must be passed in as a list or array, with one entry for each photometric filter.

## mag_src_sec: array or list

Photometric magnitude of the second (secondary) source. This must be passed in as a list or array, with one entry for each photometric filter.

## b_sff: array or list

The ratio of the source flux to the total (source + neighbors + lens) $b_{s} f f=\left(f_{S} 1+\right.$ $\left.f_{S} 2\right) /\left(f_{S} 1+f_{s} 2+f_{L}+f_{N}\right)$. This must be passed in as a list or array, with one entry for each photometric filter.

## raL: float, optional

Right ascension of the lens in decimal degrees.

## decL: float, optional

Declination of the lens in decimal degrees.

```
class model.BSPL_PhotAstromParam2(t0,u0_amp, tE, thetaE, piS,piE_E,piE_N, xSO_E, xSO_N,muS_E,
    muS_N, sep, alpha, fratio_bin, mag_base, b_sff, raL=None,
    decL=None)
```

Bases: PSPL_Param
BSPL model for astrometry and photometry - physical parameterization.
A Binary point Source Point Lens model for microlensing. This model uses a parameterization that depends on only physical quantities such as the lens mass and positions and proper motions of both the lens and source.

Note the attributes, RA (raL) and Dec (decL) are required if you are calculating a model with parallax.

## Attributes

## t0: float

Time of photometric peak, as seen from Earth (MJD.DDD)
u0_amp
[float] Angular distance between the source and the GEOMETRIC center of the lenses on the plane of the sky at closest approach in units of thetaE. Can

- positive ( $u 0$ _amp $>0$ when $u 0 \_h a t[0]>0$ ) or
- negative (u0_amp < 0 when u0_hat[0] < 0).

Note, since this is a binary source, we are expressing the nominal source position as that of the primary star in the source binary system.
tE
[float] Einstein crossing time (days).

## thetaE

[float] The size of the Einstein radius in (mas).
piS
[float] Amplitude of the parallax ( $1 \mathrm{AU} / \mathrm{dS}$ ) of the source. (mas)
piE_E
[float] The microlensing parallax in the East direction in units of thetaE
piE_N
[float] The microlensing parallax in the North direction in units of thetaE
xS0_E
[float] R.A. of source position on sky at $t=t 0(\operatorname{arcsec})$ in an arbitrary ref. frame. This should be the position of the source primary.

## xSO_N

[float] Dec. of source position on sky at $\mathrm{t}=\mathrm{t} 0(\mathrm{arcsec})$ in an arbitrary ref. frame.
muS_E
[float] RA Source proper motion (mas/yr) Identical proper motions are assumed for the source primary and secondary.
muS_N
[float] Dec Source proper motion (mas/yr) Identical proper motions are assumed for the source primary and secondary.

## sep: float

Angular separation of the source scondary from the source primary (mas).

## alpha: float

Angle made between the binary source axis and North; measured in degrees East of North.

## fratio_bin: float

Flux ratio of secondary flux / primary flux.

## mag_base

[array or list] Photometric magnitude of the base. This must be passed in as a list or array, with one entry for each photometric filter. Note that

$$
\text { flux }_{\text {base }}=f_{\text {src } 1}+f_{\text {src } 2}+f_{\text {blend }}
$$

## such that

$$
b_{s} f f=\left(f_{s r c 1}+f_{s r c 2}\right) /\left(f_{s r c 1}+f_{s r c 2}+f_{b l e n d}\right)
$$

## b_sff: array or list

The ratio of the source flux to the total (source + neighbors + lens) $b_{s} f f=\left(f_{S 1}+\right.$ $\left.f_{S 2}\right) /\left(f_{S 1}+f_{s 2}+f_{L}+f_{N}\right)$. This must be passed in as a list or array, with one entry for each photometric filter.
raL: float, optional
Right ascension of the lens in decimal degrees.

## decL: float, optional

Declination of the lens in decimal degrees.
class model. BSPL_PhotAstromParam3 $\left(t 0\right.$, u 0 _amp, $t E, \log 10 \_$thetaE, $p i S, p i E \_E, p i E \_N, x S O \_E, x S O \_N$, muS_E, muS_N, sep, alpha, fratio_bin, mag_base, b_sff, raL=None, dec $L=$ None)

Bases: PSPL_Param
BSPL model for astrometry and photometry - physical parameterization.
A Binary point Source Point Lens model for microlensing. This model uses a parameterization that depends on only physical quantities such as the lens mass and positions and proper motions of both the lens and source.

Note the attributes, RA (raL) and Dec ( dec L ) are required if you are calculating a model with parallax.

## Attributes

## t0: float

Time of photometric peak, as seen from Earth (MJD.DDD)
u0_amp
[float] Angular distance between the source and the GEOMETRIC center of the lenses on the plane of the sky at closest approach in units of thetaE. Can

- positive (u0_amp >0 when u0_hat[0] >0) or
- negative (u0_amp < 0 when u0_hat[0] < 0).

Note, since this is a binary source, we are expressing the nominal source position as that of the primary star in the source binary system.

## tE

[float] Einstein crossing time (days).
$\log 10 \_$thetaE
[float] The size of the Einstein radius in (mas).

## piS

[float] Amplitude of the parallax (1AU/dS) of the source. (mas)
piE_E
[float] The microlensing parallax in the East direction in units of thetaE
piE_N
[float] The microlensing parallax in the North direction in units of thetaE

## xS0_E

[float] R.A. of source position on sky at $t=t 0(\operatorname{arcsec})$ in an arbitrary ref. frame. This should be the position of the source primary.
xSO_N
[float] Dec. of source position on sky at $t=t 0(\operatorname{arcsec})$ in an arbitrary ref. frame.
muS_E
[float] RA Source proper motion (mas/yr) Identical proper motions are assumed for the source primary and secondary.
muS_N
[float] Dec Source proper motion (mas/yr) Identical proper motions are assumed for the source primary and secondary.
sep: float
Angular separation of the source scondary from the source primary (mas).

## alpha: float

Angle made between the binary source axis and North; measured in degrees East of North.

## fratio_bin: float

Flux ratio of secondary flux / primary flux.

## mag_base

[array or list] Photometric magnitude of the base. This must be passed in as a list or array, with one entry for each photometric filter. Note that

$$
\text { flux }_{b} \text { ase }=f_{\text {src1 } 1+f_{s r c 2+f_{b l e n d}}}
$$

## such that

$$
b_{s} f f=\left(f_{s r c 1}+f_{s r c 2}\right) /\left(f_{s r c 1}+f_{s r c 2}+f_{b l e n d}\right)
$$

## b_sff: array or list

The ratio of the source flux to the total (source + neighbors + lens) $b_{s} f f=\left(f_{S 1}+\right.$ $\left.f_{S 2}\right) /\left(f_{S 1}+f_{s 2}+f_{L}+f_{N}\right)$. This must be passed in as a list or array, with one entry for each photometric filter.

## raL: float, optional

Right ascension of the lens in decimal degrees.

## decL: float, optional

Declination of the lens in decimal degrees.

### 7.1.2 GP Models

```
class model.BSPL_GP_PhotParam1 (t0, u0_amp,tE, piE_E, piE_N, sep, phi,mag_src_pri,mag_src_sec,b_sff,
                gp_log_sigma,gp_rho,gp_log_omega04_SO,gp_log_omega0, raL=None,
                        decL=None)
```

Bases: BSPL_PhotParam1
BSPL model for photometry only, with GP.
A Binary point Source Point Lens model for microlensing.
Note the attributes, RA (raL) and $\operatorname{Dec}(\mathrm{decL})$ are required if you are calculating a model with parallax.

## Parameters

## t0: float

Time of photometric peak, as seen from Earth (MJD.DDD)
u0_amp: float
Angular distance between the lens and source on the plane of the sky at closest approach in units of thetaE. It can be

- positive ( $u 0$ _amp > 0 when $u 0 \_$hat $[0]>0$ ) or
- negative (u0_amp < 0 when u0_hat[0] < 0).

Note, since this is a binary source, we are expressing the nominal source position as that of the primary star in the source binary system.

## tE: float

Einstein crossing time. [MJD]
piE_E: float
The microlensing parallax in the East direction in units of thetaE

## piE_N: float

The microlensing parallax in the North direction in units of thetaE

## sep: float

Angular separation of the source scondary from the source primary (in units of thetaE).

## phi: float

Angle made between the binary axis and the relative proper motion vector, measured in degrees.

## mag_src_pri: array or list

Photometric magnitude of the first (primary) source. This must be passed in as a list or array, with one entry for each photometric filter.

## mag_src_sec: array or list

Photometric magnitude of the second (secondary) source. This must be passed in as a list or array, with one entry for each photometric filter.
b_sff: array or list
The ratio of the source flux to the total (source + neighbors + lens) $b_{s} f f=\left(f_{S 1}+\right.$ $\left.f_{S 2}\right) /\left(f_{S 1}+f_{s 2}+f_{L}+f_{N}\right)$. This must be passed in as a list or array, with one entry for each photometric filter.

## gp_log_sigma: float

Guassian process $\log (\sigma)$ for the Matern $3 / 2$ kernel.

## gp_rho: float

Guassian process $\rho$ for the Matern 3/2 kernel.

## gp_log_omega04_S0: float

Guassian process $\log \left(\omega_{0}^{4} * S_{0}\right)$ from the SHO kernel.

## gp_log_omega0: float

Guassian process $\log \left(\omega_{0}\right)$ from the SHO kernel.

## raL: float, optional

Right ascension of the lens in decimal degrees.

## decL: float, optional

Declination of the lens in decimal degrees.
class model.BSPL_GP_PhotAstromParam1 ( $m L$, $t 0$, beta, $d L, d L \_d S, x S O \_E, x S O \_N, m u L \_E, m u L \_N$, $m u S \_E$, muS_N, sep, alpha, mag_src_pri, mag_src_sec, b_sff, gp_log_sigma, gp_rho, gp_log_omega04_SO, gp_log_omega0, raL=None, $\operatorname{dec} L=$ None $)$
Bases: BSPL_PhotAstromParam1
BSPL model for astrometry and photometry with GP - physical parameterization.
A Binary point Source Point Lens model for microlensing. This model uses a parameterization that depends on only physical quantities such as the lens mass and positions and proper motions of both the lens and source.

Note the attributes, RA (raL) and Dec (decL) are required if you are calculating a model with parallax.

## Attributes

mL : float
Mass of the lens (Msun)

## t0: float

Time of photometric peak, as seen from Earth (MJD.DDD)

## beta: float

Angular distance between the lens and primary source on the plane of the sky (mas). Can be

- positive (u0_amp > 0 when u0_hat[0] < 0) or
- negative (u0_amp < 0 when $u 0 \_$hat[0] > 0 ).

Note, since this is a binary source, we are expressing the nominal source position as that of the primary star in the source binary system.

## dL: float

Distance from the observer to the lens (pc)

## dL_dS: float

Ratio of Distance from the obersver to the lens to Distance from the observer to the source

## xS0_E: float

RA Source position on sky at $\mathrm{t}=\mathrm{t} 0(\operatorname{arcsec})$ in an arbitrary ref. frame. This should be the position of the source primary.
xS0_N: float
Dec source position on sky at $\mathrm{t}=\mathrm{t} 0(\operatorname{arcsec})$ in an arbitrary ref. frame. This should be the position of the source primary.

## muL_E: float

RA Lens proper motion (mas/yr)

## muL_N: float

Dec Lens proper motion (mas/yr)

## muS_E: float

RA Source proper motion (mas/yr) Identical proper motions are assumed for the source primary and secondary.

## muS_N: float

Dec Source proper motion (mas/yr) Identical proper motions are assumed for the source primary and secondary.

## sep: float

Angular separation of the source scondary from the source primary (mas).

## alpha: float

Angle made between the binary source axis and North; measured in degrees East of North.

## mag_src_pri: array or list

Photometric magnitude of the first (primary) source. This must be passed in as a list or array, with one entry for each photometric filter.
mag_src_sec: array or list
Photometric magnitude of the second (secondary) source. This must be passed in as a list or array, with one entry for each photometric filter.

## b_sff: array or list

The ratio of the source flux to the total (source + neighbors + lens) $b_{s} f f=\left(f_{S 1}+\right.$ $\left.f_{S 2}\right) /\left(f_{S 1}+f_{s 2}+f_{L}+f_{N}\right)$. This must be passed in as a list or array, with one entry for each photometric filter.

## gp_log_sigma: float

Guassian process $\log (\sigma)$ for the Matern 3/2 kernel.

## gp_rho: float

Guassian process $\rho$ for the Matern 3/2 kernel.

## gp_log_omega04_S0: float

Guassian process $\log \left(\omega_{0}^{4} * S_{0}\right)$ from the SHO kernel.

## gp_log_omega0: float

Guassian process $\log \left(\omega_{0}\right)$ from the SHO kernel.

## raL: float, optional

Right ascension of the lens in decimal degrees.

## decL: float, optional

Declination of the lens in decimal degrees.
class model.BSPL_GP_PhotAstromParam2 ( $t 0, u 0_{-} a m p, t E$, thetaE, $p i S, p i E_{-} E, p i E \_N, x S O \_E, x S 0 \_N, m u S \_E$, muS_N, sep, alpha, fratio_bin, mag_base, b_sff, gp_log_sigma, $g p \_r h o, g p \_l o g \_o m e g a 04 \_S 0, g p \_l o g \_o m e g a 0, r a L=N o n e, ~$ $\operatorname{dec} L=$ None)
Bases: BSPL_PhotAstromParam2
BSPL model for astrometry and photometry with GP - physical parameterization.
A Binary point Source Point Lens model for microlensing. This model uses a parameterization that depends on only physical quantities such as the lens mass and positions and proper motions of both the lens and source.

Note the attributes, RA (raL) and $\operatorname{Dec}(\mathrm{decL})$ are required if you are calculating a model with parallax.

## Attributes <br> t0: float <br> Time of photometric peak, as seen from Earth (MJD.DDD) <br> u0_amp the plane of the sky at closest approach in units of thetaE. Can be <br> - positive (u0_amp > 0 when u0_hat[0] >0) or <br> - negative (u0_amp < 0 when u0_hat[0] < 0).

[float] Angular distance between the source and the GEOMETRIC center of the lenses on

Note, since this is a binary source, we are expressing the nominal source position as that of the primary star in the source binary system.
tE
[float] Einstein crossing time (days).

## thetaE

[float] The size of the Einstein radius in (mas).

## piS

[float] Amplitude of the parallax (1AU/dS) of the source. (mas)
piE_E
[float] The microlensing parallax in the East direction in units of thetaE
piE_N
[float] The microlensing parallax in the North direction in units of thetaE

## xS0_E

[float] R.A. of source position on sky at $t=t 0(\operatorname{arcsec})$ in an arbitrary ref. frame. This should be the position of the source primary.

## xS0_N

[float] Dec. of source position on sky at $\mathrm{t}=\mathrm{t} 0(\operatorname{arcsec})$ in an arbitrary ref. frame.
muS_E
[float] RA Source proper motion (mas/yr) Identical proper motions are assumed for the source primary and secondary.
muS_N
[float] Dec Source proper motion (mas/yr) Identical proper motions are assumed for the source primary and secondary.

## sep: float

Angular separation of the source scondary from the source primary (mas).

## alpha: float

Angle made between the binary source axis and North; measured in degrees East of North.

## fratio_bin: float

Flux ratio of secondary flux / primary flux.

## mag_base

[array or list] Photometric magnitude of the base. This must be passed in as a list or array, with one entry for each photometric filter. Note that

$$
f l u x_{\text {base }}=f_{\text {src } 1}+f_{\text {src } 2}+f_{\text {blend }}
$$

## such that

$$
b_{s f f}=\left(f_{s r c 1}+f_{s r c 2}\right) /\left(f_{s r c 1}+f_{s r c 2}+f_{b l e n d}\right)
$$

## b_sff: array or list

The ratio of the source flux to the total (source + neighbors + lens) :math:` $\mathrm{b}_{-}\{\mathrm{sff}\}=\left(\mathrm{f}_{-}\{\mathrm{S} 1\}\right.$ $\left.+f_{-}\{S 2\}\right) /\left(f_{-}\{S 1\}+f_{-}\{s 2\}+f_{-} L+f_{-} N\right)^{\prime}$. This must be passed in as a list or array, with one entry for each photometric filter.

## gp_log_sigma: float

Guassian process $\log (\sigma)$ for the Matern 3/2 kernel.

## gp_rho: float

Guassian process $\rho$ for the Matern $3 / 2$ kernel.

## gp_log_omega04_S0: float

Guassian process $\log \left(\omega_{0}^{4} * S_{0}\right)$ from the SHO kernel.

## gp_log_omega0: float

Guassian process $\log \left(\omega_{0}\right)$ from the SHO kernel.

## raL: float, optional

Right ascension of the lens in decimal degrees.

## decL: float, optional

Declination of the lens in decimal degrees.

```
class model.BSPL_GP_PhotAstromParam3 (t0,u0_amp, tE, log10_thetaE,piS, piE_E,piE_N, xS0_E, xSO_N,
                muS_E,muS_N, sep, alpha, fratio_bin, mag_base, b_sff,
                        gp_log_sigma,gp_rho,gp_log_omega04_S0,gp_log_omega0,
                        raL=None, decL=None)
```

Bases: BSPL_PhotAstromParam3
Point Source Point Lens with GP model for microlensing. This model includes proper motions of the source and the source position on the sky. It is the same as PSPL_PhotAstromParam4 except it fits for log10(thetaE) instead of thetaE.

## Attributes

## t0: float

Time of photometric peak, as seen from Earth (MJD.DDD)

## u0_amp: float

Angular distance between the lens and source on the plane of the sky at closest approach in units of thetaE. Can be

- positive (u0_amp > 0 when $u 0 \_$hat[0] > 0 ) or
- negative (u0_amp < 0 when u0_hat[0] < 0).


## tE: float

Einstein crossing time (days).

## $\log 10 \_$thetaE: float

$\log 10$ of the size of the Einstein radius in (mas).

## piS: float

Amplitude of the parallax (1AU/dS) of the source. (mas)

## piE_E: float

The microlensing parallax in the East direction in units of thetaE

## piE_N: float

The microlensing parallax in the North direction in units of thetaE

## xSO_E: float

RA Source position on sky at $\mathrm{t}=\mathrm{t} 0(\operatorname{arcsec})$ in an arbitrary ref. frame.

## xS0_N: float

Dec source position on sky at $\mathrm{t}=\mathrm{t} 0(\operatorname{arcsec})$ in an arbitrary ref. frame.

## muS_E: float

RA Source proper motion (mas/yr)

## muS_N: float

Dec Source proper motion (mas/yr)

## sep: float

Angular separation of the source scondary from the source primary (mas).

## alpha: float

Angle made between the binary source axis and North; measured in degrees East of North.

## fratio_bin: float

Flux ratio of secondary flux / primary flux.

## mag_base

[array or list] Photometric magnitude of the base. This must be passed in as a list or array, with one entry for each photometric filter. Note that

$$
f_{l u x_{b a s e}}=f_{s r c 1}+f_{\text {src } 2}+f_{\text {blend }}
$$

## such that

$$
b_{s f f}=\left(f_{s r c 1}+f_{s r c 2}\right) /\left(f_{s r c 1}+f_{s r c 2}+f_{b l e n d}\right)
$$

## b_sff: array or list

The ratio of the source flux to the total (source + neighbors + lens) :math: ${ }^{\text {b }}$ _ $\{\mathrm{sff}\}=\left(\mathrm{f}_{-}\{\mathrm{S} 1\}\right.$ $\left.+f_{-}\{S 2\}\right) /\left(f_{-}\{S 1\}+f_{-}\{s 2\}+f_{-} L+f_{-} N\right)^{`}$. This must be passed in as a list or array, with one entry for each photometric filter.

## gp_log_sigma: float

Guassian process $\log (\sigma)$ for the Matern 3/2 kernel.

## gp_rho: float

Guassian process $\rho$ for the Matern $3 / 2$ kernel.

## gp_log_omega04_S0: float

Guassian process $\log \left(\omega_{0}^{4} * S_{0}\right)$ from the SHO kernel.
gp_log_omega0: float
Guassian process $\log \left(\omega_{0}\right)$ from the SHO kernel.

### 7.2 Data Class Family

class model.BSPL
Bases: PSPL

## Methods

| animate(tE, time_steps, frame_time, name, ...) | Produces animation of microlensing event. |
| :--- | :--- |
| get_amplification(t[, filt_idx]) | Parallax: Get the photometric amplification term at a <br> set of times,. |
| get_resolved_amplification(t) | Parallax: Get the photometric amplification term at a <br> set of times, t for both the plus and minus images. |
| get_u(t) | Parameters |


| get_chi2_photometry |  |
| :--- | :--- |
| get_lnL_constant |  |
| get_photometry |  |
| log_likely_photometry |  |
| log_likely_photometry_each |  |

get_u $(t)$

## Parameters

t
[numpy array] Times at which to evaluate the source-lens separation.

## Returns

u
[numpy array] Shape $=[$ len $(t), 2$ sources, 2 directions on sky]
get_resolved_amplification $(t)$
Parallax: Get the photometric amplification term at a set of times, t for both the plus and minus images.

## Parameters

t: Array of times in MJD.DDD

## Returns

A_resolved
[numpy array] [shape $=\operatorname{len}(\mathrm{t})$, len(sources), 2]

## Notes

## For each time $t$ and each source, we have:

- A_plus is the amplification for the plus image.
- A_minus is the amplification for the minus image.


## In other words,

$\mathrm{xS}[0,0,0]$ returns the amplification of the first source's "plus" image at the first time.
Similarly,
$x S[0,0,1]$ returns the amplification of the first source's "minus" image at the first time.
get_amplification ( $t$, filt_idx=0)
Parallax: Get the photometric amplification term at a set of times, t .
Note that this is a convenience function that combines amplifications from multiple sources. The returned amplification is
..math:
$\mathrm{A}=(\mathrm{f} 1 * \mathrm{~A} 1+\mathrm{f} 2 * \mathrm{~A} 2) /(\mathrm{f} 1+\mathrm{f} 2)$
where the fluxes are the intrinsic source flux in the specified filter.

## Parameters

t: Array of times in MJD.DDD

## Returns

A
[numpy array]
Array of combined amplifications in the specified filter.
Shape $=[\operatorname{len}(\mathrm{t})]$
animate (tE, time_steps, frame_time, name, size, zoom, astrometry)
Produces animation of microlensing event. This function takes the PSPL and makes an animation, the input variables are as follows

## Parameters

## tE:

number of einstein crossings times before/after the peak you want the animation to plot
e.g $\mathrm{tE}=2$ => graph will go from -2 tE to 2 tE
time_steps:
number of time steps before/after peak, so total number of time steps will be 2 times this value

## frame_time:

times in ms of each frame in the animation
name: string
the animation will be saved as name.html
size: list
[horizontal, vertical] cm's

## zoom:

\# of einstein radii plotted in vertical direction

## class model.BSPL_Phot

Bases: BSPL, PSPL_Phot
Contains methods for model a BSPL photometry only. This is a Data-type class in our hierarchy. It is abstract and should not be instantiated.

## Attributes

t0
tE
u0_amp
u0_E
u0_N

## piE_E:

valid only if parallax model
piE_N:
valid only if parallax model
piE_amp
b_sff[\#]
mag_src[\#]:
add in
mag_base[\#]:
add in
raL:
if parallax model
decL:
if parallax model

## Methods

| animate(tE, time_steps, frame_time, name, ...) | Produces animation of microlensing event. <br> get_amplification(t[, filt_idx]) <br> get_astrometry(t[, ast_filt_idx]) <br> set of times, t. |
| :--- | :--- |
| get_resolved_amplification(t) | Position of the observed (unresolved) source position <br> in Einstein radii. |
| get_resolved_astrometry(t) | Get the astrometry of the source if the lens didn't ex- <br> ist. |
| get_resolved_astrometry_unlensed(t) | Parallax: Get the photometric amplification term at a <br> set of times, t for both the plus and minus images. |
| get_u(t) | Position of the observed source position in Einstein <br> radii. |
| Get the astrometry of the source if the lens didn't ex- <br> ist. |  |


| get_centroid_shift |  |
| :--- | :--- |
| get_chi2_photometry |  |
| get_lens_astrometry |  |
| get_InL_constant |  |
| get_photometry |  |
| log_likely_astrometry |  |
| log_likely_photometry |  |
| log_likely_photometry_each |  |

## get_resolved_astrometry_unlensed ( $t$ )

Get the astrometry of the source if the lens didn't exist. Note, this is a photometry only model, so units are in Einstein radii.

## Returns

xS_resolved_unlensed
[numpy array, [shape = len(t_obs), N_sources, 2]] The unlensed positions of the sources in Einstein radii.

## In other words,

$\mathrm{xS}[0,0,:]$ returns the 2D sky position of the first source at the first time.
Similarly,
$\mathrm{xS}[0,1,:]$ returns the 2D sky position of the second source at the first time.

## get_astrometry_unlensed ( $t$, ast_filt_idx=0)

Get the astrometry of the source if the lens didn't exist. Note, this is a photometry only model, so units are in Einstein radii.

## Returns

xS_unlensed
[numpy array, dtype=float, shape $=$ len(t_obs) x 2] The unlensed positions of the source in Einstein radii.

## get_resolved_astrometry $(t)$

Position of the observed source position in Einstein radii.

## Parameters

t
[array_like, shape $=[\mathrm{N}$ _times $]$ A Array of times to model.

## Returns

model_pos
[array_like. shape $=\left[\mathrm{N} \_\right.$times, $\mathrm{N} \_$images, 2] Array of vector positions of the centroid at each t_obs.

```
get_astrometry(t, ast_fil_idx=0)
```

Position of the observed (unresolved) source position in Einstein radii.

## Parameters

t: array_like
Array of times to model.

## Returns

## model_pos

[array_like] Array of vector positions of the centroid at each t_obs.
animate (tE, time_steps, frame_time, name, size, zoom, astrometry)
Produces animation of microlensing event. This function takes the PSPL and makes an animation, the input variables are as follows

## Parameters

tE:
number of einstein crossings times before/after the peak you want the animation to plot
e.g $\mathrm{tE}=2$ => graph will go from -2 tE to 2 tE
time_steps:
number of time steps before/after peak, so total number of time steps will be 2 times this value
frame_time:
times in ms of each frame in the animation
name: string
the animation will be saved as name.html
size: list
[horizontal, vertical] cm's

## zoom:

\# of einstein radii plotted in vertical direction
get_amplification ( $t$, filt_id $x=0$ )
Parallax: Get the photometric amplification term at a set of times, $t$.
Note that this is a convenience function that combines amplifications from multiple sources. The returned amplification is
..math:
$A=(f 1 * A 1+f 2 * A 2) /(f 1+f 2)$
where the fluxes are the intrinsic source flux in the specified filter.

## Parameters

t: Array of times in MJD.DDD

## Returns

A
[numpy array]
Array of combined amplifications in the specified filter.
Shape $=[\operatorname{len}(t)]$
get_resolved_amplification $(t)$
Parallax: Get the photometric amplification term at a set of times, $t$ for both the plus and minus images.

## Parameters

t: Array of times in MJD.DDD

## Returns

## A_resolved

[numpy array] [shape $=\operatorname{len}(\mathrm{t})$, len(sources), 2]

## Notes

## For each time $t$ and each source, we have:

- A_plus is the amplification for the plus image.
- A_minus is the amplification for the minus image.


## In other words,

$\mathrm{xS}[0,0,0]$ returns the amplification of the first source's "plus" image at the first time.

## Similarly,

 $\mathrm{xS}[0,0,1]$ returns the amplification of the first source's "minus" image at the first time.```
get_u(t)
```


## Parameters

t
[numpy array] Times at which to evaluate the source-lens separation.

## Returns

u
[numpy array] Shape $=[$ len $(t), 2$ sources, 2 directions on sky]

## class model.BSPL_PhotAstrom

Bases: BSPL, PSPL_PhotAstrom

## Methods

| animate(tE, time_steps, frame_time, name, ...) | Produces animation of microlensing event. <br> get_amplification(t[f, fil__idx])Parallax: Get the photometric amplification term at a <br> set of times, t. |
| :--- | :--- |
| get_astrometry(t[, ast_filt_idx]) | Parallax: Get unresolved astrometry for binary <br> source, point lens. |
| get_astrometry_unlensed(t[, ast_filt_idx]) | Get the astrometry of the sources if the lens didn't <br> exist. |
| get_centroid_shift(t[, ast_filt_idx]) | Parallax: Get the centroid shift (in mas) for a list of <br> observation times (in MJD). |
| get_resolved_amplification(t) | Parallax: Get the photometric amplification term at a a <br> set of times, t for both the plus and minus images. |
| get_resolved_astrometry(t) | Parallax: For each source, get the x, y astrometry for <br> the two lensed source images. |
| get_resolved_astrometry_unlensed(t) | Get the astrometry of the source if the lens didn't ex- <br> ist. |
| get_u(t) | Parameters |


| get_chi2_astrometry |  |
| :--- | :--- |
| get_chi2_photometry |  |
| get_lnL_constant |  |
| get_photometry |  |
| log_likely_astrometry |  |
| log_likely_astrometry_each |  |
| log_likely_photometry |  |
| log_likely_photometry_each |  |

## get_resolved_astrometry_unlensed ( $t$ )

Get the astrometry of the source if the lens didn't exist.

## Returns

xS_resolved_unlensed
[numpy array, [shape = len(t_obs), N_sources, 2]] The unlensed positions of the sources in arcseconds.

## In other words,

$\mathrm{xS}[0,0,:]$ returns the 2 D sky position of the first source at the first time.
Similarly,
$\mathrm{xS}[0,1,:]$ returns the 2 D sky position of the second source at the first time.

```
get_astrometry_unlensed(t,ast_filt_idx=0)
```

Get the astrometry of the sources if the lens didn't exist.

## Returns

xS_unlensed
[numpy array, dtype=float, shape $=$ len(t_obs) x 2 directions]
The unlensed positions of the combined sources in arcseconds.
Shape $=[\operatorname{len}(\mathrm{t}), 2$ directions $]$
get_resolved_astrometry $(t)$
Parallax: For each source, get the $x$, $y$ astrometry for the two lensed source images. For each source, we label the two images as plus and minus.

## Returns

xS_resolved
[numpy array] [shape $=\operatorname{len}(\mathrm{t})$, len(sources), 2, 2]

## Notes

## For each time $t$ and each source, we have:

- xS_plus is the vector position of the plus image.
- xS_minus is the vector position of the minus image.

In other words, $x S[0,0,0,:]$ returns the 2 D sky position of the first source's "plus" image at the first time.

## Similarly,

 $\mathrm{xS}[0,0,1,:]$ returns the 2D sky position of the first source's "minus" image at the first time.get_astrometry (t, ast_fil_idx=0)
Parallax: Get unresolved astrometry for binary source, point lens.

## Parameters

t:
Array of times in MJD.DDD

## Returns

xS_lensed
Returns flux-weighted average of lensed source positions.
get_centroid_shift ( $t$, ast_fil_idx $=0$ )
Parallax: Get the centroid shift (in mas) for a list of observation times (in MJD).
Returns the flux-weighted centroid of all the sources lensed images.

## Parameters

t:
Array of times in MJD.DDD

## Returns

centroid_shift
[numpy array] [shape = len(t), 2]
animate ( $t$ E, time_steps, frame_time, name, size, zoom, astrometry)
Produces animation of microlensing event. This function takes the PSPL and makes an animation, the input variables are as follows

## Parameters

## tE:

number of einstein crossings times before/after the peak you want the animation to plot
e.g $\mathrm{tE}=2=>$ graph will go from -2 tE to 2 tE

## time_steps:

number of time steps before/after peak, so total number of time steps will be 2 times this value
frame_time:
times in ms of each frame in the animation
name: string
the animation will be saved as name.html
size: list
[horizontal, vertical] cm's

## zoom:

\# of einstein radii plotted in vertical direction

## get_amplification(t, filt_idx=0)

Parallax: Get the photometric amplification term at a set of times, $t$.
Note that this is a convenience function that combines amplifications from multiple sources. The returned amplification is
..math:

```
A = (f1 * A1 + f2 * A2) / (f1 + f2)
```

where the fluxes are the intrinsic source flux in the specified filter.

## Parameters

t: Array of times in MJD.DDD

## Returns

A
[numpy array]
Array of combined amplifications in the specified filter.
Shape $=[\operatorname{len}(\mathrm{t})]$

## get_resolved_amplification( $t$ )

Parallax: Get the photometric amplification term at a set of times, $t$ for both the plus and minus images.

## Parameters

t: Array of times in MJD.DDD

## Returns

A_resolved
[numpy array] [shape $=\operatorname{len}(t)$, len(sources $), 2]$

Notes

## For each time $t$ and each source, we have:

- A_plus is the amplification for the plus image.
- A_minus is the amplification for the minus image.


## In other words,

 $\mathrm{xS}[0,0,0]$ returns the amplification of the first source's "plus" image at the first time.
## Similarly,

 $x \mathrm{x}[0,0,1]$ returns the amplification of the first source's "minus" image at the first time.get_u( $t$ )

## Parameters

t
[numpy array] Times at which to evaluate the source-lens separation.

## Returns

u
[numpy array] Shape = [len(t), 2 sources, 2 directions on sky]

### 7.3 Parallax Class Family

## class model.BSPL_noParallax

Bases: PSPL_noParallax

## Methods

| get_amplification(t) | noParallax: Get the photometric amplification term <br> at a set of times, t. |
| :--- | :--- |
| get_astrometry(t_obs[, ast_filt_idx]) | noParallax: Position of the observed source position <br> in arcsec. |
| get_astrometry_unlensed(t_obs) | noParallax: Get the astrometry of the source if the <br> lens didn't exist. |
| get_centroid_shift(t) | noParallax: Get the centroid shift (in mas) for a list <br> of observation times (in MJD). |
| get_lens_astrometry(t_obs) | Equation of motion for just the foreground lens. |
| get_resolved_amplification(t) | Get the photometric amplification term at a set of <br> times, t for both the plus and minus images. |
| get_resolved_astrometry(t_obs) | Get the $x, y$ astrometry for each of the two source <br> images, which we label plus and minus. |

## calc_piE_ecliptic

## get_amplification $(t)$

noParallax: Get the photometric amplification term at a set of times, t .

## Parameters

t:
Array of times in MJD.DDD
get_astrometry (t_obs, ast_filt_idx=0)
noParallax: Position of the observed source position in arcsec.
get_astrometry_unlensed (t_obs)
noParallax: Get the astrometry of the source if the lens didn't exist.

## Returns

xS_unlensed
[numpy array, dtype=float, shape $=$ len(t_obs) x 2] The unlensed positions of the source in arcseconds.

## get_centroid_shift ( $t$ )

noParallax: Get the centroid shift (in mas) for a list of observation times (in MJD).

```
get_lens_astrometry (t_obs)
```

Equation of motion for just the foreground lens.

## Parameters

t_obs
[array_like] Time (in MJD).

## get_resolved_amplification ( $t$ )

Get the photometric amplification term at a set of times, t for both the plus and minus images.

## Parameters

t:
Array of times in MJD.DDD
get_resolved_astrometry (t_obs)
Get the $\mathrm{x}, \mathrm{y}$ astrometry for each of the two source images, which we label plus and minus.

## Returns

[xS_plus, xS_minus]
[list of numpy arrays]

- xS_plus is the vector position of the plus image in arcsec
- xS_minus is the vector position of the plus image in arcsec
class model.BSPL_Parallax
Bases: PSPL_Parallax


## Methods

| calc_piE_ecliptic() | Parallax: Get piE_ecliptic |
| :--- | :--- |
| get_amplification(t) | Parallax: Get the photometric amplification term at a <br> set of times, t. |
| get_astrometry(t_obs[, ast_filt_idx]) | Parallax: Get astrometry <br> get_astrometry_unlensed(t_obs) <br> ist. |
| get_centroid_shift(t[, ast_filt_idx]) | Parallax: Get the centroid shift (in mas) for a list of <br> observation times (in MJD). |
| get_geoproj_ast_params(t0par) |  |
| get_geoproj_params(t0par) | Parallax: Get lens astrometry <br> set of times, t for both the plus and minus images. |
| get_lens_astrometry(t_obs) | Parallax: Get the x, y astrometry for each of the two <br> source images, which we label plus and minus. |
| get_resolved_amplification(t) |  |

## start

calc_piE_ecliptic()
Parallax: Get piE_ecliptic
get_amplification $(t)$
Parallax: Get the photometric amplification term at a set of times, $t$.

## Parameters

t:
Array of times in MJD.DDD
get_astrometry (t_obs, ast_fil_idx=0)
Parallax: Get astrometry

## get_astrometry_unlensed (t_obs)

Get the astrometry of the source if the lens didn't exist.

## Returns

xS_unlensed
[numpy array, dtype=float, shape $=$ len(t_obs) $\times 2$ 2] The unlensed positions of the source in arcseconds.
get_centroid_shift (t, ast_fil_idx=0)
Parallax: Get the centroid shift (in mas) for a list of observation times (in MJD).

```
get_lens_astrometry(t_obs)
```

Parallax: Get lens astrometry

## get_resolved_amplification $(t)$

Parallax: Get the photometric amplification term at a set of times, t for both the plus and minus images.

## Parameters

t:
Array of times in MJD.DDD
get_resolved_astrometry (t_obs)
Parallax: Get the $\mathrm{x}, \mathrm{y}$ astrometry for each of the two source images, which we label plus and minus.

## Returns

[xS_plus, xS_minus]
[list of numpy arrays]

- xS_plus is the vector position of the plus image.
- xS_minus is the vector position of the plus image.


## FSPL(FINITE-SOURCE POINT LENS) MODELS - NOT DONE YET...

 PLACE HOLDERS
### 8.1 FSPL Classes

class model.FSPL
Bases: PSPL

## Methods

| animate $(\mathrm{tE}$, time_steps, frame_time, name, ...) | Produces animation of microlensing event. |
| :--- | :--- |
| get_photometry(t_obs[, filt_idx, amp_arr, ...]) | Get the photometry for each of the lensed source im- <br> ages. |
| get_source_outline_astrometry(r, n, center) | Return astrometric points that outline the outer cir- <br> cumference of the source star. |


| get_chi2_photometry |  |
| :--- | :--- |
| get_lnL_constant |  |
| log_likely_photometry |  |
| log_likely_photometry_each |  |

get_source_outline_astrometry ( $r$, $n$, center)
Return astrometric points that outline the outer circumference of the source star.

The outline is described as a circle of radius self.radius and is evaluated at self.n_outline number of points.
takes in the radius of the circle, centre position and number of points we are approximating the circle by and returns a numpy array of positions
e.g: $(((1,0),(0,1),(-1,0),(0,-1)))$ if $n=4$ and radius $=1$

## Returns

source_points
[numpy array] Returns an array of shape = [2, self.n_outline, len(time)]
get_photometry (t_obs, filt_idx=0, amp_arr=None, print_warning=True)
Get the photometry for each of the lensed source images.

## Parameters

t_obs
[array_like] Array of times to model.

## Other Parameters

## amp_arr

[array_like] Amplifications of each individual image at each time, i.e. amp_arr.shape $=$ (len(t_obs), number of images at each t_obs).

This will over-ride t_obs; but is more efficient when calculating both photometry and astrometry. If None, then just use t_obs.

## Returns

mag_model
[array_like] Magnitude of the centroid at t_obs.
animate (tE, time_steps, frame_time, name, size, zoom, astrometry)
Produces animation of microlensing event. This function takes the PSPL and makes an animation, the input variables are as follows

## Parameters

tE:
number of einstein crossings times before/after the peak you want the animation to plot
e.g tE $=2=>$ graph will go from -2 tE to 2 tE

## time_steps:

number of time steps before/after peak, so total number of time steps will be 2 times this value

## frame_time:

times in ms of each frame in the animation
name: string
the animation will be saved as name.html
size: list
[horizontal, vertical] cm's
zoom:
\# of einstein radii plotted in vertical direction
class model.FSPL_PhotAstrom
Bases: FSPL, PSPL_PhotAstrom
Contains methods for model a PSPL photometry + astrometry. This is a Data-type class in our hierarchy. It is abstract and should not be instantiated.

## Attributes

Available class variables that should be defined.
t0
tE
u0_amp
u0_E
u0_N
beta
piE_E:
valid only if parallax model
piE_N:
valid only if parallax model
piE_amp
mL
thetaE_amp
thetaE_E
thetaE_N
xS0_E
xS0_N
xL0_E
xL0_N
muS_E
muS_N
muL_E
muL_N
muRel_E
muRel_N
muRel_amp
piS
piL
dL
dS
dL_dS (dL over dS)
radius
n
b_sff[\#]
mag_src[\#] - add in
mag_base[\#] - add in
raL:
if parallax model
decL:
if parallax model

## Methods

| animate(tE, time_steps, frame_time, name, ...) | Produces animation of microlensing event. |
| :--- | :--- |
| get_astrometry_unlensed(t) | Outputs position of source unlensed. |
| get_lens_astrometry(t_obs) | Get the photometry for each of the lensed source im- <br> ages. |
| get_photometry(t_obs[, filt_idx, amp_arr, ...]) | Return astrometric points that outline the outer cir- <br> cumference of the source star. |
| get_source_outline_astrometry(r, n, center) |  |


| get_chi2_astrometry |  |
| :--- | :--- |
| get_chi2_photometry |  |
| get_lnL_constant |  |
| get_resolved_astrometry |  |
| log_likely_astrometry |  |
| log_likely_astrometry_each |  |
| log_likely_photometry |  |
| log_likely_photometry_each |  |

get_photometry (t_obs, filt_idx=0, amp_arr=None, print_warning=True)
Get the photometry for each of the lensed source images.

## Parameters

t_obs
[array_like] Array of times to model.

## Other Parameters

amp_arr
[array_like] Amplifications of each individual image at each time, i.e. amp_arr.shape $=$ (len(t_obs), number of images at each $t \_o b s$ ).

This will over-ride t_obs; but is more efficient when calculating both photometry and astrometry. If None, then just use t_obs.

## Returns

mag_model
[array_like] Magnitude of the centroid at t_obs.
animate (tE, time_steps, frame_time, name, size, zoom, astrometry)
Produces animation of microlensing event. This function takes the PSPL and makes an animation, the input variables are as follows

## Parameters

tE:
number of einstein crossings times before/after the peak you want the animation to plot
e.g $\mathrm{tE}=2$ => graph will go from -2 tE to 2 tE
time_steps:
number of time steps before/after peak, so total number of time steps will be 2 times this value

## frame_time:

times in ms of each frame in the animation
name: string
the animation will be saved as name.html
size: list
[horizontal, vertical] cm's

## zoom:

\# of einstein radii plotted in vertical direction
get_astrometry_unlensed $(t)$
Outputs position of source unlensed.
Input a list of times and it will output the position of the source had it not been lensed at each of the times in the list
e.g if $n=4$, and say $v=[1,0] \&$ the times are $[0,1,2]$ in years.

This will return
$((((1, \theta),(\theta, 1),(-1, \theta),(\theta,-1)),((2, \theta),(1,1),(\theta, \theta),(1,-1)),($ $(3,0),(2,1),(1,0),(2,-1))) \ldots$
$=($ positions at $\mathrm{t}=0)$, (positions at $\mathrm{t}=1),($ positions at $\mathrm{t}=2)$
so np.array (positions) is an array which contains an array for each time step with the positions of all the points on the boundary of the source.
get_source_outline_astrometry ( $r, n$, center )
Return astrometric points that outline the outer circumference of the source star.

The outline is described as a circle of radius self.radius and is evaluated at self.n_outline number of points.
takes in the radius of the circle, centre position and number of points we are approximating the circle by and returns a numpy array of positions
e.g: $(((1, \theta),(\theta, 1),(-1, \theta),(\theta,-1)))$ if $n=4$ and radius $=1$

## Returns

source_points
[numpy array] Returns an array of shape = [2, self.n_outline, len(time)]
class model.FSPL_noParallax
Bases: PSPL_noParallax

## Methods

| get_amplification(t) | noParallax: Get the photometric amplification term <br> at a set of times, t. |
| :--- | :--- |
| get_astrometry(t_obs[, ast_filt_idx]) | noParallax: Position of the observed source position <br> in arcsec. |
| get_astrometry_unlensed(t_obs) | noParallax: Get the astrometry of the source if the <br> lens didn't exist. |
| get_centroid_shift(t) | noParallax: Get the centroid shift (in mas) for a list <br> of observation times (in MJD). |
| get_lens_astrometry(t_obs) | Equation of motion for just the foreground lens. |
| get_resolved_amplification(t) | Get the photometric amplification term at a set of <br> times, t for both the plus and minus images. |
| get_resolved_astrometry(t_obs) | Get the $x, y$ astrometry for each of the two source <br> images, which we label plus and minus. |

## calc_piE_ecliptic

## get_amplification $(t)$

noParallax: Get the photometric amplification term at a set of times, t .

## Parameters

t:
Array of times in MJD.DDD
get_astrometry (t_obs, ast_filt_idx=0)
noParallax: Position of the observed source position in arcsec.
get_astrometry_unlensed (t_obs)
noParallax: Get the astrometry of the source if the lens didn't exist.

## Returns

xS_unlensed
[numpy array, dtype=float, shape $=$ len(t_obs) x 2] The unlensed positions of the source in arcseconds.

## get_centroid_shift $(t)$

noParallax: Get the centroid shift (in mas) for a list of observation times (in MJD).

```
get_lens_astrometry(t_obs)
```

Equation of motion for just the foreground lens.

## Parameters

t_obs
[array_like] Time (in MJD).
get_resolved_amplification( $t$ )
Get the photometric amplification term at a set of times, t for both the plus and minus images.

## Parameters

t:
Array of times in MJD.DDD

## get_resolved_astrometry (t_obs)

Get the x , y astrometry for each of the two source images, which we label plus and minus.

## Returns

[xS_plus, xS_minus]
[list of numpy arrays]

- xS_plus is the vector position of the plus image in arcsec
- xS_minus is the vector position of the plus image in arcsec
class model.FSPL_Parallax
Bases: PSPL_Parallax

Methods

| calc_piE_ecliptic() | Parallax: Get piE_ecliptic |
| :--- | :--- |
| get_amplification(t) | Parallax: Get the photometric amplification term at a <br> set of times,. |
| get_astrometry(t_obs[, ast_filt_idx]) | Parallax: Get astrometry |
| get_astrometry_unlensed(t_obs) | Get the astrometry of the source if the lens didn't ex- <br> ist. |
| get_centroid_shift(t[, ast_filt_idx]) | Parallax: Get the centroid shift (in mas) for a list of <br> observation times (in MJD). |
| get_geoproj_ast_params(t0par) |  |
| get_geoproj_params(t0par) | Parallax: Get lens astrometry <br> set of times, the for both the plus and minus images. |
| get_lens_astrometry(t_obs) | Parallax: Get the x, y astrometry for each of the two <br> source images, which we label plus and minus. |
| get_resolved_amplification(t) |  |

## start

## calc_piE_ecliptic()

Parallax: Get piE_ecliptic

## get_amplification $(t)$

Parallax: Get the photometric amplification term at a set of times, t .

## Parameters

t:
Array of times in MJD.DDD
get_astrometry (t_obs, ast_filt_idx=0)
Parallax: Get astrometry
get_astrometry_unlensed(t_obs)
Get the astrometry of the source if the lens didn't exist.

## Returns

## xS_unlensed

[numpy array, dtype=float, shape $=$ len(t_obs) x 2] The unlensed positions of the source in arcseconds.
get_centroid_shift ( $t$, ast_filt_idx=0)
Parallax: Get the centroid shift (in mas) for a list of observation times (in MJD).

```
get_lens_astrometry(t_obs)
```

Parallax: Get lens astrometry

```
get_resolved_amplification(t)
```

Parallax: Get the photometric amplification term at a set of times, t for both the plus and minus images.

## Parameters

t:
Array of times in MJD.DDD

## get_resolved_astrometry (t_obs)

Parallax: Get the $\mathrm{x}, \mathrm{y}$ astrometry for each of the two source images, which we label plus and minus.

## Returns

[xS_plus, xS_minus]
[list of numpy arrays]

- xS_plus is the vector position of the plus image.
- xS_minus is the vector position of the plus image.
class model.FSPL_PhotAstromParam $1\left(m L, t 0\right.$, beta, $d L, d S, x S O_{-} E, x S O_{-} N, m u L_{-} E, m u L \_N, m u S \_E, m u S \_N$, radius, $b$ _sff, mag_src, $n \_$outline $\left.=10, r a L=N o n e, d e c L=N o n e\right)$
Bases: PSPL_Param
PSPL model for astrometry and photometry - physical parameterization.
A Point Source Point Lens model for microlensing. This model uses a parameterization that depends on only physical quantities such as the lens mass and positions and proper motions of both the lens and source.
Note the attributes, RA (raL) and Dec (decL) are required if you are calculating a model with parallax.


## Parameters

mL : float
Mass of the lens (Msun)

## t0: float

Time of photometric peak, as seen from Earth (MJD.DDD)

## beta: float

Angular distance between the lens and source on the plane of the sky (mas). Can be

- positive (u0_amp > 0 when u0_hat[0] (East component) < 0) or
- negative (u0_amp < 0 when u0_hat[0] (East component) >0).


## dL: float

Distance from the observer to the lens (pc)

## dL_dS: float

Ratio of Distance from the obersver to the lens to Distance from the observer to the source

## xS0_E: float

RA Source position on sky at $\mathrm{t}=\mathrm{t} 0(\operatorname{arcsec})$ in an arbitrary ref. frame.

## xS0_N: float

Dec source position on sky at $\mathrm{t}=\mathrm{t} 0(\operatorname{arcsec})$ in an arbitrary ref. frame.
muL_E: float
RA Lens proper motion (mas/yr)
muL_N: float
Dec Lens proper motion (mas/yr)
muS_E: float
RA Source proper motion (mas/yr)
muS_N: float
Dec Source proper motion (mas/yr)

## radius: float

Projected radius of the star in arcsec on the sky plane.
b_sff: float
The ratio of the source flux to the total (source + neighbors + lens) $b_{s} f f=f_{S} /\left(f_{S}+f_{L}+\right.$ $\left.f_{N}\right)$. This must be passed in as a list or array, with one entry for each photometric filter.
mag_src: float
Photometric magnitude of the source. This must be passed in as a list or array, with one entry for each photometric filter.

## n_outline: int

Number of boundary points to use when approximating the source outline. Calculation time scales approximately linearly with ' $n$ _outline'.

## raL: float, optional

Right ascension of the lens in decimal degrees.

## decL: float, optional

Declination of the lens in decimal degrees.

```
class model.FSPL_PhotAstrom
```

Bases: FSPL, PSPL_PhotAstrom
Contains methods for model a PSPL photometry + astrometry. This is a Data-type class in our hierarchy. It is abstract and should not be instantiated.

## Attributes

Available class variables that should be defined.
t0
tE
u0_amp
u0_E
u0_N
beta
piE_E:
valid only if parallax model
piE_N:
valid only if parallax model
piE_amp
mL
thetaE_amp
thetaE_E
thetaE_N
xS0_E
xS0_N
xL0_E
xL0_N
muS_E
muS_N
muL_E
muL_N
muRel_E
muRel_N
muRel_amp
piS
piL
dL
dS
dL_dS (dL over dS)
radius
n
b_sff[\#]
mag_src[\#] - add in
mag_base[\#] - add in
raL:
if parallax model
decL:
if parallax model

## Methods

| animate $(\mathrm{tE}$, time_steps, frame_time, name, ...) | Produces animation of microlensing event. |
| :--- | :--- |
| get_astrometry_unlensed(t) | Outputs position of source unlensed. |
| get_lens_astrometry(t_obs) | Get the photometry for each of the lensed source im- <br> ages. |
| get_photometry(t_obs[, filt_idx, amp_arr, ...]) | Return astrometric points that outline the outer cir- <br> cumference of the source star. |
| get_source_outline_astrometry(r, n, center) |  |


| get_chi2_astrometry |  |
| :--- | :--- |
| get_chi2_photometry |  |
| get_lnL_constant |  |
| get_resolved_astrometry |  |
| log_likely_astrometry |  |
| log_likely_astrometry_each |  |
| log_likely_photometry |  |
| log_likely_photometry_each |  |

get_photometry (t_obs, filt_idx=0, amp_arr=None, print_warning=True)
Get the photometry for each of the lensed source images.

## Parameters

## t_obs

[array_like] Array of times to model.

## Other Parameters

## amp_arr

[array_like] Amplifications of each individual image at each time, i.e. amp_arr.shape $=$ (len(t_obs), number of images at each $t$ _obs).

This will over-ride t_obs; but is more efficient when calculating both photometry and astrometry. If None, then just use t_obs.

## Returns

mag_model
[array_like] Magnitude of the centroid at t_obs.
animate (tE, time_steps, frame_time, name, size, zoom, astrometry)
Produces animation of microlensing event. This function takes the PSPL and makes an animation, the input variables are as follows

## Parameters

tE:
number of einstein crossings times before/after the peak you want the animation to plot
e.g tE $=2$ => graph will go from -2 tE to 2 tE
time_steps:
number of time steps before/after peak, so total number of time steps will be 2 times this value

## frame_time:

times in ms of each frame in the animation
name: string
the animation will be saved as name.html
size: list
[horizontal, vertical] cm's
zoom:
\# of einstein radii plotted in vertical direction

## get_astrometry_unlensed $(t)$

Outputs position of source unlensed.
Input a list of times and it will output the position of the source had it not been lensed at each of the times in the list
e.g if $n=4$, and say $v=[1,0] \&$ the times are $[0,1,2]$ in years.

This will return
$(((1,0),(0,1),(-1,0),(\theta,-1)),((2,0),(1,1),(0,0),(1,-1)),($
$(3,0),(2,1),(1,0),(2,-1))) \ldots$
$=($ positions at $\mathrm{t}=0),($ positions at $\mathrm{t}=1),($ positions at $\mathrm{t}=2)$
so np. array (positions) is an array which contains an array for each time step with the positions of all the points on the boundary of the source.
get_source_outline_astrometry ( $r, n$, center )
Return astrometric points that outline the outer circumference of the source star.

The outline is described as a circle of radius self.radius and is evaluated at self.n_outline number of points.
takes in the radius of the circle, centre position and number of points we are approximating the circle by and returns a numpy array of positions
e.g: $(((1,0),(0,1),(-1,0),(0,-1)))$ if $n=4$ and radius $=1$

## Returns

source_points
[numpy array] Returns an array of shape = [2, self.n_outline, len(time)]

### 8.2 FSPL Limb Classes

## class model.FSPL_Limb

Bases: FSPL

## Methods

| animate(crossings, time_steps, frame_time, ...) | Produces animation of microlensing event. |
| :--- | :--- |
| get_photometry(t_obs[, filt_idx, amp_arr, ...]) | Get the photometry for each of the lensed source im- <br> ages. |
| get_source_outline_astrometry(r, n, center) | Return astrometric points that outline the outer cir- <br> cumference of the source star. |


| F |  |
| :--- | :--- |
| get_amplification |  |
| get_chi2_photometry |  |
| get_lnL_constant |  |
| log_likely_photometry |  |
| log_likely_photometry_each |  |

get_photometry (t_obs, filt_idx=0, amp_arr=None, print_warning=True)
Get the photometry for each of the lensed source images.

## Parameters

## t_obs

[array_like] Array of times to model.

## Returns

mag_model
[array_like] Magnitude of the centroid at t_obs.
animate(crossings, time_steps, frame_time, name, size, zoom)
Produces animation of microlensing event. This function takes the PSPL and makes an animation, the input variables are as follows

## Parameters

tE:
number of einstein crossings times before/after the peak you want the animation to plot
e.g tE $=2=>$ graph will go from -2 tE to 2 tE
time_steps:
number of time steps before/after peak, so total number of time steps will be 2 times this value
frame_time:
times in ms of each frame in the animation
name: string
the animation will be saved as name.html
size: list
[horizontal, vertical] cm's
zoom:
\# of einstein radii plotted in vertical direction
get_source_outline_astrometry $(r, n$, center $)$
Return astrometric points that outline the outer circumference of the source star.

The outline is described as a circle of radius self.radius and is evaluated at self.n_outline number of points.
takes in the radius of the circle, centre position and number of points we are approximating the circle by and returns a numpy array of positions
e.g: $(((1,0),(0,1),(-1,0),(0,-1)))$ if $n=4$ and radius $=1$

## Returns

source_points
[numpy array] Returns an array of shape = [2, self.n_outline, len(time)]
class model.FSPL_Limb_noParallax
Bases: FSPL_noParallax

## Methods

| get_amplification(t) | noParallax: Get the photometric amplification term <br> at a set of times, t. |
| :--- | :--- |
| get_astrometry(t_obs[, ast_filt_idx]) | noParallax: Position of the observed source position <br> in arcsec. |
| get_astrometry_unlensed(t_obs) | noParallax: Get the astrometry of the source if the <br> lens didn't exist. |
| get_centroid_shift(t) | noParallax: Get the centroid shift (in mas) for a list <br> of observation times (in MJD). |
| get_lens_astrometry(t_obs) | Equation of motion for just the foreground lens. |
| get_resolved_amplification(t) | Get the photometric amplification term at a set of <br> times, t for both the plus and minus images. |
| get_resolved_astrometry(t_obs) | Get the $x, y$ astrometry for each of the two source <br> images, which we label plus and minus. |

## calc_piE_ecliptic

## get_amplification $(t)$

noParallax: Get the photometric amplification term at a set of times, t .

## Parameters

t:
Array of times in MJD.DDD
get_astrometry (t_obs, ast_filt_idx=0)
noParallax: Position of the observed source position in arcsec.
get_astrometry_unlensed (t_obs)
noParallax: Get the astrometry of the source if the lens didn't exist.

## Returns

xS_unlensed
[numpy array, dtype=float, shape $=$ len(t_obs) x 2] The unlensed positions of the source in arcseconds.

## get_centroid_shift $(t)$

noParallax: Get the centroid shift (in mas) for a list of observation times (in MJD).

```
get_lens_astrometry(t_obs)
```

Equation of motion for just the foreground lens.

## Parameters

t_obs
[array_like] Time (in MJD).
get_resolved_amplification( $t$ )
Get the photometric amplification term at a set of times, t for both the plus and minus images.

## Parameters

t:
Array of times in MJD.DDD

## get_resolved_astrometry (t_obs)

Get the x , y astrometry for each of the two source images, which we label plus and minus.

## Returns

[xS_plus, xS_minus]
[list of numpy arrays]

- xS_plus is the vector position of the plus image in arcsec
- xS_minus is the vector position of the plus image in arcsec


## class model.FSPL_Limb_Parallax

Bases: FSPL_Parallax

## Methods

| calc_piE_ecliptic() | Parallax: Get piE_ecliptic |
| :--- | :--- |
| get_amplification(t) | Parallax: Get the photometric amplification term at a <br> set of times,. |
| get_astrometry(t_obs[, ast_filt_idx]) | Parallax: Get astrometry |
| get_astrometry_unlensed(t_obs) | Get the astrometry of the source if the lens didn't ex- <br> ist. |
| get_centroid_shift(t[, ast_filt_idx]) | Parallax: Get the centroid shift (in mas) for a list of <br> observation times (in MJD). |
| get_geoproj_ast_params(t0par) |  |
| get_geoproj_params(t0par) | Parallax: Get lens astrometry <br> set of times, the for both the plus and minus images. |
| get_lens_astrometry(t_obs) | Parallax: Get the x, y astrometry for each of the two <br> source images, which we label plus and minus. |
| get_resolved_amplification(t) |  |

## start

## calc_piE_ecliptic()

Parallax: Get piE_ecliptic

## get_amplification $(t)$

Parallax: Get the photometric amplification term at a set of times, t .

## Parameters

t:
Array of times in MJD.DDD
get_astrometry (t_obs, ast_filt_idx=0)
Parallax: Get astrometry
get_astrometry_unlensed(t_obs)
Get the astrometry of the source if the lens didn't exist.

## Returns

## xS_unlensed

[numpy array, dtype=float, shape $=$ len(t_obs) x 2] The unlensed positions of the source in arcseconds.
get_centroid_shift ( $t$, ast_fil_id $x=0$ )
Parallax: Get the centroid shift (in mas) for a list of observation times (in MJD).
get_lens_astrometry (t_obs)
Parallax: Get lens astrometry

## get_resolved_amplification $(t)$

Parallax: Get the photometric amplification term at a set of times, t for both the plus and minus images.

## Parameters

t:
Array of times in MJD.DDD
get_resolved_astrometry (t_obs)
Parallax: Get the x , y astrometry for each of the two source images, which we label plus and minus.

## Returns

[xS_plus, xS_minus]
[list of numpy arrays]

- xS_plus is the vector position of the plus image.
- xS_minus is the vector position of the plus image.
class model.FSPL_Limb_PhotAstromParam1 (lens_mass, $t 0, x S 0$, beta, muL, muS, $d L, d S, n$, radius, utilde, $n r$, mag_src)
Bases: PSPL_Param


## GENERAL USE AND SHARED FUNCTIONS

### 9.1 Shared Functions

model.u0_hat_from_thetaE_hat (thetaE_hat, beta)
Calculate the closest approach vector direction. Define the beta sign convention as Andy Gould does with

- beta $>0$ means u0_E $>0$
- u0_amp > 0 mean u0_E > 0

See Gould 2004, pg 320, bottom right
$u 0>0 \rightarrow$ lens passes to the right side of the source as seen from Earth
theta $X 0=x S 0-x L 0=u 0 *$ theta $E$
which implies that:

- u0_E > 0 for $\mathrm{u} 0>0$
- u0_E < 0 for $\mathrm{u} 0<0$
which is what we use.
model.parallax_in_direction ( $R A$, Dec, mjd)
Memoized version of parallax_in_direction(RA, Dec, mjd)
R.A. in degrees. (J2000)

Dec. in degrees. (J2000)
MJD

Equations following MulensModel.
model.dparallax_dt_in_direction ( $R A, D e c, m j d$ )
R.A. in degrees. (J2000) Dec. in degrees. (J2000) MJD

Equations following MulensModel. Time derivative $->$ units are $\mathrm{yr}^{\wedge}-1$
model.sun_position (mjd, radians=False)

## NAME:

SUNPOS
PURPOSE:
To compute the RA and Dec of the Sun at a given date.

## INPUTS:

mjd - The modified Julian date of the day (and time), scalar or vector
OUTPUTS:
ra:

The right ascension of the sun at that date in DEGREES
double precision, same number of elements as jd
dec:
The declination of the sun at that date in DEGREES
elong:
Ecliptic longitude of the sun at that date in DEGREES.
obliquity:
the obliquity of the ecliptic, in DEGREES

## OPTIONAL INPUT KEYWORD:

RADIAN [def=False] - If this keyword is set to True, then all output variables are given in Radians rather than Degrees

## NOTES:

Patrick Wallace (Rutherford Appleton Laboratory, UK) has tested the accuracy of a C adaptation of the sunpos.pro code and found the following results. From 1900-2100 SUNPOS gave 7.3 arcsec maximum error, 2.6 arcsec RMS. Over the shorter interval 1950-2050 the figures were $6.4 \operatorname{arcsec}$ max, 2.2 arcsec RMS.

The returned RA and Dec are in the given date's equinox.
Procedure was extensively revised in May 1996, and the new calling sequence is incompatible with the old one.

## METHOD:

Uses a truncated version of Newcomb's Sun. Adapted from the IDL routine SUN_POS by CD Pike, which was adapted from a FORTRAN routine by B. Emerson (RGO).

## EXAMPLE:

(1) Find the apparent RA and Dec of the Sun on May 1, 1982

IDL> jdcnv, 1982, 5, 1,0 ,jd ;Find Julian date jd $=2445090.5$
IDL> sunpos, jd, ra, dec
IDL> print,adstring(ra,dec,2)
$023132.61+145434.9$

The Astronomical Almanac gives $023132.58+145434.9$ so the error in SUNPOS for this case is $<0.5$ ".
(2) Find the apparent RA and Dec of the Sun for every day in 1997

IDL> jdenv, 1997,1,1,0, jd ;Julian date on Jan 1, 1997
IDL> sunpos, jd+ dindgen(365), ra, dec ;RA and Dec for each day

MODIFICATION HISTORY:

- Written by Michael R. Greason, STX, 28 October 1988.
- Accept vector arguments, W. Landsman - April, 1989
- Eliminated negative right ascensions - MRG, Hughes STX, 6 May 1992.
- Rewritten using the 1993 Almanac. Keywords added. MRG, HSTX, 10 February 1994.
- Major rewrite, improved accuracy, always return values in degrees - W. Landsman May, 1996
- Added /RADIAN keyword; W. Landsman; August, 1997
- Converted to IDL V5.0; W. Landsman; September 1997
- Converted to python; J. R. Lu; August 2016
model.get_angular_einstein_radius $(m, d 1, d 2)$
model.get_unit_vector $(x)$
model.get_u $\boldsymbol{O}\left(t h e t a E \_h a t\right.$, beta, thetaE_amp)
model.get_uhat(thetaE_hat, beta)
model.get_einstein_time(theta, $v$, days)
model.get_thetas(source, lens)
model.get_amplitudes(vectors)
model.get_unit_vectors(vectors)
model.get_plus(amps, hats, pos, lens, radius)
model.get_minus(amps, hats, pos, lens, radius)
model. oned_int(centre, function1, function2, ymax, ymin, $n, x$, middle, centres)
model.twod_int (centre, function1, function2, xmax, xmin, ymax, ymin, $n x, n y$, middle, centres)
model.twod_cent_x_int (centre, function1, function2, xmax, xmin, ymax, ymin, nx, ny, middle, centres)
model.oned_x_int(centre, functionl, function2, ymax, ymin, $n, x$, middle, centres)
model.twod_cent_y_int(centre, function1, function2, xmax, xmin, ymax, ymin, $n x$, ny, middle, centres)
model. oned_y_int(centre, function1, function2, ymax, ymin, $n, x$, middle, centres)
model.get_image $(y 0, m 1, d, R)$
Function to find the images of the star


## Parameters

y 0 :
position of the cente of the source star, in units of anguler Einstein radius
m 1 :
Mass of rightmost lens divided by the total mass
d:
separation of the lenses in angular Einstein radii
R:
angular radius of the source in angular Einstein radii

## MODEL FITTER

### 10.1 PSPL_Solver

class model_fitter.PSPL_Solver(data, model_class, custom_additional_param_names=None, add_error_on_photometry=False, multiply_error_on_photometry=False, use_phot_optional_params=True, use_ast_optional_params=True, wrapped_params=None, importance_nested_sampling=False, multimodal=True, const_efficiency_mode=False, n_live_points=300, evidence_tolerance $=0.5$, sampling_efficiency $=0.8$, n_iter_before_update $=100$, null_log_evidence $=-1 e+90$, max_modes $=100$, mode_tolerance $=-1 e+90$, outputfiles_basename $=$ 'chains $/ 1-$ ', seed $=-1$, verbose $=$ False, resume $=$ False, context $=0$, write_output=True, $l o g \_z e r o=-1 e+100$, max_iter $=0$, init_MPI=False, dump_callback='default')
Bases: Solver
A PyMultiNest solver to find the optimal PSPL parameters, given data and a microlensing model from model.py. DESPITE THE NAME YOU CAN ALSO USE IT TO FIT PSBL!

## Examples

Assuming that a data dictionary has been instantiated with the above keys, and that a model has been loaded in from model.py, PSPL_Solver can be run with the following commands: .. code:

```
fitter = PSPL_Solver(data,
    PSPL_PhotAstrom_Par_Param1,
    add_error_on_photometry=True,
    custom_additional_param_names=['dS', 'tE'],
    outputfiles_basename='./model_output/test_')
fitter.solve()
```


## Attributes

## data

[dictionary] Observational data used to fit a microlensing model. What the data must contain depends on what type of microlensing model you are solving for.
The data dictionary must always photometry information of at least one filter. This data must contain the times, magnitudes, and magnitude errors of the observations. The keys to these arrays are:

- t_photl (MJD)
- mag1 (magnitudes)
- mag_errl (magnitudes)

PSPL_Solver supports multiple photometric filters. For each additional filter, increments the extension of the above keys by one. For example, a second filter would be:

- t_phot2 (MJD)
- mag2 (magnitudes)
- mag_err2 (magnitudes)

PSPL_Solver supports solving microlensing models that calculate with parallax. These models must be accompanied with data that contains the right ascenscion and declination of the target. These keys are:

- raL (decimal degrees)
- decL (decimal degrees)

PSPL_Solver supports solving microlensing models that fit astrometry. These models must be accompanied with data that contains astrometric observations in the following keys:

- t_ast (MJD)
- xpos (arcsec along East-West increasing to the East)
- ypos (arcsec along the North-South increasing to the North)
- xpos_err (arcsec)
- ypos_err (arcsec)


## model_class

PSPL_Solver must be provided with the microlensing model that you are trying to fit to your data. These models are written out in model.py, along with extensive documentation as to their content and construction in the file's docstring. The model can support either

1. photometric data or photometric and astrometric data,
2. parallax or no parallax, and
3. different parameterizations of the model.

For example, a model with accepts both astrometric and photometric data, uses parallax, and uses a parameterization that includes the distance to the source and the lens is: PSPL_PhotAstrom_Par_Param1.
custom_additional_param_names
[list, optional] If provided, the fitter will override the default additional_param_names of the model_class. These are the parameters, besides those that are being fitted for, that are written out to disk for posterior plotting after the fit has completed. To see the default additional_param_names run:

```
print(model_class.additional _param_names)
```

add_error_on_photometry
[boolean, optional] If set to True, the fitter will fit for an additive error to the photometric magnitudes in the fitting process. This error will have the name $a d d \_e r r N$, with an $N$ equal to the filter number.
multiply_error_on_photometry
[boolean, optional] If set to True, the fitter will fit for a multiplicative error to the photometric
magnitudes in the fitting process. This error will have the name mult_err $N$, with an $N$ equal to the filter number.

## All other parameters

See pymultinest.run() for a description of all other parameters.

## Methods

| LogLikelihood(cube[, ndim, n_params]) | This is just a wrapper because PyMultinest requires <br> passing in the ndim and nparams. |
| :--- | :--- |
| Prior(cube[, ndim, nparams]) |  |
| Prior_from_post(cube[, ndim, nparams]) | Get the bin midpoints |
| calc_best_fit(tab, smy[, s_idx, def_best]) | Returns best-fit parameters, where best-fit can be me- <br> dian, maxl, or MAP. |
| calc_chi2([params, verbose]) | Parameters |

calc_chi2_manual([params, verbose])

## Parameters

| get_best_fit([def_best]) | Returns best-fit parameters, where best-fit can be me- <br> dian, maxl, or MAP. |
| :--- | :--- |
| get_best_fit_model([def_best]) | Identify best-fit model |
| get_best_fit_modes([def_best]) | Returns a list of best-fit parameters, where best-fit can <br> be median, maxl, or MAP. |
| load_mnest_modes([remake_fits]) | Load up the separate modes results into an astropy <br> table. |
| load_mnest_modes_results_for_dynesty([...]) | Make a Dynesty-style results object that can be used <br> in the nicer plotting codes. |
| load_mnest_results([remake_fits]) | Load up the MultiNest results into an astropy table. |
| load_mnest_results_for_dynesty([remake_fits]) | Make a Dynesty-style results object that can be used <br> in the nicer plotting codes. |
| load_mnest_summary([remake_fits]) | Load up the MultiNest results into an astropy table. |
| log_likely(cube[, verbose]) |  |

## Parameters

| make_default_priors() | Setup our prior distributions (i.e. |
| :--- | :---: |
| plot_dynesty_style([sim_vals, fit_vals, ...]) |  |

## Parameters

| plot_model_and_data(model[, input_model, ...]) | Make and save the model and data plots. |
| :--- | :--- |
| plot_model_and_data_modes([def_best]) | Plots photometry data, along with n random draws <br> from the posterior. |
| print_likelihood([params, verbose]) | Randomly sample from a multinest posterior distri- <br> bution. |
| sample_post(binmids, cdf, bininds) | Reads in the fits for the different modes <br> (post_separate.dat) and splits it into a .dat file <br> per mode. |
| separate_modes() | Run a MultiNest fit to find the optimal parameters <br> (and their posteriors) given the data. |
| solve() | Write a YAML file that contains the parameters to re- <br> initialize this object, if desired. |
| write_params_yaml() |  |


| Prior_copy |  |
| :--- | :--- |
| callback_plotter |  |
| check_data |  |
| dyn_log_likely |  |
| dyn_prior |  |
| get_best_fit_modes_model |  |
| get_model |  |
| get_modified_mag_err |  |
| log_likely_astrometry |  |
| log_likely_photometry |  |
| setup_params |  |
| summarize_results |  |
| summarize_results_modes |  |
| write_summary_maxL |  |

## make_default_priors()

Setup our prior distributions (i.e. random samplers). We will draw from these in the Prior() function. We set them up in advance because they depend on properties of the data. Also, they can be over-written by custom priors as desired.
To make your own custom priors, use the make_gen() functions with different limits.
Prior_from_post (cube, ndim=None, nparams=None)
Get the bin midpoints
sample_post (binmids, $c d f$, bininds)
Randomly sample from a multinest posterior distribution.

## Parameters

Nparams:
number of parameters
Nbins:
number of histogram bins per dimension

## Nnzero:

number of histogram bins with non-zero probability

## binmids

[list of length $N$, each list entry is an array of shape ( $M$,$) ] The centers of the bins for each$ parameter

## cdf

[(Nnzero, ) array] CDF of the distribution. Only the non-zero probability entries.

## bininds

[(Nnzero, Nparams) array] Histogram indices of the non-zero probability entries.
LogLikelihood (cube, ndim=None, n_params=None)
This is just a wrapper because PyMultinest requires passing in the ndim and nparams.
log_likely (cube, verbose=False)

## Parameters

cube
[list or dict] The dictionary or cube of the model parameters.

```
write_params_yaml()
```

Write a YAML file that contains the parameters to re-initialize this object, if desired.

```
solve()
```

Run a MultiNest fit to find the optimal parameters (and their posteriors) given the data.
Note we will ALWAYS tell multinest to be verbose.

```
separate_modes()
```

Reads in the fits for the different modes (post_separate.dat) and splits it into a dat file per mode.
Is there a more intelligent way to deal with all the indices??? Write better later, but it seems to work for now...
calc_best_fit (tab, smy, s_idx=0, def_best='maxl')
Returns best-fit parameters, where best-fit can be median, maxl, or MAP. Default is maxl.
If best-fit is median, then also return $+/-1$ sigma uncertainties.
If best-fit is MAP, then also need to indicate which row of summary table to use. Default is $s_{-} i d x=0$ (global solution). $s_{-} i d x=1,2, \ldots, n$ for the n different modes.
tab $=$ self.load_mnest_results() smy $=$ self.load_mnest_summary()

```
get_best_fit(def_best='maxl')
```

Returns best-fit parameters, where best-fit can be median, maxl, or MAP. Default is maxl.
If def_best is median, then also return $+/-1$ sigma uncertainties.

## Returns

Either a dicitonary or a tuple of length=2 holding two dictionaries, one for values and one for uncertainty ranges. See calc_best_fit() for details.
get_best_fit_modes(def_best='maxl')
Returns a list of best-fit parameters, where best-fit can be median, maxl, or MAP. Default is maxl.
If def_best is median, then also return $+/-1$ sigma uncertainties.

## Returns

Either a list of dicitonaries or a list where each entry is a tuple of length $=\mathbf{2}$ holding two dictionaries, one for values and one for uncertainty ranges.
See calc_best_fit() for details.

```
get_best_fit_model(def_best='maxl')
```

Identify best-fit model

## Parameters

## def_best

[str] Choices are 'map' (maximum a posteriori), 'median', or 'maxl' (maximum likelihood)
load_mnest_results(remake_fits=False)
Load up the MultiNest results into an astropy table.
load_mnest_summary (remake_fits=False)
Load up the MultiNest results into an astropy table.

## load_mnest_modes(remake_fits=False)

Load up the separate modes results into an astropy table.
load_mnest_results_for_dynesty (remake_fits=False)
Make a Dynesty-style results object that can be used in the nicer plotting codes.

## load_mnest_modes_results_for_dynesty(remake_fits=False)

Make a Dynesty-style results object that can be used in the nicer plotting codes.
plot_dynesty_style(sim_vals=None, fit_vals=None, remake_fits=False, dims=None, traceplot=True, cornerplot=True, $k d e=$ True)

## Parameters

sim_vals
[dict] Dictionary of simulated input or comparison values to overplot on posteriors.
fit_vals
[str] Choices are 'map' (maximum a posteriori), 'mean', or 'maxl' (maximum likelihood)
plot_model_and_data(model, input_model=None, mnest_results=None, suffix=", zoomx=None, zoomy=None, zoomy_res=None, fitter $=$ None, N_traces $=50$ )
Make and save the model and data plots.
zoomx, xoomy, zoomy_res : list the same length as self.n_phot_sets Each entry of the list is a list [a,b] cooresponding to the plot limits

```
plot_model_and_data_modes(def_best='maxl')
```

Plots photometry data, along with n random draws from the posterior.
print_likelihood(params='best', verbose=True)

## Parameters

## model_params

[str or dict, optional]
model_params = 'best' will load up the best solution and calculate
the chi^ ${ }^{\wedge}$ based on those values. Alternatively, pass in a dictionary with the model parameters to use.
calc_chi2 $($ params='best', verbose $=$ False $)$

## Parameters

params
[str or dict, optional] model_params = 'best' will load up the best solution and calculate the chi^2 based on those values. Alternatively, pass in a dictionary with the model parameters to use.
calc_chi2_manual (params='best', verbose=False)

## Parameters

params
[str or dict, optional] model_params = 'best' will load up the best solution and calculate the chi^ $^{\wedge} 2$ based on those values. Alternatively, pass in a dictionary with the model parameters to use.
class model_fitter.PSPL_Solver_weighted(data, model_class, custom_additional_param_names=None, add_error_on_photometry=False, multiply_error_on_photometry=False, use_phot_optional_params=True, use_ast_optional_params=True, wrapped_params=None, importance_nested_sampling=False, multimodal=True, const_efficiency_mode $=$ False, $n \_l i v e \_p o i n t s=300$, evidence_tolerance $=0.5$, sampling_efficiency $=0.8$, n_iter_before_update $=100$, null_log_evidence $=-1 e+90$, max_modes $=100$, mode_tolerance $=-1 e+90$, outputfiles_basename='chains/l-', seed=-1, verbose $=$ False, resume $=$ False, context $=0$, write_output=True, log_zero $=-1 e+100$, max_iter $=0$, init_MPI=False, dump_callback=None, weights='phot_ast_equal')
Bases: PSPL_Solver
Soliver where the likelihood function has each data set weigthed equally (i.e. not the natural weighting by the number of points; but rather each contributes $1 / n \_k$ where $n$ is the number of data points and $k$ is the data set.

## Methods

| LogLikelihood(cube[, ndim, n_params]) | This is just a wrapper because PyMultinest requires passing in the ndim and nparams. |
| :---: | :---: |
| Prior(cube[, ndim, nparams]) |  |
| Prior_from_post(cube[, ndim, nparams]) | Get the bin midpoints |
| calc_best_fit(tab, smy[, s_idx, def_best]) | Returns best-fit parameters, where best-fit can be median, maxl, or MAP. |
| calc_chi2([params, verbose]) Parameters |  |
|  |  |
| calc_chi2_manual([params, verbose]) |  |
|  | Parameters |
| calc_weights(weights) | order of weight_arr is `[phot_1, phot_2,. |
| get_best_fit([def_best]) | Returns best-fit parameters, where best-fit can be median, maxl, or MAP. |
| get_best_fit_model([def_best]) | Identify best-fit model |
| get_best_fit_modes([def_best]) | Returns a list of best-fit parameters, where best-fit can be median, maxl, or MAP. |
| load_mnest_modes([remake_fits]) | Load up the separate modes results into an astropy table. |
| load_mnest_modes_results_for_dynesty([...]) | Make a Dynesty-style results object that can be used in the nicer plotting codes. |
| load_mnest_results([remake_fits]) | Load up the MultiNest results into an astropy table. |
| load_mnest_results_for_dynesty([remake_fits]) | Make a Dynesty-style results object that can be used in the nicer plotting codes. |
| load_mnest_summary([remake_fits]) | Load up the MultiNest results into an astropy table. |
| log_likely(cube[, verbose]) |  |
|  | Parameters |
| make_default_priors() | Setup our prior distributions (i.e. |
| plot_dynesty_style([sim_vals, fit_vals, ...]) |  |
|  | Parameters |
| plot_model_and_data(model[, input_model, ...]) | Make and save the model and data plots. |
| plot_model_and_data_modes([def_best]) | Plots photometry data, along with n random draws from the posterior. |
| print_likelihood([params, verbose]) |  |
|  | Parameters |
| sample_post(binmids, cdf, bininds) | Randomly sample from a multinest posterior distribution. |
| separate_modes() | Reads in the fits for the different modes (post_separate.dat) and splits it into a .dat file per mode. |
| solve() | Run a MultiNest fit to find the optimal parameters (and their posteriors) given the data. |
| write_params_yaml() | Write a YAML file that contains the parameters to reinitialize this object, if desired. |

| Prior_copy |  |
| :--- | :--- |
| callback_plotter |  |
| check_data |  |
| dyn_log_likely |  |
| dyn_prior |  |
| get_best_fit_modes_model |  |
| get_model |  |
| get_modified_mag_err |  |
| log_likely_astrometry |  |
| log_likely_photometry |  |
| setup_params |  |
| summarize_results |  |
| summarize_results_modes |  |
| write_summary_maxL |  |

## calc_weights(weights)

order of weight_arr is [phot_1, phot_2, . . phot_n, ast_1, ast_2, ... ast_n]
LogLikelihood(cube, ndim=None, n_params=None)
This is just a wrapper because PyMultinest requires passing in the ndim and nparams.

```
Prior_from_post(cube, ndim=None, nparams=None)
```

Get the bin midpoints

```
calc_best_fit(tab, smy, s_idx=0, def_best='maxl')
```

Returns best-fit parameters, where best-fit can be median, maxl, or MAP. Default is maxl.
If best-fit is median, then also return +/- 1 sigma uncertainties.
If best-fit is MAP, then also need to indicate which row of summary table to use. Default is $s_{-} i d x=0$ (global solution). $s_{-} i d x=1,2, \ldots, n$ for the n different modes.
tab $=$ self.load_mnest_results() smy $=$ self.load_mnest_summary()
calc_chi2 (params='best', verbose=False)

## Parameters

 params[str or dict, optional] model_params = 'best' will load up the best solution and calculate the chi^2 based on those values. Alternatively, pass in a dictionary with the model parameters to use.
calc_chi2_manual (params='best', verbose=False)

## Parameters

## params

[str or dict, optional] model_params = 'best' will load up the best solution and calculate the chi^2 based on those values. Alternatively, pass in a dictionary with the model parameters to use.
get_best_fit(def_best='maxl')
Returns best-fit parameters, where best-fit can be median, maxl, or MAP. Default is maxl.
If def_best is median, then also return +/- 1 sigma uncertainties.

## Returns

Either a dicitonary or a tuple of length $=2$ holding two dictionaries, one for values and one for uncertainty ranges.
See calc_best_fit() for details.
get_best_fit_model (def_best='maxl')
Identify best-fit model

## Parameters

def_best
[str] Choices are 'map' (maximum a posteriori), 'median', or 'maxl' (maximum likelihood)
get_best_fit_modes(def_best='maxl')
Returns a list of best-fit parameters, where best-fit can be median, maxl, or MAP. Default is maxl.
If def_best is median, then also return +/- 1 sigma uncertainties.

## Returns

Either a list of dicitonaries or a list where each entry is
a tuple of length $=\mathbf{2}$ holding two dictionaries, one for values
and one for uncertainty ranges.
See calc_best_fit() for details.

## load_mnest_modes(remake_fits=False)

Load up the separate modes results into an astropy table.
load_mnest_modes_results_for_dynesty (remake_fits=False)
Make a Dynesty-style results object that can be used in the nicer plotting codes.

## load_mnest_results(remake_fits=False)

Load up the MultiNest results into an astropy table.

```
load_mnest_results_for_dynesty(remake_fits=False)
```

Make a Dynesty-style results object that can be used in the nicer plotting codes.

## load_mnest_summary (remake_fits=False)

Load up the MultiNest results into an astropy table.

```
log_likely(cube, verbose=False)
```


## Parameters

## cube

[list or dict] The dictionary or cube of the model parameters.
make_default_priors()
Setup our prior distributions (i.e. random samplers). We will draw from these in the Prior() function. We set them up in advance because they depend on properties of the data. Also, they can be over-written by custom priors as desired.

To make your own custom priors, use the make_gen() functions with different limits.
plot_dynesty_style(sim_vals=None, fit_vals=None, remake_fits=False, dims=None, traceplot=True, cornerplot=True, $k d e=$ True)

## Parameters

sim_vals
[dict] Dictionary of simulated input or comparison values to overplot on posteriors.

## fit_vals

[str] Choices are 'map' (maximum a posteriori), 'mean', or 'maxl' (maximum likelihood)
plot_model_and_data(model, input_model=None, mnest_results=None, suffix=", zoomx=None, zoomy=None, zoomy_res=None, fitter=None, N_traces=50)
Make and save the model and data plots.
zoomx, xoomy, zoomy_res : list the same length as self.n_phot_sets Each entry of the list is a list $[a, b]$ cooresponding to the plot limits

```
plot_model_and_data_modes(def_best='maxl')
```

Plots photometry data, along with n random draws from the posterior.

```
print_likelihood(params='best', verbose=True)
```


## Parameters

## model_params

[str or dict, optional]
model_params = 'best' will load up the best solution and calculate the chi^2 based on those values. Alternatively, pass in a dictionary with the model parameters to use.
sample_post (binmids, cdf, bininds)
Randomly sample from a multinest posterior distribution.

## Parameters

## Nparams:

number of parameters

## Nbins:

number of histogram bins per dimension

## Nnzero:

number of histogram bins with non-zero probability

## binmids

[list of length N , each list entry is an array of shape ( $\mathrm{M}, \mathrm{)}$ ] The centers of the bins for each parameter

## cdf

[(Nnzero, ) array] CDF of the distribution. Only the non-zero probability entries.
bininds
[(Nnzero, Nparams) array] Histogram indices of the non-zero probability entries.

## separate_modes()

Reads in the fits for the different modes (post_separate.dat) and splits it into a dat file per mode.
Is there a more intelligent way to deal with all the indices??? Write better later, but it seems to work for now...
solve()
Run a MultiNest fit to find the optimal parameters (and their posteriors) given the data.
Note we will ALWAYS tell multinest to be verbose.
write_params_yaml()
Write a YAML file that contains the parameters to re-initialize this object, if desired.
class model_fitter.PSPL_Solver_Hobson_Weighted (data, model_class,
custom_additional_param_names=None, add_error_on_photometry=False, multiply_error_on_photometry=False, use_phot_optional_params=True, use_ast_optional_params=True, wrapped_params=None, importance_nested_sampling=False, multimodal=True, const_efficiency_mode=False, n_live_points $=300$, evidence_tolerance $=0.5$, sampling_efficiency=0.8, n_iter_before_update $=100$, null_log_evidence $=-1 e+90$, max_modes $=100$, mode_tolerance $=-1 e+90$, outputfiles_basename $=$ 'chains $/ 1-$-', seed $=-1$, verbose $=$ False, resume $=$ False, context $=0$, write_output=True, log_zero=-le+100, max_iter=0, init_MPI=False, dump_callback='default')

## Methods

| LogLikelihood(cube[, ndim, n_params]) | This is just a wrapper because PyMultinest requires <br> passing in the ndim and nparams. |
| :--- | :--- |
| Prior(cube[, ndim, nparams]) |  |
| Prior_from_post(cube[, ndim, nparams]) | Get the bin midpoints |
| calc_best_fit(tab, smy[, s_idx, def_best]) | Returns best-fit parameters, where best-fit can be me- <br> dian, maxl, or MAP. |
| calc_chi2([params, verbose]) | Parameters |

calc_chi2_manual([params, verbose])

## Parameters

| get_best_fit([def_best]) | Returns best-fit parameters, where best-fit can be me- <br> dian, maxl, or MAP. |
| :--- | :--- |
| get_best_fit_model([def_best]) | Identify best-fit model |
| get_best_fit_modes([def_best]) | Returns a list of best-fit parameters, where best-fit can <br> be median, maxl, or MAP. |
| get_hobson_effective_weights(cube) | Return the effective weights, alpha_k, for each data <br> set. |

hobson_weight_log_likely(ln_prob_dk_giv_ak_1)Implement a data-set-specific weighting scheme by using a hyperparameter, alpha_k, for the kth data set as described in Hobson et al. 2002.

load_mnest_modes([remake_fits]) $\quad$| Load up the separate modes results into an astropy |
| :--- |
| table. |

load_mnest_modes_results_for_dynesty([...]) Make a Dynesty-style results object that can be used in the nicer plotting codes.
load_mnest_results([remake_fits]) Load up the MultiNest results into an astropy table.
load_mnest_results_for_dynesty([remake_fits]) Make a Dynesty-style results object that can be used in the nicer plotting codes.
load_mnest_summary([remake_fits]) Load up the MultiNest results into an astropy table.
log_likely(cube[, verbose]) Compute a log-likelihood where there is a hyperparameter, alpha_k, that controls the weighting between each data k set.
make_default_priors() Setup our prior distributions (i.e.
plot_dynesty_style([sim_vals, fit_vals, ...])

## Parameters

| plot_model_and_data(model[, input_model, ...]) | Make and save the model and data plots. |
| :--- | :--- |
| plot_model_and_data_modes([def_best]) | Plots photometry data, along with n random draws <br> from the posterior. |

print_likelihood([params, verbose])

## Parameters

sample_post(binmids, cdf, bininds) $\quad$| Randomly sample from a multinest posterior distri- |
| :--- |
| bution. |

separate_modes() Reads in the fits for the different modes (post_separate.dat) and splits it into a .dat file per mode.
solve()
Run a MultiNest fit to find the optimal parameters (and their posteriors) given the data.
write_params_yaml() Write a YAML file that contains the parameters to re-

| Prior_copy |  |
| :--- | :--- |
| callback_plotter |  |
| check_data |  |
| dyn_log_likely |  |
| dyn_prior |  |
| get_best_fit_modes_model |  |
| get_model |  |
| get_modified_mag_err |  |
| log_likely_astrometry |  |
| log_likely_photometry |  |
| setup_params |  |
| summarize_results |  |
| summarize_results_modes |  |
| write_summary_maxL |  |

## log_likely (cube, verbose=False)

Compute a log-likelihood where there is a hyperparameter, alpha_k, that controls the weighting between each data k set. This algorithm is described in Hobson et al. 2002.

Specifically, we are implementing Eq. 35.

## Parameters

## cube

[list or dict] The dictionary or cube of the model parameters.
hobson_weight_log_likely (ln_prob_dk_giv_ak_l)
Implement a data-set-specific weighting scheme by using a hyperparameter, alpha_k, for the kth data set as described in Hobson et al. 2002.

Specifically, we are implementing Eq. 16 and 23-27, with the prior described in Eq. 21.
We are not using the simplifications in Section 5 for now.

## get_hobson_effective_weights(cube)

Return the effective weights, alpha_k, for each data set. Photometry first, then astrometry.
LogLikelihood (cube, ndim=None, n_params=None)
This is just a wrapper because PyMultinest requires passing in the ndim and nparams.
Prior_from_post (cube, ndim=None, nparams=None)
Get the bin midpoints
calc_best_fit (tab, smy, s_idx=0, def_best='maxl')
Returns best-fit parameters, where best-fit can be median, maxl, or MAP. Default is maxl.
If best-fit is median, then also return +/- 1 sigma uncertainties.
If best-fit is MAP, then also need to indicate which row of summary table to use. Default is $s_{-} i d x=0$ (global solution). $s_{-} i d x=1,2, \ldots, n$ for the n different modes.
tab $=$ self.load_mnest_results() smy $=$ self.load_mnest_summary()
calc_chi2 $($ params='best', verbose $=$ False $)$

## Parameters

## params

[str or dict, optional] model_params = 'best' will load up the best solution and calculate the chi^ $^{\wedge} 2$ based on those values. Alternatively, pass in a dictionary with the model parameters to use.
calc_chi2_manual (params='best', verbose=False)

## Parameters

params
[str or dict, optional] model_params = 'best' will load up the best solution and calculate the chi^ $^{\wedge} 2$ based on those values. Alternatively, pass in a dictionary with the model parameters to use.

```
get_best_fit(def_best='maxl')
```

Returns best-fit parameters, where best-fit can be median, maxl, or MAP. Default is maxl.
If def_best is median, then also return +/- 1 sigma uncertainties.

## Returns

Either a dicitonary or a tuple of length $=2$ holding two dictionaries, one for values and one for uncertainty ranges. See calc_best_fit() for details.

```
get_best_fit_model(def_best='maxl')
```

Identify best-fit model

## Parameters

## def_best

[str] Choices are 'map' (maximum a posteriori), 'median', or 'maxl' (maximum likelihood)

```
get_best_fit_modes(def_best='maxl')
```

Returns a list of best-fit parameters, where best-fit can be median, maxl, or MAP. Default is maxl.
If def_best is median, then also return +/- 1 sigma uncertainties.

## Returns

Either a list of dicitonaries or a list where each entry is a tuple of length $=\mathbf{2}$ holding two dictionaries, one for values and one for uncertainty ranges.
See calc_best_fit() for details.
load_mnest_modes(remake_fits=False)
Load up the separate modes results into an astropy table.

```
load_mnest_modes_results_for_dynesty(remake_fits=False)
```

Make a Dynesty-style results object that can be used in the nicer plotting codes.

## load_mnest_results (remake_fits=False)

Load up the MultiNest results into an astropy table.
load_mnest_results_for_dynesty (remake_fits=False)
Make a Dynesty-style results object that can be used in the nicer plotting codes.
load_mnest_summary (remake_fits=False)
Load up the MultiNest results into an astropy table.

## make_default_priors()

Setup our prior distributions (i.e. random samplers). We will draw from these in the Prior() function. We set them up in advance because they depend on properties of the data. Also, they can be over-written by custom priors as desired.

To make your own custom priors, use the make_gen() functions with different limits.
plot_dynesty_style(sim_vals=None, fit_vals=None, remake_fits=False, dims=None, traceplot=True, cornerplot=True, $k d e=$ True)

## Parameters

sim_vals
[dict] Dictionary of simulated input or comparison values to overplot on posteriors.
fit_vals
[str] Choices are 'map' (maximum a posteriori), 'mean', or 'maxl' (maximum likelihood)
plot_model_and_data (model, input_model=None, mnest_results=None, suffix $=$ ", zoomx=None, zoomy=None, zoomy_res=None, fitter=None, N_traces=50)
Make and save the model and data plots.
zoomx, xoomy, zoomy_res : list the same length as self.n_phot_sets Each entry of the list is a list [a,b] cooresponding to the plot limits

```
plot_model_and_data_modes(def_best='maxl')
```

Plots photometry data, along with n random draws from the posterior.
print_likelihood(params='best', verbose=True)

## Parameters

model_params
[str or dict, optional]
model_params = 'best' will load up the best solution and calculate
the chi^2 based on those values. Alternatively, pass in a dictionary with the model parameters to use.
sample_post (binmids, $c d f$, bininds)
Randomly sample from a multinest posterior distribution.

## Parameters

Nparams:
number of parameters
Nbins:
number of histogram bins per dimension

## Nnzero:

number of histogram bins with non-zero probability

## binmids

[list of length N , each list entry is an array of shape ( M,$)_{\text {] }}$ ] The centers of the bins for each parameter
cdf
[(Nnzero, ) array] CDF of the distribution. Only the non-zero probability entries.
bininds
[(Nnzero, Nparams) array] Histogram indices of the non-zero probability entries.
separate_modes()
Reads in the fits for the different modes (post_separate.dat) and splits it into a dat file per mode.
Is there a more intelligent way to deal with all the indices??? Write better later, but it seems to work for now...
solve()
Run a MultiNest fit to find the optimal parameters (and their posteriors) given the data.
Note we will ALWAYS tell multinest to be verbose.
write_params_yaml()
Write a YAML file that contains the parameters to re-initialize this object, if desired.

### 10.2 Prior Generators

```
model_fitter.make_gen(min, max)
model_fitter.make_norm_gen(mean, std)
model_fitter.make_lognorm_gen(mean, std)
```

Make a natural-log normal distribution for a variable. The specified mean and std should be in the $\ln ()$ space.

```
model_fitter.make_log10norm_gen(mean_in_log10, std_in_log10)
```

Scale scipy lognorm from natural $\log$ to base 10 . Note the mean and std should be in the $\log 10()$ space already.

## Parameters

## mean:

 mean of the underlying $\log 10$ gaussian (i.e. a $\log 10$ quantity)std: variance of underlying $\log 10$ gaussian
model_fitter.make_truncnorm_gen(mean, std, lo_cut, hi_cut)
lo_cut and hi_cut are in the units of sigma
model_fitter.make_truncnorm_gen_with_bounds (mean, std, low_bound, hi_bound)
low_bound and hi_bound are in the same units as mean and std
model_fitter.make_t0_gen ( $t$, mag)
Get an approximate t0 search range by finding the brightest point and then searching days where flux is higher than $80 \%$ of this peak.

```
model_fitter.make_mag_base_gen(mag)
```

Make a prior for baseline magnitude using the data.

```
model_fitter.make_mag_src_gen(mag)
```

Make a prior for source magnitude using the data. Allow negative blending.

```
model_fitter.make_xSO_gen(pos)
model_fitter.make_xSO_norm_gen(pos)
```

model_fitter.make_muS_EN_gen (t, pos, scale_factor=100.0)
Get an approximate muS search range by looking at the best fit straight line to the astrometry. Then allows lots of free space.

## Returns

gen: uniform generator for velocity in mas/yr
model_fitter.make_muS_EN_norm_gen( $t$, pos, $n \_u s e=$ None, scale_factor=10.0)
Get an approximate muS search range by looking at the best fit straight line to the astrometry. Then allows lots of free space.

## Parameters

t: array of times in days
pos:
array of positions in arcsec

## Returns

gen:
uniform generator for velocity in mas/yr
model_fitter.make_invgamma_gen (t_arr)
ADD DESCRIPTION

## Parameters

t_arr:
time array
model_fitter.compute_invgamma_params (t_arr)

Based on function of same name from
Fran Bartolic's caustic package: https://github.com/fbartolic/caustic | Returns parameters of an inverse gamma distribution s.t.

- $1 \%$ of total prob. mass is assigned to values of $t<t_{\text {min }}$ and
- $1 \%$ of total prob. masss to values greater than $t_{-}\{$tmax $\}$.
$t_{\_}\{\min \}$ is defined to be the median spacing between consecutive data points in the time series and $t_{-}\{\max \}$ is the total duration of the time series.


## Parameters

t_arr
[array] Array of times

## Returns

invgamma_a, invgamma_b
[float (?)] The parameters $a, b$ of the inverse gamma function.
model_fitter.make_piS()
model_fitter.make_fdfdt()
model_fitter.random_prob(generator, $x$ )
model_fitter.weighted_quantile(values, quantiles, sample_weight=None, values_sorted=False, old_style=False)
Very close to numplt.percentile, but supports weights.

### 10.2.1 Parameters

values:
numplt.array with data
quantiles:
array-like with many quantiles needed
sample_weight:
array-like of the same length as array
values_sorted: bool,
if True, then will avoid sorting of initial array
old_style:
if True, will correct output to be consistent with numplt.percentile.

## Returns

arr:
numplt.array with computed quantiles.

Notes

Note: quantiles should be in $[0,1]$ !
model_fitter.split_param_filter_index1(s)
Split a parameter name into the <string><number> components where <string> is the parameter name and <number> is the filter index ( 1 -based). If there is no number at the end for a filter index, then return None for the second argument.

## Returns

param_name
[str] The name of the parameter.
filt_index
[int (or None)] The 1-based filter index.
model_fitter.generate_params_dict(params, fitter_param_names)
Take a list, dictionary, or astropy Row of fit parameters and extra parameters and convert it into a well-formed dictionary that can be fed straight into a model object.

The output object will only contain parameters specified by name in fitter_param_names. Multi-filter photometry parameters are treated specially and grouped together into an array such as ['mag_src'] = [mag_src1, mag_src2].

## Parameters

params
[list, dict, Row] Contains values of parameters. Note that if the params are in a list, they need to be in the same order as fitter_param_names. If the params are in a dict or Row, then order is irrelevant.
fitter_param_names
[list] The names of the parameters that will be delivered, in order, in the output.

## Returns

params_dict
[dict] Dictionary of the parameter names and values.

### 10.3 GENERAL USE AND SHARED FUNCTIONS

### 10.3.1 Shared Functions

model_fitter.pointwise_likelihood(data, model, filt_index=0)
Makes some plots to diagnose weirdness in GP fits.
model_fitter.debug_gp_nan(data, model, filt_index=0)
Makes some plots to diagnose weirdness in GP fits.
model_fitter.plot_params(model)
Print parameters
model_fitter.plot_photometry(data, model, input_model=None, dense_time=True, residuals=True, filt_index $=0$, zoomx $=$ None, zoomy=None, zoomy_res=None, mnest_results=None, $N_{-}$traces=50, gp=False, fitter=None)
Get the data out.
model_fitter.plot_photometry_gp(data, model, input_model=None, dense_time=True, residuals=True, filt_index $=0$, zoomx $=$ None, zoomy=None, zoomy_res $=$ None, mnest_results=None, N_traces $^{\prime}=50$, gp $=$ False )
model_fitter.plot_astrometry (data, model, input_model=None, dense_time=True, residuals=True, n_phot_sets $=0$, filt_index $=0$, ast_filt_index $=0$, mnest_results $=$ None, N_traces=50, fitter=None)
Astrometry on the sky
model_fitter.plot_astrometry_on_sky (data, model, ast_filt_index=0)
model_fitter.plot_astrometry_proper_motion_removed(data, model, ast_filt_index=0)
Proper Motion Subtracted
model_fitter.quantiles (mnest_results, sigma=1)
Calculate the median and N sigma credicble interval.

## Parameters

## mnest_results

[astropy table] The table that comes out of load_mnest_results.
sigma
[int, optional] 1, 2, or 3 sigma to determine which credible interval to return.
model_fitter.get_mnest_results(root_name, parameters)

## Parameters

root_name
[str] The directory and base name of the MultiNest output.

## parameters

[list or array] A list of strings with the parameter names to be displayed. There should be one name for each parameter in MultiNest and in the order that they appeared in the hyper-cube.
model_fitter.calc_AIC $(k, \max \log L)$

Calculate Akaike Information Criterion.
$\mathrm{k}=$ number of parameters
max $\log \mathrm{L}=$ maximum $\log$ likelihood
model_fitter.calc_BIC( $n, k, \max \log L)$

Calculate Bayesian Information Criterion.
$\mathrm{n}=$ sample size
$\mathrm{k}=$ number of parameters
$\max \log \mathrm{L}=$ maximum $\log$ likelihood
model_fitter.postplot (results, span=None, quantiles=[0.025, 0.5, 0.975], q_color='gray', smooth=0.02, post_color='blue', post_kwargs=None, $k d e=T r u e, n k d e=1000$, max_n_ticks=5, use_math_text=False, labels=None, label_kwargs=None, show_titles=False, title_fmt $=$ '. $2 f^{\prime}$, title_kwarg $s=$ None, truths $1=$ None, truths $2=$ None, truth_color $1=$ 'red', truth_color $2=$ 'blue', truth_kwargs $1=$ None, truth_kwargs $2=$ None, verbose $=$ False, fig=None)
Plot marginalized posteriors for each parameter. Basically copied half of traceplot.

## Parameters

## results

[Results instance] A Results instance from a nested sampling run. Compatible with results derived from nestle.
span
[iterable with shape (ndim,), optional] A list where each element is either a length-2 tuple containing lower and upper bounds or a float from (O., 1.] giving the fraction of (weighted) samples to include. If a fraction is provided, the bounds are chosen to be equal-tailed. An example would be:

```
span = [(0., 10.), 0.95, (5., 6.)]
```

Default is 0.999999426697 (5-sigma credible interval) for each parameter.

## quantiles

[iterable, optional] A list of fractional quantiles to overplot on the 1-D marginalized posteriors as vertical dashed lines. Default is [0.025, 0.5, 0.975 ] (the $95 \% / 2$-sigma credible interval).

## smooth

[float or iterable with shape (ndim,), optional] The standard deviation (either a single value or a different value for each subplot) for the Gaussian kernel used to smooth the 1-D marginalized posteriors, expressed as a fraction of the span. Default is 0.02 ( $2 \%$ smoothing). If an integer is provided instead, this will instead default to a simple (weighted) histogram with bins=smooth.

## post_color

[str or iterable with shape (ndim,), optional] A ~matplotlib-style color (either a single color or a different value for each subplot) used when plotting the histograms. Default is 'blue'.

## post_kwargs

[dict, optional] Extra keyword arguments that will be used for plotting the marginalized 1-D posteriors.
kde
[bool, optional] Whether to use kernel density estimation to estimate and plot the PDF of the importance weights as a function of log-volume (as opposed to the importance weights themselves). Default is True.
nkde
[int, optional] The number of grid points used when plotting the kernel density estimate. Default is 1000 .

## max_n_ticks

[int, optional] Maximum number of ticks allowed. Default is 5 .

## use_math_text

[bool, optional] Whether the axis tick labels for very large/small exponents should be displayed as powers of 10 rather than using $e$. Default is False.

## labels

[iterable with shape (ndim,), optional] A list of names for each parameter. If not provided, the default name used when plotting will follow $x_{i}$ style.

## label_kwargs

[dict, optional] Extra keyword arguments that will be sent to the ~matplotlib.axes.Axes.set_xlabel and ~matplotlib.axes.Axes.set_ylabel methods.
show_titles
[bool, optional] Whether to display a title above each 1-D marginalized posterior showing the 0.5 quantile along with the upper/lower bounds associated with the 0.025 and 0.975 ( $95 \% / 2$ sigma credible interval) quantiles. Default is True.

## title_fmt

[str, optional] The format string for the quantiles provided in the title. Default is ' $.2 f$ '.

## title_kwargs

[dict, optional] Extra keyword arguments that will be sent to the ~matplotlib.axes.Axes.set_title command.

## truths

[iterable with shape (ndim,), optional] A list of reference values that will be overplotted on the traces and marginalized 1-D posteriors as solid horizontal/vertical lines. Individual values can be exempt using None. Default is None.

## truth_color

[str or iterable with shape (ndim,), optional] A ~matplotlib-style color (either a single color or a different value for each subplot) used when plotting truths. Default is 'red'.

## truth_kwargs

[dict, optional] Extra keyword arguments that will be used for plotting the vertical and horizontal lines with truths.

## verbose

[bool, optional] Whether to print the values of the computed quantiles associated with each parameter. Default is False.
fig
[(~matplotlib.figure.Figure, ~matplotlib.axes.Axes), optional] If provided, overplot the traces and marginalized 1-D posteriors onto the provided figure. Otherwise, by default an internal figure is generated.

## Returns

## traceplot

[(~matplotlib.figure.Figure, ~matplotlib.axes.Axes)] Output trace plot.
model_fitter.cornerplot_2truth(results, dims=None, span=None, quantiles=[0.025, 0.5, 0.975],
color='black', smooth $=0.02$, quantiles_2d=None, hist_kwargs=None, hist2d_kwargs=None, labels=None, label_kwargs=None, show_titles=False, title_fmt='.2f', title_kwargs=None, truths $1=$ None, truth_colorl='red', truth_kwargs $1=$ None, truths $2=$ None, truth_color $2=$ 'blue', truth_kwargs $2=$ None, max_n_ticks=5, top_ticks=False, use_math_text=False, verbose=False, fig=None)
Generate a corner plot of the 1-D and 2-D marginalized posteriors.

## Parameters

## results

[Results instance] A Results instance from a nested sampling run. Compatible with results derived from nestle.

## dims

[iterable of shape (ndim,), optional] The subset of dimensions that should be plotted. If not provided, all dimensions will be shown.

## span

[iterable with shape (ndim,), optional] A list where each element is either a length-2 tuple containing lower and upper bounds or a float from (0., 1.] giving the fraction of (weighted) samples to include. If a fraction is provided, the bounds are chosen to be equal-tailed. An example would be:
. . code: :

$$
\operatorname{span}=[(0 ., 10 .), 0.95,(5 ., 6 .)]
$$

Default is 0.999999426697 (5-sigma credible interval).

## quantiles

[iterable, optional] A list of fractional quantiles to overplot on the 1-D marginalized posteriors as vertical dashed lines. Default is [0.025, 0.5, 0.975] (spanning the $95 \% / 2$-sigma credible interval).

## color

[str or iterable with shape (ndim,), optional] A ~matplotlib-style color (either a single color or a different value for each subplot) used when plotting the histograms. Default is 'black'.

## smooth

[float or iterable with shape (ndim,), optional] The standard deviation (either a single value or a different value for each subplot) for the Gaussian kernel used to smooth the 1-D and 2-D marginalized posteriors, expressed as a fraction of the span. Default is 0.02 ( $2 \%$ smoothing). If an integer is provided instead, this will instead default to a simple (weighted) histogram with bins=smooth.
quantiles_2d
[iterable with shape (nquant,), optional] The quantiles used for plotting the smoothed 2-D distributions. If not provided, these default to $0.5,1,1.5$, and 2 -sigma contours roughly corresponding to quantiles of [0.1, 0.4, 0.65, 0.85].

## hist_kwargs

[dict, optional] Extra keyword arguments to send to the 1-D (smoothed) histograms.

## hist2d_kwargs

[dict, optional] Extra keyword arguments to send to the 2-D (smoothed) histograms.

## labels

[iterable with shape (ndim,), optional] A list of names for each parameter. If not provided, the default name used when plotting will follow $x_{i}$ style.

## label_kwargs

[dict, optional] Extra keyword arguments that will be sent to the ~matplotlib.axes.Axes.set_xlabel and ~matplotlib.axes.Axes.set_ylabel methods.

## show_titles

[bool, optional] Whether to display a title above each 1-D marginalized posterior showing the 0.5 quantile along with the upper/lower bounds associated with the 0.025 and 0.975 ( $95 \% / 2-$ sigma credible interval) quantiles. Default is True.

## title_fmt

[str, optional] The format string for the quantiles provided in the title. Default is '. $2 f^{\prime}$ '.

## title_kwargs

[dict, optional] Extra keyword arguments that will be sent to the ~matplotlib.axes.Axes.set_title command.

## truths

[iterable with shape (ndim,), optional] A list of reference values that will be overplotted on the traces and marginalized 1-D posteriors as solid horizontal/vertical lines. Individual values can be exempt using None. Default is None.

## truth_color

[str or iterable with shape (ndim,), optional] A ~matplotlib-style color (either a single color or a different value for each subplot) used when plotting truths. Default is 'red'.

## truth_kwargs

[dict, optional] Extra keyword arguments that will be used for plotting the vertical and horizontal lines with truths.

## max_n_ticks

[int, optional] Maximum number of ticks allowed. Default is 5 .
top_ticks
[bool, optional] Whether to label the top (rather than bottom) ticks. Default is False.

## use_math_text

[bool, optional] Whether the axis tick labels for very large/small exponents should be displayed as powers of 10 rather than using $e$. Default is False.

## verbose

[bool, optional] Whether to print the values of the computed quantiles associated with each parameter. Default is False.
fig
[(~matplotlib.figure.Figure, ~matplotlib.axes.Axes), optional] If provided, overplot the traces and marginalized 1-D posteriors onto the provided figure. Otherwise, by default an internal figure is generated.

## Returns

## cornerplot

[(~matplotlib.figure.Figure, $\sim$ matplotlib.axes.Axes $)]$ Output corner plot.
model_fitter.contour2d_alpha ( $x, y$, smooth=0.02, span=None, weights=None, sigma_levels=[1, 2, 3], $a x=$ None, color='gray', plot_density=True, plot_contours=True,
contour_kwargs=None, **kwargs)

Simplified/modified from dynesty's plotting._hist2d function. Plots non-filled 2D contours, where the contours are the $0.5,1,1.5,2$ sigma contours (note this)

## Parameters

x
[interable with shape (nsamps,)] Sample positions in the first dimension.
y
[iterable with shape (nsamps,)] Sample positions in the second dimension.

## span

[iterable with shape (ndim,), optional] A list where each element is either a length-2 tuple containing lower and upper bounds or a float from (0., 1.] giving the fraction of (weighted) samples to include. If a fraction is provided, the bounds are chosen to be equal-tailed. An example would be:

```
`span = [(0., 10.), 0.95, (5., 6.)]`
```

Default is 0.999999426697 (5-sigma credible interval).

## weights

[iterable with shape (nsamps,)] Weights associated with the samples. Default is None (no weights).

## sigma_levels

[iterable, optional] The contour levels to draw. Default are [1, 2, 3]-sigma. UNITS ARE IN SIGMA
ax
[~matplotlib.axes.Axes, optional] An ~matplotlib.axes.axes instance on which to add the 2-D histogram. If not provided, a figure will be generated.

## color

[str, optional] The ~matplotlib-style color used to draw lines and color cells and contours. Default is 'gray'.

## plot_density

[bool, optional] Whether to draw the density colormap. Default is True.
plot_contours
[bool, optional] Whether to draw the contours. Default is True.
contour_kwargs
[dict] Any additional keyword arguments to pass to the contour method.
data_kwargs
[dict] Any additional keyword arguments to pass to the plot method when adding the individual data points.
model_fitter.traceplot_custom(results_list, quantiles=[0.025, 0.5, 0.975], smooth $=0.02$, thin $=1$, dims =None, contour_labels_list=None, post_color_list=['blue'], post_kwargs=None, $k d e=T r u e, n k d e=1000$, trace_cmap='plasma', trace_color=None, trace_kwargs=None, connect=False, connect_highlight=10, connect_color='red', connect_kwargs=None, max_n_ticks=5, use_math_text=False, labels=None, label_kwargs=None, show_titles $=$ False, title_fmt='. $2 f^{\prime}$, title_kwargs $=$ None, truths $=$ None, truth_color='red', truth_kwargs=None, verbose=False, fig=None)

Plot traces and marginalized posteriors for each parameter. Allows you to plot multiple trace plots on top of each other. The keywords are mostly the same as the dynesty default, only listing the new keywords here.

## Parameters

results_list
[list of Results instance] A Results instance from a nested sampling run. Compatible with results derived from nestle.
color_list
[list of length the same as results_list] List of ~matplotlib-style colors.
contour_labels_list
[list of length the same as results_list] List of strings for labelling each contour.

## Returns

traceplot
[(~matplotlib.figure.Figure, ~matplotlib.axes.Axes)] Output trace plot.
model_fitter.cornerplot_custom(results_list, dims=None, quantiles=[0.025, 0.5, 0.975], color_list=['blue'], smooth=0.02, quantiles_2d=None, hist_kwargs=None, hist2d_kwargs=None, labels=None, label_kwargs=None, contour_labels_list=None, show_titles=False, title_fmt='. $2 f^{\prime}$, title_kwargs=None, truths=None, truth_color='red', truth_kwargs=None, max_n_ticks=5, top_ticks=False, use_math_text=False, verbose=False, fig=None)
Generate a corner plot of the 1-D and 2-D marginalized posteriors. Allows you to plot multiple corner plots on top of each other. The keywords are mostly the same as dynesty default, only listing the new keywords here.

## Parameters

## results_list

[list of Results instance] A Results instance from a nested sampling run. Compatible with results derived from nestle.
color_list
[list of length the same as results_list] List of $\sim$ matplotlib-style colors.
contour_labels_list
[list of length the same as results_list] List of strings for labelling each contour.

## Returns

cornerplot
[(~matplotlib.figure.Figure, ~matplotlib.axes.Axes)] Output corner plot.

## A

animate() (model.BSPL method), 237
animate() (model.BSPL_Phot method), 240
animate() (model.BSPL_PhotAstrom method), 243
animate() (model.FSPL method), 250
animate() (model.FSPL_Limb method), 261
animate() (model.FSPL_PhotAstrom method), 252, 259
animate() (model.FSPL_PhotAstrom_Par_Param1 method), 167
animate() (model.PSBL method), 203
animate() (model.PSBL_Phot method), 206
animate() (model.PSBL_Phot_noPar_GP_Param1 method), 155
animate() (model.PSBL_Phot_noPar_Param1 method), 121
animate() (model.PSBL_Phot_Par_GP_Param1 method), 161
animate() (model.PSBL_Phot_Par_Param1 method), 126
animate() (model.PSBL_PhotAstrom method), 211
animate() (model.PSBL_PhotAstrom_noPar_GP_Param1 method), 132
animate() (model.PSBL_PhotAstrom_noPar_GP_Param2 method), 137
animate() (model.PSBL_PhotAstrom_noPar_Param1 method), 86
animate() (model.PSBL_PhotAstrom_noPar_Param2 method), 91
animate() (model.PSBL_PhotAstrom_Par_GP_Param1 method), 143
animate() (model.PSBL_PhotAstrom_Par_GP_Param2 method), 149
animate() (model.PSBL_PhotAstrom_Par_Param1 method), 97
animate() (model.PSBL_PhotAstrom_Par_Param2 method), 103
animate() (model.PSBL_PhotAstrom_Par_Param3 method), 109
animate() (model.PSBL_PhotAstrom_Par_Param4 method), 115
animate() (model.PSPL method), 184
animate() (model.PSPL_Astrom method), 187
animate() (model.PSPL_Astrom_Par_Param3 method), 37
animate() (model.PSPL_Astrom_Par_Param4 method), 35
animate() (model.PSPL_Phot method), 185
animate() (model.PSPL_Phot_noPar_GP_Paraml method), 56
animate() (model.PSPL_Phot_noPar_GP_Param2 method), 58
animate() (model.PSPL_Phot_noPar_Param1 method), 39
animate() (model.PSPL_Phot_noPar_Param2 method), 41
animate() (model.PSPL_Phot_Par_GP_Param1 method), 47
animate() (model.PSPL_Phot_Par_GP_Param1_2 method), 52
animate() (model.PSPL_Phot_Par_GP_Param2 method), 49
animate() (model.PSPL_Phot_Par_GP_Param2_2 method), 54
animate() (model.PSPL_Phot_Par_Param1 method), 43
animate() (model.PSPL_Phot_Par_Param2 method), 45
animate() (model.PSPL_PhotAstrom method), 186
animate() (model.PSPL_PhotAstrom_LumLens_Par_Param1 method), 19
animate() (model.PSPL_PhotAstrom_LumLens_Par_Param2 method), 21
animate() (model.PSPL_PhotAstrom_LumLens_Par_Param4 method), 23
animate() (model.PSPL_PhotAstrom_noPar_GP_Param1 method), 81
animate() (model.PSPL_PhotAstrom_noPar_GP_Param2 method), 83
animate() (model.PSPL_PhotAstrom_noPar_Param1 method), 15
animate() (model.PSPL_PhotAstrom_noPar_Param2 method), 27
animate() (model.PSPL_PhotAstrom_noPar_Param4 method), 33

```
animate() (model.PSPL_PhotAstrom_Par_GP_Param1
    method), }6
animate() (model.PSPL_PhotAstrom_Par_GP_Param2
    method),64
animate() (model.PSPL_PhotAstrom_Par_GP_Param3
    method),66
animate() (model.PSPL_PhotAstrom_Par_GP_Param4
    method),69
animate()(model.PSPL_PhotAstrom_Par_LumLens_GP_RahbenpiE_ecliptic() (model.FSPL_Limb_Parallax
    method),71 method),263
animate()(model.PSPL_PhotAstrom_Par_LumLens_GP_RahkonpiE_ecliptic() (model.FSPL_Parallax
    method),74 method),255
animate()(model.PSPL_PhotAstrom_Par_LumLens_GP_RahkenpiE_ecliptic()
    method),76 (model.FSPL_PhotAstrom_Par_Param1
animate()(model.PSPL_PhotAstrom_Par_LumLens_GP_Param4 method),168
    method),79 calc_piE_ecliptic()
animate() (model.PSPL_PhotAstrom_Par_Param1 (model.PSBL_Phot_Par_GP_Param1 method),
        method), 17
animate() (model.PSPL_PhotAstrom_Par_Param2 calc_piE_ecliptic()
        method),25
animate() (model.PSPL_PhotAstrom_Par_Param3
        method), }2
animate() (model.PSPL_PhotAstrom_Par_Param4
        method), 31
```


## B

```
BSPL (class in model), 236
BSPL_GP_PhotAstromParam1 (class in model), 231
BSPL_GP_PhotAstromParam2 (class in model), 233
BSPL_GP_PhotAstromParam3 (class in model), 234
BSPL_GP_PhotParam1 (class in model), 230
BSPL_noParallax (class in model), 245
BSPL_Parallax (class in model), 246
BSPL_Phot (class in model), 238
BSPL_PhotAstrom (class in model), 241
BSPL_PhotAstromParam1 (class in model), 226
BSPL_PhotAstromParam2 (class in model), 227
BSPL_PhotAstromParam3 (class in model), 228
BSPL_PhotParam1 (class in model), 225
```


## C

```
calc_AIC() (in module model_fitter), 290
calc_best_fit() (model_fitter.PSPL_Solver method), 274
calc_best_fit() (model_fitter.PSPL_Solver_Hobson_W method), 283
calc_best_fit() (model_fitter.PSPL_Solver_weighted method), 278
calc_BIC() (in module model_fitter), 290
calc_chi2() (model_fitter.PSPL_Solver method), 275
calc_chi2() (model_fitter.PSPL_Solver_Hobson_Weighte method), 283
calc_chi2() (model_fitter.PSPL_Solver_weighted method), 278
```

calc_chi2_manual() method), 275
calc_chi2_manual() (model_fitter.PSPL_Solver_Hobson_Weighted method), 284
calc_chi2_manual() (model_fitter.PSPL_Solver_weighted method), 278
calc_piE_ecliptic() (model.BSPL_Parallax method), 246
(model.FSPL_Limb_Parallax
(model.FSPL_Parallax method), 255
RankupiE_ecliptic () (model.FSPL_PhotAstrom_Par_Param1
calc_piE_ecliptic() (model.PSBL_Phot_Par_GP_Param1 method), 162
calc_piE_ecliptic() (model.PSBL_Phot_Par_Param1 method), 127
calc_piE_ecliptic()
(model.PSBL_PhotAstrom_Par_GP_Param1 method), 144
calc_piE_ecliptic()
(model.PSBL_PhotAstrom_Par_GP_Param2
method), 150
calc_piE_ecliptic() (model.PSBL_PhotAstrom_Par_Param1 method), 98
calc_piE_ecliptic()
(model.PSBL_PhotAstrom_Par_Param2 method), 104
calc_piE_ecliptic() (model.PSBL_PhotAstrom_Par_Param3 method), 110
calc_piE_ecliptic() (model.PSBL_PhotAstrom_Par_Param4 method), 116
calc_piE_ecliptic() (model.PSPL_Astrom_Par_Param3 method), 38
calc_piE_ecliptic() (model.PSPL_Astrom_Par_Param4 method), 36
calc_piE_ecliptic() (model.PSPL_Parallax method), 194, 195, 199, 215
calc_piE_ecliptic()
(model.PSPL_Parallax_LumLens method), 197
galc_piE_ecliptic()
(model.PSPL_Phot_Par_GP_Param1 method), 47
calc_piE_ecliptic()

```
    (model.PSPL_Phot_Par_GP_Param1_2
    method),52
calc_piE_ecliptic()
    (model.PSPL_Phot_Par_GP_Param2 method),
    50
calc_piE_ecliptic()
    (model.PSPL_Phot_Par_GP_Param2_2
    method),54
calc_piE_ecliptic()
    (model.PSPL_Phot_Par_Param1 method),
    43 ()
calc_piE_ecliptic()
    (model.PSPL_Phot_Par_Param2 method),
    45
calc_piE_ecliptic()
        (model.PSPL_PhotAstrom_LumLens_Par_Param1
        method), 20
calc_piE_ecliptic()
    (model.PSPL_PhotAstrom_LumLens_Par_Param2dparallax_dt_in_direction() (in module model),
    method),22 265
calc_piE_ecliptic()
    (model.PSPL_PhotAstrom_LumLens_Par_Param&
    method), 24
calc_piE_ecliptic()
    (model.PSPL_PhotAstrom_Par_GP_Param1
    method),62
calc_piE_ecliptic()
    (model.PSPL_PhotAstrom_Par_GP_Param2
    method),64
calc_piE_ecliptic()
    (model.PSPL_PhotAstrom_Par_GP_Param3
    method),67
calc_piE_ecliptic()
    (model.PSPL_PhotAstrom_Par_GP_Param4
    method),69
calc_piE_ecliptic()
    FSPL_PhotAstromParam1 (class in model), }25
    (model.PSPL_PhotAstrom_Par_LumLens_GP_Pa{q|\{_size (model.Celerite_GP_Model property),188
    method),}7
calc_piE_ecliptic()
    (model.PSPL_PhotAstrom_Par_LumLens_GP_Param2
    method),74 (in module model_fitter),
calc_piE_ecliptic()
    (model.PSPL_PhotAstrom_Par_LumLens_GP_Pa
    method),77
calc_piE_ecliptic()
    (model.PSPL_PhotAstrom_Par_LumLens_GP_Param4 method), 156
    method),79
calc_piE_ecliptic()
    (model.PSPL_PhotAstrom_Par_Param1
    method),18
calc_piE_ecliptic()
    (model.PSPL_PhotAstrom_Par_Param2
    method), 26
calc_piE_ecliptic()
```

(model.PSPL_PhotAstrom_Par_Param3 method), 30
calc_piE_ecliptic() (model.PSPL_PhotAstrom_Par_Param4 method), 32
calc_weights() (model_fitter.PSPL_Solver_weighted method), 278
Celerite_GP_Model (class in model), 187
compute_gradient() (model.Celerite_GP_Model method), 188
compute_invgamma_params() (in module model_fitter), 287
contour2d_alpha() (in module model_fitter), 293
cornerplot_2truth() (in module model_fitter), 292
cornerplot_custom() (in module model_fitter), 295
1
D
debug_gp_nan() (in module model_fitter), 289 265
calc_piE_ecliptic()
(model.PSPL_PhotAstrom_LumLens_Par_Param\&
method), 24
freeze_all_parameters() (model.Celerite_GP_Model method), 188
freeze_parameter () (model.Celerite_GP_Model method), 188
FSPL (class in model), 249
FSPL_Limb (class in model), 260
FSPL_Limb_noParallax (class in model), 261
FSPL_Limb_Parallax (class in model), 263
FSPL_Limb_PhotAstromParam1 (class in model), 264
FSPL_noParallax (class in model), 253
FSPL_Parallax (class in model), 255
FSPL_PhotAstrom (class in model), 250, 257
FSPL_PhotAstrom_Par_Param1 (class in model), 166
FSPL_PhotAstromParam1 (class in model), 256
calc_piE_ecliptic()
Paytul_size (model.Celerite_GP_Model property), 188
G
Param 2
generate_params_dict() (in module model_fitter), 288
get_all_arrays() (model.PSBL method), 203
get_all_arrays() (model.PSBL_Phot method), 206
get_all_arrays() (model.PSBL_Phot_noPar_GP_Param1
get_all_arrays() (model.PSBL_Phot_noPar_Param1 method), 121
get_all_arrays() (model.PSBL_Phot_Par_GP_Param1 method), 162
get_all_arrays() (model.PSBL_Phot_Par_Param1 method), 127
get_all_arrays() (model.PSBL_PhotAstrom method), 211

```
get_all_arrays()(model.PSBL_PhotAstrom_noPar_GP_getrcamplification()
    method),132 method),246
get_all_arrays() (model.PSBL_PhotAstrom_noPar_GP_geatrcamplification() (model.BSPL_Phot method),
    method),138 240
get_all_arrays() (model.PSBL_PhotAstrom_noPar_Pargett_amplification() (model.BSPL_PhotAstrom
    method), 87 method), 243
get_all_arrays() (model.PSBL_PhotAstrom_noPar_Pargme_amplification() (model.FSPL_Limb_noParallax
    method), }92\mathrm{ method), 262
get_all_arrays()(model.PSBL_PhotAstrom_Par_GP_Pegrettn_mmplification() (model.FSPL_Limb_Parallax
    method),144 method),263
get_all_arrays() (model.PSBL_PhotAstrom_Par_GP_P@gretm_amplification() (model.FSPL_noParallax
    method),150 method),254
get_all_arrays() (model.PSBL_PhotAstrom_Par_Paramget_amplification() (model.FSPL_Parallax
    method),98 method),255
get_all_arrays() (model.PSBL_PhotAstrom_Par_Paramget_amplification()
    method),104 (model.FSPL_PhotAstrom_Par_Param1
get_all_arrays() (model.PSBL_PhotAstrom_Par_Param3 method),168
    method),110 get_amplification()
get_all_arrays() (model.PSBL_PhotAstrom_Par_Param4 (model.PSBL_Phot_noPar_GP_Param1
    method),116
get_amp_arr() (model.PSBL method), 201 get_amplification()
get_amp_arr() (model.PSBL_Phot method), 206
get_amp_arr() (model.PSBL_Phot_noPar_GP_Param1
    method),156
get_amp_arr() (model.PSBL_Phot_noPar_Param1 (model.PSBL_Phot_Par_GP_Paraml method),
    method),121 162
get_amp_arr() (model.PSBL_Phot_Par_GP_Paraml get_amplification()
    method),162 (model.PSBL_Phot_Par_Param1 method),
get_amp_arr() (model.PSBL_Phot_Par_Paraml 127
    method),127 
    get_amplification()
get_amp_arr() (model.PSBL_PhotAstrom method), 211 (model.PSBL_PhotAstrom_noPar_GP_Param1
get_amp_arr() (model.PSBL_PhotAstrom_noPar_GP_Param1 method),133
    method),132 get_amplification()
get_amp_arr() (model.PSBL_PhotAstrom_noPar_GP_Param2 (model.PSBL_PhotAstrom_noPar_GP_Param2
    method),138 method),138
get_amp_arr() (model.PSBL_PhotAstrom_noPar_Paramlget_amplification()
    method), 87 (model.PSBL_PhotAstrom_noPar_Param1
get_amp_arr() (model.PSBL_PhotAstrom_noPar_Param2 method), 87
    method), 92 get_amplification()
get_amp_arr()(model.PSBL_PhotAstrom_Par_GP_Param1 (model.PSBL_PhotAstrom_noPar_Param2
    method),144 method), }9
get_amp_arr() (model.PSBL_PhotAstrom_Par_GP_ParangRt_amplification()
    method),150
get_amp_arr() (model.PSBL_PhotAstrom_Par_Param1
    method),}9
get_amp_arr()(model.PSBL_PhotAstrom_Par_Param2
    method),104
get_amp_arr() (model.PSBL_PhotAstrom_Par_Param3 get_amplification()
        method), 110
get_amp_arr()(model.PSBL_PhotAstrom_Par_Param4
    method),116
get_amplification() (model.BSPL method), 237
get_amplification() (model.BSPL_noParallax
    method), 245
    method), }14
    get_amplification()
        (model.PSBL_PhotAstrom_Par_GP_Param2
    method), 151
    get_amplification()
        (model.PSBL_PhotAstrom_Par_Param2
        method), }10
    get_amplification()
```

```
    (model.PSBL_PhotAstrom_Par_Param3
    method),111
get_amplification()
    (model.PSBL_PhotAstrom_Par_Param4
    method),117
get_amplification()
    (model.PSPL_Astrom_Par_Param3 method),
    38
get_amplification()
    (model.PSPL_Astrom_Par_Param4 method),
    36
get_amplification() (model.PSPL_noParallax
    method), 191, }21
get_amplification()
    (model.PSPL_noParallax_LumLens method),
    196
get_amplification() (model.PSPL_Parallax
    method), 193, 194, 198, }21
get_amplification()
    (model.PSPL_Parallax_LumLens method),
    197
get_amplification()
    (model.PSPL_Phot_noPar_GP_Param1
    method),57
get_amplification()
    (model.PSPL_Phot_noPar_GP_Param2
    method),59
get_amplification()
    (model.PSPL_Phot_noPar_Paraml method),
    39
get_amplification()
    (model.PSPL_Phot_noPar_Param2 method),
    4 1
get_amplification()
    (model.PSPL_Phot_Par_GP_Paraml method),
    48
get_amplification()
    (model.PSPL_Phot_Par_GP_Param1_2
    method),52
get_amplification()
    (model.PSPL_Phot_Par_GP_Param2 method),
    50
get_amplification()
    (model.PSPL_Phot_Par_GP_Param2_2
    method),55
get_amplification()
    (model.PSPL_Phot_Par_Param1 method),
    4 3
get_amplification()
    (model.PSPL_Phot_Par_Param2 method),
    45
get_amplification() get_amplification()
    (model.PSPL_PhotAstrom_LumLens_Par_Param1 (model.PSPL_PhotAstrom_Par_Param3
    method),20
get_amplification()
        (model.PSPL_PhotAstrom_LumLens_Par_Param2
        method), }2
get_amplification()
        (model.PSPL_PhotAstrom_LumLens_Par_Param4
        method), 24
get_amplification()
        (model.PSPL_PhotAstrom_noPar_GP_Param1
        method), }8
get_amplification()
(model.PSPL_PhotAstrom_noPar_GP_Param2
    method), }8
get_amplification()
        (model.PSPL_PhotAstrom_noPar_Param1
        method),16
get_amplification()
        (model.PSPL_PhotAstrom_noPar_Param2
        method), 28
get_amplification()
        (model.PSPL_PhotAstrom_noPar_Param4
        method),33
get_amplification()
        (model.PSPL_PhotAstrom_Par_GP_Param1
        method),62
get_amplification()
        (model.PSPL_PhotAstrom_Par_GP_Param2
        method),64
get_amplification()
    (model.PSPL_PhotAstrom_Par_GP_Param3
    method),67
get_amplification()
    (model.PSPL_PhotAstrom_Par_GP_Param4
    method),69
get_amplification()
        (model.PSPL_PhotAstrom_Par_LumLens_GP_Param1
        method), 72
get_amplification()
        (model.PSPL_PhotAstrom_Par_LumLens_GP_Param2
        method), }7
get_amplification()
        (model.PSPL_PhotAstrom_Par_LumLens_GP_Param3
        method),77
get_amplification()
        (model.PSPL_PhotAstrom_Par_LumLens_GP_Param4
        method), }7
get_amplification()
        (model.PSPL_PhotAstrom_Par_Param1
        method),18
get_amplification()
        (model.PSPL_PhotAstrom_Par_Param2
        method),26
        model.PSPL_PhotAstrom_Par_Param3
        method), 30
```


## BAGLE, Release 0.1

get_amplification()
(model.PSPL_PhotAstrom_Par_Param4 method), 32
get_amplitudes() (in module model), 267
get_angular_einstein_radius() (in module model), 267
get_astrometry() (model.BSPL_noParallax method), 245
get_astrometry() (model.BSPL_Parallax method), 246
get_astrometry() (model.BSPL_Phot method), 239
get_astrometry() (model.BSPL_PhotAstrom method), 242
get_astrometry() (model.FSPL_Limb_noParallax method), 262
get_astrometry() (model.FSPL_Limb_Parallax method), 263
get_astrometry() (model.FSPL_noParallax method), 254
get_astrometry() (model.FSPL_Parallax method), 255
get_astrometry() (model.FSPL_PhotAstrom_Par_Param1 method), 168
get_astrometry() (model.PSPL_Phot_Par_GP_Param1_2 method), 52
get_astrometry() (model.PSBL_Phot method), 205
get_astrometry() (model.PSPL_Astrom_Par_Param3 method), 38
get_astrometry() (model.PSPL_Astrom_Par_Param4 method), 36
get_astrometry() (model.PSPL_noParallax method), 192, 215
get_astrometry() (model.PSPL_noParallax_LumLens method), 196
get_astrometry() (model.PSPL_Parallax method), 193, 194, 198, 214
get_astrometry() (model.PSPL_Parallax_LumLens method), 197
get_astrometry() (model.PSPL_Phot_noPar_GP_Param1 method), 57
get_astrometry() (model.PSPL_Phot_noPar_GP_Param2 method), 59
get_astrometry() (model.PSPL_Phot_noPar_Param1 method), 40
get_astrometry() (model.PSPL_Phot_noPar_Param2 method), 41
get_astrometry() (model.PSPL_Phot_Par_GP_Param1
get_astrometry() (model.PSBL_Phot_noPar_GP_Paramget_astrometry() (model.PSPL_Phot_Par_GP_Param2 method), 156 method), 50
get_astrometry() (model.PSBL_Phot_noPar_Param1 get_astrometry() (model.PSPL_Phot_Par_GP_Param2_2 method), 122 method), 55
get_astrometry() (model.PSBL_Phot_Par_GP_Paraml get_astrometry() (model.PSPL_Phot_Par_Param1 method), 163 method), 44
get_astrometry() (model.PSBL_Phot_Par_Param1 get_astrometry() (model.PSPL_Phot_Par_Param2 method), 128
get_astrometry() (model.PSBL_PhotAstrom method), get_astrometry() (model.PSPL_PhotAstrom_LumLens_Par_Param1 210 method), 20
get_astrometry() (model.PSBL_PhotAstrom_noPar_GPgeatrcaositrometry() (model.PSPL_PhotAstrom_LumLens_Par_Param2 method), 133 method), 22
get_astrometry() (model.PSBL_PhotAstrom_noPar_GPgletrcanszrometry() (model.PSPL_PhotAstrom_LumLens_Par_Param4 method), 139 method), 24
get_astrometry() (model.PSBL_PhotAstrom_noPar_Parglett_astrometry() (model.PSPL_PhotAstrom_noPar_GP_Param1 method), 88 method), 82
get_astrometry() (model.PSBL_PhotAstrom_noPar_Parguee_astrometry) ( (model.PSPL_PhotAstrom_noPar_GP_Param2 method), 93 method), 84
get_astrometry() (model.PSBL_PhotAstrom_Par_GP_Pgreten_Astrometry() (model.PSPL_PhotAstrom_noPar_Param1 method), 145 method), 16
get_astrometry() (model.PSBL_PhotAstrom_Par_GP_Pgyetenastrometry() (model.PSPL_PhotAstrom_noPar_Param2 method), 151
method), 28
get_astrometry() (model.PSBL_PhotAstrom_Par_Paramget_astrometry() (model.PSPL_PhotAstrom_noPar_Param4 method), 99 method), 34
get_astrometry() (model.PSBL_PhotAstrom_Par_Param@et_astrometry() (model.PSPL_PhotAstrom_Par_GP_Param1 method), 105 method), 62
get_astrometry() (model.PSBL_PhotAstrom_Par_Paramget_astrometry() (model.PSPL_PhotAstrom_Par_GP_Param2 method), 111 method), 64
get_astrometry() (model.PSBL_PhotAstrom_Par_Paramget_astrometry() (model.PSPL_PhotAstrom_Par_GP_Param3 method), 117 method), 67

```
get_astrometry()(model.PSPL_PhotAstrom_Par_GP_Param4 128
    method),69 get_astrometry_unlensed()
get_astrometry()(model.PSPL_PhotAstrom_Par_LumLens_GP_P(amowhdl.PSBL_PhotAstrom method), 210
    method), 72 get_astrometry_unlensed()
get_astrometry() (model.PSPL_PhotAstrom_Par_LumLens_GP_P(an\omega\omegahel.PSBL_PhotAstrom_noPar_GP_Param1
    method),74 method),133
get_astrometry() (model.PSPL_PhotAstrom_Par_LumLeget_GBtRometřy_unlensed()
    method),77 (model.PSBL_PhotAstrom_noPar_GP_Param2
get_astrometry()(model.PSPL_PhotAstrom_Par_LumLens_GP_Pameatholl), 139
    method),79 get_astrometry_unlensed()
get_astrometry() (model.PSPL_PhotAstrom_Par_Param1 (model.PSBL_PhotAstrom_noPar_Param1
    method),18
    method), }8
get_astrometry() (model.PSPL_PhotAstrom_Par_Paramget_astrometry_unlensed()
    method), 26 (model.PSBL_PhotAstrom_noPar_Param2
get_astrometry() (model.PSPL_PhotAstrom_Par_Param3 method), 93
    method), 30 get_astrometry_unlensed()
get_astrometry() (model.PSPL_PhotAstrom_Par_Param4 (model.PSBL_PhotAstrom_Par_GP_Param1
    method), }3
get_astrometry_unlensed()
    (model.BSPL_noParallax method), }24
get_astrometry_unlensed() (model.BSPL_Parallax
    method), 247
get_astrometry_unlensed() (model.BSPL_Phot
    method), 239
get_astrometry_unlensed()
        (model.BSPL_PhotAstrom method), }24
get_astrometry_unlensed()
        (model.FSPL_Limb_noParallax method), get_astrometry_unlensed()
        262
get_astrometry_unlensed()
        (model.FSPL_Limb_Parallax method), 263
get_astrometry_unlensed()
        (model.FSPL_noParallax method), 254
get_astrometry_unlensed() (model.FSPL_Parallax
        method), }25
get_astrometry_unlensed()
    (model.FSPL_PhotAstrom method), 253, get_astrometry_unlensed()
    259
get_astrometry_unlensed()
        (model.FSPL_PhotAstrom_Par_Param1
        method),168
get_astrometry_unlensed() (model.PSBL_Phot
        method), 205
get_astrometry_unlensed()
        (model.PSBL_Phot_noPar_GP_Param1
        method),157
get_astrometry_unlensed()
        (model.PSBL_Phot_noPar_Paraml method), get_astrometry_unlensed()
        122
get_astrometry_unlensed()
        (model.PSBL_Phot_Par_GP_Paraml method), get_astrometry_unlensed()
        163
get_astrometry_unlensed()
        (model.PSBL_Phot_Par_Param1 method), get_astrometry_unlensed()
```

(model.PSPL_Phot_noPar_GP_Param2
method), 59
get_astrometry_unlensed()
(model.PSPL_Phot_noPar_Param1 method), 40
get_astrometry_unlensed()
(model.PSPL_Phot_noPar_Param2 method), 42
get_astrometry_unlensed()
(model.PSPL_Phot_Par_GP_Param1 method), 48
get_astrometry_unlensed()
(model.PSPL_Phot_Par_GP_Param1_2
method), 52
get_astrometry_unlensed()
(model.PSPL_Phot_Par_GP_Param2 method), 50
get_astrometry_unlensed()
(model.PSPL_Phot_Par_GP_Param2_2
method), 55
get_astrometry_unlensed()
(model.PSPL_Phot_Par_Param1 method), 44
get_astrometry_unlensed()
(model.PSPL_Phot_Par_Param2 method), 46
get_astrometry_unlensed()
(model.PSPL_PhotAstrom_LumLens_Par_Param1
method), 20
get_astrometry_unlensed()
(model.PSPL_PhotAstrom_LumLens_Par_Param2
method), 22
get_astrometry_unlensed()
(model.PSPL_PhotAstrom_LumLens_Par_Param4
method), 24
get_astrometry_unlensed()
(model.PSPL_PhotAstrom_noPar_GP_Param1
method), 82
get_astrometry_unlensed()
(model.PSPL_PhotAstrom_noPar_GP_Param2
method), 84
get_astrometry_unlensed()
(model.PSPL_PhotAstrom_noPar_Param1
method), 16
get_astrometry_unlensed()
(model.PSPL_PhotAstrom_noPar_Param2
method), 28
get_astrometry_unlensed()
(model.PSPL_PhotAstrom_noPar_Param4
method), 34
get_astrometry_unlensed()
(model.PSPL_PhotAstrom_Par_GP_Param1
method), 62
get_astrometry_unlensed()
(model.PSPL_PhotAstrom_Par_GP_Param2 method), 64
get_astrometry_unlensed() (model.PSPL_PhotAstrom_Par_GP_Param3 method), 67
get_astrometry_unlensed()
(model.PSPL_PhotAstrom_Par_GP_Param4 method), 69
get_astrometry_unlensed()
(model.PSPL_PhotAstrom_Par_LumLens_GP_Param1 method), 72
get_astrometry_unlensed()
(model.PSPL_PhotAstrom_Par_LumLens_GP_Param2 method), 74
get_astrometry_unlensed() (model.PSPL_PhotAstrom_Par_LumLens_GP_Param3 method), 77
get_astrometry_unlensed()
(model.PSPL_PhotAstrom_Par_LumLens_GP_Param4
method), 79
get_astrometry_unlensed()
(model.PSPL_PhotAstrom_Par_Param1
method), 18
get_astrometry_unlensed()
(model.PSPL_PhotAstrom_Par_Param2
method), 26
get_astrometry_unlensed()
(model.PSPL_PhotAstrom_Par_Param3
method), 30
get_astrometry_unlensed()
(model.PSPL_PhotAstrom_Par_Param4
method), 32
get_best_fit() (model_fitter.PSPL_Solver method), 274
get_best_fit() (model_fitter.PSPL_Solver_Hobson_Weighted method), 284
get_best_fit() (model_fitter.PSPL_Solver_weighted method), 278
get_best_fit_model() (model_fitter.PSPL_Solver method), 274
get_best_fit_model()
(model_fitter.PSPL_Solver_Hobson_Weighted method), 284
get_best_fit_model() (model_fitter.PSPL_Solver_weighted method), 279
get_best_fit_modes() (model_fitter.PSPL_Solver method), 274
get_best_fit_modes() (model_fitter.PSPL_Solver_Hobson_Weighted method), 284
get_best_fit_modes()
(model_fitter.PSPL_Solver_weighted method), 279

```
get_centroid() (model.PSBL_PhotAstrom method), method),134
    2 1 0 ~ g e t \_ c e n t r o i d \_ s h i f t ( )
get_centroid() (model.PSBL_PhotAstrom_noPar_GP_Param1 (model.PSBL_PhotAstrom_noPar_GP_Param2
    method),133 method),139
get_centroid()(model.PSBL_PhotAstrom_noPar_GP_Poyetmzentroid_shift()
    method),139 (model.PSBL_PhotAstrom_noPar_Param1
get_centroid() (model.PSBL_PhotAstrom_noPar_Param1 method), 88
    method), 88 get_centroid_shift()
get_centroid()(model.PSBL_PhotAstrom_noPar_Param2 (model.PSBL_PhotAstrom_noPar_Param2
    method), }93\mathrm{ method), }9
get_centroid()(model.PSBL_PhotAstrom_Par_GP_Pargret_centroid_shift()
    method),145 (model.PSBL_PhotAstrom_Par_GP_Param1
get_centroid()(model.PSBL_PhotAstrom_Par_GP_Param2 method),145
    method),151 get_centroid_shift()
get_centroid() (model.PSBL_PhotAstrom_Par_Param1 (model.PSBL_PhotAstrom_Par_GP_Param2
    method),99 method),151
get_centroid()(model.PSBL_PhotAstrom_Par_Param2 get_centroid_shift()
    method),105 (model.PSBL_PhotAstrom_Par_Param1
get_centroid() (model.PSBL_PhotAstrom_Par_Param3 method), }9
    method),111 get_centroid_shift()
get_centroid() (model.PSBL_PhotAstrom_Par_Param4 (model.PSBL_PhotAstrom_Par_Param2
        method),117 method),105
get_centroid_shift() (model.BSPL_noParallax get_centroid_shift()
        method), 245
get_centroid_shift() (model.BSPL_Parallax
        method), }24
get_centroid_shift() (model.BSPL_PhotAstrom
        method), }24
get_centroid_shift()
        (model.FSPL_Limb_noParallax method),
        262
get_centroid_shift() (model.FSPL_Limb_Parallax
        method), 264
get_centroid_shift() (model.FSPL_noParallax
        method), }25
get_centroid_shift() (model.FSPL_Parallax
        method), }25
get_centroid_shift()
        (model.FSPL_PhotAstrom_Par_Param1
        method),168
get_centroid_shift()
        (model.PSBL_Phot_noPar_GP_Paraml
        method),157
get_centroid_shift()
        (model.PSBL_Phot_noPar_Param1 method), get_centroid_shift()
        1 2 2 ~ ( m o d e l . P S P L \_ P h o t \_ n o P a r \_ G P \& P a r a m 1 ~
get_centroid_shift() method),57
        (model.PSBL_Phot_Par_GP_Paraml method), get_centroid_shift()
        1 6 3 ~ ( m o d e l . P S P L \_ P h o t \_ n o P a r \_ G P \_ P a r a m 2 ~
get_centroid_shift() method),59
    (model.PSBL_Phot_Par_Paraml method), get_centroid_shift()
        128 (model.PSPL_Phot_noPar_Param1 method),
get_centroid_shift()
        40
        (model.PSBL_PhotAstrom_noPar_GP_Paraml get_centroid_shift()
```

```
    (model.PSPL_Phot_noPar_Param2 method),
    42
get_centroid_shift()
    (model.PSPL_Phot_Par_GP_Paraml method),
    48
get_centroid_shift()
    (model.PSPL_Phot_Par_GP_Param1_2
    method),53
get_centroid_shift()
    (model.PSPL_Phot_Par_GP_Param2 method),
    50
get_centroid_shift()
    (model.PSPL_Phot_Par_GP_Param2_2
    method),55
get_centroid_shift()
    (model.PSPL_Phot_Par_Param1 method),
    4 4
get_centroid_shift()
    (model.PSPL_Phot_Par_Param2 method),
    4 6
get_centroid_shift()
    (model.PSPL_PhotAstrom_LumLens_Par_Param1
    method),20
get_centroid_shift()
    (model.PSPL_PhotAstrom_LumLens_Par_Param2
    method), 22
get_centroid_shift()
    (model.PSPL_PhotAstrom_LumLens_Par_Param4get_complex_pos() (model.PSBL_Phot_noPar_GP_Param1
    method), 24
get_centroid_shift()
    (model.PSPL_PhotAstrom_noPar_GP_Param1
    method), }8
get_centroid_shift()
    (model.PSPL_PhotAstrom_noPar_GP_Param2
    method), }8
get_centroid_shift()
    (model.PSPL_PhotAstrom_noPar_Param1
    method),16
get_centroid_shift()
    (model.PSPL_PhotAstrom_noPar_Param2
    method), }2
get_centroid_shift()
    (model.PSPL_PhotAstrom_noPar_Param4
    method), 34
get_centroid_shift()
    (model.PSPL_PhotAstrom_Par_GP_Paraml
    method),62
get_centroid_shift()
    (model.PSPL_PhotAstrom_Par_GP_Param2
    method), }6
get_centroid_shift()
    (model.PSPL_PhotAstrom_Par_GP_Param3
    method),67
get_centroid_shift()
```

method), 111
get_complex_pos() (model.PSBL_PhotAstrom_Par_Paraynet_image_pos_arr_old() (model.PSBL method), 202 method), 117
get_einstein_time() (in module model), 267
get_hobson_effective_weights()
(model_fitter.PSPL_Solver_Hobson_Weighted method), 283
get_image() (in module model), 267
get_image_pos_arr() (model.PSBL method), 202
get_image_pos_arr() (model.PSBL_Phot method), 207
get_image_pos_arr()
(model.PSBL_Phot_noPar_GP_Param1
method), 157
get_image_pos_arr()
(model.PSBL_Phot_noPar_Param1 method), 123
get_image_pos_arr()
(model.PSBL_Phot_Par_GP_Param1 method), 163
get_image_pos_arr()
(model.PSBL_Phot_Par_Param1 method), 128
get_image_pos_arr() (model.PSBL_PhotAstrom
method), 212
get_image_pos_arr()
(model.PSBL_PhotAstrom_noPar_GP_Param1
method), 134
get_image_pos_arr()
(model.PSBL_PhotAstrom_noPar_GP_Param2
method), 140
get_image_pos_arr()
(model.PSBL_PhotAstrom_noPar_Param1
method), 89
get_image_pos_arr()
(model.PSBL_PhotAstrom_noPar_Param2
method), 94
get_image_pos_arr()
(model.PSBL_PhotAstrom_Par_GP_Param1
method), 146
get_image_pos_arr()
(model.PSBL_PhotAstrom_Par_GP_Param2
method), 152
get_image_pos_arr()
(model.PSBL_PhotAstrom_Par_Param1
method), 100
get_image_pos_arr()
(model.PSBL_PhotAstrom_Par_Param2
method), 106
get_image_pos_arr()
(model.PSBL_PhotAstrom_Par_Param3
method), 112
get_image_pos_arr()
(model.PSBL_PhotAstrom_Par_Param4
method), 118
get_image_pos_arr_old() (model.PSBL_Phot method), 207
get_image_pos_arr_old() (model.PSBL_Phot_noPar_GP_Param1 method), 158
get_image_pos_arr_old() (model.PSBL_Phot_noPar_Param1 method), 123
get_image_pos_arr_old()
(model.PSBL_Phot_Par_GP_Param1 method), 164
get_image_pos_arr_old()
(model.PSBL_Phot_Par_Param1 method), 129
get_image_pos_arr_old() (model.PSBL_PhotAstrom method), 212
get_image_pos_arr_old()
(model.PSBL_PhotAstrom_noPar_GP_Param1 method), 134
get_image_pos_arr_old()
(model.PSBL_PhotAstrom_noPar_GP_Param2
method), 140
get_image_pos_arr_old()
(model.PSBL_PhotAstrom_noPar_Param1
method), 89
get_image_pos_arr_old()
(model.PSBL_PhotAstrom_noPar_Param2
method), 94
get_image_pos_arr_old()
(model.PSBL_PhotAstrom_Par_GP_Param1 method), 146
get_image_pos_arr_old()
(model.PSBL_PhotAstrom_Par_GP_Param2
method), 152
get_image_pos_arr_old()
(model.PSBL_PhotAstrom_Par_Param1
method), 100
get_image_pos_arr_old()
(model.PSBL_PhotAstrom_Par_Param2
method), 106
get_image_pos_arr_old()
(model.PSBL_PhotAstrom_Par_Param3
method), 112
get_image_pos_arr_old()
(model.PSBL_PhotAstrom_Par_Param4
method), 118
get_lens_astrometry() (model.BSPL_noParallax method), 245
get_lens_astrometry() (model.BSPL_Parallax method), 247
get_lens_astrometry()
(model.FSPL_Limb_noParallax method),

262
get_lens_astrometry() (model.FSPL_Limb_Parallax method), 264
get_lens_astrometry() (model.FSPL_noParallax method), 254
get_lens_astrometry() (model.FSPL_Parallax method), 256
get_lens_astrometry()
(model.FSPL_PhotAstrom_Par_Param1
method), 168
get_lens_astrometry()
(model.PSBL_Phot_noPar_GP_Param1 method), 158
get_lens_astrometry()
(model.PSBL_Phot_noPar_Param1 method), 124
get_lens_astrometry()
(model.PSBL_Phot_Par_GP_Param1 method), 164
get_lens_astrometry()
(model.PSBL_Phot_Par_Param1 method), 129
get_lens_astrometry() (model.PSBL_PhotAstrom
method), 210
get_lens_astrometry()
(model.PSBL_PhotAstrom_noPar_GP_Param1 method), 135
get_lens_astrometry()
(model.PSBL_PhotAstrom_noPar_GP_Param2 method), 141
get_lens_astrometry()
(model.PSBL_PhotAstrom_noPar_Param1 method), 89
get_lens_astrometry()
(model.PSBL_PhotAstrom_noPar_Param2 method), 95
get_lens_astrometry()
(model.PSBL_PhotAstrom_Par_GP_Param1 method), 147
get_lens_astrometry()
(model.PSBL_PhotAstrom_Par_GP_Param2
method), 153
get_lens_astrometry()
(model.PSBL_PhotAstrom_Par_Param1
method), 101
get_lens_astrometry()
(model.PSBL_PhotAstrom_Par_Param2 method), 107
get_lens_astrometry()
(model.PSBL_PhotAstrom_Par_Param3
method), 113
get_lens_astrometry()
(model.PSBL_PhotAstrom_Par_Param4 method), 119
get_lens_astrometry()
(model.PSPL_Astrom_Par_Param3 method), 38
get_lens_astrometry() (model.PSPL_Astrom_Par_Param4 method), 36
get_lens_astrometry() (model.PSPL_noParallax method), 191, 215
get_lens_astrometry() (model.PSPL_noParallax_LumLens method), 196
get_lens_astrometry() (model.PSPL_Parallax method), 193, 194, 198, 214
get_lens_astrometry()
(model.PSPL_Parallax_LumLens method), 197
get_lens_astrometry()
(model.PSPL_Phot_noPar_GP_Param1
method), 57
get_lens_astrometry()
(model.PSPL_Phot_noPar_GP_Param2 method), 59
get_lens_astrometry() (model.PSPL_Phot_noPar_Param1 method), 40
get_lens_astrometry()
(model.PSPL_Phot_noPar_Param2 method), 42
get_lens_astrometry()
(model.PSPL_Phot_Par_GP_Param1 method), 48
get_lens_astrometry()
(model.PSPL_Phot_Par_GP_Param1_2
method), 53
get_lens_astrometry()
(model.PSPL_Phot_Par_GP_Param2 method), 50
get_lens_astrometry()
(model.PSPL_Phot_Par_GP_Param2_2
method), 55
get_lens_astrometry()
(model.PSPL_Phot_Par_Param1 method), 44
get_lens_astrometry() (model.PSPL_Phot_Par_Param2 method), 46
get_lens_astrometry() (model.PSPL_PhotAstrom_LumLens_Par_Param1 method), 20
get_lens_astrometry()
(model.PSPL_PhotAstrom_LumLens_Par_Param2
method), 22
get_lens_astrometry()
(model.PSPL_PhotAstrom_LumLens_Par_Param4

```
    method), 24
get_lens_astrometry()
    (model.PSPL_PhotAstrom_noPar_GP_Param1
    method), }8
get_lens_astrometry()
    (model.PSPL_PhotAstrom_noPar_GP_Param2
    method), }8
get_lens_astrometry()
    (model.PSPL_PhotAstrom_noPar_Param1
    method),16
get_lens_astrometry()
    (model.PSPL_PhotAstrom_noPar_Param2
    method), }2
get_lens_astrometry()
    (model.PSPL_PhotAstrom_noPar_Param4
    method), 34
get_lens_astrometry()
    (model.PSPL_PhotAstrom_Par_GP_Param1
    method), 62
get_lens_astrometry()
    (model.PSPL_PhotAstrom_Par_GP_Param2
    method),65
get_lens_astrometry()
    (model.PSPL_PhotAstrom_Par_GP_Param3
    method),67
get_lens_astrometry()
    (model.PSPL_PhotAstrom_Par_GP_Param4
    method), 70
get_lens_astrometry()
    (model.PSPL_PhotAstrom_Par_LumLens_GP_Param1
    method),}7
    get_log_det_covariance()
get_lens_astrometry() (model.PSPL_Phot_Par_GP_Param2 method),
    (model.PSPL_PhotAstrom_Par_LumLens_GP_Param2 50
    method),75 get_log_det_covariance()
get_lens_astrometry() (model.PSPL_Phot_Par_GP_Param2_2
    (model.PSPL_PhotAstrom_Par_LumLens_GP_Param3 method),55
    method),77
get_log_det_covariance()
get_lens_astrometry() (model.PSPL_PhotAstrom_noPar_GP_Paraml
    (model.PSPL_PhotAstrom_Par_LumLens_GP_Param4 method), }8
    method),80 get_log_det_covariance()
get_lens_astrometry()
    (model.PSPL_PhotAstrom_Par_Param1
    method),18
get_lens_astrometry()
    (model.PSPL_PhotAstrom_Par_Param2
    method), 26
get_lens_astrometry()
    (model.PSPL_PhotAstrom_Par_Param3
    method),30
get_lens_astrometry()
    (model.PSPL_PhotAstrom_Par_Param4
    method), }3
get_log_det_covariance()
    (model.PSBL_Phot_noPar_GP_Param1
    (model.PSPL_PhotAstrom_noPar_GP_Param2
    method), }8
get_log_det_covariance()
        (model.PSPL_PhotAstrom_Par_GP_Param1
        method),62
get_log_det_covariance()
        (model.PSPL_PhotAstrom_Par_GP_Param2
        method),65
get_log_det_covariance()
        (model.PSPL_PhotAstrom_Par_GP_Param3
        method),67
get_log_det_covariance()
        (model.PSPL_PhotAstrom_Par_GP_Param4
        method), 70
```

get_log_det_covariance()
get_photometry() (model.PSBL_PhotAstrom_Par_Param1
(model.PSPL_PhotAstrom_Par_LumLens_GP_Param1 method), 101
method), 72 get_photometry() (model.PSBL_PhotAstrom_Par_Param2
get_log_det_covariance() method), 107
(model.PSPL_PhotAstrom_Par_LumLens_GP_Pagent2photometry () (model.PSBL_PhotAstrom_Par_Param3
method), 75
get_log_det_covariance() method), 113
(model.PSPL_PhotAstrom_Par_LumLens_GP_Param3 method), 119
method), 77 get_photometry_with_gp()
get_log_det_covariance() (model.PSBL_Phot_noPar_GP_Paraml
(model.PSPL_PhotAstrom_Par_LumLens_GP_Param4 method), 159
method), 80
get_minus() (in module model), 267
get_mnest_results() (in module model_fitter), 289
get_photometry_with_gp()
get_parameter() (model.Celerite_GP_Model method) 188
get_parameter_bounds() (model.Celerite_GP_Model method), 188
get_parameter_dict() (model.Celerite_GP_Model method), 189
get_parameter_names() (model.Celerite_GP_Model method), 189
get_parameter_vector() (model.Celerite_GP_Model method), 189
get_photometry() (model.FSPL method), 250
get_photometry() (model.FSPL_Limb method), 260
get_photometry() (model.FSPL_PhotAstrom method), get_photometry_with_gp() (model.PSPL_GP 252, 258 method), 190
get_photometry () (model.FSPL_PhotAstrom_Par_Paramget_photometry_with_gp() method), 168 (model.PSPL_Phot_noPar_GP_Param1
get_photometry() (model.PSBL method), 203 method), 57
get_photometry() (model.PSBL_Phot method), 208 get_photometry_with_gp()
get_photometry() (model.PSBL_Phot_noPar_GP_Param1 (model.PSPL_Phot_noPar_GP_Param2 method), 158 method), 59
get_photometry() (model.PSBL_Phot_noPar_Paraml get_photometry_with_gp() method), 124
get_photometry() (model.PSBL_Phot_Par_GP_Param1 method), 165
(model.PSPL_Phot_Par_GP_Param1 method), 48
 method), 129
method), 53
get_photometry() (model.PSBL_PhotAstrom method), get_photometry_with_gp() 213 (model.PSPL_Phot_Par_GP_Param2 method),
get_photometry() (model.PSBL_PhotAstrom_noPar_GP_Param1 50 method), 135 get_photometry_with_gp()
get_photometry() (model.PSBL_PhotAstrom_noPar_GP_Param2 (model.PSPL_Phot_Par_GP_Param2_2 method), 141 method), 55
get_photometry() (model.PSBL_PhotAstrom_noPar_Pargrett_photometry_with_gp() method), 90 (model.PSPL_PhotAstrom_noPar_GP_Param1
get_photometry() (model.PSBL_PhotAstrom_noPar_Param2 method), 82 method), 95 get_photometry_with_gp()
get_photometry() (model.PSBL_PhotAstrom_Par_GP_Param1 (model.PSPL_PhotAstrom_noPar_GP_Param2 method), 147 method), 84
get_photometry() (model.PSBL_PhotAstrom_Par_GP_Pgretm@hotometry_with_gp() method), 153 (model.PSPL_PhotAstrom_Par_GP_Param1

```
    method),62
get_photometry_with_gp()
    (model.PSPL_PhotAstrom_Par_GP_Param2
    method), }6
get_photometry_with_gp()
    (model.PSPL_PhotAstrom_Par_GP_Param3
    method), }6
get_photometry_with_gp()
    (model.PSPL_PhotAstrom_Par_GP_Param4
    method), }7
get_photometry_with_gp()
    (model.PSPL_PhotAstrom_Par_LumLens_GP_Param1 method),90
    method), 72
get_photometry_with_gp() (model.PSBL_PhotAstrom_noPar_Param2
    (model.PSPL_PhotAstrom_Par_LumLens_GP_Param2 method), }9
    method),75 get_resolved_amplification()
get_photometry_with_gp() (model.PSBL_PhotAstrom_Par_GP_Param1
    (model.PSPL_PhotAstrom_Par_LumLens_GP_Param3 method), }14
    method),77 get_resolved_amplification()
get_photometry_with_gp() (model.PSBL_PhotAstrom_Par_GP_Param2
    (model.PSPL_PhotAstrom_Par_LumLens_GP_Param4 method), 153
    method), }8
get_plus() (in module model), 267
get_resolved_amplification() (model.BSPL
    method),236
get_resolved_amplification()
    (model.BSPL_noParallax method), 245
get_resolved_amplification()
    (model.BSPL_Parallax method), 247
get_resolved_amplification() (model.BSPL_Phot
    method), 240
get_resolved_amplification()
    (model.BSPL_PhotAstrom method), 244
get_resolved_amplification()
    (model.FSPL_Limb_noParallax method),
    262
get_resolved_amplification()
    (model.FSPL_Limb_Parallax method), 264
get_resolved_amplification()
    (model.FSPL_noParallax method), 254
get_resolved_amplification()
    (model.FSPL_Parallax method), 256
get_resolved_amplification()
    (model.FSPL_PhotAstrom_Par_Param1
    method),169
get_resolved_amplification()
    (model.PSBL_Phot_noPar_GP_Param1
    method),159
get_resolved_amplification()
    (model.PSBL_Phot_noPar_Param1 method),
    124
get_resolved_amplification()
    (model.PSBL_Phot_Par_GP_Paraml method),
    165
get_resolved_amplification()
        (model.PSBL_PhotAstrom_Par_Paraml
    method), 101
get_resolved_amplification()
    (model.PSBL_PhotAstrom_Par_Param2
    method),107
get_resolved_amplification()
        (model.PSBL_PhotAstrom_Par_Param3
    method), 113
get_resolved_amplification()
        (model.PSBL_PhotAstrom_Par_Param4
        method),119
get_resolved_amplification()
        (model.PSPL_Astrom_Par_Param3 method),
        38
get_resolved_amplification()
        (model.PSPL_Astrom_Par_Param4 method),
        36
get_resolved_amplification()
        (model.PSPL_noParallax method), 192,
        216
get_resolved_amplification()
        (model.PSPL_noParallax_LumLens method),
        196
get_resolved_amplification()
        (model.PSPL_Parallax method), 193, 195,
        199,214
get_resolved_amplification()
        (model.PSPL_Parallax_LumLens method),
        197
get_resolved_amplification()
    (model.PSPL_Phot_noPar_GP_Param1
    method),57
```

```
get_resolved_amplification()
    (model.PSPL_Phot_noPar_GP_Param2
    method),60
get_resolved_amplification()
    (model.PSPL_Phot_noPar_Param1 method),
    40
get_resolved_amplification()
    (model.PSPL_Phot_noPar_Param2 method),
    42
get_resolved_amplification()
    (model.PSPL_Phot_Par_GP_Param1 method),
    48
get_resolved_amplification()
    (model.PSPL_Phot_Par_GP_Param1_2
    method),53
get_resolved_amplification()
    (model.PSPL_Phot_Par_GP_Param2 method),
    50
get_resolved_amplification()
    (model.PSPL_Phot_Par_GP_Param2_2
    method),55
get_resolved_amplification()
    (model.PSPL_Phot_Par_Param1 method),
    4 4
get_resolved_amplification()
    (model.PSPL_Phot_Par_Param2 method),
    4 6
get_resolved_amplification()
    (model.PSPL_PhotAstrom_LumLens_Par_Param1
    method), 20
get_resolved_amplification()
    (model.PSPL_PhotAstrom_LumLens_Par_Param2
    method), }2
get_resolved_amplification()
    (model.PSPL_PhotAstrom_LumLens_Par_Param4
    method), 24
get_resolved_amplification()
    (model.PSPL_PhotAstrom_noPar_GP_Param1
    method), }8
get_resolved_amplification()
    (model.PSPL_PhotAstrom_noPar_GP_Param2
    method), }8
get_resolved_amplification()
    (model.PSPL_PhotAstrom_noPar_Param1
    method),16
get_resolved_amplification()
    (model.PSPL_PhotAstrom_noPar_Param2
    method),28
get_resolved_amplification()
    (model.PSPL_PhotAstrom_noPar_Param4
    method), }3
get_resolved_amplification()
    (model.PSPL_PhotAstrom_Par_GP_Param1
    method),62
```

method), 205
get_resolved_astrometry()
(model.PSBL_Phot_noPar_GP_Param1
method), 159
get_resolved_astrometry()
(model.PSBL_Phot_noPar_Param1 method), 124
get_resolved_astrometry()
(model.PSBL_Phot_Par_GP_Param1 method), 165
get_resolved_astrometry()
(model.PSBL_Phot_Par_Param1 method),
130
get_resolved_astrometry()
(model.PSBL_PhotAstrom method), 209
get_resolved_astrometry()
(model.PSBL_PhotAstrom_noPar_GP_Param1
method), 135
get_resolved_astrometry()
(model.PSBL_PhotAstrom_noPar_GP_Param2
method), 141
get_resolved_astrometry()
(model.PSBL_PhotAstrom_noPar_Param1
method), 90
get_resolved_astrometry()
(model.PSBL_PhotAstrom_noPar_Param2
method), 95
get_resolved_astrometry()
(model.PSBL_PhotAstrom_Par_GP_Param1
method), 147
get_resolved_astrometry()
(model.PSBL_PhotAstrom_Par_GP_Param2
method), 153
get_resolved_astrometry()
(model.PSBL_PhotAstrom_Par_Param1
method), 101
get_resolved_astrometry()
(model.PSBL_PhotAstrom_Par_Param2
method), 107
get_resolved_astrometry()
(model.PSBL_PhotAstrom_Par_Param3
method), 113
get_resolved_astrometry()
(model.PSBL_PhotAstrom_Par_Param4
method), 119
get_resolved_astrometry()
(model.PSPL_Astrom_Par_Param3 method), 38
get_resolved_astrometry()
(model.PSPL_Astrom_Par_Param4 method), 36
get_resolved_astrometry()
(model.PSPL_noParallax method), 192, 216
get_resolved_astrometry()
(model.PSPL_noParallax_LumLens method), 196
get_resolved_astrometry() (model.PSPL_Parallax method), 193, 195, 199, 214
get_resolved_astrometry()
(model.PSPL_Parallax_LumLens method), 198
get_resolved_astrometry()
(model.PSPL_Phot_noPar_GP_Param1
method), 58
get_resolved_astrometry()
(model.PSPL_Phot_noPar_GP_Param2
method), 60
get_resolved_astrometry() (model.PSPL_Phot_noPar_Param1 method), 40
get_resolved_astrometry()
(model.PSPL_Phot_noPar_Param2 method), 42
get_resolved_astrometry() (model.PSPL_Phot_Par_GP_Param1 method), 48
get_resolved_astrometry()
(model.PSPL_Phot_Par_GP_Param1_2
method), 53
get_resolved_astrometry()
(model.PSPL_Phot_Par_GP_Param2 method), 51
get_resolved_astrometry() (model.PSPL_Phot_Par_GP_Param2_2 method), 55
get_resolved_astrometry() (model.PSPL_Phot_Par_Param1 method), 44
get_resolved_astrometry()
(model.PSPL_Phot_Par_Param2 method), 46
get_resolved_astrometry()
(model.PSPL_PhotAstrom_LumLens_Par_Param1 method), 20
get_resolved_astrometry()
(model.PSPL_PhotAstrom_LumLens_Par_Param2 method), 22
get_resolved_astrometry() (model.PSPL_PhotAstrom_LumLens_Par_Param4 method), 24
get_resolved_astrometry() (model.PSPL_PhotAstrom_noPar_GP_Param1 method), 82
get_resolved_astrometry() (model.PSPL_PhotAstrom_noPar_GP_Param2 method), 85
get_resolved_astrometry()

```
    (model.PSPL_PhotAstrom_noPar_Param1
    method),17
get_resolved_astrometry()
    (model.PSPL_PhotAstrom_noPar_Param2
    method), 28
get_resolved_astrometry()
    (model.PSPL_PhotAstrom_noPar_Param4
    method), }3
get_resolved_astrometry()
    (model.PSPL_PhotAstrom_Par_GP_Paraml
    method),62
get_resolved_astrometry()
    (model.PSPL_PhotAstrom_Par_GP_Param2
    method),65
get_resolved_astrometry()
    (model.PSPL_PhotAstrom_Par_GP_Param3
    method),67
get_resolved_astrometry()
    (model.PSPL_PhotAstrom_Par_GP_Param4
    method), }7
get_resolved_astrometry()
    (model.PSPL_PhotAstrom_Par_LumLens_GP_Param1 method), }9
    method), }7
get_resolved_astrometry() (model.PSBL_PhotAstrom_Par_GP_Param1
    (model.PSPL_PhotAstrom_Par_LumLens_GP_Param2 method), 148
    method),75 get_resolved_lens_astrometry()
get_resolved_astrometry() (model.PSBL_PhotAstrom_Par_GP_Param2
    (model.PSPL_PhotAstrom_Par_LumLens_GP_Param3 method), 154
    method),77 get_resolved_lens_astrometry()
get_resolved_astrometry() (model.PSBL_PhotAstrom_Par_Paraml
    (model.PSPL_PhotAstrom_Par_LumLens_GP_Param4 method),101
    method), }8
get_resolved_astrometry()
    (model.PSPL_PhotAstrom_Par_Param1
    method),19
get_resolved_astrometry()
    (model.PSPL_PhotAstrom_Par_Param2
    method),26
get_resolved_astrometry()
    (model.PSPL_PhotAstrom_Par_Param3
    method), }3
get_resolved_astrometry()
    (model.PSPL_PhotAstrom_Par_Param4
    method), }3
get_resolved_astrometry_unlensed()
    (model.BSPL_Phot method), }23
get_resolved_astrometry_unlensed()
    (model.BSPL_PhotAstrom method), }24
get_resolved_lens_astrometry()
    (model.PSBL_Phot method), 205
get_resolved_lens_astrometry()
    (model.PSBL_Phot_noPar_GP_Param1
    method),159
get_resolved_lens_astrometry()
    get_resolved_lens_astrometry()
        (model.PSBL_PhotAstrom_Par_Param2
        method),107
    get_resolved_lens_astrometry()
        (model.PSBL_PhotAstrom_Par_Param3
        method),113
    get_resolved_lens_astrometry()
        (model.PSBL_PhotAstrom_Par_Param4
        method),119
    get_resolved_photometry() (model.PSBL method),
        203
    get_resolved_photometry() (model.PSBL_Phot
        method), }20
    get_resolved_photometry()
        (model.PSBL_Phot_noPar_GP_Param1
        method),159
    get_resolved_photometry()
        (model.PSBL_Phot_noPar_Paraml method),
        125
get_resolved_photometry()
        (model.PSBL_Phot_Par_GP_Param1 method),
        166
    get_resolved_photometry()
```

(model.PSBL_Phot_Par_Param1 method), H 130
get_resolved_photometry()
(model.PSBL_PhotAstrom method), 213
get_resolved_photometry()
(model.PSBL_PhotAstrom_noPar_GP_Param1 method), 136
get_resolved_photometry()
(model.PSBL_PhotAstrom_noPar_GP_Param2
method), 142
get_resolved_photometry()
(model.PSBL_PhotAstrom_noPar_Param1 method), 90
get_resolved_photometry()
(model.PSBL_PhotAstrom_noPar_Param2
method), 95
get_resolved_photometry()
(model.PSBL_PhotAstrom_Par_GP_Param1
method), 148
get_resolved_photometry()
(model.PSBL_PhotAstrom_Par_GP_Param2
method), 154
get_resolved_photometry()
(model.PSBL_PhotAstrom_Par_Param1
method), 101
get_resolved_photometry()
(model.PSBL_PhotAstrom_Par_Param2
method), 107
get_resolved_photometry()
(model.PSBL_PhotAstrom_Par_Param3
method), 113
get_resolved_photometry()
(model.PSBL_PhotAstrom_Par_Param4
method), 119
get_source_outline_astrometry() (model.FSPL
method), 249
get_source_outline_astrometry()
(model.FSPL_Limb method), 261
get_source_outline_astrometry()
(model.FSPL_PhotAstrom method), 253, 260
get_source_outline_astrometry()
(model.FSPL_PhotAstrom_Par_Param1 method), 169
get_thetas() (in module model), 267
get_u() (model.BSPL method), 236
get_u() (model.BSPL_Phot method), 241
get_u() (model.BSPL_PhotAstrom method), 244
get_u0() (in module model), 267
get_uhat () (in module model), 267
get_unit_vector() (in module model), 267
get_unit_vectors() (in module model), 267
get_value() (model.Celerite_GP_Model method), 188

## L

hobson_weight_log_likely()
(model_fitter.PSPL_Solver_Hobson_Weighted method), 283
load_mnest_modes() (model_fitter.PSPL_Solver method), 274
load_mnest_modes() (model_fitter.PSPL_Solver_Hobson_Weighted method), 284
load_mnest_modes() (model_fitter.PSPL_Solver_weighted method), 279
load_mnest_modes_results_for_dynesty() (model_fitter.PSPL_Solver method), 275
load_mnest_modes_results_for_dynesty() (model_fitter.PSPL_Solver_Hobson_Weighted method), 284
load_mnest_modes_results_for_dynesty() (model_fitter.PSPL_Solver_weighted method), 279
load_mnest_results() (model_fitter.PSPL_Solver method), 274
load_mnest_results() (model_fitter.PSPL_Solver_Hobson_Weighted method), 284
load_mnest_results() (model_fitter.PSPL_Solver_weighted method), 279
load_mnest_results_for_dynesty() (model_fitter.PSPL_Solver method), 275
load_mnest_results_for_dynesty() (model_fitter.PSPL_Solver_Hobson_Weighted method), 284
load_mnest_results_for_dynesty() (model_fitter.PSPL_Solver_weighted method), 279
load_mnest_summary () (model_fitter.PSPL_Solver method), 274
load_mnest_summary() (model_fitter.PSPL_Solver_Hobson_Weighted method), 284
load_mnest_summary() (model_fitter.PSPL_Solver_weighted method), 279
log_likely() (model_fitter.PSPL_Solver method), 273
log_likely() (model_fitter.PSPL_Solver_Hobson_Weighted method), 283
log_likely() (model_fitter.PSPL_Solver_weighted method), 279
log_likely_photometry() (model.PSBL_Phot_noPar_GP_Param1 method), 160
log_likely_photometry() (model.PSBL_Phot_Par_GP_Param1 method),

166
log_likely_photometry()
(model.PSBL_PhotAstrom_noPar_GP_Param1
method), 136
log_likely_photometry()
(model.PSBL_PhotAstrom_noPar_GP_Param2
method), 142
log_likely_photometry() (model.PSBL_PhotAstrom_Par_GP_Param1
method), 148
log_likely_photometry()
(model.PSBL_PhotAstrom_Par_GP_Param2
method), 154
log_likely_photometry() (model.PSPL_GP
method), 190, 191
log_likely_photometry()
(model.PSPL_Phot_noPar_GP_Param1
method), 58
log_likely_photometry()
(model.PSPL_Phot_noPar_GP_Param2
method), 60
log_likely_photometry()
(model.PSPL_Phot_Par_GP_Param1 method), 48
log_likely_photometry()
(model.PSPL_Phot_Par_GP_Param1_2
method), 53
log_likely_photometry()
(model.PSPL_Phot_Par_GP_Param2 method), 51
log_likely_photometry()
(model.PSPL_Phot_Par_GP_Param2_2
method), 55
log_likely_photometry()
(model.PSPL_PhotAstrom_noPar_GP_Param1 method), 83
log_likely_photometry()
(model.PSPL_PhotAstrom_noPar_GP_Param2
method), 85
log_likely_photometry()
(model.PSPL_PhotAstrom_Par_GP_Param1
method), 63
log_likely_photometry()
(model.PSPL_PhotAstrom_Par_GP_Param2
method), 65
log_likely_photometry()
(model.PSPL_PhotAstrom_Par_GP_Param3 method), 68
log_likely_photometry()
(model.PSPL_PhotAstrom_Par_GP_Param4 method), 70
log_likely_photometry()
(model.PSPL_PhotAstrom_Par_LumLens_GP_Pa method), 73
log_likely_photometry()
(model.PSPL_PhotAstrom_Par_LumLens_GP_Param2 method), 75
log_likely_photometry()
(model.PSPL_PhotAstrom_Par_LumLens_GP_Param3 method), 78
log_likely_photometry()
(model.PSPL_PhotAstrom_Par_LumLens_GP_Param4 method), 80
log_prior () (model.Celerite_GP_Model method), 189
LogLikelihood() (model_fitter.PSPL_Solver method), 273
LogLikelihood() (model_fitter.PSPL_Solver_Hobson_Weighted method), 283
LogLikelihood() (model_fitter.PSPL_Solver_weighted method), 278
M
make_default_priors() (model_fitter.PSPL_Solver method), 273
make_default_priors() (model_fitter.PSPL_Solver_Hobson_Weighted method), 284
make_default_priors() (model_fitter.PSPL_Solver_weighted method), 279
make_fdfdt() (in module model_fitter), 287
make_gen() (in module model_fitter), 286
make_invgamma_gen() (in module model_fitter), 287
make_log10norm_gen() (in module model_fitter), 286
make_lognorm_gen() (in module model_fitter), 286
make_mag_base_gen() (in module model_fitter), 286
make_mag_src_gen() (in module model_fitter), 286
make_muS_EN_gen() (in module model_fitter), 286
make_muS_EN_norm_gen() (in module model_fitter), 287
make_norm_gen() (in module model_fitter), 286
make_piS() (in module model_fitter), 287
make_t0_gen() (in module model_fitter), 286
make_truncnorm_gen() (in module model_fitter), 286
make_truncnorm_gen_with_bounds() (in module model_fitter), 286
make_xSO_gen() (in module model_fitter), 286
make_xSQ_norm_gen() (in module model_fitter), 286

## O

oned_int() (in module model), 267
oned_x_int() (in module model), 267
oned_y_int() (in module model), 267
P
parallax_in_direction() (in module model), 265
Parambleter_vector (model.Celerite_GP_Model property), 189
plot_astrometry() (in module model_fitter), 289
plot_astrometry_on_sky() (in module model_fitter), 289
plot_astrometry_proper_motion_removed() module model_fitter), 289
plot_dynesty_style() (model_fitter.PSPL_Solver method), 275
plot_dynesty_style() (model_fitter.PSPL_Solver_Hobson_Weighted method), 285
plot_dynesty_style()
(model_fitter.PSPL_Solver_weighted method), 279
plot_model_and_data() (model_fitter.PSPL_Solver method), 275
plot_model_and_data() (model_fitter.PSPL_Solver_Hobson_Weighted method), 285
plot_model_and_data() (model_fitter.PSPL_Solver_weighted method), 280
plot_model_and_data_modes() (model_fitter.PSPL_Solver method), 275
plot_model_and_data_modes() (model_fitter.PSPL_Solver_Hobson_Weighted method), 285
plot_model_and_data_modes() (model_fitter.PSPL_Solver_weighted method), 280
plot_params() (in module model_fitter), 289
plot_photometry() (in module model_fitter), 289
plot_photometry_gp() (in module model_fitter), 289
pointwise_likelihood() (in module model_fitter), 289
postplot() (in module model_fitter), 290
print_likelihood() (model_fitter.PSPL_Solver method), 275
print_likelihood() (model_fitter.PSPL_Solver_Hobson method), 285
print_likelihood() (model_fitter.PSPL_Solver_weighte method), 280
Prior_from_post() (model_fitter.PSPL_Solver method), 273
Prior_from_post() (model_fitter.PSPL_Solver_Hobson_ method), 283
Prior_from_post() (model_fitter.PSPL_Solver_weighted method), 278
PSBL (class in model), 201
PSBL_GP_PhotAstromParam1 (class in model), 224
PSBL_GP_PhotAstromParam2 (class in model), 224
PSBL_GP_PhotParam1 (class in model), 224
PSBL_Phot (class in model), 204
PSBL_Phot_noPar_GP_Param1 (class in model), 154
PSBL_Phot_noPar_Param1 (class in model), 120

PSBL_Phot_Par_GP_Param1 (class in model), 160
PSBL_Phot_Par_Param1 (class in model), 125
PSBL_PhotAstrom (class in model), 208
PSBL_PhotAstrom_noPar_GP_Param1 (class in model), 131
PSBL_PhotAstrom_noPar_GP_Param2 (class in model), 136
PSBL_PhotAstrom_noPar_Param1 (class in model), 85
PSBL_PhotAstrom_noPar_Param2 (class in model), 91
PSBL_PhotAstrom_Par_GP_Param1 (class in model), 142
PSBL_PhotAstrom_Par_GP_Param2 (class in model), 148
PSBL_PhotAstrom_Par_Param1 (class in model), 96
PSBL_PhotAstrom_Par_Param2 (class in model), 102
PSBL_PhotAstrom_Par_Param3 (class in model), 108
PSBL_PhotAstrom_Par_Param4 (class in model), 114
PSBL_PhotAstromParam1 (class in model), 216
PSBL_PhotAstromParam2 (class in model), 217
PSBL_PhotAstromParam3 (class in model), 218
PSBL_PhotAstromParam4 (class in model), 219
PSBL_PhotAstromParam5 (class in model), 220
PSBL_PhotAstromParam6 (class in model), 222
PSBL_PhotParam1 (class in model), 223
PSPL (class in model), 184
PSPL_Astrom (class in model), 186
PSPL_Astrom_Par_Param3 (class in model), 36
PSPL_Astrom_Par_Param4 (class in model), 34
PSPL_AstromParam3 (class in model), 172
PSPL_AstromParam4 (class in model), 171
PSPL_GP (class in model), 190
PSPL_GP_PhotAstromParam1 (class in model), 181 PSPL_GP_PhotAstromParam2 (class in model), 181 PSPL_GP_PhotAstromParam3 (class in model), 181 PSPL_GP_PhotAstromParam4 (class in model), 182
PSPL_GP_PhotParam1 (class in model), 180
PSPL_GP_PhotParam1_2 (class in model), 180
PlsPLgherdPhotParam2 (class in model), 180
PSPL_GP_PhotParam2_2 (class in model), 180
dPSPL_noParallax (class in model), 191, 215
PSPL_noParallax_LumLens (class in model), 195
PSPL_Parallax (class in model), 192, 194, 198, 213
PSPL_Parallax_LumLens (class in model), 196
Mspyhtedram (class in model), 171
PSPL_Phot (class in model), 184
PSPL_Phot_noPar_GP_Param1 (class in model), 56
PSPL_Phot_noPar_GP_Param2 (class in model), 58
PSPL_Phot_noPar_Param1 (class in model), 39
PSPL_Phot_noPar_Param2 (class in model), 40
PSPL_Phot_Par_GP_Param1 (class in model), 46
PSPL_Phot_Par_GP_Param1_2 (class in model), 51
PSPL_Phot_Par_GP_Param2 (class in model), 49
PSPL_Phot_Par_GP_Param2_2 (class in model), 53
PSPL_Phot_Par_Param1 (class in model), 42

PSPL_Phot_Par_Param2 (class in model), 44
PSPL_PhotAstrom (class in model), 185
PSPL_PhotAstrom_LumLens_Par_Param1 (class in model), 19
PSPL_PhotAstrom_LumLens_Par_Param2 (class in model), 21
PSPL_PhotAstrom_LumLens_Par_Param4 (class in model), 23
PSPL_PhotAstrom_noPar_GP_Param1 (class in model), 80
PSPL_PhotAstrom_noPar_GP_Param2 (class in model), 83
PSPL_PhotAstrom_noPar_Param1 (class in model), 15
PSPL_PhotAstrom_noPar_Param2 (class in model), 27
PSPL_PhotAstrom_noPar_Param4 (class in model), 32
PSPL_PhotAstrom_Par_GP_Param1 (class in model), 60
PSPL_PhotAstrom_Par_GP_Param2 (class in model), 63
PSPL_PhotAstrom_Par_GP_Param3 (class in model), 65
PSPL_PhotAstrom_Par_GP_Param4 (class in model), 68
PSPL_PhotAstrom_Par_LumLens_GP_Param1 (class in model), 70
PSPL_PhotAstrom_Par_LumLens_GP_Param2 (class in model), 73
PSPL_PhotAstrom_Par_LumLens_GP_Param3 (class in model), 75
PSPL_PhotAstrom_Par_LumLens_GP_Param4 (class in model), 78
PSPL_PhotAstrom_Par_Param1 (class in model), 17
PSPL_PhotAstrom_Par_Param2 (class in model), 25
PSPL_PhotAstrom_Par_Param3 (class in model), 28
PSPL_PhotAstrom_Par_Param4 (class in model), 30
PSPL_PhotAstromParam1 (class in model), 175
PSPL_PhotAstromParam2 (class in model), 176
PSPL_PhotAstromParam3 (class in model), 177
PSPL_PhotAstromParam4 (class in model), 178
PSPL_PhotAstromParam5 (class in model), 179
PSPL_PhotParam1 (class in model), 173
PSPL_PhotParam2 (class in model), 174
PSPL_Solver (class in model_fitter), 269
PSPL_Solver_Hobson_Weighted (class in model_fitter), 280
PSPL_Solver_weighted (class in model_fitter), 275

## Q

quantiles() (in module model_fitter), 289

## R

random_prob() (in module model_fitter), 287
rescale_complex_pos() (model.PSBL method), 202
rescale_complex_pos() (model.PSBL_Phot method), 208
rescale_complex_pos()
(model.PSBL_Phot_noPar_GP_Param1
method), 160
rescale_complex_pos()
(model.PSBL_Phot_noPar_Param1 method), 125
rescale_complex_pos()
(model.PSBL_Phot_Par_GP_Param1 method), 166
rescale_complex_pos()
(model.PSBL_Phot_Par_Param1 method), 130
rescale_complex_pos() (model.PSBL_PhotAstrom
method), 213
rescale_complex_pos()
(model.PSBL_PhotAstrom_noPar_GP_Param1
method), 136
rescale_complex_pos()
(model.PSBL_PhotAstrom_noPar_GP_Param2
method), 142
rescale_complex_pos()
(model.PSBL_PhotAstrom_noPar_Param1
method), 90
rescale_complex_pos()
(model.PSBL_PhotAstrom_noPar_Param2
method), 96
rescale_complex_pos()
(model.PSBL_PhotAstrom_Par_GP_Param1
method), 148
rescale_complex_pos()
(model.PSBL_PhotAstrom_Par_GP_Param2
method), 154
rescale_complex_pos()
(model.PSBL_PhotAstrom_Par_Param1
method), 102
rescale_complex_pos()
(model.PSBL_PhotAstrom_Par_Param2
method), 108
rescale_complex_pos()
(model.PSBL_PhotAstrom_Par_Param3
method), 114
in rescale_complex_pos()
(model.PSBL_PhotAstrom_Par_Param4
method), 120
S
sample_post() (model_fitter.PSPL_Solver method), 273
sample_post() (model_fitter.PSPL_Solver_Hobson_Weighted method), 285
sample_post() (model_fitter.PSPL_Solver_weighted method), 280

```
separate_modes()(model_fitter.PSPL_Solver method),
    2 7 4
separate_modes() (model_fitter.PSPL_Solver_Hobson_Weighted
    method), }28
separate_modes() (model_fitter.PSPL_Solver_weighted
    method), }28
set_parameter() (model.Celerite_GP_Model method),
        189
set_parameter_vector() (model.Celerite_GP_Model
    method),189
solve() (model_fitter.PSPL_Solver method), }27
solve() (model_fitter.PSPL_Solver_Hobson_Weighted
    method), 286
solve()(model_fitter.PSPL_Solver_weighted method),
    280
split_param_filter_index1() (in module
    model_fitter), }28
sun_position() (in module model), }26
```

```
T
thaw_all_parameters() (model.Celerite_GP_Model
    method),189
thaw_parameter() (model.Celerite_GP_Model
    method),189
traceplot_custom() (in module model_fitter), 294
twod_cent_x_int() (in module model), }26
twod_cent_y_int() (in module model), }26
twod_int() (in module model), }26
U
u0_hat_from_thetaE_hat() (in module model), 265
V
vector_size(model.Celerite_GP_Model property),190
```

```
W
weighted_quantile() (in module model_fitter), 287
write_params_yaml() (model_fitter.PSPL_Solver
    method), 273
write_params_yaml()
    (model_fitter.PSPL_Solver_Hobson_Weighted
    method), 286
write_params_yaml()
    (model_fitter.PSPL_Solver_weighted method),
    280
```

