

Refactoring MagnificationCurves and Derivatives

Finite Source Calculations

$$A_{FSPL} = A_{PSPL}(B_0 - \gamma B_1)$$

$$\begin{aligned} \frac{\delta A_{FSPL}}{\delta \rho} &= A_{PSPL} \left(\frac{\delta B_0}{\delta z} \frac{\delta z}{\delta \rho} - \gamma \frac{\delta B_1}{\delta z} \frac{\delta z}{\delta \rho} \right) \quad \text{where} \quad z \equiv \frac{u}{\rho} \\ &= A_{PSPL} \left(\frac{-u}{\rho^2} \right) \left(\frac{\delta B_0}{\delta z} - \gamma \frac{\delta B_1}{\delta z} \right) \end{aligned}$$

Part of Model.get_magnification(), but not stored.

Related, but one is in PointLens(), the other is in FitData()

$$A_{FSPL} = A_{PSPL}(B_0 - \gamma B_1)$$

$$\frac{\delta A_{FSPL}}{\delta \rho} = A_{PSPL} \left(\frac{-u}{\rho^2} \right) \left(\frac{\delta B_0}{\delta z} - \gamma \frac{\delta B_1}{\delta z} \right)$$

$$\frac{\delta A_{FSPL}}{\delta a} = \left(\frac{\delta u}{\delta a} \right) \left[\left(\frac{A_{PSPL}}{\rho} \right) \left(\frac{\delta B_0}{\delta z} - \gamma \frac{\delta B_1}{\delta z} \right) + \frac{\delta A_{PSPL}}{\delta u} (B_0 - \gamma B_1) \right]$$

where $a = (t_0, u_0, t_E, \pi_{E,N}, \pi_{E,E})$

Separate calculation that requires recalculating u.

```
class PointLensMagnificationCurve():
    def __init__(self, trajectory=None, parameters=None):
        self.trajectory = trajectory
        if parameters is None:
            self.parameters = {'t_0': None, 'u_0': None, 't_E': None}

    def _check_parameters(self):
        pass

    def get_magnification(self):
        return point_lens_magnification(trajectory)

    def get_d_A_d_params(self):
        return d_A_d_params
```

```
class FiniteSourceGould94MagnificationCurve(PointLensMagnificationCurve):
```

```
    def __init__(self, **kwargs):
        PointLensMagnificationCurve.__init__(**kwargs)
        self.pspl_magnification = None
        self.B_0 = None
```

Also contains the
read_B0_B1
function...

```
    def _check_parameters(self):
        PointLensMagnificationCurve._check_parameters()
        # Check for rho
```

```
    def get_pspl_magnification(self):
        self.pspl_magnification = PointLensMagnificationCurve.get_magnification()
        return self.pspl_magnification
```

```
    def get_B_0(self):
        return self.B_0
```

```
class FiniteSourceGould94MagnificationCurve(PointLensMagnificationCurve):
```

```
    def get_magnification(self):
```

```
        if self.pspl_magnification is None:
```

```
            self.get_pspl_magnification()
```

```
        if self.B_0 is None:
```

```
            self.get_B_0()
```

```
        self.fspl_magnification = self.pspl_magnification() * self.B_0
```

```
    def get_d_A_d_params(self):
```

```
        PointLensMagnificationCurve.get_d_A_d_params()
```

```
        # Modifications for FSPL
```

```
        return d_A_d_params
```

```
class FiniteSourceYoo04MagnificationCurve(
    FiniteSourceGould94MagnificationCurve):

    def __init__(self, **kwargs):
        FiniteSourceGould94MagnificationCurve.__init__(**kwargs)
        self.B_1 = None

    def _check_parameters(self):
        FiniteSourceGould94MagnificationCurve._check_parameters()
        # Check for gamma

    def get_B_1(self):
        return self.B_1

    def get_magnification(self):
        FiniteSourceGould94MagnificationCurve.get_magnification()
        if self.B_1 is None:
            self.get_B_1()

        self.fspl_magnification += self.parameters.gamma * self.B_1

    def get_d_A_d_params(self):
        FiniteSourceGould94MagnificationCurve.get_d_A_d_params()
        # Modifications for B_1 term
        return d_A_d_params
```

```
def _get_d_u_d_params(self, parameters):
```

```
    """
```

```
    Calculate d u / d parameters
```

```
    Returns a *dict*.
```

```
    """
```

```
    # Setup
```

```
    gradient = {param: 0 for param in parameters}
```

```
    as_dict = self.model.parameters.as_dict()
```

```
    # Get source location
```

```
    trajectory = self.get_dataset_trajectory()
```

```
    u_ = np.sqrt(trajectory.x**2 + trajectory.y**2)
```

```
    # Calculate derivatives
```

```
    d_u_d_x = trajectory.x / u_
```

```
    d_u_d_y = trajectory.y / u_
```

```
    dt = self.dataset.time - as_dict['t_0']
```

Current FitData method

`self.trajectory.
get_du_d_params()`



```

# Exactly 2 out of (u_0, t_E, t_eff) must be defined and
# gradient depends on which ones are defined.
t_E = self.model.parameters.t_E
t_eff = self.model.parameters.t_eff
if 't_eff' not in as_dict:
    gradient['t_0'] = -d_u_d_x / t_E
    gradient['u_0'] = d_u_d_y
    gradient['t_E'] = d_u_d_x * -dt / t_E**2
elif 't_E' not in as_dict:
    gradient['t_0'] = -d_u_d_x * as_dict['u_0'] / t_eff
    gradient['u_0'] = (d_u_d_y + d_u_d_x * dt / t_eff)
    gradient['t_eff'] = (d_u_d_x * -dt * as_dict['u_0'] / t_eff**2)
elif 'u_0' not in as_dict:
    gradient['t_0'] = -d_u_d_x / t_E
    gradient['t_E'] = (d_u_d_x * dt - d_u_d_y * t_eff) / t_E**2
    gradient['t_eff'] = d_u_d_y / t_E
else:
    raise KeyError(
        'Something is wrong with ModelParameters in ' +
        'FitData.calculate_chi2_gradient():\n', as_dict)

```

self.PLMC.
get_d_A_d_params

```
# Below we deal with parallax only.  
if 'pi_E_N' in parameters or 'pi_E_E' in parameters:  
    delta_N = trajectory.parallax_delta_N_E['N']  
    delta_E = trajectory.parallax_delta_N_E['E']  
  
    gradient['pi_E_N'] = d_u_d_x * delta_N + d_u_d_y * delta_E  
    gradient['pi_E_E'] = d_u_d_x * delta_E - d_u_d_y * delta_N  
  
return gradient
```



```
PLMC.get_d_A_d_params():  
gradient[piEE,N] =  
self.trajectory.d_u_d_x[piEE,N] *  
self.trajectory.parallax_delta_N_E[E,N]...
```

Deprecated?

`Pointlens().`

`_get_point_lens_finite_source_magnification()`

`get_point_lens_limb_darkening_magnification()`

`get_point_lens_uniform_integrated_magnification()`

`get_point_lens_LD_integrated_magnification()`

Need a property `self.direct`

`FiniteSourceUniformGould94MC()`

`FiniteSourceLDYoo04MC()`

`FiniteSourceUniformLee09MC()`

`FiniteSourceLDLee09MC()`

Raise error for `d_A_d_params`

The Problem:

$$A_{FSPL} = A_{PSPL}(B_0 - \gamma B_1)$$

Comes from `mm.Model.get_magnification(times,
satellite_skycoords, gamma/bandpass)`

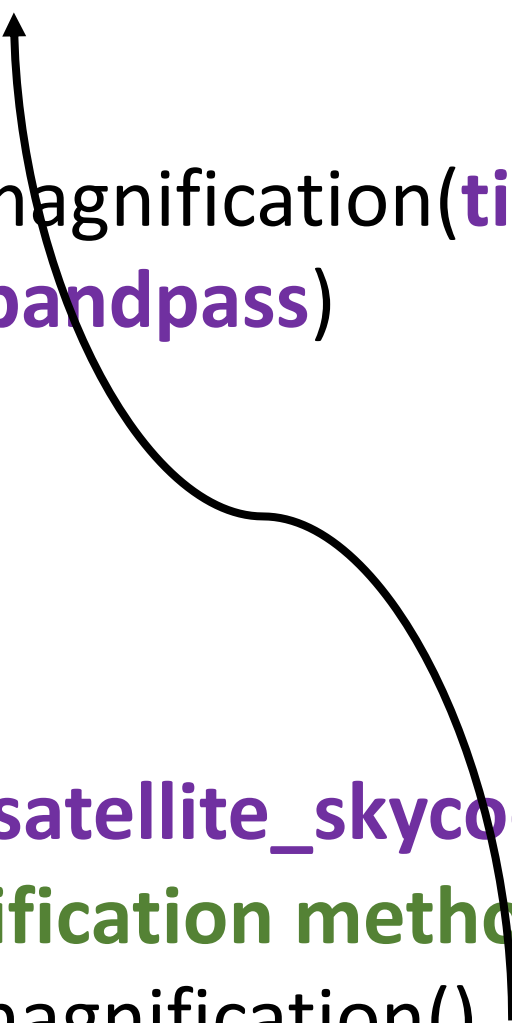
`get_magnification()`

→ `_get_magnification()`

→ `_magnification_1_source()`

→ `MagnificationCurve(times, satellite_skycoords, gamma,
parameters, parallax, magnification methods)`

→ `MagnificationCurve().get_magnification()`



The Problem: $A_{FSPL} = A_{PSPL}(B_0 - \gamma B_1)$

MagnificationCurve().get_magnification() →

- Trajectory()
- get_point_lens_magnification() →
 - get_pspl_magnification → **pspl_magnification**
 - **u**
- PointLens(parameters) →
get_point_lens_limb_darkening_magnification(u,
pspl_magnification, gamma)

The Problem: $A_{FSPL} = A_{PSPL}(B_0 - \gamma B_1)$

PointLens.get_point_lens_limb_darkening_magnification(u,
pspl_magnification, gamma) →

_read_B0B1_file()

→ PointLens._B0_interpolation(), PointLens._B0_minus_B1_interpolation()