Refactoring MagnificationCurves and Derivatives

Finite Source Calculations

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

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

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

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

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

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

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

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

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

Separate calculation that requires recalculating u.

Reader Mode

```python
class FSPLDerivs:

    _B0B1_file_read = False

    def __init__(self, fit):
        # Another problem with this code: the ephemerides file is tied to
        # the dataset, so satellite parallax properties need to be defined
        # relative to the dataset *not* the model (which may not have an
        # ephemerides file). This creates bookkeeping problems.

        # Define the initialization functions
        def _get_u():
            # This calculation gets repeated = Not efficient. Should
            # trajectory be a property of model? No. Wouldn't include
            # dataset ephemerides.
            trajectory = self.fit.get_dataset_trajectory()
            u_ = np.sqrt(trajectory.x ** 2 + trajectory.y ** 2)
            return u_
```

```python
def _get_dataset_satellite_skycoord():
    satellite_skycoord = None
    if self.dataset.ephemerides_file is not None:
        satellite_skycoord = self.dataset.satellite_skycoord

    return satellite_skycoord

def _get_magnification_curve():
    # This code was copied directly from model.py --> indicates a
    # refactor is needed.
    # Also, shouldn't satellite_skycoord be stored as a property of
    # mm.Model?
    # We also have to access a lot of private functions of model,
    # which is bad.
    magnification_curve = MagnificationCurve(
        self.dataset.time, parameters=self.model.parameters,
        parallax=self.model._parallax, coords=self.model.coords,
```

```python
def _get_b0_gamma_b1_and_derivs():
    z_ = self.u_ / self.model.parameters.rho

    b0_gamma_b1 = np.ones(len(self.dataset.time))
    db0_gamma_db1 = np.zeros(len(self.dataset.time))
    # This section was copied from magnificationcurve.py. Addl
    # evidence a refactor is needed.
    methods = np.array(
        self._magnification_curve._methods_for_epochs())
    for method in set(methods):
        kwargs = {}
        if (self._magnification_curve._methods_parameters is not
                None):
            if method in self._magnification_curve._methods_parameters.keys():
                kwargs = self._magnification_curve._methods_parameters[method]
            if kwargs != {}:
                raise ValueError(
```

```python
def _read_B0B1_file(self):
    """Read file with pre-computed function values"""
    # Adapted from PointLens
    file_ = os.path.join(
        mm.DATA_PATH, 'interpolation_table_b0b1_v3.dat')
    if not os.path.exists(file_):
        raise ValueError(
            'File with FSPL data does not exist.\n' + file_)
    (z, B0, B0_minus_B1, B1, B0_prime, B1_prime) = np.loadtxt(
        file_, unpack=True)
    FitData.FSPLDerivs._z_max = z[-1]
    FitData.FSPLDerivs._get_B0 = interp1d(
        z, B0, kind='cubic', bounds_error=False, fill_value=1.0)
    FitData.FSPLDerivs._get_B1 = interp1d(
        z, B1, kind='cubic', bounds_error=False, fill_value=0.0)
    FitData.FSPLDerivs._get_B0_prime = interp1d(
        z, B0_prime, kind='cubic', bounds_error=False, fill_value=0.0)
    FitData.FSPLDerivs._get_B1_prime = interp1d(
```

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()

General Constraints

- There are multiple methods for calculating the magnification
- Different methods may be applied to different parts of the data
- The underlying parameter that controls magnification is the source position, which is calculated by Trajectory
- Different datasets may have different trajectories
- The model needs to be universal and independent of the datasets

Proposed Solution: Create separate classes for each type of magnification curve.

```python
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
```

```python
class FiniteSourceGould94MagnificationCurve(PointLensMagnificationCurve):

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

    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
```

Also contains the read_B0_B1 function...

```python
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
```

```python
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
```

```python
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()
```

```python
# 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

```python
# 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

Open Questions

- How do these new classes interface with FitData, Model and MagnificationCurve?
- Is MagnificationCurve still needed?
- How are different methods applied to different parts of the data?
- How do we fix the bookkeeping problem of the ephemerides files (which may be attached to datasets)?