Friedrich-Alexander-Universität
Erlangen-Nürnberg

ERLANGEN CENTRE FOR ASTROPARTICLE PHYSICS

# Implementation of Priors in Gammapy

Katrin Streil
Erlangen, 23.10.2023

# What are priors?

Based on Bayesian statistics

Model Para.

Nuisance Para. = parameters not of physical interest

$$P(\mu(\theta), \nu(\delta); c) \propto \mathcal{L}(\mu(\theta) + \nu(\delta); c) \cdot G(\delta)$$

Posterior

Likelihood

$$\mathcal{L}(\mu(\boldsymbol{\theta}); c) = \prod_{i=1}^{N} \frac{\mu_i(\boldsymbol{\theta})^{c_i}}{c_i!} \exp(-\mu_i(\boldsymbol{\theta}))$$

Prior:
Incorporating prior knowledge

Model Prediction

Counts

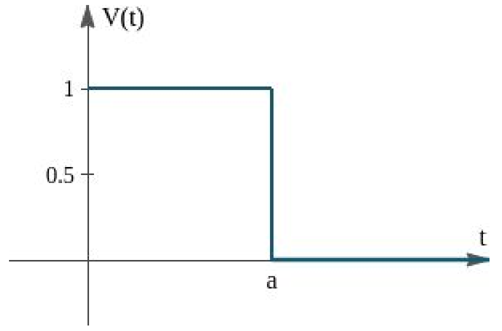
Additional Model Prediction
(optional)

# Use cases of priors

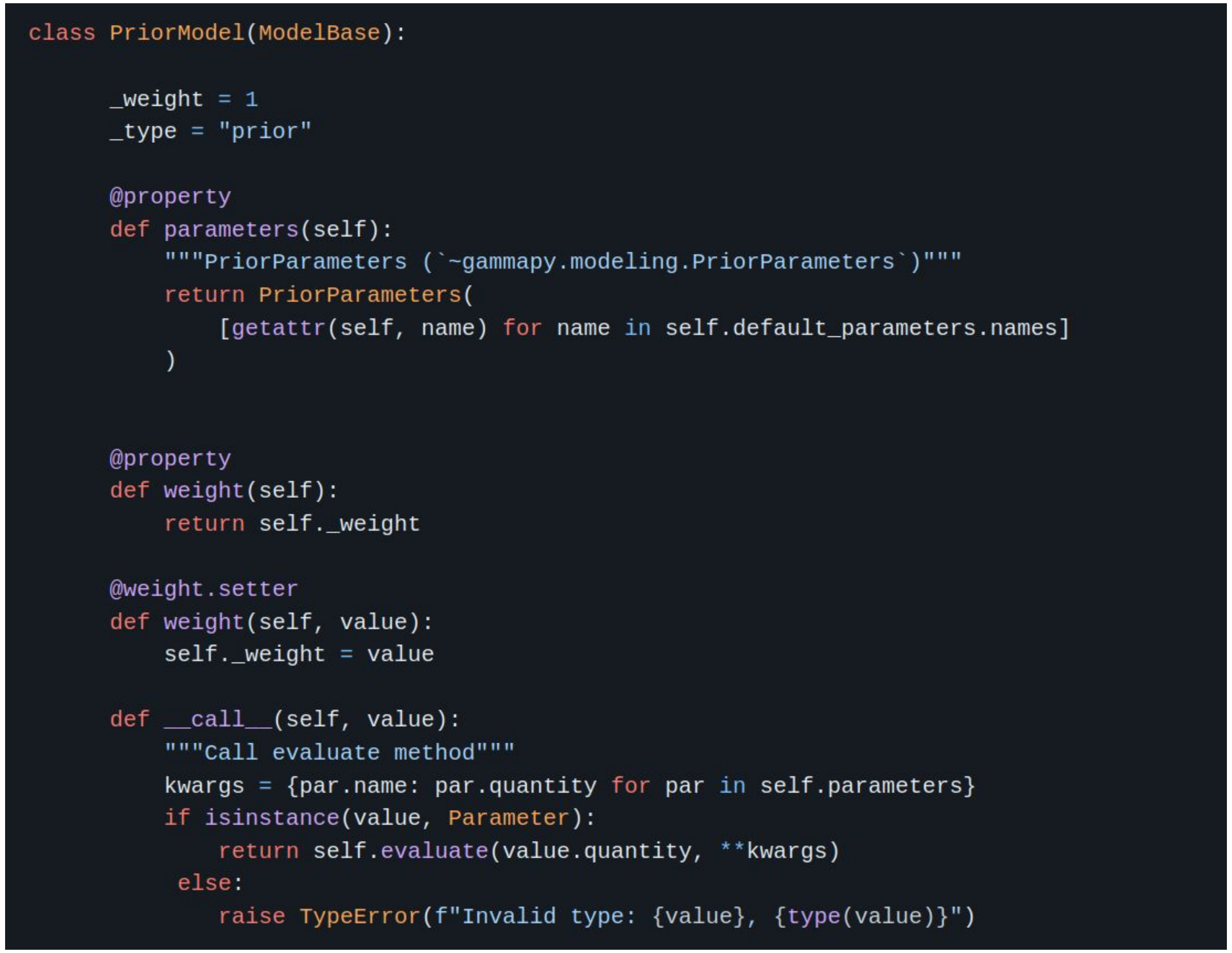
- Generally: incorporate any kind of prior knowledge to the Fit (astrophysical, instrumental, statistical,..)
- **BKG systematics**:
  - Standard: Norm + tilt (unpenalized nuisance parameters)
  - Now: Advanced modifications with multiple nuisance parameters (i.e. one normalization parameters per energy bin (PieceWiseSpectralModel), spatial corrections like tilt, etc.)
  - Incorporate educated guess of the magnitude of the systematic in the prior

# Use cases of priors

- **Positive flux values**
  - Standard: interpret negative results as ‘nonphysical’, set min = 0
  - Now: favor positive values by setting a “inverted heaviside function” as prior
- **Support unfolding methods for spectral flux points**
  - Triknov regularization with Triknov matrix set in multi-dimensional prior



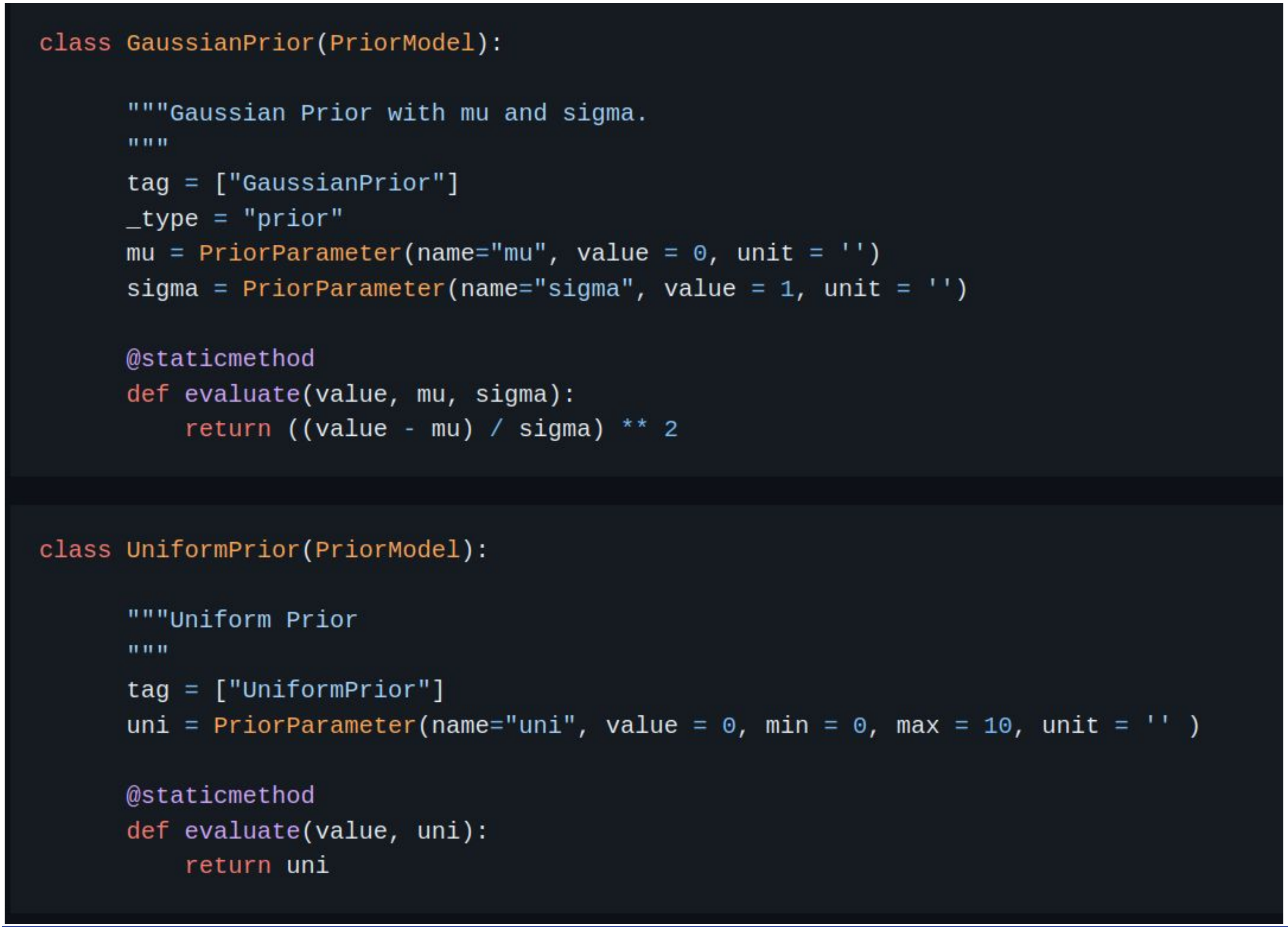
# Implementation: PriorModel



```
class PriorModel(ModelBase):

      _weight = 1
      _type = "prior"

      @property
      def parameters(self):
          """PriorParameters (`~gammapy.modeling.PriorParameters`)"""
          return PriorParameters(
              [getattr(self, name) for name in self.default_parameters.names]
          )


      @property
      def weight(self):
          return self._weight

      @weight.setter
      def weight(self, value):
          self._weight = value

      def __call__(self, value):
          """Call evaluate method"""
          kwargs = {par.name: par.quantity for par in self.parameters}
          if isinstance(value, Parameter):
              return self.evaluate(value.quantity, **kwargs)
           else:
              raise TypeError(f"Invalid type: {value}, {type(value)}")
```

- The parameters of the **PriorModel** are **PriorParameters** and **PriorParameter**.
- They inherit from the **Parameters** and **Parameter** classes (with limited attributes)
- Assumption: **PriorParameter** set in the same unit as **Parameter**
- Future additional properties of the **PriorModel** classes:
  - **Write/read** from/to a yaml file, ideally also when the corresponding model is written/read (see serialisation example below)
  - **Prior** registry system
  - Different prior **subclasses** depending on the use cases

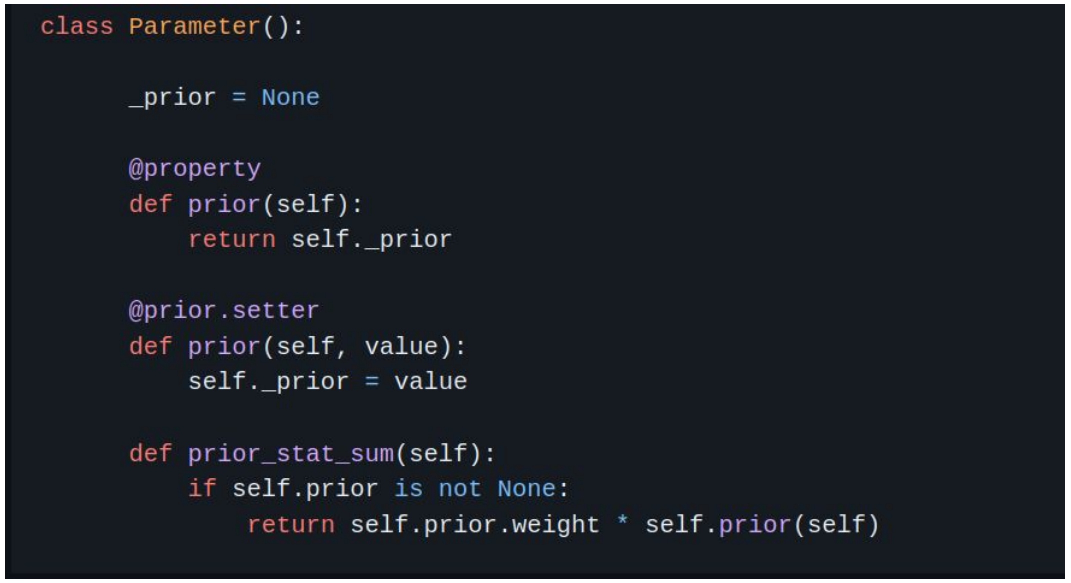
# Implementation: Subclasses



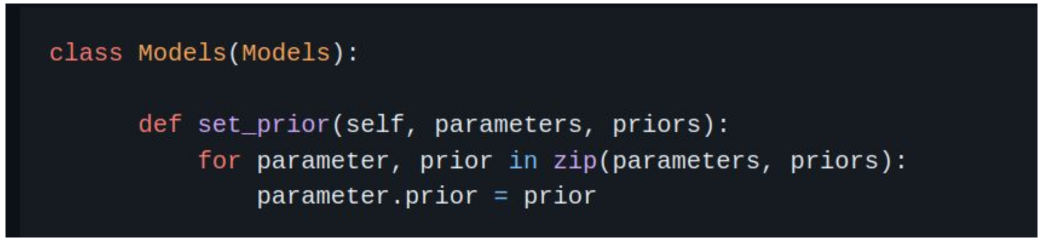
```python
class GaussianPrior(PriorModel):

        """Gaussian Prior with mu and sigma.
        """
        tag = ["GaussianPrior"]
        _type = "prior"
        mu = PriorParameter(name="mu", value = 0, unit = '')
        sigma = PriorParameter(name="sigma", value = 1, unit = '')

        @staticmethod
        def evaluate(value, mu, sigma):
            return ((value - mu) / sigma) ** 2
```

```python
class UniformPrior(PriorModel):

        """Uniform Prior
        """
        tag = ["UniformPrior"]
        uni = PriorParameter(name="uni", value = 0, min = 0, max = 10, unit = '' )

        @staticmethod
        def evaluate(value, uni):
            return uni
```

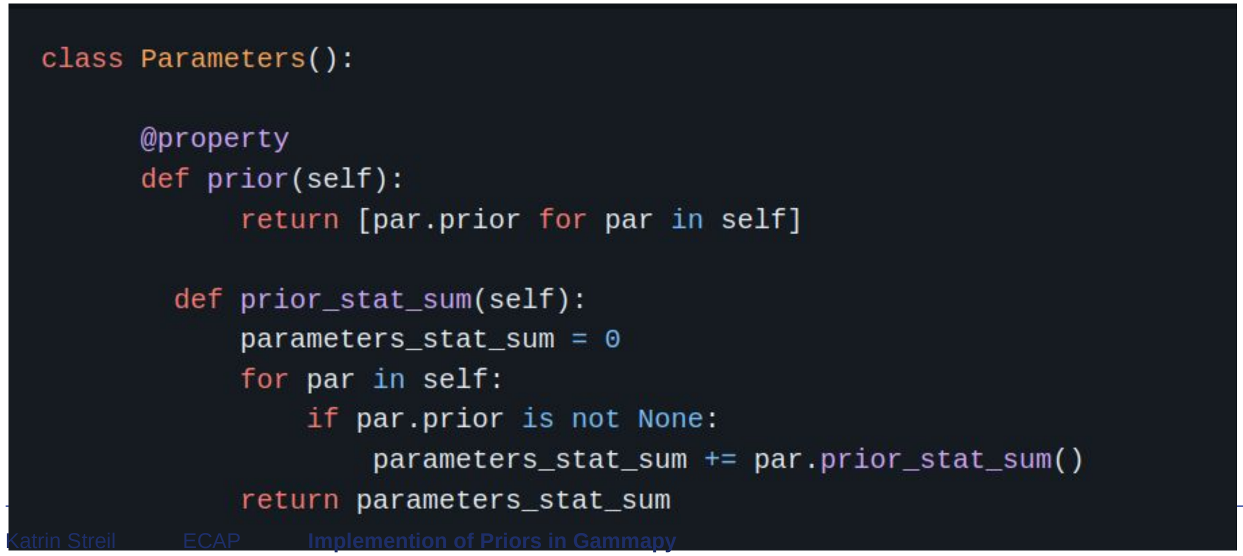
# Implementation: Setting Priors



```
class Parameter():

      _prior = None

      @property
      def prior(self):
          return self._prior

      @prior.setter
      def prior(self, value):
          self._prior = value

      def prior_stat_sum(self):
          if self.prior is not None:
              return self.prior.weight * self.prior(self)
```

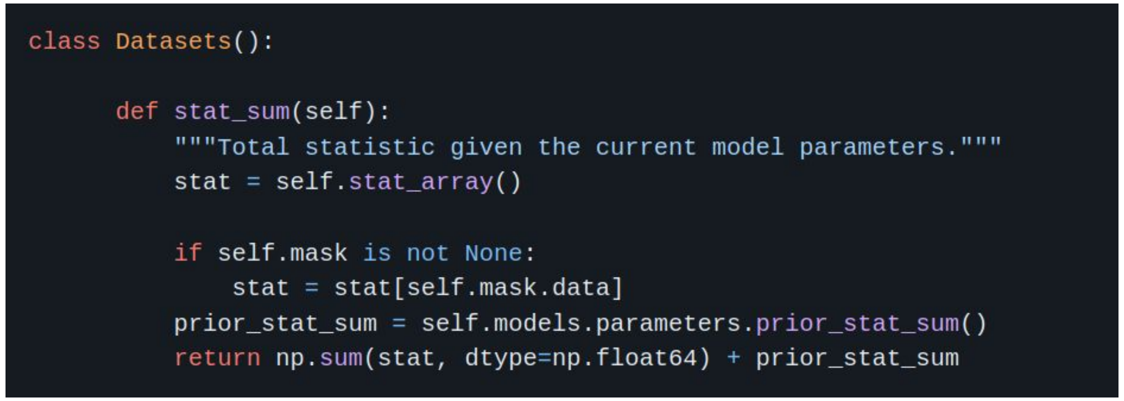


```
class Models(Models):

      def set_prior(self, parameters, priors):
          for parameter, prior in zip(parameters, priors):
              parameter.prior = prior
```



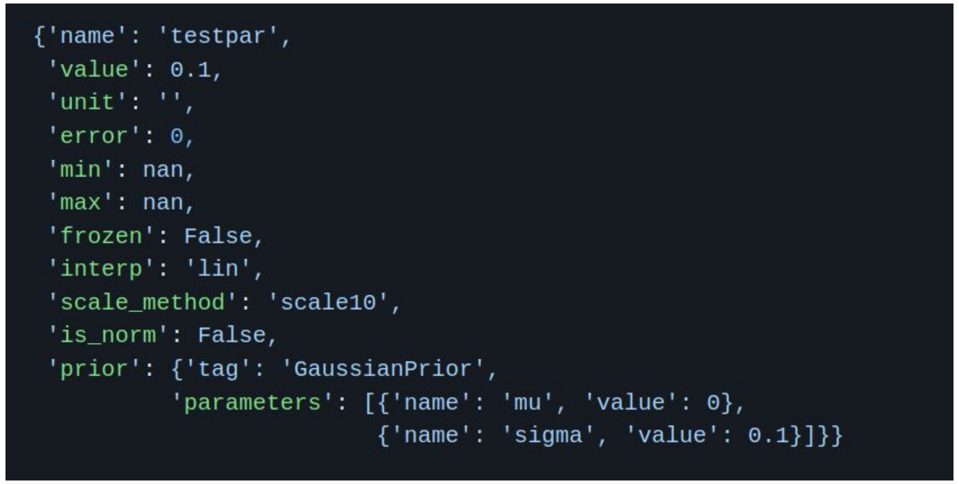
```
class Parameters():

      @property
      def prior(self):
            return [par.prior for par in self]

        def prior_stat_sum(self):
            parameters_stat_sum = 0
            for par in self:
                if par.prior is not None:
                    parameters_stat_sum += par.prior_stat_sum()
            return parameters_stat_sum
```

# Implementation: Evaluation



```
class Datasets():

      def stat_sum(self):
          """Total statistic given the current model parameters."""
          stat = self.stat_array()

          if self.mask is not None:
              stat = stat[self.mask.data]
          prior_stat_sum = self.models.parameters.prior_stat_sum()
          return np.sum(stat, dtype=np.float64) + prior_stat_sum
```

# Implementation: Serialization

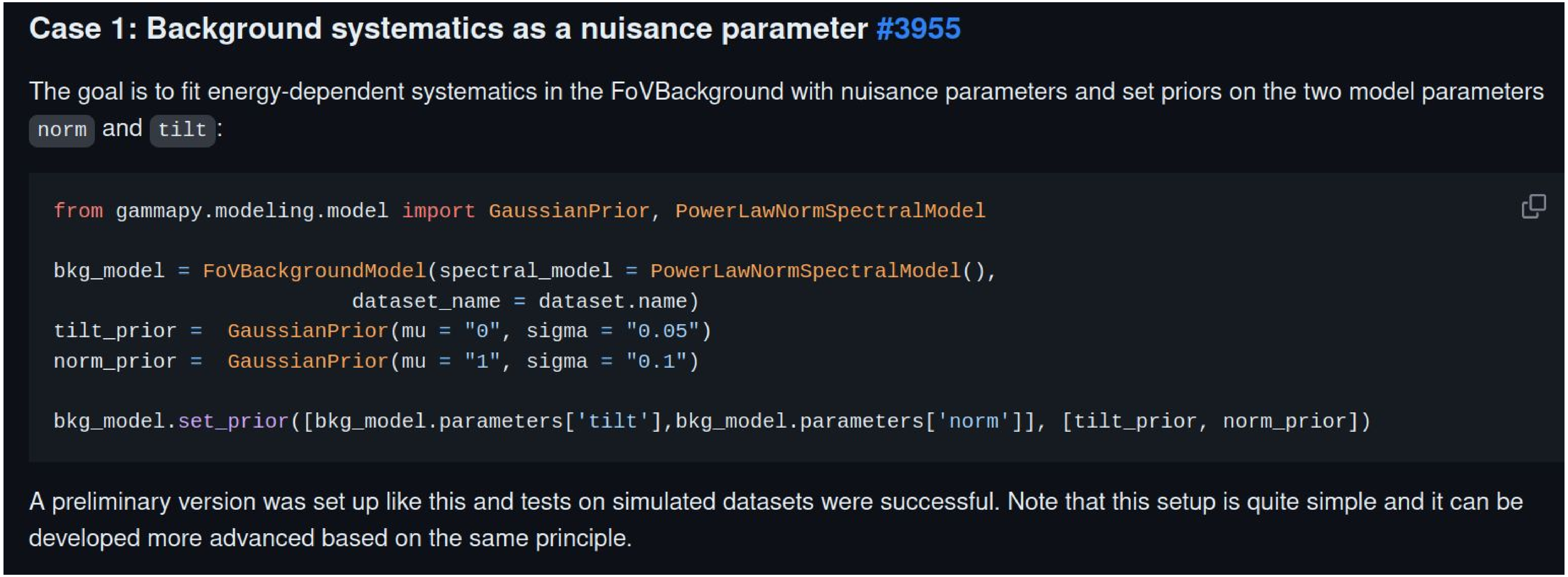


```
{'name': 'testpar',
 'value': 0.1,
 'unit': '',
 'error': 0,
 'min': nan,
 'max': nan,
 'frozen': False,
 'interp': 'lin',
 'scale_method': 'scale10',
 'is_norm': False,
 'prior': {'tag': 'GaussianPrior',
           'parameters': [{'name': 'mu', 'value': 0},
                          {'name': 'sigma', 'value': 0.1}]}}
```

## Summary and Addressing Comments

- **Prior** base class similar structured as **Model** class (subclasses, Parameters, evaluate,..)
- Priors set on model parameter. (after discussions if better set on Model or Parameter)
- Can be set after model initialization
- Unit handeling: for now assumed parameter unit is the same as the priorparameter unit
- Bounds on priorparameters: can be set equivalent to bounds on parameters
- Global weight set on prior (“regularization strength tau”)
- Suggestion for a “LogUniformPrior” to set on logscale parameters (amplitudes)
- Value setting of the Prior for maximum flexibility in the use cases

## Backup

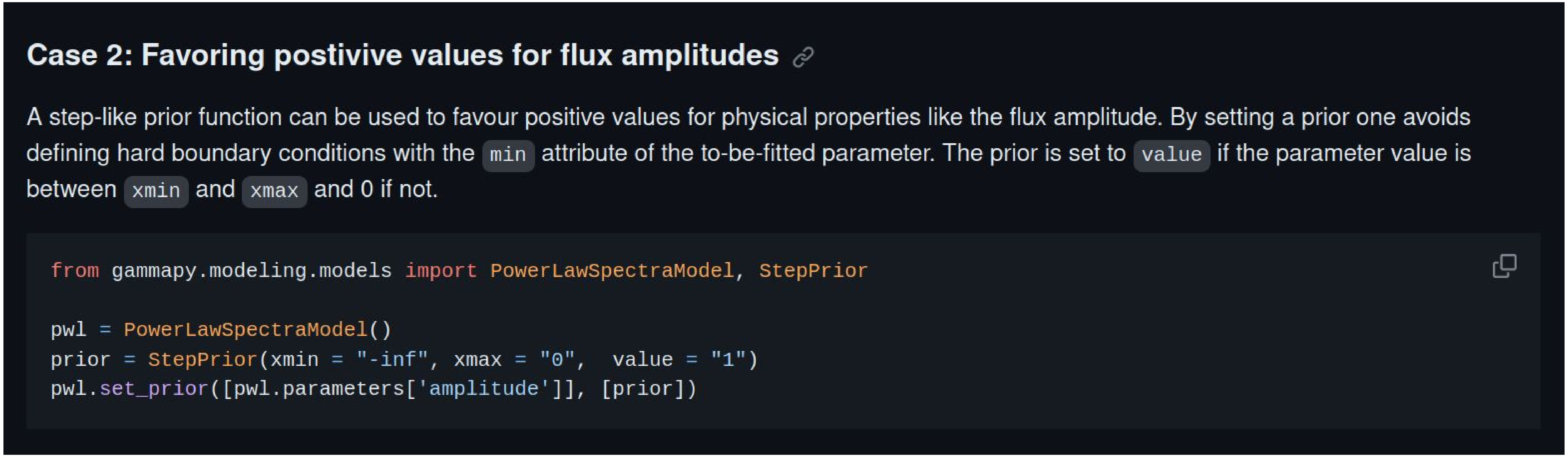


## Case 1: Background systematics as a nuisance parameter #3955

The goal is to fit energy-dependent systematics in the FoVBackground with nuisance parameters and set priors on the two model parameters `norm` and `tilt`:

```
from gammapy.modeling.model import GaussianPrior, PowerLawNormSpectralModel

bkg_model = FoVBackgroundModel(spectral_model = PowerLawNormSpectralModel(),
                         dataset_name = dataset.name)
tilt_prior =  GaussianPrior(mu = "0", sigma = "0.05")
norm_prior =  GaussianPrior(mu = "1", sigma = "0.1")

bkg_model.set_prior([bkg_model.parameters['tilt'],bkg_model.parameters['norm']], [tilt_prior, norm_prior])
```

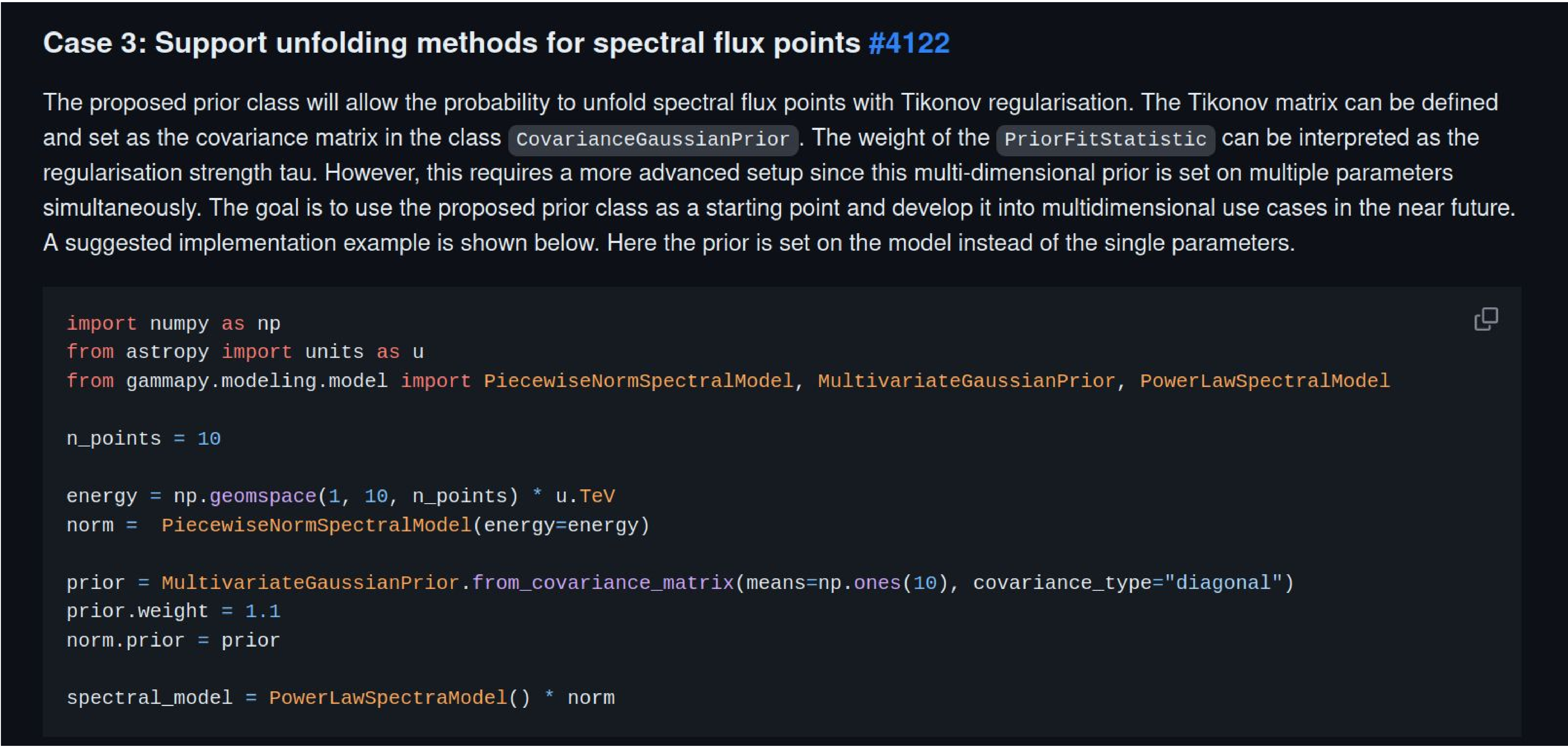
A preliminary version was set up like this and tests on simulated datasets were successful. Note that this setup is quite simple and it can be developed more advanced based on the same principle.



## Case 2: Favoring postivive values for flux amplitudes

A step-like prior function can be used to favour positive values for physical properties like the flux amplitude. By setting a prior one avoids defining hard boundary conditions with the `min` attribute of the to-be-fitted parameter. The prior is set to `value` if the parameter value is between `xmin` and `xmax` and 0 if not.

```python
from gammapy.modeling.models import PowerLawSpectraModel, StepPrior

pwl = PowerLawSpectraModel()
prior = StepPrior(xmin = "-inf", xmax = "0",  value = "1")
pwl.set_prior([pwl.parameters['amplitude']], [prior])
```



## Case 3: Support unfolding methods for spectral flux points #4122

The proposed prior class will allow the probability to unfold spectral flux points with Tikonov regularisation. The Tikonov matrix can be defined and set as the covariance matrix in the class `CovarianceGaussianPrior`. The weight of the `PriorFitStatistic` can be interpreted as the regularisation strength tau. However, this requires a more advanced setup since this multi-dimensional prior is set on multiple parameters simultaneously. The goal is to use the proposed prior class as a starting point and develop it into multidimensional use cases in the near future. A suggested implementation example is shown below. Here the prior is set on the model instead of the single parameters.

```python
import numpy as np
from astropy import units as u
from gammapy.modeling.model import PiecewiseNormSpectralModel, MultivariateGaussianPrior, PowerLawSpectralModel

n_points = 10

energy = np.geomspace(1, 10, n_points) * u.TeV
norm =  PiecewiseNormSpectralModel(energy=energy)

prior = MultivariateGaussianPrior.from_covariance_matrix(means=np.ones(10), covariance_type="diagonal")
prior.weight = 1.1
norm.prior = prior

spectral_model = PowerLawSpectraModel() * norm
```