FitStatistics API Discussion

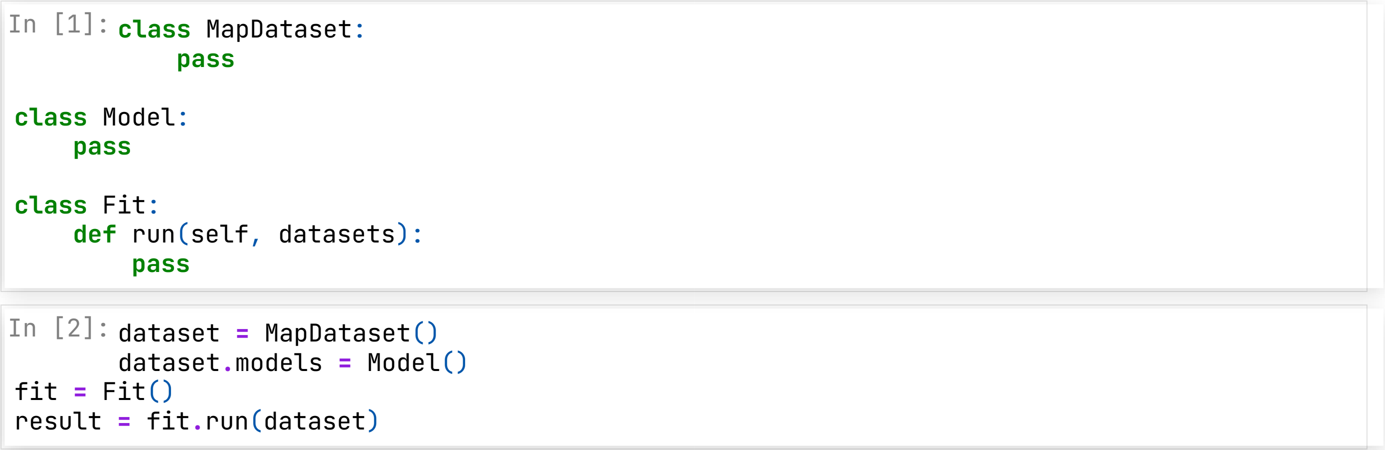
Noah Biederbeck
2022-12-05
TU Dortmund University

This talk summarizes my understanding of the current implementation, shows possible ways to improve, and is a warm invitation to contribute.

Gammapy API as of now

Currently, the FitStatistics/Likelihood API is quite limited.
There has already been some discussion about it and feature requests tangential to it:

- 2019-03-07: **Add sigma v estimator functionality for dark matter use case #2075**
- 2022-05-17: **Background systematics as a nuisance parameter #3955**
- 2020-06-10: **PIG 20 - Global Model API**



```
In [1]: class MapDataset:
            pass

class Model:
    pass

class Fit:
    def run(self, datasets):
        pass
```

```
In [2]: dataset = MapDataset()
        dataset.models = Model()
fit = Fit()
result = fit.run(dataset)
```

What happens internally?

```
stat_sum = sum(stat_array)
stat_array = cash(dataset.counts, dataset.npred())
dataset.npred():
    model/evaluater.compute_npred()
```

See **code**

Why am I here?

I came to this issue because of the **unfolding (#4122)** I want to implement.
In unfolding (here: `RUN`, `R`egularized `UN`folding) you regularize the solution via e.g. Tikhonov Regularization (flat second derivative).
You can't do this with the current Gammapy API.

Regularization is independent of measured and predicted counts, it is only dependent on model parameters.
Priors and Nuisance are of similar nature.

Things I hope we already are of the same opinion

It would be great if

1. The fitstatistic is not hardcoded on the dataset
    - one fitstatistic per dataset is good
2. Priors, Nuisance, Regularization is handled

Bonus

1. The model lives independently from the dataset(s)

Proposals for the FitStatistic API

Use Cases

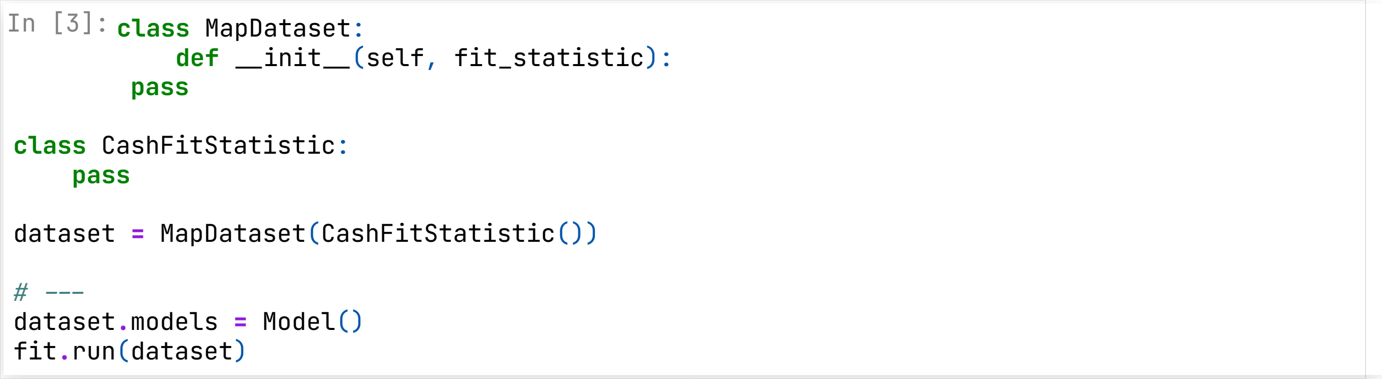
- Unfolding with spectral regularization
- Including IRF systematics see **https://github.com/open-gamma-ray-astro/joint-crab/blob/master/joint_crab/fit_systematics.py#L62**
- Nuissance for DM analysis, see **https://github.com/gammapy/gammapy/pull/2075/files#diff-fcd6006c3dbf2eaa23394b5472d3a5111ee749782a3efe52f32472525b4df305R486**
- Nuissance parameters for background: **https://github.com/gammapy/gammapy/issues/3955**
- Nuissance parameter for background spatial correction, regularize: **https://github.com/gammapy/gammapy/pull/4208**
- ... (add your own)

Which API to evaluate the fit statistic?

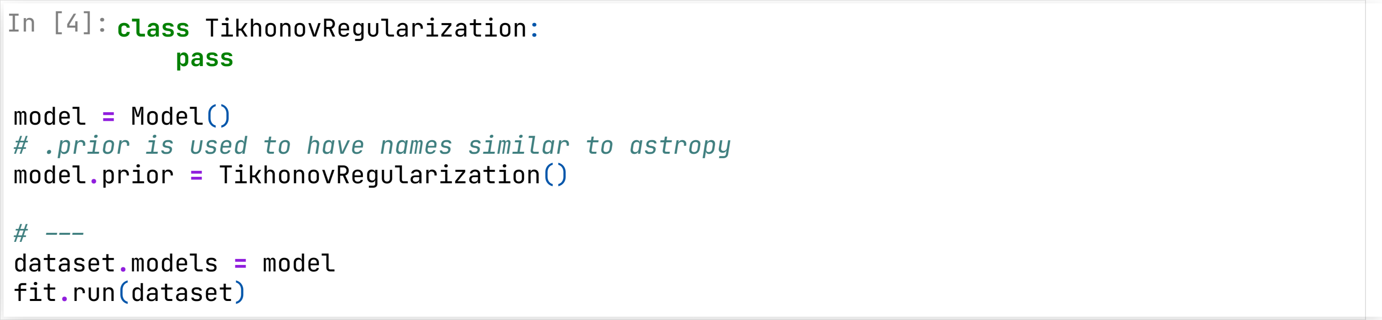
- Serialise fit statistics?
- There might be nuisance parameters?
- Separation of Concerns
- ...

Requirements

- Fulfill all use cases
- As general as possible
- Fit statistic are additive
- Independent of model dimension
- ...

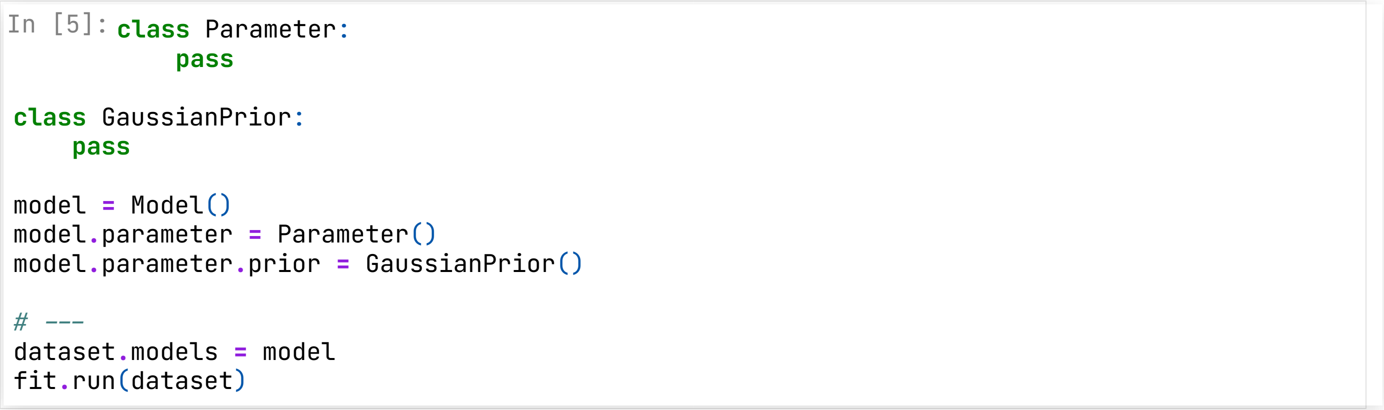


```
class MapDataset:
    def __init__(self, fit_statistic):
        pass

class CashFitStatistic:
    pass

dataset = MapDataset(CashFitStatistic())

# ---
dataset.models = Model()
fit.run(dataset)
```



In [4]:

```python
class TikhonovRegularization:
    pass

model = Model()
# .prior is used to have names similar to astropy
model.prior = TikhonovRegularization()

# ---
dataset.models = model
fit.run(dataset)
```

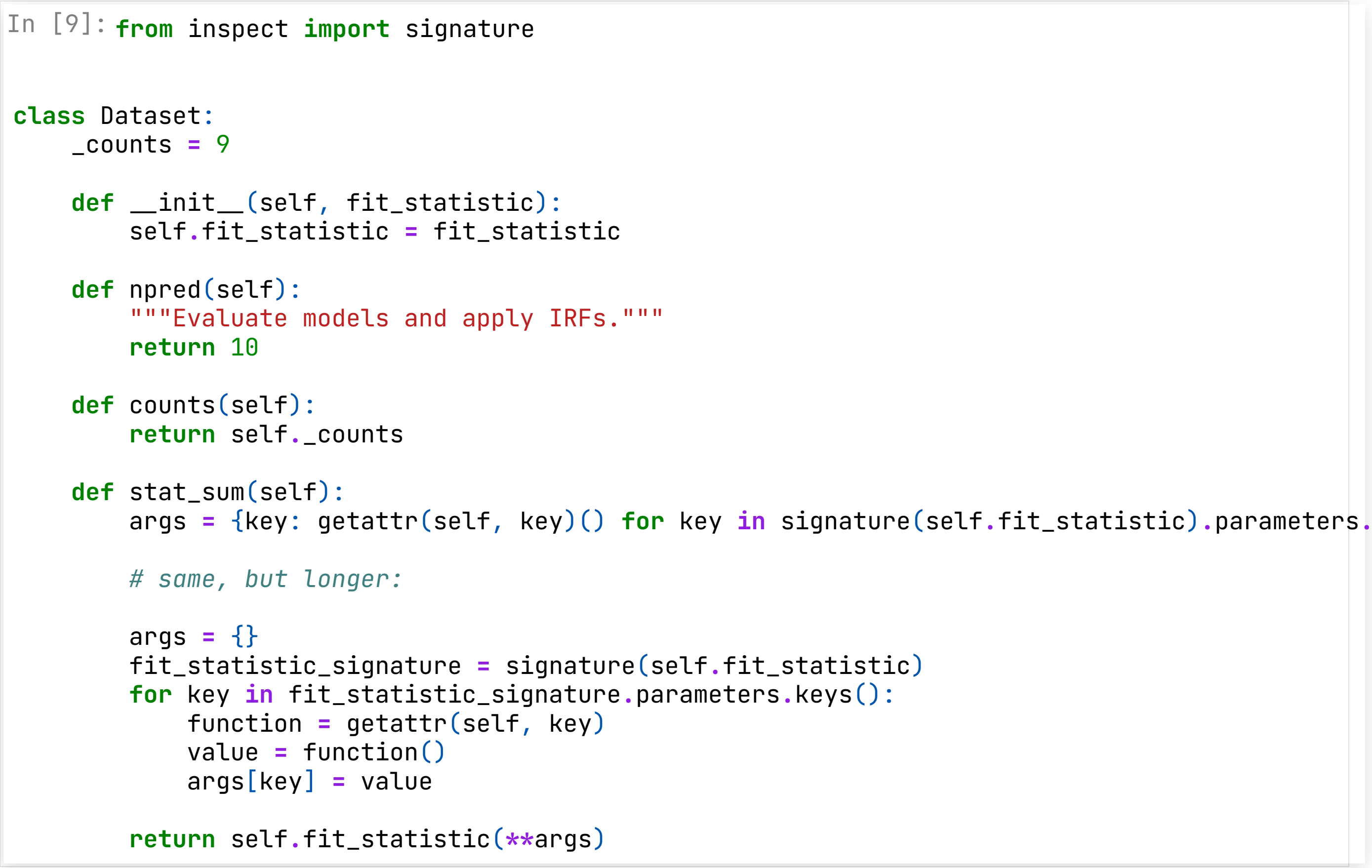


```
In [5]: class Parameter:
            pass

class GaussianPrior:
    pass

model = Model()
model.parameter = Parameter()
model.parameter.prior = GaussianPrior()

# ---
dataset.models = model
fit.run(dataset)
```

Implementations

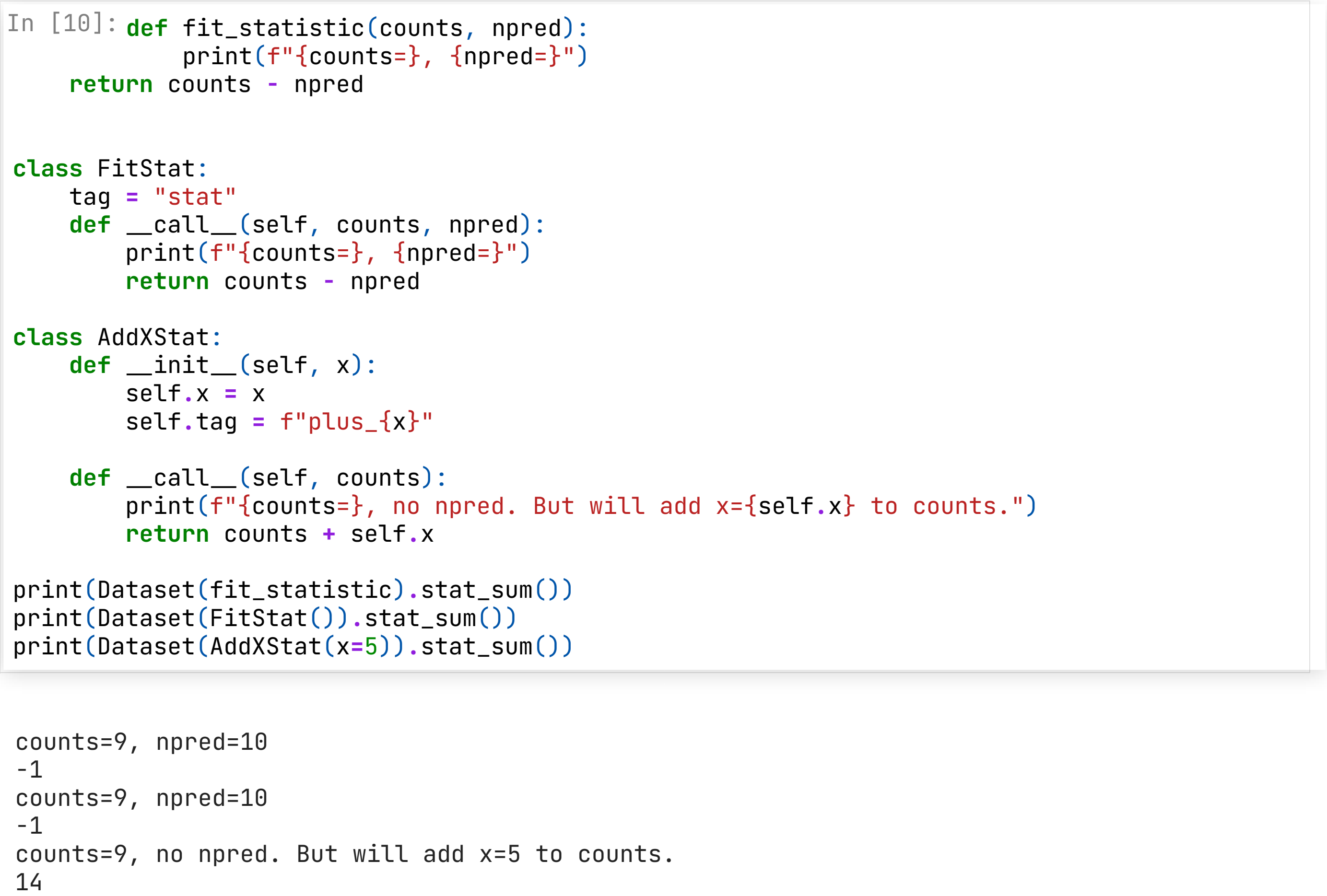
```
In [8]: class CashFitStatistic:
            pass

class MSEFitStatistic:
    tag = "mse"

    def stat_sum(self, dataset):
        counts = dataset.counts
        npred = dataset.npred()
        return (counts - npred) ** 2

class MapDataset:
    def __init__(self, *, fit_statistic=None):
        self.fit_statistic = fit_statistic or CashFitStatistic()
        self.counts = 10

    def npred(self):
        """Evaluate models and apply IRFs."""
        return 9

    def stat_sum(self):
        return self.fit_statistic.stat_sum(self)

dataset = MapDataset(fit_statistic=MSEFitStatistic())
print(dataset.stat_sum())
```

```
1
```



In [9]:

```
from inspect import signature


class Dataset:
    _counts = 9

    def __init__(self, fit_statistic):
        self.fit_statistic = fit_statistic

    def npred(self):
        """Evaluate models and apply IRFs."""
        return 10

    def counts(self):
        return self._counts

    def stat_sum(self):
        args = {key: getattr(self, key)() for key in signature(self.fit_statistic).parameters.keys()

        # same, but longer:

        args = {}
        fit_statistic_signature = signature(self.fit_statistic)
        for key in fit_statistic_signature.parameters.keys():
            function = getattr(self, key)
            value = function()
            args[key] = value

        return self.fit_statistic(**args)
```



```
In [10]: def fit_statistic(counts, npred):
           print(f"{counts=}, {npred=}")
       return counts - npred


class FitStat:
    tag = "stat"
    def __call__(self, counts, npred):
        print(f"{counts=}, {npred=}")
        return counts - npred

class AddXStat:
    def __init__(self, x):
        self.x = x
        self.tag = f"plus_{x}"

    def __call__(self, counts):
        print(f"{counts=}, no npred. But will add x={self.x} to counts.")
        return counts + self.x

print(Dataset(fit_statistic).stat_sum())
print(Dataset(FitStat()).stat_sum())
print(Dataset(AddXStat(x=5)).stat_sum())
```

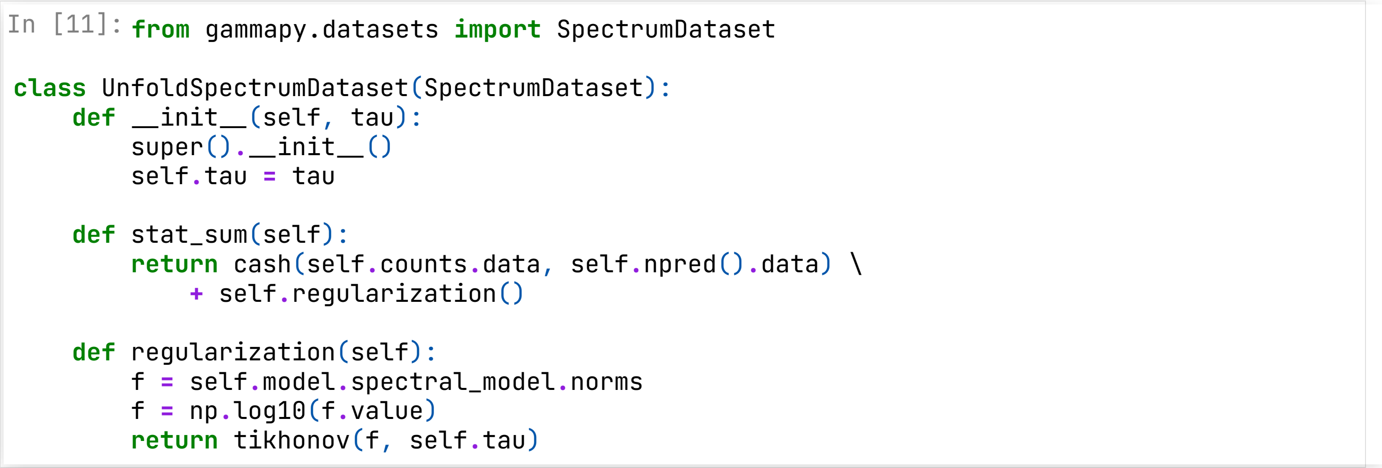
```
counts=9, npred=10
-1
counts=9, npred=10
-1
counts=9, no npred. But will add x=5 to counts.
14
```

Shown were only FitStatistics, but very similar things can be done for Priors/Nuisance on Models.

When we calculate *anything* on a `dataset`, we always have access to `dataset.models`, and thus `dataset.models.prior`.

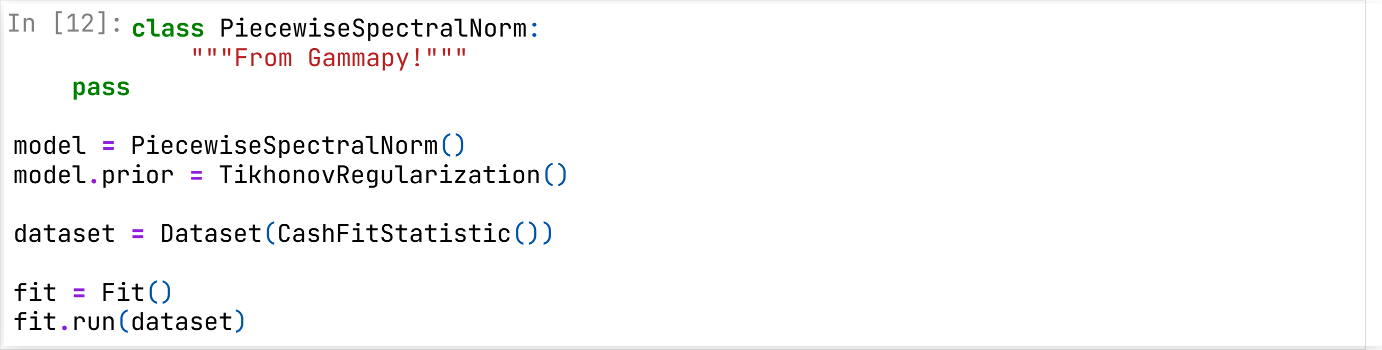
That depends somewhat on the Fitting API, shown later.

Unfolding

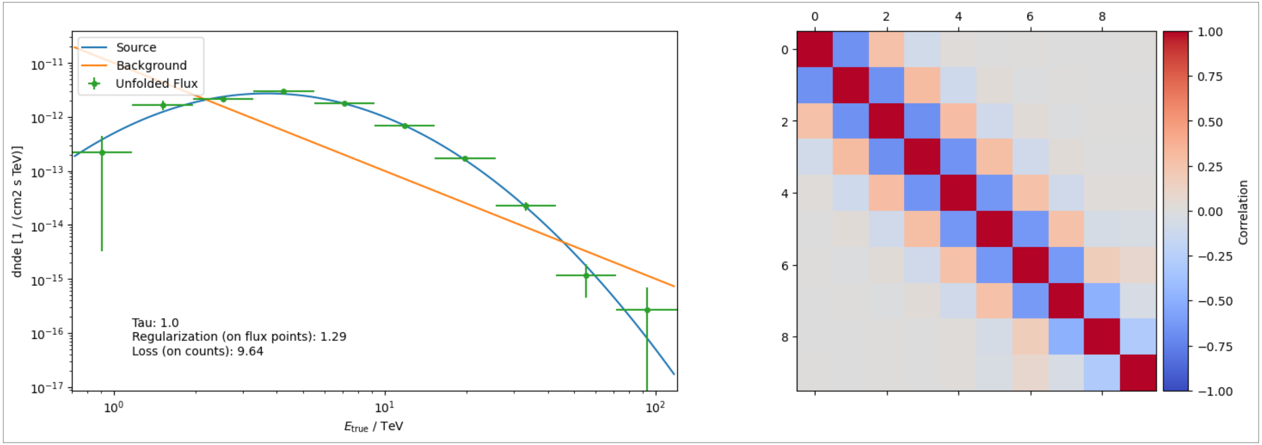


In [11]:

```
from gammapy.datasets import SpectrumDataset

class UnfoldSpectrumDataset(SpectrumDataset):
    def __init__(self, tau):
        super().__init__()
        self.tau = tau

    def stat_sum(self):
        return cash(self.counts.data, self.npred().data) \
            + self.regularization()

    def regularization(self):
        f = self.model.spectral_model.norms
        f = np.log10(f.value)
        return tikhonov(f, self.tau)
```



```
In [12]: class PiecewiseSpectralNorm:
            """From Gammapy!"""
    pass

model = PiecewiseSpectralNorm()
model.prior = TikhonovRegularization()

dataset = Dataset(CashFitStatistic())

fit = Fit()
fit.run(dataset)
```


Source
Background
Unfolded Flux
Tau: 1.0
Regularization (on flux points): 1.29
Loss (on counts): 9.64
dnde [1 / (cm2 s TeV)]
10^{-11}
10^{-12}
10^{-13}
10^{-14}
10^{-15}
10^{-16}
10^{-17}
10^{0}
10^{1}
10^{2}
E_{true} / TeV
0
2
4
6
8
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
Correlation

Proposal for the Fit API

Proposal 1

Similar to Astropy

```
In [13]: class Fit:
             def run(self, model, datasets):
         pass
```

```
In [14]: model = Model()

dataset.fit_statistic = CashFitStatistic()

fit = Fit()

fit.run(model, dataset)
```

Proposal 2

Similar to Sherpa

```
In [15]: class Fit:
            def __init__(self, model, dataset):
        pass

    def run(self):
        pass
```

```
In [16]: model = Model()

dataset.fit_statistic = CashFitStatistic()

fit = Fit(model, dataset)

fit.run()
```

Proposal 3

Similar to scikit-learn

```
In [17]: class Model:
             def fit(self, datasets):
         pass
```

```
In [18]: model = Model()

dataset.fit_statistic = CashFitStatistic()

model.fit(dataset)
```

Proposal 4

```
In [19]: class Fit:
      def __init__(self, model):
    pass

  def run(self, datasets):
    pass
```

```
In [20]: model = Model()

dataset.fit_statistic = CashFitStatistic()

fit = Fit(model)

fit.run(dataset)
```

Depending on whether or not this Fitting API is included in the FitStatistics API the implementation details of the evaluation of `model.prior` and `model.parameter.prior` change.

Coding Sprint

I want and need your help.

Goals

- Collect (more) use-cases
- Discuss implementations
- Write a PIG