PYTHON MODELING & FITTING PACKAGES

Christoph Deil, MPIK Heidelberg
Deil.Christoph@gmail.com

HSF PyHEP WG call
Sep 11, 2019

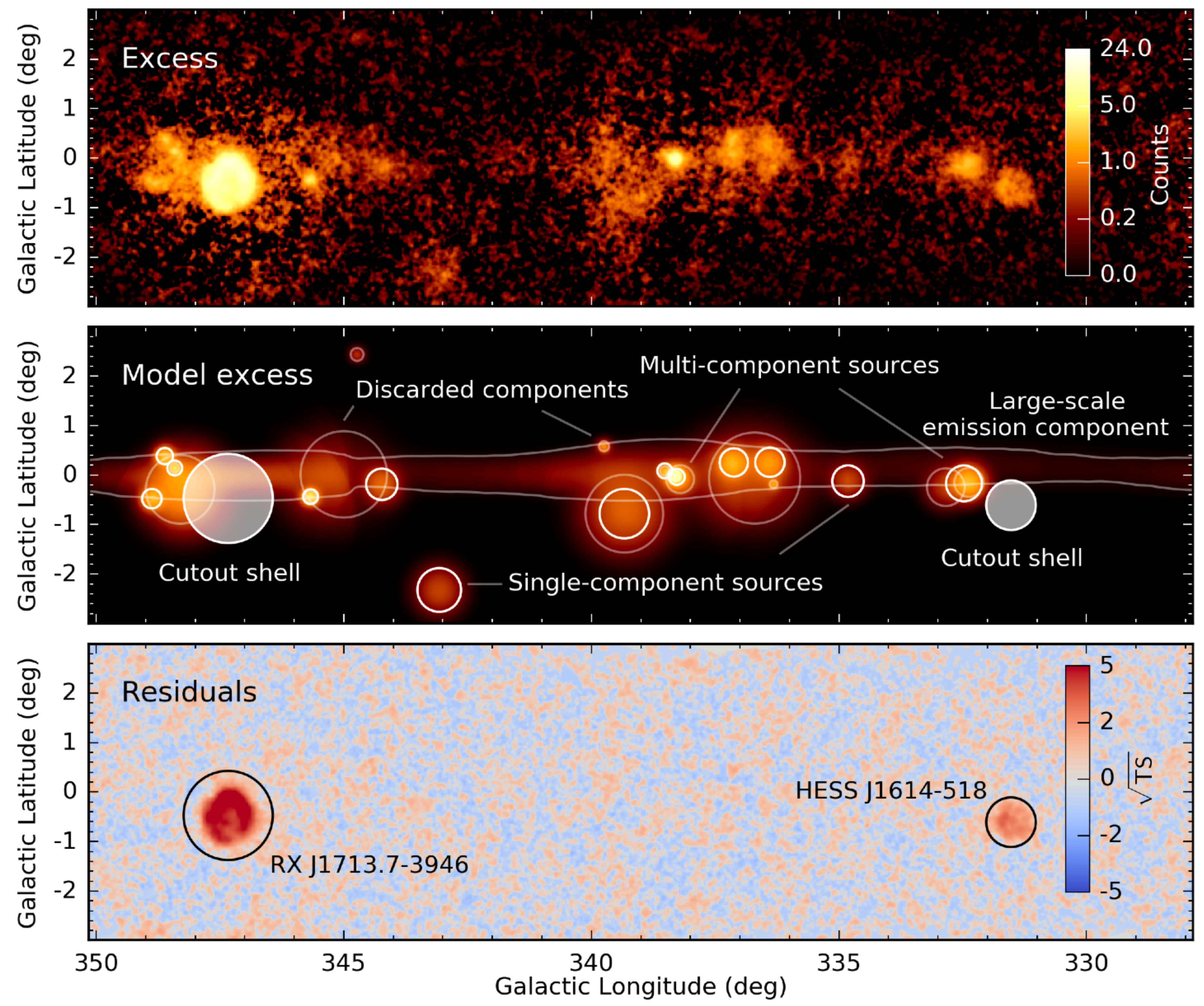
MY KEY POINTS

1. Python is great for modeling & fitting.
2. Python is terrible for modeling & fitting.
3. We should make it better and collaborate more.

ABOUT ME

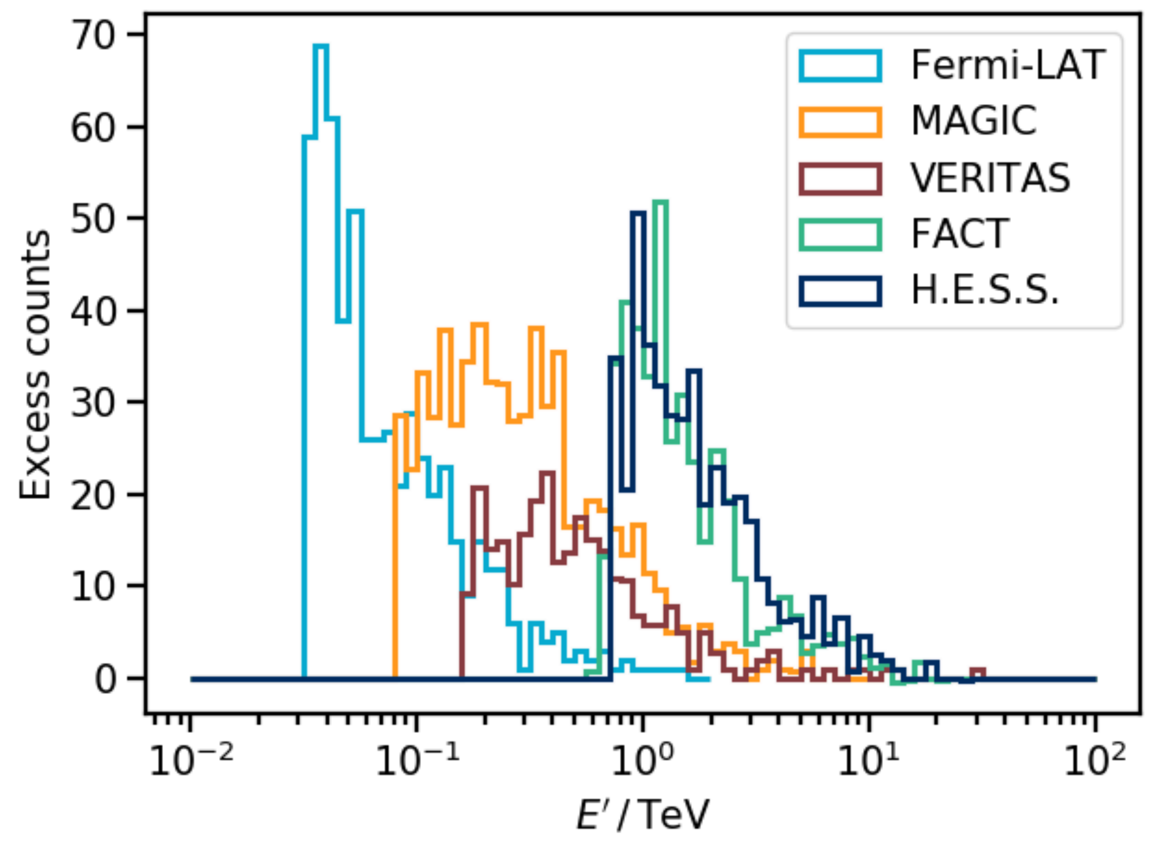
- Christoph Deil, gamma-ray astronomer from Heidelberg
- Started with C++ & ROOT, discovered Python 10 years ago.
- Heavy user and very interested in modeling & fitting in Python
- Use many packages: scipy, iminuit, scikit-learn, Astropy, Sherpa, …
- Develop Gammapy for modeling & fitting gamma-ray astronomy data

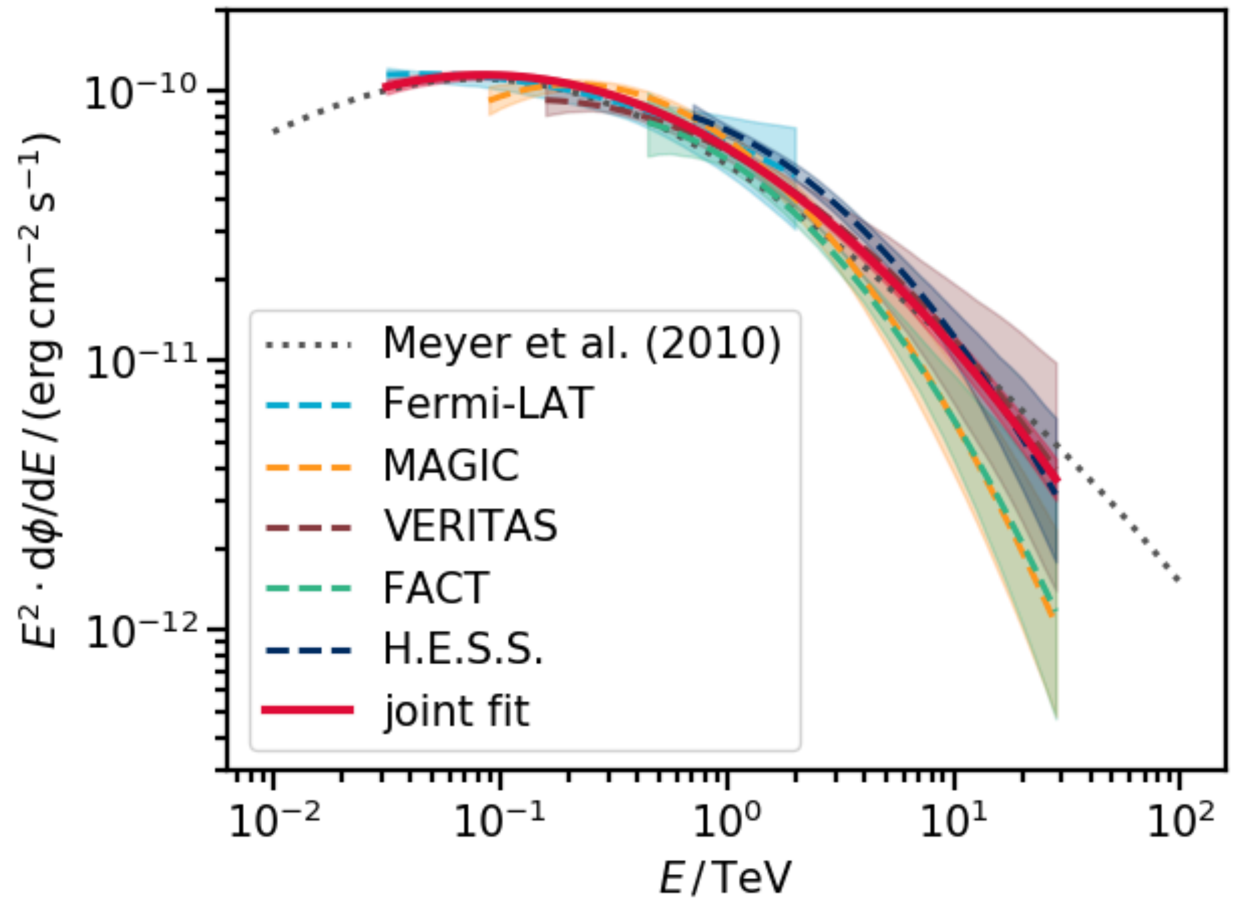
MAX-PLANCK-INSTITUT FÜR KERNPHYSIK

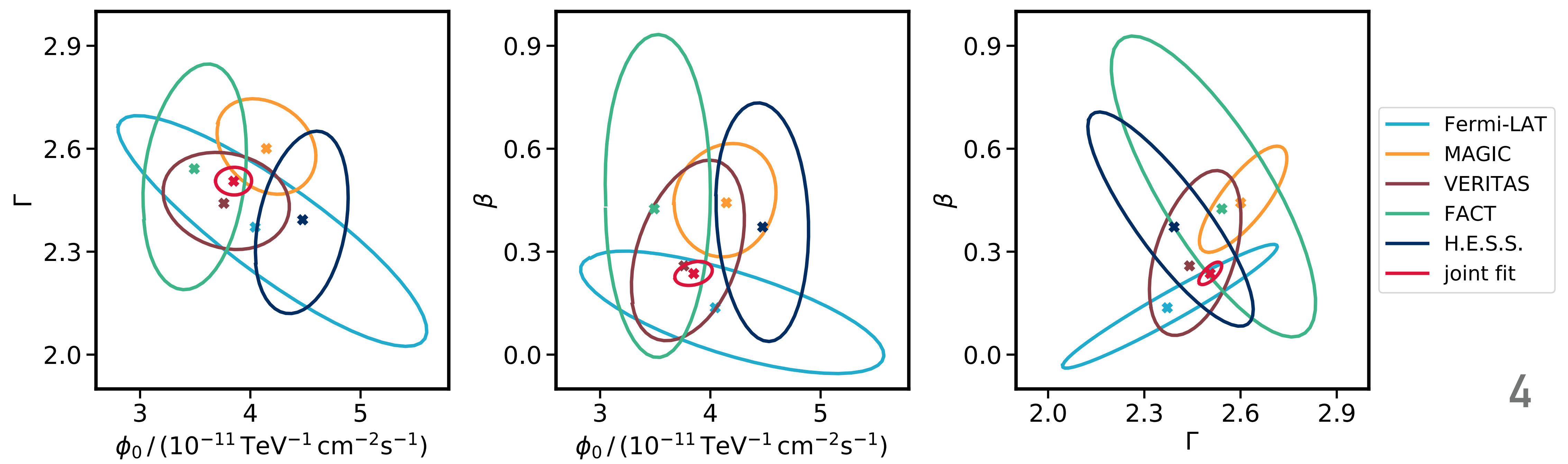


GAMMA-RAY ASTRONOMY

- High-level analysis similar to HEP?
- Data: unbinned event lists (lon, lat, energy, time), or binned 1d, 2d, 3d counts
- Instrument response: effective area, plus spatial (PSF) and energy dispersion
- Models: Spatial (lon/lat) and spectra (energy)
- Fitting: Forward-fold model with instrument response. Poisson likelihood.
- Goal: inference about models & parameters







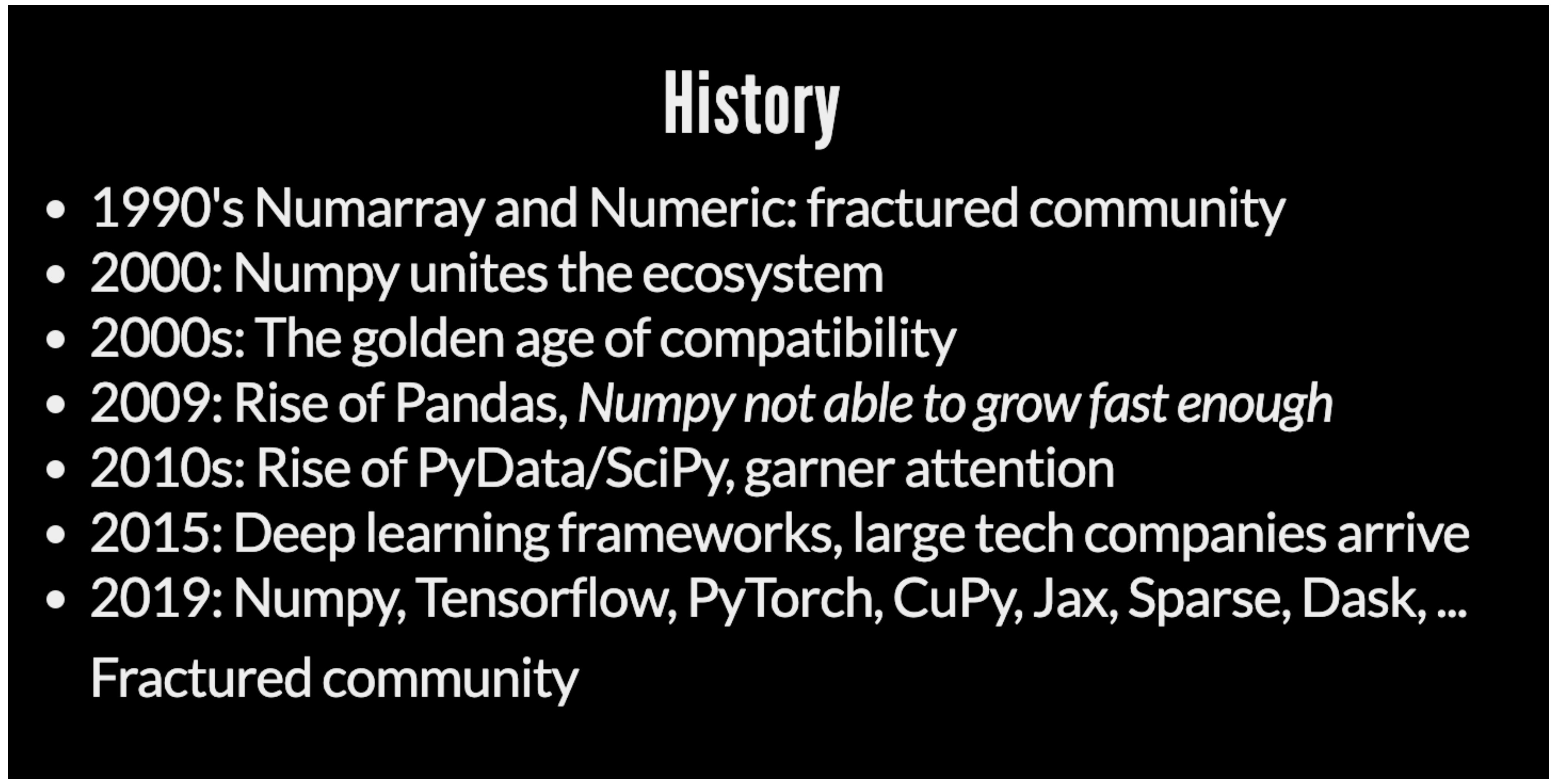
PYTHON IS GREAT FOR MODELING & FITTING

- Very dynamic and easy language
- Many packages available:
  - low-level optimisation packages: scipy.optimize, emcee, …
  - mid-level fitting packages: lmfit, statsmodels, scikit-learn, iminuit
  - high-level modeling frameworks: Sherpa, astropy.modeling, …
  - all-inclusive solutions: RooFit, Tensorflow, pytorch, pymc, …
- Can write very complex analyses in a day.
- Can write general or domain-specific modeling & fitting package in a week.

PYTHON IS TERRIBLE FOR MODELING AND FITTING

- Eco-system is fractured, a lot of duplication of effort and little interoperability.
  Hard to use scipy.optimize & iminuit & pymc & tensorflow & … together.
  Have to choose one framework and write all models & analysis code there.

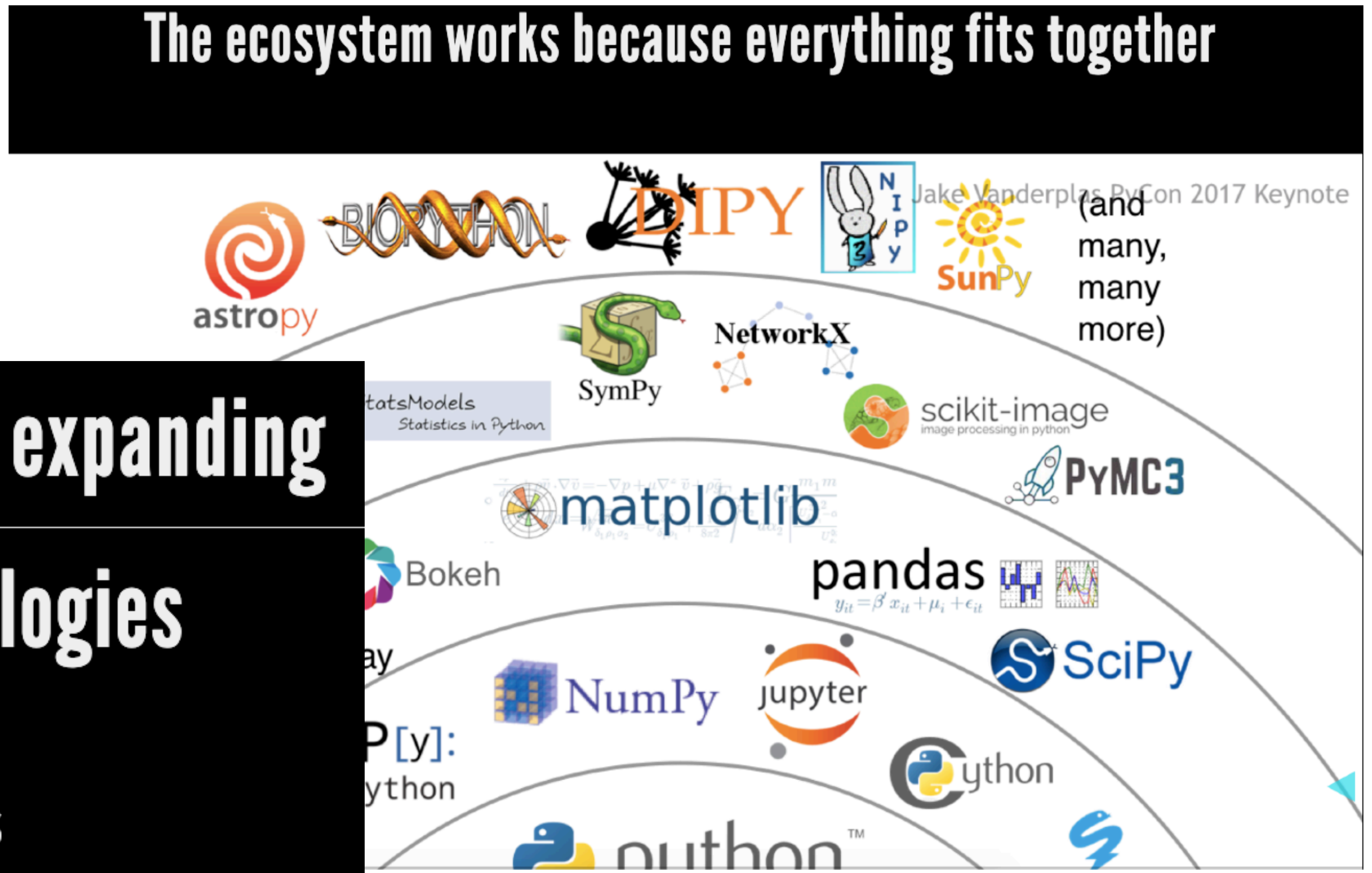
- As a user (astronomer): what to use to implement my analysis?
- As a domain-specific package developer (Gammapy): what packages to build on?
- As a potential framework contributor: where to put my effort?



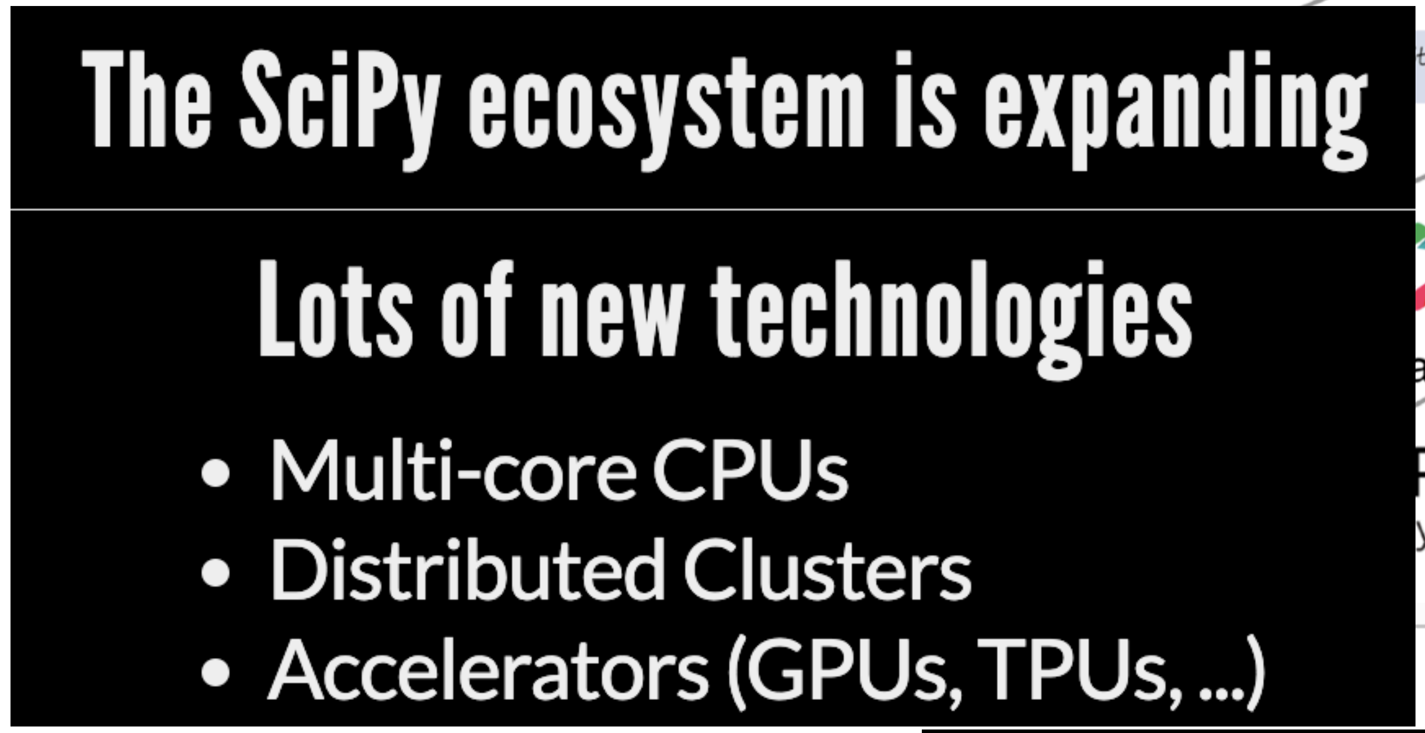
History

- 1990's Numarray and Numeric: fractured community
- 2000: Numpy unites the ecosystem
- 2000s: The golden age of compatibility
- 2009: Rise of Pandas, *Numpy not able to grow fast enough*
- 2010s: Rise of PyData/SciPy, garner attention
- 2015: Deep learning frameworks, large tech companies arrive
- 2019: Numpy, Tensorflow, PyTorch, CuPy, Jax, Sparse, Dask, ...

  Fractured community



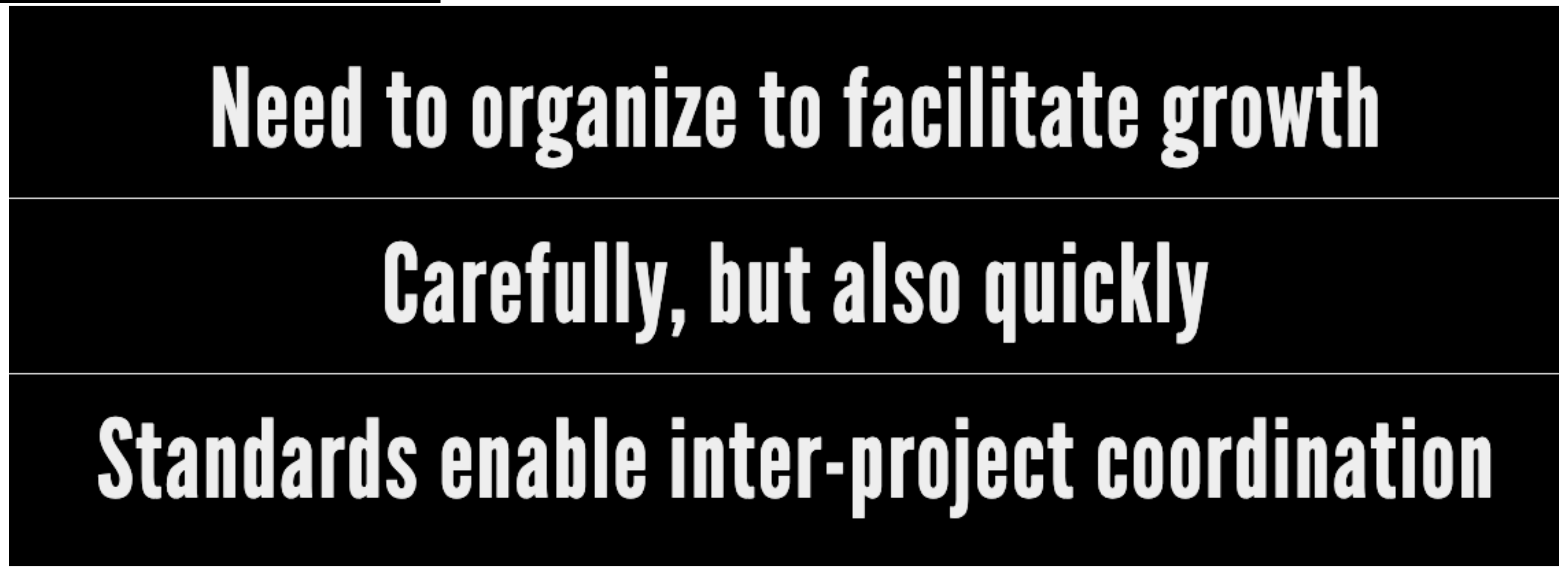
The ecosystem works because everything fits together



The SciPy ecosystem is expanding

Lots of new technologies

- Multi-core CPUs
- Distributed Clusters
- Accelerators (GPUs, TPUs, ...)



Need to organize to facilitate growth

Carefully, but also quickly

Standards enable inter-project coordination

FRACTURED COMMUNITY

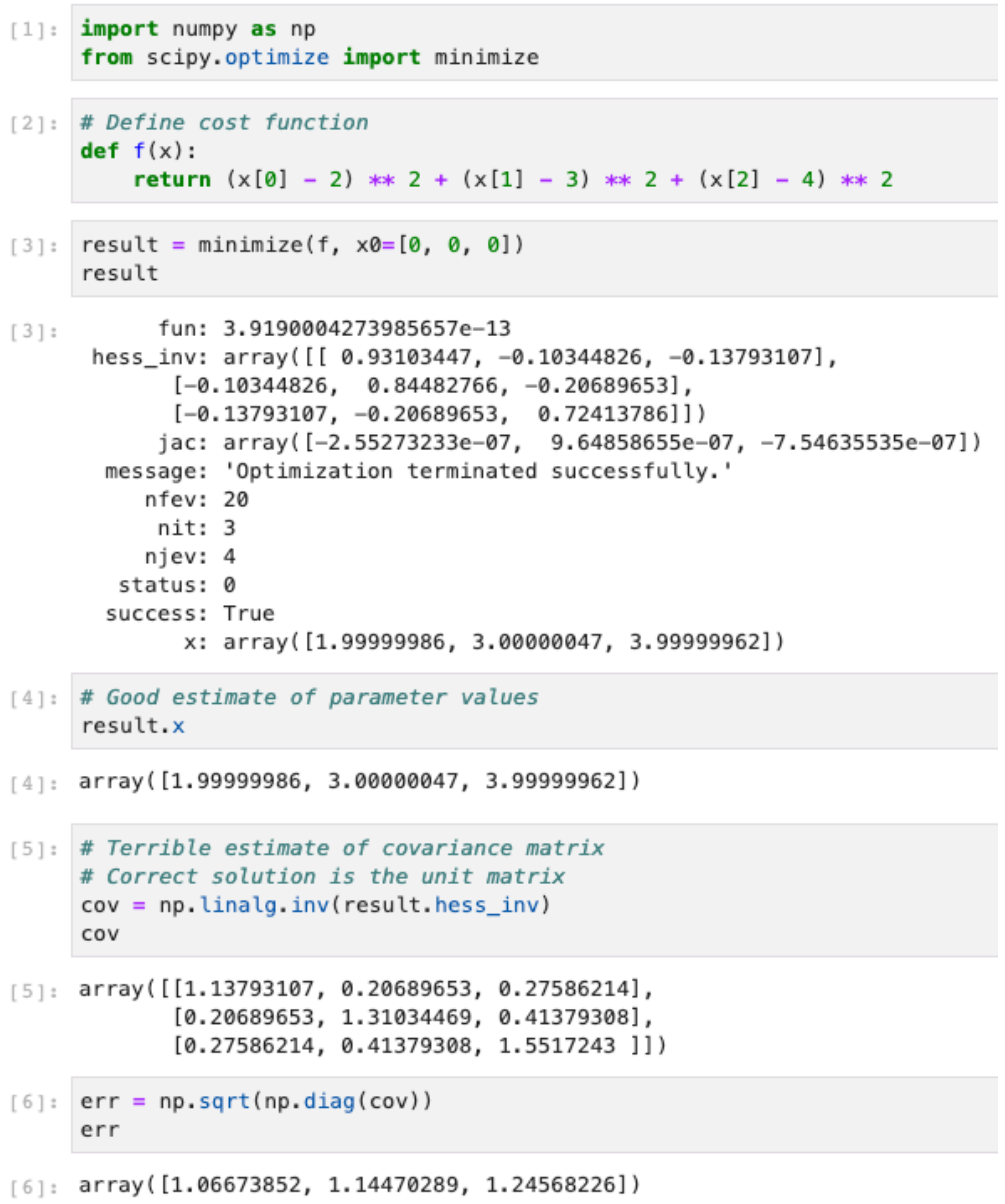
- Matthew Rocklin at Scipy 2019 "*Refactoring the Ecosystem for Heterogeneity*" See slides and video.
- Matthew talks about scientific computing in Python, but I think the same comments apply for modeling & fitting
- Even before Tensorflow et al., there never was a common standard or leading Python package for modeling & fitting *(beyond scipy.optimize, which is very low-level)*

INTERFACES, LIBRARIES AND FRAMEWORKS

- Why are the Python modelling & fitting packages so fragmented?
- Scientific computing packages like scipy, scikit-image, scikit-learn, Astropy, … are interoperable because they are **libraries** using common objects (Python & Numpy) or at least common compatible interfaces (Numpy & dark arrays).
- The one clear **interface** that exists is the cost function passed to an optimiser. See e.g. scipy.optimize.minimize and iminuit.minimize and emcee.
- As soon as a Model or Parameter class is introduced, a **framework** is created. And it's incompatible with any other existing framework for modeling & fitting.
- It's very hard to impossible to avoid creating a framework, if you want things like parameter limits, linked parameters, units, compound models, 1D/2D/3D data, …

EXISTING PACKAGES

Quick overview of a few that I've used (astronomer bias). There are 100s in widespread use.



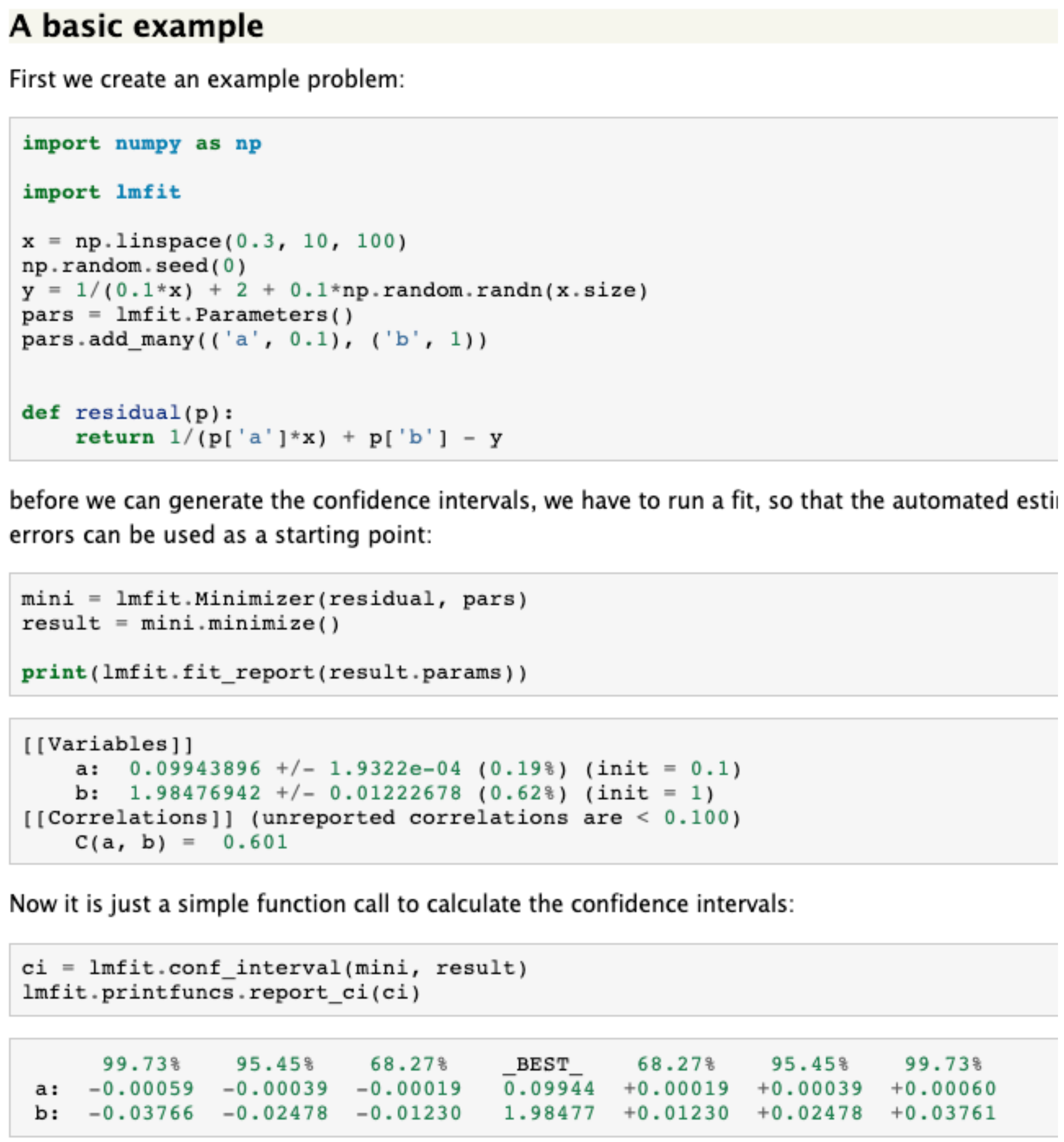
```
[1]: import numpy as np
     from scipy.optimize import minimize

[2]: # Define cost function
     def f(x):
         return (x[0] - 2) ** 2 + (x[1] - 3) ** 2 + (x[2] - 4) ** 2

[3]: result = minimize(f, x0=[0, 0, 0])
     result

[3]:       fun: 3.9190004273985657e-13
      hess_inv: array([[ 0.93103447, -0.10344826, -0.13793107],
            [-0.10344826,  0.84482766, -0.20689653],
            [-0.13793107, -0.20689653,  0.72413786]])
           jac: array([-2.55273233e-07,  9.64858655e-07, -7.54635535e-07])
       message: 'Optimization terminated successfully.'
          nfev: 20
           nit: 3
          njev: 4
        status: 0
       success: True
             x: array([1.99999986, 3.00000047, 3.99999962])

[4]: # Good estimate of parameter values
     result.x

[4]: array([1.99999986, 3.00000047, 3.99999962])

[5]: # Terrible estimate of covariance matrix
     # Correct solution is the unit matrix
     cov = np.linalg.inv(result.hess_inv)
     cov

[5]: array([[1.13793107, 0.20689653, 0.27586214],
            [0.20689653, 1.31034469, 0.41379308],
            [0.27586214, 0.41379308, 1.5517243 ]])

[6]: err = np.sqrt(np.diag(cov))
     err

[6]: array([1.06673852, 1.14470289, 1.24568226])
```

SCIPY.OPTIMIZE

- ➤ https://docs.scipy.org/doc/scipy/reference/tutorial/optimize.html
- ➤ Wraps and re-implements some common optimisation algorithms.
- ➤ Doesn’t do likelihood profile analysis or parameter error estimation (apart from least_square and curve_fit special case)
- ➤ Single-Function API
- ➤ User interface is the cost function, either with Python or Numpy objects (optional; can use analytical gradient)
- ➤ A low-level library, not a framework. Wrapped by others (lmfit, statsmodels, …)

STATSMODELS & SCIKIT-LEARN

- https://www.statsmodels.org
  - Statistical models & tests
  - Big focus on parameter error estimation
- https://scikit-learn.org
  - Machine learning
  - Little or no parameter error estimation
- Focus on specific pre-defined very commonly used models.
- Not clear to me if useful for more general applications, like in HEP & astro.
- Partly scipy.optimize based, partly custom optimisers used in the background



A basic example

First we create an example problem:

```
import numpy as np

import lmfit

x = np.linspace(0.3, 10, 100)
np.random.seed(0)
y = 1/(0.1*x) + 2 + 0.1*np.random.randn(x.size)
pars = lmfit.Parameters()
pars.add_many(('a', 0.1), ('b', 1))


def residual(p):
    return 1/(p['a']*x) + p['b'] - y
```

before we can generate the confidence intervals, we have to run a fit, so that the automated esti
errors can be used as a starting point:

```
mini = lmfit.Minimizer(residual, pars)
result = mini.minimize()

print(lmfit.fit_report(result.params))
```

```
[[Variables]]
    a:  0.09943896 +/- 1.9322e-04 (0.19%) (init = 0.1)
    b:  1.98476942 +/- 0.01222678 (0.62%) (init = 1)
[[Correlations]] (unreported correlations are < 0.100)
    C(a, b) =  0.601
```

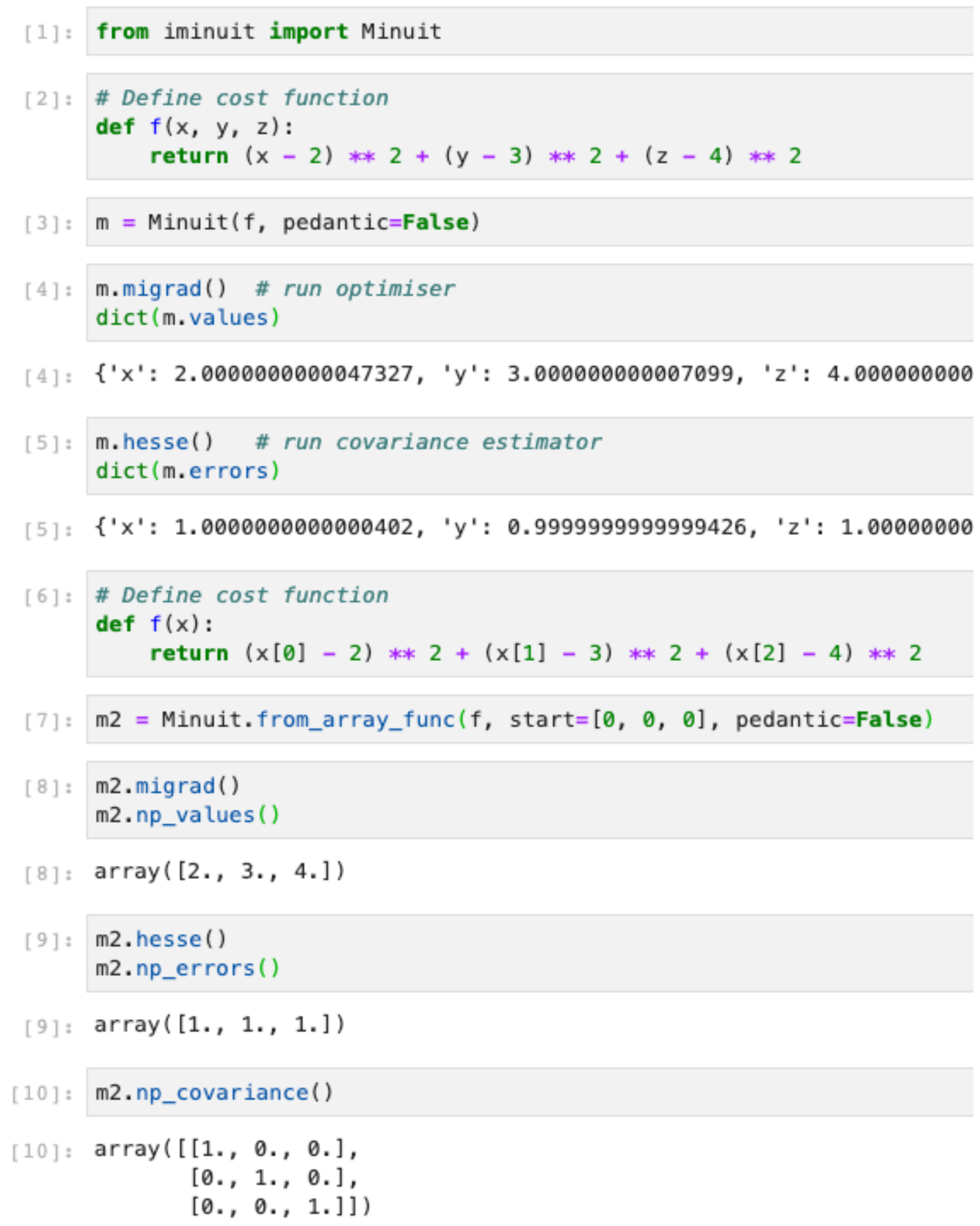
Now it is just a simple function call to calculate the confidence intervals:

```
ci = lmfit.conf_interval(mini, result)
lmfit.printfuncs.report_ci(ci)
```

```
      99.73%    95.45%    68.27%    _BEST_    68.27%    95.45%    99.73%
 a:  -0.00059  -0.00039  -0.00019   0.09944  +0.00019  +0.00039  +0.00060
 b:  -0.03766  -0.02478  -0.01230   1.98477  +0.01230  +0.02478  +0.03761
```

LMFIT

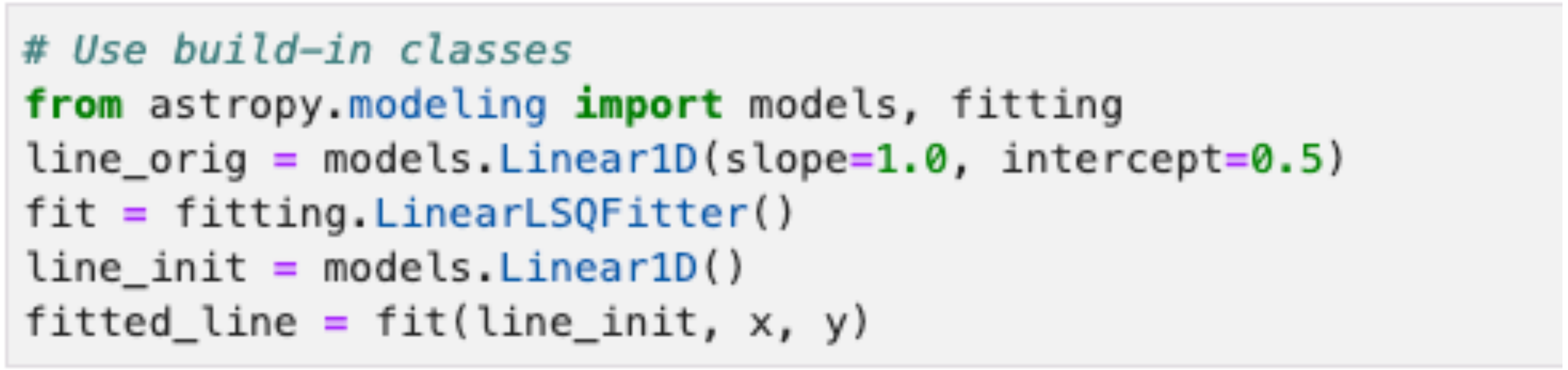
- https://lmfit.github.io/lmfit-py/
- Thin layer on top of scipy.optimize, a mini framework with classes:
  Parameter, Parameters, Model
- Handles parameter bounds, constraints, computes errors via likelihood profile analysis.
- Not clear to me if it's a general framework we could use for any analysis, or if parts are hard-coded on least squares and Levenberg-Marquardt method
- Developed & maintained by a physicist: Matt Newville (Chicago).



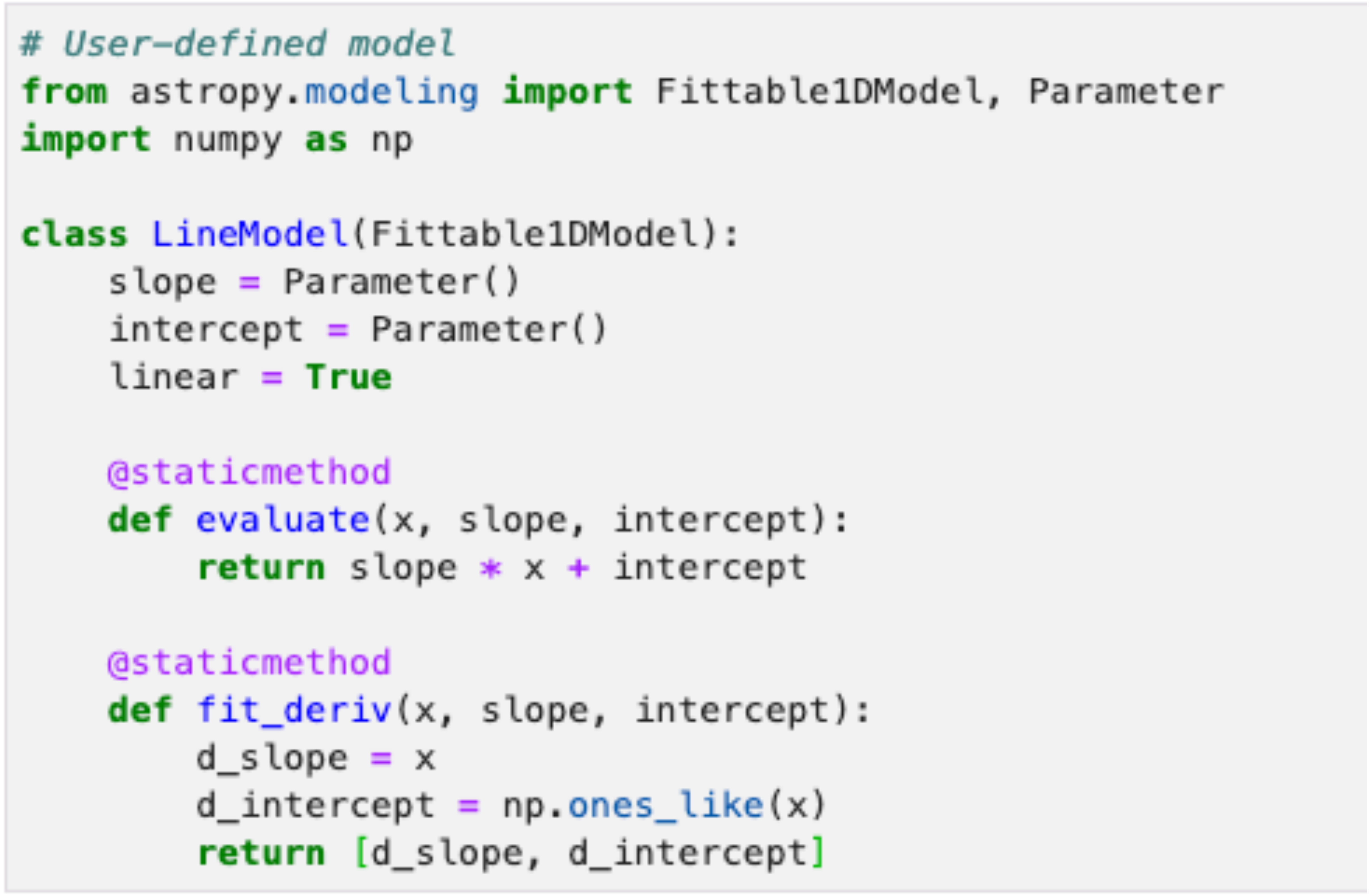
```
[1]: from iminuit import Minuit

[2]: # Define cost function
     def f(x, y, z):
         return (x - 2) ** 2 + (y - 3) ** 2 + (z - 4) ** 2

[3]: m = Minuit(f, pedantic=False)

[4]: m.migrad()  # run optimiser
     dict(m.values)

[4]: {'x': 2.0000000000047327, 'y': 3.000000000007099, 'z': 4.000000000

[5]: m.hesse()   # run covariance estimator
     dict(m.errors)

[5]: {'x': 1.0000000000000402, 'y': 0.9999999999999426, 'z': 1.00000000

[6]: # Define cost function
     def f(x):
         return (x[0] - 2) ** 2 + (x[1] - 3) ** 2 + (x[2] - 4) ** 2

[7]: m2 = Minuit.from_array_func(f, start=[0, 0, 0], pedantic=False)

[8]: m2.migrad()
     m2.np_values()

[8]: array([2., 3., 4.])

[9]: m2.hesse()
     m2.np_errors()

[9]: array([1., 1., 1.])

[10]: m2.np_covariance()

[10]: array([[1., 0., 0.],
             [0., 1., 0.],
             [0., 0., 1.]])
```

IMINUIT

- https://iminuit.readthedocs.io
- Python wrapper for MINUIT C++ library
- Single-class API
- User interface is the cost function, either with Python or Numpy objects (optional; can use analytical gradient)
- Great if it does what you need.
- Not easy to build upon: stateful interface and API is not a layered and extensible.
- Mini framework: the Minuit object has methods to handle parameters or make likelihood profiles and plots



```
# Use build-in classes
from astropy.modeling import models, fitting
line_orig = models.Linear1D(slope=1.0, intercept=0.5)
fit = fitting.LinearLSQFitter()
line_init = models.Linear1D()
fitted_line = fit(line_init, x, y)
```

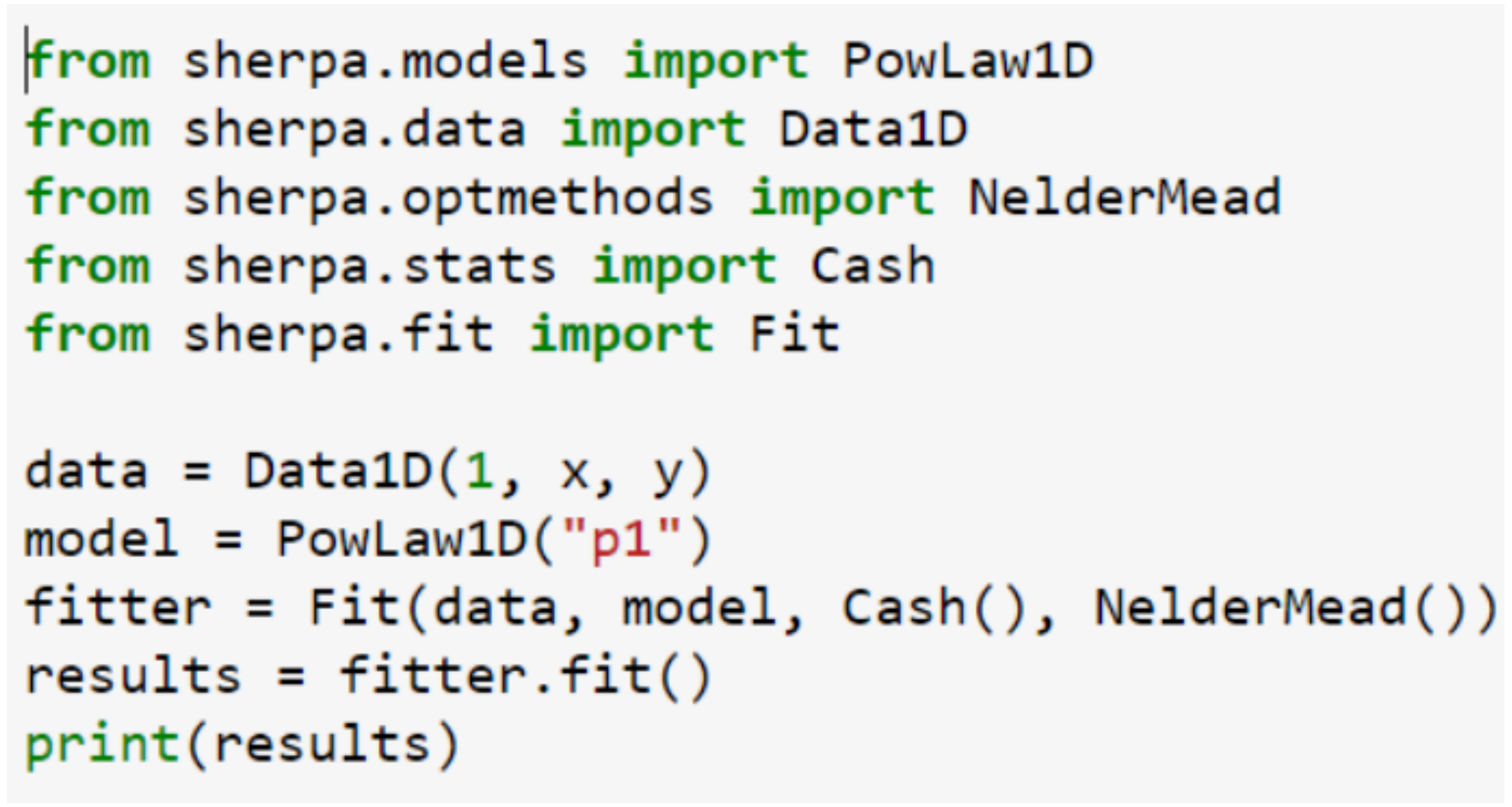


```
# User-defined model
from astropy.modeling import Fittable1DModel, Parameter
import numpy as np

class LineModel(Fittable1DModel):
    slope = Parameter()
    intercept = Parameter()
    linear = True

    @staticmethod
    def evaluate(x, slope, intercept):
        return slope * x + intercept

    @staticmethod
    def fit_deriv(x, slope, intercept):
        d_slope = x
        d_intercept = np.ones_like(x)
        return [d_slope, d_intercept]
```

ASTROPY.MODELING

- https://docs.astropy.org/en/latest/modeling/
- A framework: Parameter, Model, FittableModel, Fittable1DModel, …
- Some Fitter classes (mostly connecting to scipy.optimize), no Data classes
- Supports complex models: linked parameters, compound models, transformations, units, …
- Recently simplified Parameter and compound model design, see here.
- Could also be interesting for others: ASDF (Advanced Scientific Data Format) for model serialisation, see https://asdf.readthedocs.io

SHERPA



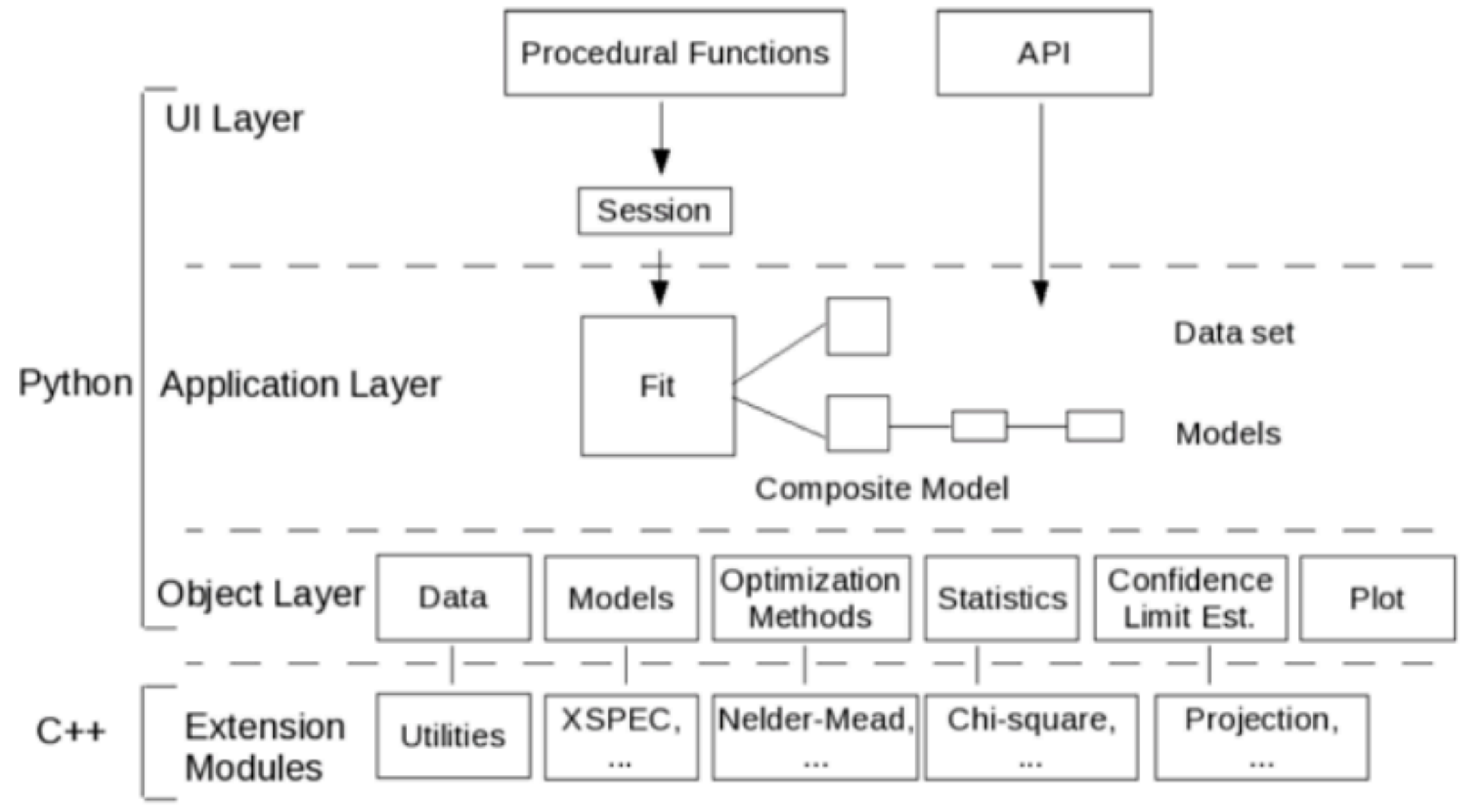
```python
from sherpa.models import PowLaw1D
from sherpa.data import Data1D
from sherpa.optmethods import NelderMead
from sherpa.stats import Cash
from sherpa.fit import Fit

data = Data1D(1, x, y)
model = PowLaw1D("p1")
fitter = Fit(data, model, Cash(), NelderMead())
results = fitter.fit()
print(results)
```

Proceedings of the 8th Python in Science Conference (SciPy 2009)

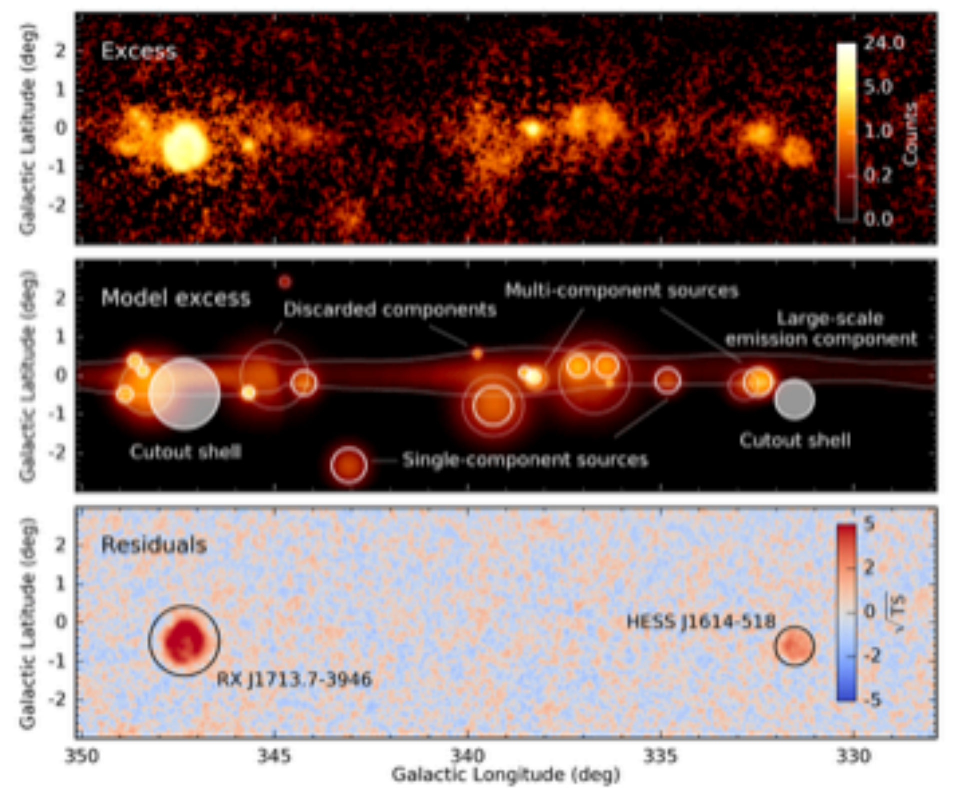
Sherpa: 1D/2D modeling and fitting in Python

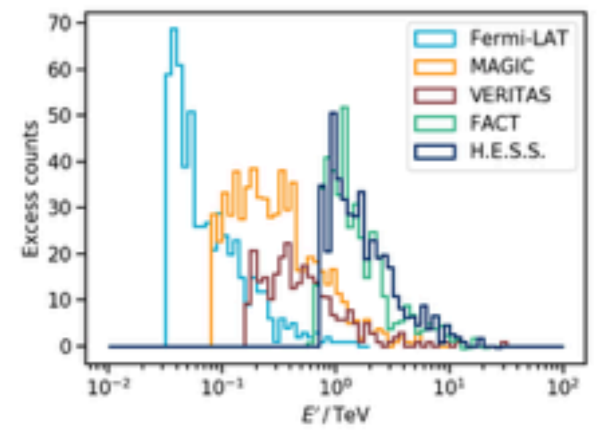
2009

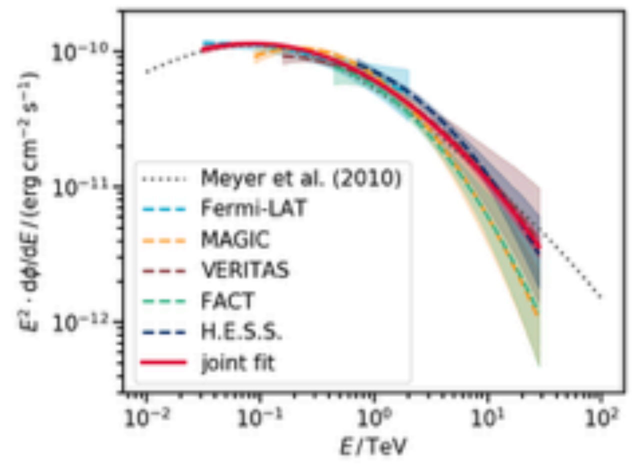
Brian L. Refsdal (brefsdal@head.cfa.harvard.edu) – *Harvard-Smithsonian Center for Astrophysics*, USA
Stephen M. Doe (sdoe@head.cfa.harvard.edu) – *Harvard-Smithsonian Center for Astrophysics*, USA
Dan T. Nguyen (dtn@head.cfa.harvard.edu) – *Harvard-Smithsonian Center for Astrophysics*, USA
Aneta L. Siemiginowska (aneta@head.cfa.harvard.edu) – *Harvard-Smithsonian Center for Astrophysics*, USA

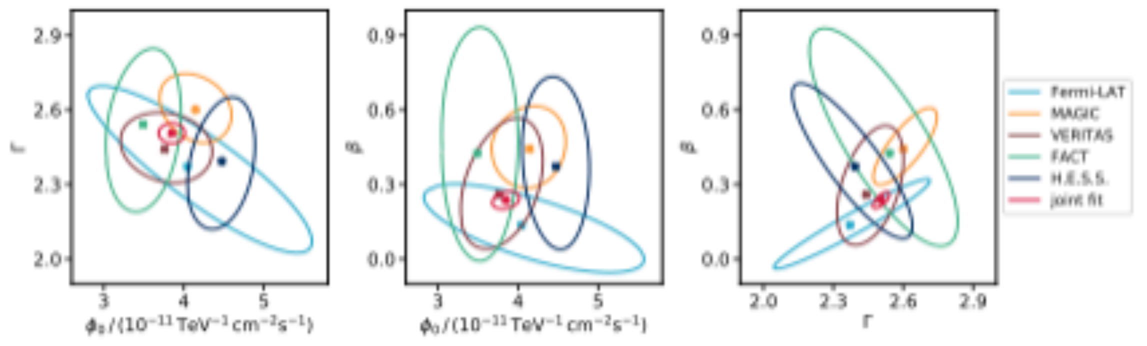


- https://sherpa.readthedocs.io
  https://github.com/sherpa/sherpa
- Modeling & fitting package for Chandra X-ray satellite. Started 20 years ago with S-lang, migrated to Python in 2007, moved to Github in 2015, continually improved …
- Built-in optimisers, error estimators, plotting code … a full framework.
- Based on Numpy. Object-oriented, extensible API, and procedural session-based user interface on top.
- Bridge to astropy: SABA
- References: Scipy 2009, Scipy 2011, 1, 2



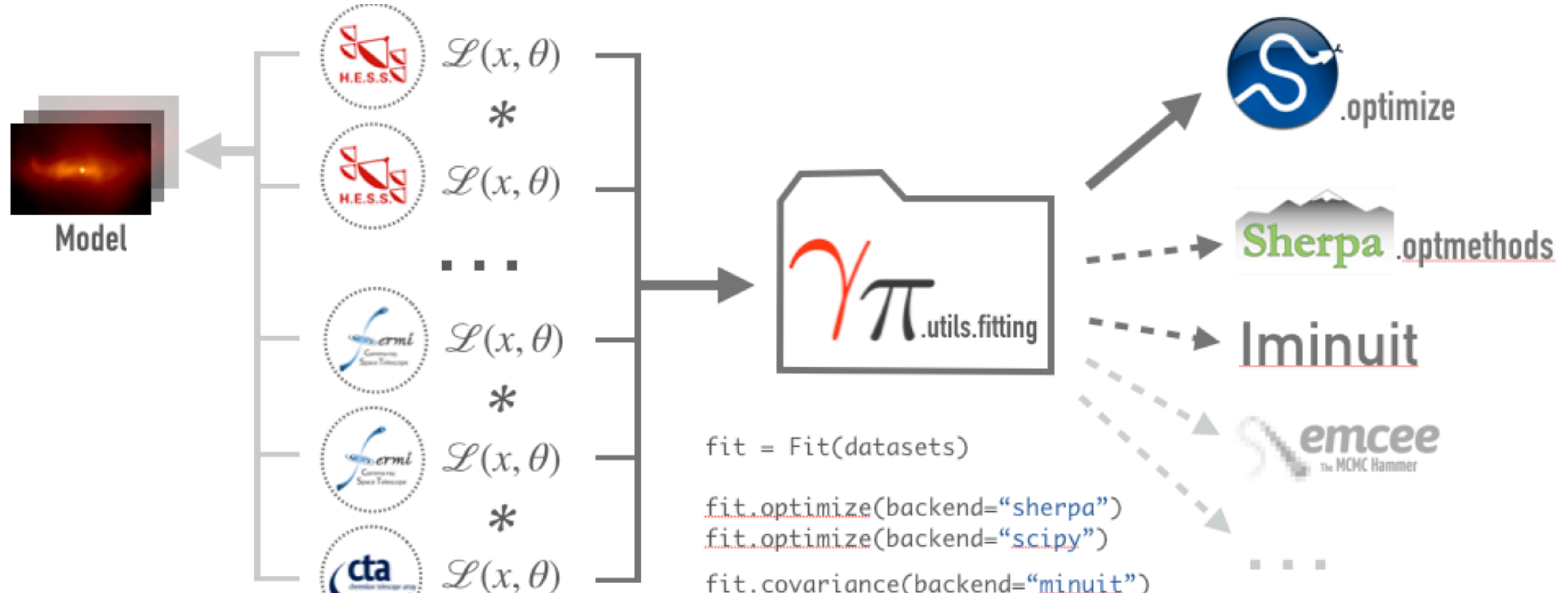






GAMMA-RAY ASTRONOMY

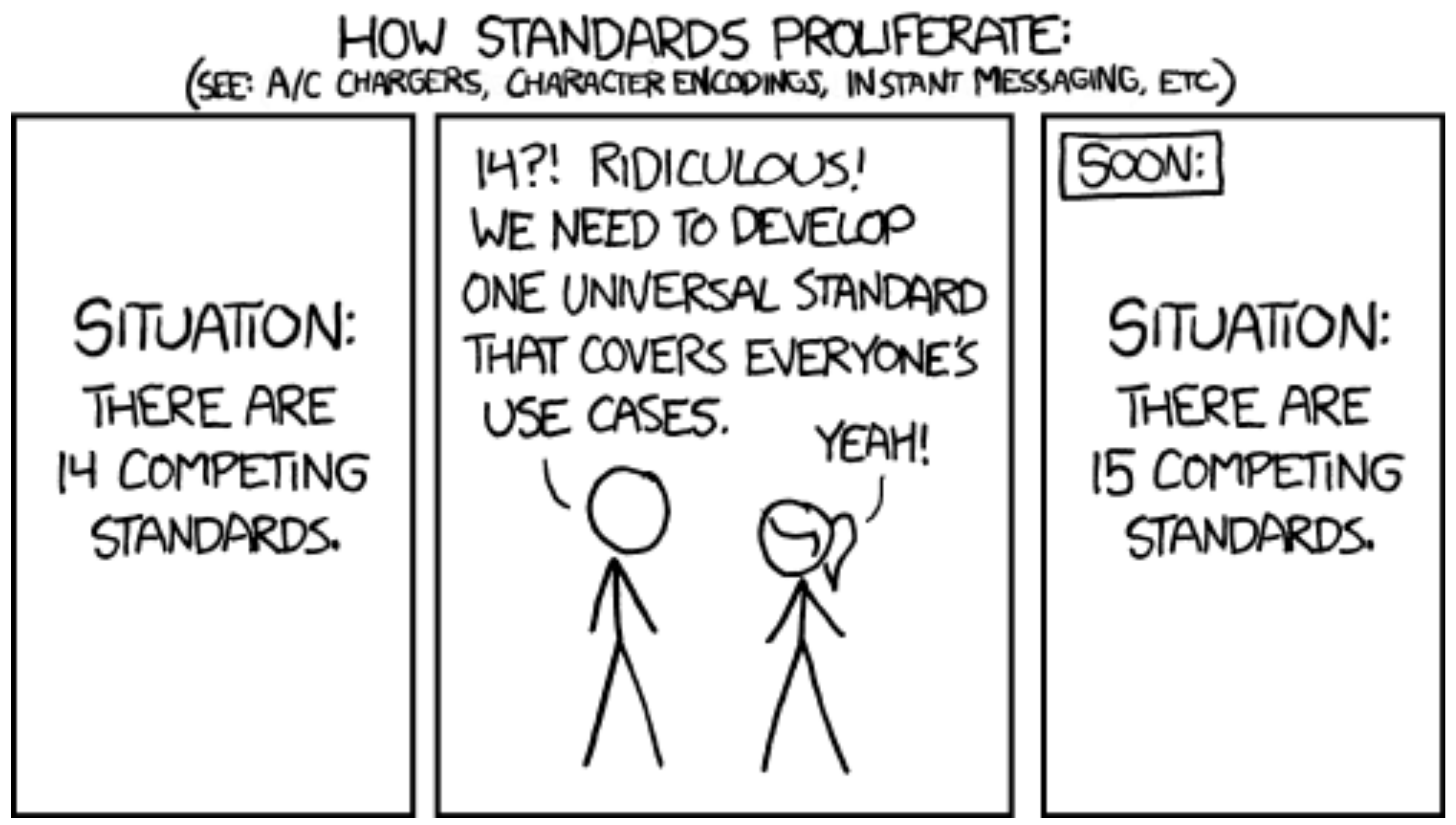
- High-level analysis similar to HEP?
- Data: unbinned event lists (lon, lat, energy, time), or binned 1d, 2d, 3d counts
- Instrument response: effective area, plus spatial (PSF) and energy dispersion
- Models: Spatial (lon/lat) and spectra (energy)
- Fitting: Forward-fold model with instrument response. Poisson likelihood.
- Goal: inference about models & parameters



```
fit = Fit(datasets)

fit.optimize(backend="sherpa")
fit.optimize(backend="scipy")

fit.covariance(backend="minuit")
fit.confidence(parameter="amplitude")

fit.likelihood_profile(parameter="amplitude")
```

GAMMAPY

- Gammapy code, docs, example
- What I work on, very domain-specific package.
- Currently we roll our own mini framework: Parameter, Parameters, Model, Dataset, Datasets and Fit classes, most code in built-in models.
- *Idea: Dataset has list of model components and data and defines the likelihood. Fit class is the manager and interface to optimiser and error estimator backends. Linked parameters across datasets via multiple references to Python Parameter objects.*
- *Very much work in progress, feedback welcome. Would like to switch to a solution that supports parallelism (multi-core, maybe CPUs), and gradients, ideally with autodiff. Sounds like Tensorflow, but is it stable enough for us and can we teach our users & devs?*


HOW STANDARDS PROLIFERATE:
(SEE: A/C CHARGERS, CHARACTER ENCODINGS, INSTANT MESSAGING, ETC)
SITUATION: THERE ARE 14 COMPETING STANDARDS.
14?! RIDICULOUS! WE NEED TO DEVELOP ONE UNIVERSAL STANDARD THAT COVERS EVERYONE'S USE CASES.
YEAH!
SOON:
SITUATION: THERE ARE 15 COMPETING STANDARDS.

https://xkcd.com/927/

http://www.juliaopt.org

QUO VADIS?

- Contribute to existing packages?
- Create new packages?
- Create a standard and abstract interfaces?

- Seems possible, but difficult.
  Both technically and sociologically.

- Significant effort exists based on Tensorflow (e.g. tensorflow probability) or in other languages (e.g. Julia, Swift)
- But what about Numpy & pytorch & … ?

SUMMARY & DISCUSSION

1. Python is great for modeling & fitting.
2. Python is terrible for modeling & fitting.
3. Can we make it better and collaborate on a few standards or packages?
   Or are use cases and requirements in various domains too different to converge?