From a model to a fit

LIKELIHOOD ESTIMATE

$$\hat{\tau} = \frac{n}{\sum_i (y_i - \overline{y})^2}$$

COMPLETE EXAMPLE

A Gaussian mean and precision

```
1  %%%
2  Eight readings with one mean and precision.
3  %%%
4  flatppl_compat = "0.1"
6  observed = external(cartpow(reals, 8))
8  tau = elementof(posreals)
9  mu = elementof(reals)
10 sigma = 1.0 / sqrt(tau)
12 y ~ iid(Normal(mu, sigma), 8)
13 K = kernelof(record(y = y), mu = mu, tau = tau)
15 L = likelihoodof(K, record(y = observed))
17 # Mean and precision at any named point.
18 point = record(mu = 1.15, tau = 3.0)
19 log_L = logdensityof(L, point)
```

Save and run the fit

Save the model as `fitting-from-python.flatppl`. Copy the five Python blocks in reading order into `fit.py` beside it. Run `python fit.py` with FlatPPL, JAX, NumPy and Optimistix installed.

Python imports

```
from pathlib import Path

import jax
import numpy as np
import optimistix as optx
from flatppl import load
```

Bind the data

```
data = np.array([1.2, 0.4, 2.1, 1.7,
                 0.9, 1.5, 0.3, 1.1])
path = Path(__file__).with_name(
    "fitting-from-python.flatppl")
model = load(path).set(observed=data)
```

`Path` locates the model beside the driver, independently of the current working directory.

ANNOTATED LINES

A External observations — 6

```
observed = external(cartpow(reals, 8))
```

Declare the eight readings as an external real vector. Supply them from Python with `.set(observed=data)`. A new vector of the same shape can use the same model.

B Choose fitting coordinates — 8–10

```
tau = elementof(posreals)
mu = elementof(reals)
sigma = 1.0 / sqrt(tau)
```

Declare a real mean and a positive precision. Pass $\frac{1}{\sqrt{\tau}}$ as the Normal standard deviation.

C Name the kernel inputs — 12–13

```
y ~ iid(Normal(mu, sigma), 8)
K = kernelof(record(y = y), mu = mu, tau = tau)
```

Use `iid` for eight independent readings with a common law. `kernelof` takes the two fitted parameters. Its output record contains the reading vector under `y`.

D Select the likelihood from Python — 15

```
L = likelihoodof(K, record(y = observed))
```

Bind `y` to the eight readings with `likelihoodof`. Select the resulting binding in Python by its name, `"L"`. Minimize the negative log likelihood.

Analytic maximum

The eight observations have mean 1.15 and sum of squared deviations 2.68. At the maximum, $\log L = -6.97701$.

Parameter	Fit coordinate	Model estimate
Mean `mu`	μ	1.15
Precision `tau`	$\log \tau$	$\frac{8}{2.68} \approx 2.98507$

Why the variance uses eight

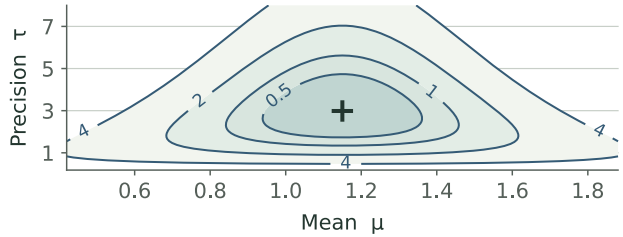
At $\mu = \overline{y}$, the likelihood equation for τ is

$$\frac{\partial \ell}{\partial \tau} = \frac{n}{2\tau} - \frac{1}{2}\sum_i (y_i - \overline{y})^2 = 0.$$

Solving for τ gives the likelihood estimate with divisor n in the variance.

PYTHON DRIVER

Likelihood contours in mean and precision



Contours show drops in log likelihood from its maximum. The cross marks the fitted mean and precision in their model coordinates.

Evaluate in free coordinates

```
with jax.enable_x64(True):
    fit = model.logdensity("L", jacobian=False,
                           dtype="float64")
    def objective(z, args):
        return fit.potential(z)
```

Set `jacobian=False` to optimize in log precision while preserving the likelihood at the corresponding model point.

Return to model coordinates

```
    solver = optx.BFGS(rtol=1e-9, atol=1e-9)
    result = optx.minimise(objective, solver,
                           fit.init)
    point = fit.constrain(result.value)
```

Keep the minimization inside the 64-bit context. Apply `constrain` to recover the fitted `mu` and positive `tau`.

Check the result

```
print(float(point["mu"]), float(point["tau"]))
np.testing.assert_allclose(
    [point["mu"], point["tau"]],
    [data.mean(), 1 / data.var()], rtol=1e-7)
```

NumPy divides by eight for the default variance of these observations. Its reciprocal gives the maximum-likelihood precision.

Jessica Cox · bcox@mpp.mpg.de