Python for Data Science

Parameter Estimation

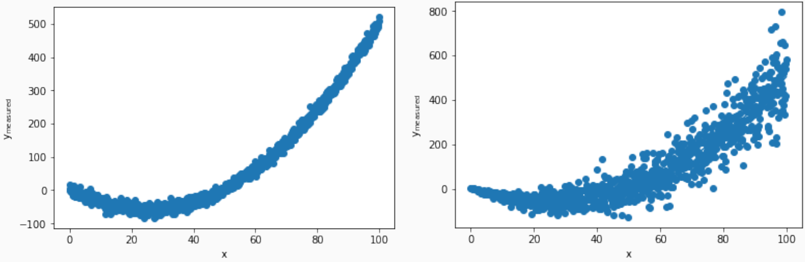
Anna Scaife

University of Manchester

Parameter Estimation

Parameter estimation is one of the most common applications in numerical data science.

Parameter Estimation



Parameter Estimation

We think that the model which describes the data has this form:

$$y = ax^2 - mx + c$$

and we know the value of the measurement noise for each data point, σ_i.

What we don't know is the value of the parameters a, m and c.

Cost Function

To fit the model to our data we need to specify a *cost function*. This is a function that evaluates the deviation of the model from the measurements.

The most commonly used cost function is chi-squared:

$$\chi^2 = \sum_i \frac{(d_i - m_i)^2}{\sigma_i^2}$$

The cost function that we specify expresses our *a priori* knowledge of the data.

Direct Optimisation

Import the library:

```
import scipy.optimize as op
```

Define the function you want to fit:

```
def model(p,x):

    a,m,c = p
    y = a*x**2 - m*x + c

    return y
```

Direct Optimisation

Define the cost function. Here it is the Gaussian loglikelihood:

```
def ll(p,x,y,sigma):

    y_try = model(p,x)
    diff = y_try - y

    ll = -0.5*np.sum(diff**2/sigma**2)

    return ll
```

`scipy` optimisation requires the negative loglikelihood:

```
nll = lambda *args: -ll(*args)
```

Direct Optimisation

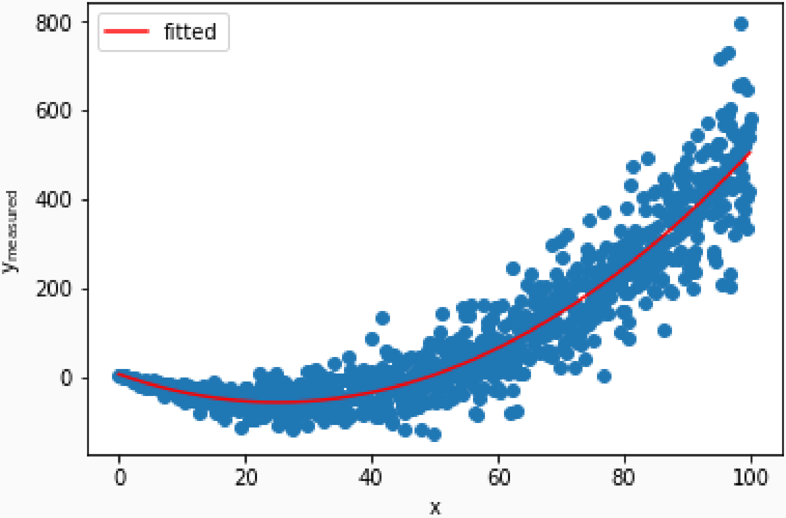
```
initial = np.array([1.,1.,1.])
```

```
result = op.minimize(nll, initial, args=(x, y_meas, sigma))
a, m, c = result["x"]
```

```
In [12]: print result

      fun: 482.94548559080283
 hess_inv: array([[1.80041525e-07, 1.79870092e-05, 2.99227436e-04],
       [1.79870092e-05, 1.91700675e-03, 3.58894926e-02],
       [2.99227436e-04, 3.58894926e-02, 8.96796830e-01]])
      jac: array([-7.62939453e-06,  0.00000000e+00, -3.81469727e-06])
  message: 'Optimization terminated successfully.'
     nfev: 50
      nit: 7
     njev: 10
   status: 0
  success: True
        x: array([0.10051117, 5.05231612, 5.89715686])
```

Direct Optimisation



Markov Chain Monte Carlo

Import the library:

```
import emcee
```

Set up the MCMC sampler:

```
ndim, nwalkers = len(initial), 10
p0 = initial + 1e-8 * np.random.randn(nwalkers, ndim)
sampler = emcee.EnsembleSampler(nwalkers, ndim, ll,
                                args=(x,y_meas,sigma))
```

Markov Chain Monte Carlo

```python
print("Running burn-in...")
p0,_,_ = sampler.run_mcmc(p0, 1000)
sampler.reset()
```

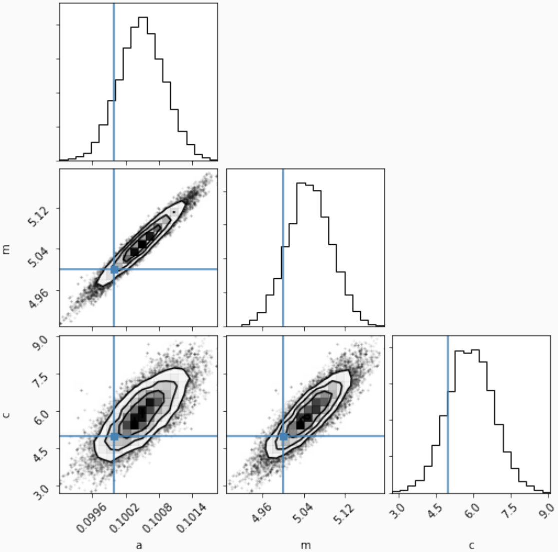
```python
print("Running production...")
sampler.run_mcmc(p0, 3000)
```

Markov Chain Monte Carlo

```
import corner

tri_labels = [r"a", r"m", r"c"]
tri_truths = [0.1,5.,5.]
tri_range = [(0, 1.), (0, 10), (0, 10)]
inds = np.array([0,1,2])
corner.corner(sampler.flatchain[:, inds], truths=tri_truths,
                                              labels=tri_labels)
```

Markov Chain Monte Carlo



(see visualisation talk: seaborn library)