Likelihood estimation examples

Oliver Maclaren
oliver.maclaren@auckland.ac.nz

Part I: Constant model

Consider the model

$$y = \theta + e$$
$$e \sim \mathcal{N}(0, 1)$$

Suppose we take 5 iid observations from this model, i.e.

$$y_i = \theta + e_i$$
$$e_i \sim \mathcal{N}(0, 1)$$

for $i = 1, ..., 5$.

- Write down an appropriate form of the likelihood function for samples of this size.
- Show that the maximum likelihood estimate of θ is $\bar{y}$, the sample mean.
- Write code to plot the full likelihood for any sample consisting of 5 iid observations.
- Write code to find the maximum likelihood estimate by optimising the likelihood numerically and compare it to the analytical answer for data generated from some parameter value you choose.
- Write code to form approximate 95% confidence interval estimates for any sample realisation of 5 observations.
- Choose a true parameter value and carry out a *coverage* experiment for this confidence interval procedure: repeatedly generate new sample data of 5 observations, form the confidence interval, and record whether it contains the true parameter value. The estimated coverage is the proportion of your intervals that contain the true parameter value.

Part II: Linear regression model

Here we consider profiling for the simple linear (in parameters) model

$$y_i = f(x_i, a, b) + e_i = ax_i + bx_i^2 + e_i, \quad e_i \sim \mathcal{N}(\mu = 0, \sigma = 0.5),$$

i.e. $y_i \sim \mathcal{N}(f(x_i, a, b), 0.5)$, for $i = 1, ...N$, meaning that we have Gaussian observation model with $\sigma = 0.5$. We wish to compute the profiles $\mathcal{L}_p(a; y)$ and $\mathcal{L}_p(b; y)$, where y denotes the combined vector of observations.

When computing profile likelihoods like $\mathcal{L}_p(a; y)$, for example, the key computations involve solving a series of least squares problems of the form

$$\inf_b \frac{1}{2\sigma^2} \|y - f(a, b)\|^2$$

for a grid of a values, and where model outputs at all x observation locations have been collected into a vector, here denoted by $f(a, b)$. The relative likelihood function is then computed by taking the maximum across all a values to obtain the overall minimiser for the denominator, and then taking exponentials to regain the likelihoods. We assume that the true likelihood is sufficiently smooth so that we can extend the likelihood as calculated at gridded values to a likelihood defined at all values via interpolation between grid points. Similarly, we compute $\mathcal{L}_p(b; y)$ by switching the roles of a and b as interest and nuisance parameters.

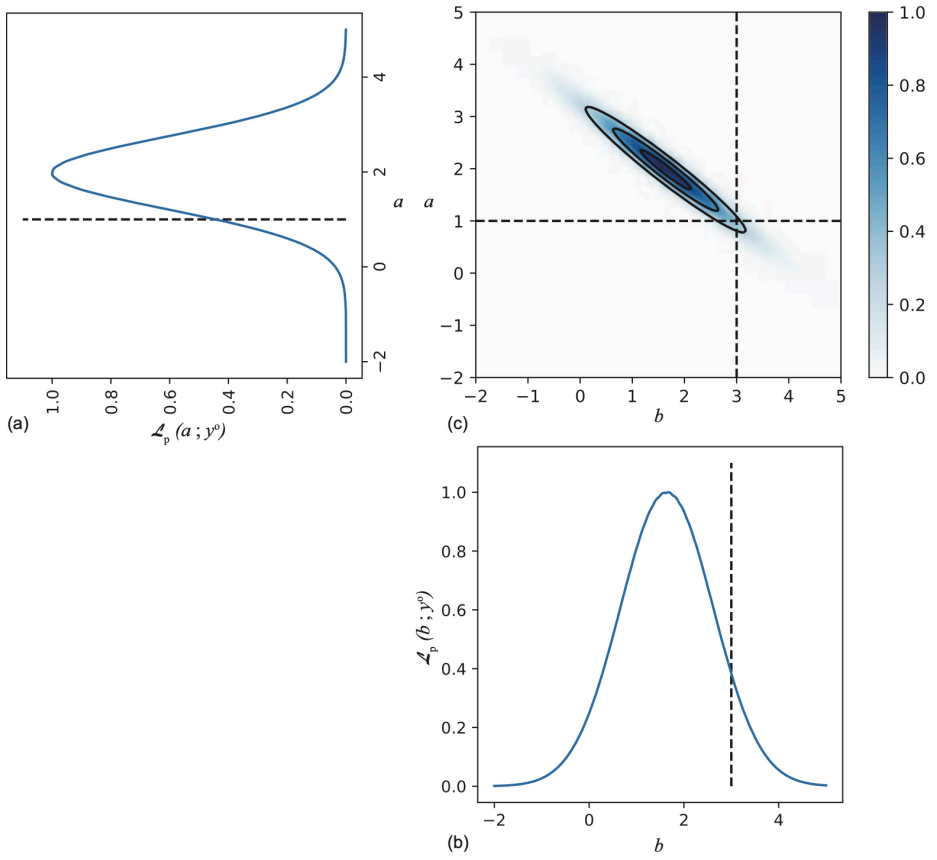

Figure 1: Example of profiling the two-dimensional likelihood function corresponding to the linear regression model.

A simple example of profiling the two-dimensional likelihood function corresponding to the model above is shown in Figure 1. Here we generated 40 samples from the model over a regularly spaced grid for $x \in [0, 1]$. The true parameters used were $a = 1, b = 3$. The two-dimensional likelihood was generated by direct calculation over a 200×200 grid.

- Reproduce this example.

Part III: ODE model

Consider the system of reactions representing an enzyme with two binding sites:

$$S + E \underset{k_{-1}}{\overset{k_1}{\rightleftharpoons}} C_1 \xrightarrow{k_2} P + E$$

$$S + C_1 \underset{k_{-3}}{\overset{k_3}{\rightleftharpoons}} C_2 \xrightarrow{k_4} P + C_1$$

The equations governing this system are

$$\begin{aligned}
\frac{d[S]}{dt} &= -J_1 + J_{-1} - J_3 + J_{-3} \\
\frac{d[E]}{dt} &= -J_1 + J_{-1} + J_2 \\
\frac{d[C_1]}{dt} &= J_1 - J_{-1} - J_2 - J_3 + J_{-3} + J_4 \\
\frac{d[C_2]}{dt} &= J_3 - J_{-3} - J_4 \\
\frac{d[P]}{dt} &= J_2 + J_4
\end{aligned}$$

where

$$\begin{aligned}
J_1 &= k_1[S][E] \\
J_{-1} &= k_{-1}[C_1] \\
J_2 &= k_2[C_1] \\
J_3 &= k_3[S][C_1] \\
J_{-3} &= k_{-3}[C_2] \\
J_4 &= k_4[C_2]
\end{aligned}$$

- Implement this model in a programming language like Python, R, Matlab etc.
- Next construct a script to solve the ODE above using the following settings:
  - Initial conditions: [S] = 15 μM, [E] = 0.5 μM, all other species set to 0 μM.
  - $k_1 = 2\ \mu M^{-1}s^{-1}$, $k_{-1} = 11\ s^{-1}$, $k_2 = 5\ s^{-1}$, $k_3 = 9\ \mu M^{-1}s^{-1}$, $k_{-3} = 21\ s^{-1}$, and $k_4 = 11\ s^{-1}$.
- Once constructed, run your script in order to simulate the system for a period of time such that it reaches steady state.

Suppose we had a vector of experimental data $[P]_{\text{data}}$ of product concentrations at the times (in s)

$$\text{times (s)} = [0, 1, 2, ..., 10].$$

The $[P]_{\text{data}}$ is given by

$$[P]_{\text{data}}\ (\mu M) = [0.26, 4.92, 9.53, 11.56, 14.04, 14.04, 14.18, 14.61, 14.62, 15.24, 15.62].$$

There is measurement noise in this data with standard deviation of 0.2.

We happen to know that the true system for which the data was collected can be described by the above model, with all the parameters the same as in the previous question, **except that k_3 is different**. We know that the k_3 for this system is somewhere between 10 and 50 $\mu M^{-1}s^{-1}$, but unfortunately we don't know the exact value and need to estimate it based on $[P]_{\text{data}}$. We can do this as follows.

Firstly, notice that for any k_3 we can simulate our system to get $[P]_{\text{sim}}$ and compare the result to the given data $[P]_{\text{data}}$. We can hence define an 'objective function' to measure how well a given k_3 can 'fit' the given $[P]_{\text{data}}$ via its implied $[P]_{\text{sim}}$ values. Here is a Python version of a possible objective function (it calculates the sum of squared differences between observed and simulated data at the given time points):

```
import numpy as np
from scipy.integrate import solve_ivp
def objective(k3):
    P_data = np.array([ 0.26,  4.92,  9.53, 11.56, 14.04, 14.04, \
        14.18, 14.61, 14.62, 15.24, 15.62])
    t_data = np.linspace(0,10,11)
    prm[3] = k3
    dydt = lambda t, y: dydt_p(t,y,prm)
    sol = solve_ivp(fun=dydt,t_span=[0,10],t_eval=t_data,y0=y0)
    P_sim = sol.y[4,:]
    return np.linalg.norm(P_sim-P_data,2)**2
```

In Matlab this would be something like:

```
function obj = objective(k3)
    P_data = [0.26,  4.92,  9.53, 11.56, 14.04, 14.04, 14.18, 14.61, 14.62, 15.24, 15.62];
    t_data = linspace(0,10,11);
    prm(4) = k3
    sol=ode15s(@lab2ode,[0 10],ics,options,prm);
    X = deval(sol,t_data)
    P_sim=X(5,:)
    obj = norm(P_sim-P_data)^2;
end
```

Note that I've left out some variables like initial conditions etc that you'd need to define within, or provide externally to, the objective function to simulate the ODE.

Given all of the required information, our goal is to minimise the difference between the model predictions and the given data, by varying k_3 and minimising the above objective.

- Explain under what conditions minimising this objective maximises a likelihood function
- Evaluate the above objective function over a grid of 100 equally-spaced values of k_3 between 10 and 50, and plot the result. By looking at the graph, state what you think would be a reasonable guess for the k_3 that generated the data.

We can also use Matlab/Python's optimisation tools to minimise the above function to get a more precise estimate. Using Python we can do:

```
from scipy.optimize import minimize
k3_initial_guess = 1
res = minimize(objective, x0=k3_initial_guess,tol=1e-3)
```

In Matlab we can use fminsearch and do:

```
k3_initial_guess = 1;
k3_sol = fminsearch(@objective,k3_initial_guess)
```

- Running the above will determine a k_3 that 'best' fits the given data, according to our objective. Run it and then extract (if using Python) the estimate for k_3 that it returns and report this. Does

this estimate (roughly) agree with your previous estimate based on visual inspection of the objective function? Would you expect to get exactly the same answer from a new experiment on the same system? Why/why not?

- Convert your objective function plot to a proper likelihood plot.