Lesson 3: Likelihood-based Inference for POMP Models

Aaron A. King Edward L. Ionides Translated in pypomp by Kunyang He

2025-12-23

Table of contents I

Table of contents II

Table of contents III

This lesson develops likelihood-based inference for POMP models, with a focus on the particle filter algorithm for computing the likelihood.

Learning Objectives

Students completing this lesson will:

1. Gain an understanding of the nature of the problem of likelihood computation for POMP models.
2. Be able to explain the simplest particle filter algorithm.
3. Gain experience in the visualization and exploration of likelihood surfaces.
4. Be able to explain the tools of likelihood-based statistical inference that become available given numerical accessibility of the likelihood function.

Overview I

Conceptual links in our methodological approach

- The Monte Carlo technique called the **particle filter** is central for connecting the higher-level ideas of POMP models and likelihood-based inference to the lower-level tasks involved in carrying out data analysis.

Overview II

- We employ a standard toolkit for likelihood-based inference:
  - Maximum likelihood estimation
  - Profile likelihood confidence intervals
  - Likelihood ratio tests for model selection
  - Other likelihood-based model comparison tools such as AIC
- We seek to better understand these tools, and to figure out how to implement and interpret them in the specific context of POMP models.

The Likelihood Function I

- The basis for modern frequentist, Bayesian, and information-theoretic inference.
- Method of maximum likelihood introduced by Fisher (1922).
- The likelihood function itself is a representation of what the data have to say about the parameters.

The Likelihood Function II

Definition of the likelihood function:

- Data are a sequence of N observations, denoted $y^*_{1:N}$.
- A statistical model is a density function $f_{Y_{1:N}}(y_{1:N};\theta)$ which defines a probability distribution for each value of a parameter vector θ.
- To perform statistical inference, we must decide, among other things, for which (if any) values of θ it is reasonable to model $y^*_{1:N}$ as a random draw from $f_{Y_{1:N}}(y_{1:N};\theta)$.

The Likelihood Function III

The likelihood function is:

$$\mathcal{L}(\theta) = f_{Y_{1:N}}(y^*_{1:N};\theta),$$

the density function evaluated at the data.

It is often convenient to work with the **log-likelihood function**:

$$\ell(\theta) = \log \mathcal{L}(\theta) = \log f_{Y_{1:N}}(y^*_{1:N};\theta).$$

A Simulator is Implicitly a Statistical Model

- For simple statistical models, we may describe the model by explicitly writing the density function $f_{Y_{1:N}}(y_{1:N};\theta)$. One may then ask how to simulate a random variable $Y_{1:N} \sim f_{Y_{1:N}}(y_{1:N};\theta)$.
- For many dynamic models it is much more convenient to define the model via a procedure to **simulate** the random variable $Y_{1:N}$. This implicitly defines the corresponding density $f_{Y_{1:N}}(y_{1:N};\theta)$.
- For a complicated simulation procedure, it may be difficult or impossible to write down or even compute $f_{Y_{1:N}}(y_{1:N};\theta)$ exactly.
- It is important to bear in mind that **the likelihood function exists even when we don't know what it is!**

The Likelihood for a POMP Model I

Recall the structure of a POMP model:

- Measurements, Y_n, at time t_n depend on the latent process, X_n, at that time.
- The Markov property asserts that latent process variables depend on their value at the previous timestep.
- The distribution of X_{n+1}, conditional on X_n, is independent of X_k for $k < n$ and Y_k for $k \leq n$.
- The distribution of Y_n, conditional on X_n, is independent of all other variables.

The Likelihood for a POMP Model II

The joint density factors as:

$$f_{X_{0:N},Y_{1:N}}(x_{0:N}, y_{1:N};\theta) = f_{X_0}(x_0;\theta)\prod_{n=1}^{N} f_{X_n|X_{n-1}}(x_n|x_{n-1};\theta)\, f_{Y_n|X_n}(y_n|x$$

The Likelihood for a POMP Model III

The marginal density for the sequence of measurements, $Y_{1:N}$, evaluated at the data, $y^*_{1:N}$, is:

$$\mathcal{L}(\theta) = f_{Y_{1:N}}(y^*_{1:N};\theta) = \int f_{X_{0:N},Y_{1:N}}(x_{0:N}, y^*_{1:N};\theta)\, dx_{0:N}.$$

This integral is **high dimensional** and, except for the simplest cases, cannot be reduced analytically.

Monte Carlo Likelihood by Direct Simulation I

- First, let's rewrite the likelihood integral using an equivalent factorization:

$$\mathcal{L}(\theta) = f_{Y_{1:N}}(y^*_{1:N};\theta) = \int \left\{ \prod_{n=1}^{N} f_{Y_n|X_n}(y^*_n|x_n;\theta) \right\} f_{X_{0:N}}(x_{0:N};\theta)\, dx_0$$

- Notice that the likelihood can be written as an expectation:

$$\mathcal{L}(\theta) = \mathbb{E}\left[\prod_{n=1}^{N} f_{Y_n|X_n}(y^*_n|X_n;\theta) \right],$$

where the expectation is taken with $X_{0:N} \sim f_{X_{0:N}}(x_{0:N};\theta)$.

Monte Carlo Likelihood by Direct Simulation II

- Using a law of large numbers, we can approximate:

$$\mathcal{L}(\theta) \approx \frac{1}{J} \sum_{j=1}^{J} \prod_{n=1}^{N} f_{Y_n|X_n}(y_n^*|X_n^j;\theta),$$

where $\{X_{0:N}^j, j = 1, \dots, J\}$ is a Monte Carlo sample drawn from $f_{X_{0:N}}(x_{0:N};\theta)$.

Monte Carlo Likelihood by Direct Simulation III

Problems with this naive approach:

- This scales poorly with dimension. It requires Monte Carlo effort that scales **exponentially** with the length of the time series.
- Once a simulated trajectory diverges from the data, it will seldom come back.
- Simulations that lose track of the data make negligible contributions to the likelihood estimate.

The Particle Filter I

Fortunately, we can compute the likelihood for a POMP model by a much more efficient algorithm.

We proceed by factorizing the likelihood differently:

$$\mathcal{L}(\theta) = f_{Y_{1:N}}(y^*_{1:N};\theta) = \prod_{n=1}^{N} f_{Y_n|Y_{1:n-1}}(y^*_n|y^*_{1:n-1};\theta)$$

$$= \prod_{n=1}^{N} \int f_{Y_n|X_n}(y^*_n|x_n;\theta)\, f_{X_n|Y_{1:n-1}}(x_n|y^*_{1:n-1};\theta)\, dx_n.$$

The Particle Filter II

The prediction formula (from Markov property):

$$f_{X_n|Y_{1:n-1}}(x_n|y^*_{1:n-1};\theta) = \int f_{X_n|X_{n-1}}(x_n|x_{n-1};\theta)\, f_{X_{n-1}|Y_{1:n-1}}(x_{n-1}|y^*_{1:n-1}$$

The filtering formula (from Bayes' theorem):

$$f_{X_n|Y_{1:n}}(x_n|y^*_{1:n};\theta) = \frac{f_{Y_n|X_n}(y^*_n|x_n;\theta)\, f_{X_n|Y_{1:n-1}}(x_n|y^*_{1:n-1};\theta)}{\int f_{Y_n|X_n}(y^*_n|u_n;\theta)\, f_{X_n|Y_{1:n-1}}(u_n|y^*_{1:n-1};\theta)\, du_n}.$$

The Particle Filter III

This suggests we keep track of two key distributions at each time t_n:

- The **prediction distribution**: $f_{X_n|Y_{1:n-1}}(x_n|y^*_{1:n-1})$
- The **filtering distribution**: $f_{X_n|Y_{1:n}}(x_n|y^*_{1:n})$

The particle filter uses Monte Carlo techniques to sequentially estimate these integrals. Hence, the alternative name of **sequential Monte Carlo (SMC)**.

Basic Particle Filter Algorithm I

1. Suppose $X^F_{n-1,j}$, $j=1,\dots,J$ is a set of J points drawn from the filtering distribution at time t_{n-1}.
2. We obtain a sample $X^P_{n,j}$ from the prediction distribution at time t_n by simply **simulating** the process model:

$$X^P_{n,j} \sim \mathsf{process}(X^F_{n-1,j}, \theta), \quad j=1,\dots,J.$$

Basic Particle Filter Algorithm II

3. Having obtained $X^P_{n,j}$, we obtain a sample from the filtering distribution at time t_n by **resampling** from $\{X^P_{n,j}, j \in 1:J\}$ with weights:

$$w_{n,j} = f_{Y_n|X_n}(y^*_n|X^P_{n,j};\theta).$$

4. The conditional likelihood

$$\mathcal{L}_n(\theta) = f_{Y_n|Y_{1:n-1}}(y^*_n|y^*_{1:n-1};\theta)$$

is approximated by:

$$\hat{\mathcal{L}}_n(\theta) \approx \frac{1}{J}\sum_j f_{Y_n|X_n}(y^*_n|X^P_{n,j};\theta).$$

Basic Particle Filter Algorithm III

5. We iterate this procedure through the data, one step at a time, alternately simulating and resampling, until we reach $n = N$.

6. The full log-likelihood then has approximation:

$$\ell(\theta) = \log \mathcal{L}(\theta) = \sum_n \log \mathcal{L}_n(\theta) \approx \sum_n \log \hat{\mathcal{L}}_n(\theta).$$

Block Diagram of Particle Filter

Particle Filter Steps

1. **Initialize**: `rinit`
2. **Predict**: `rproc` (simulate forward)
3. **Weight**: `dmeas` (evaluate measurement density)
4. **Filter**: resample particles according to weights
5. Repeat steps 2-4 for N observations

The particle filter provides an **unbiased** estimate of the likelihood. This implies a consistent but biased estimate of the log-likelihood.

Import Required Packages

```
import jax.numpy as jnp
import jax
import pandas as pd
import numpy as np
import pypomp as pp
import matplotlib.pyplot as plt
import time
```

Load Data and Build Model I

```python
# Download and prepare data
meas = (pd.read_csv(
          "https://kingaa.github.io/sbied/stochsim/Measles_Consett_1948.csv")
          .loc[:, ["week", "cases"]]
          .rename(columns={"week": "time", "cases": "reports"})
          .set_index("time")
          .astype(float))

ys = meas.copy()
ys.columns = pd.Index(["reports"])
```

Load Data and Build Model II

```
# Helper functions for negative binomial
def nbinom_logpmf(x, k, mu):
    """Log PMF of NegBin(k, mu) that is robust when mu == 0."""
    x = jnp.asarray(x)
    k = jnp.asarray(k)
    mu = jnp.asarray(mu)
    logp_zero = jnp.where(x == 0, 0.0, -jnp.inf)
    safe_mu = jnp.where(mu == 0.0, 1.0, mu)
    core = (jax.scipy.special.gammaln(k + x)
            - jax.scipy.special.gammaln(k)
            - jax.scipy.special.gammaln(x + 1)
            + k * jnp.log(k / (k + safe_mu))
            + x * jnp.log(safe_mu / (k + safe_mu)))
    return jnp.where(mu == 0.0, logp_zero, core)

def rnbinom(key, k, mu):
    """Sample from NegBin(k, mu) via Gamma-Poisson mixture."""
    key_g, key_p = jax.random.split(key)
    lam = jax.random.gamma(key_g, k) * (mu / k)
    return jax.random.poisson(key_p, lam)
```

Load Data and Build Model III

```
# SIR model components
def rinit(theta_, key, covars, t0):
    """Initial state simulator for SIR model."""
    N = theta_["N"]
    eta = theta_["eta"]
    S0 = jnp.round(N * eta)
    I0 = 1.0
    R0 = jnp.round(N * (1 - eta)) - 1.0
    H0 = 0.0
    return {"S": S0, "I": I0, "R": R0, "H": H0}

def rproc(X_, theta_, key, covars, t, dt):
    """Process simulator for SIR model."""
    S, I, R, H = X_["S"], X_["I"], X_["R"], X_["H"]
    Beta = theta_["Beta"]
    mu_IR = theta_["mu_IR"]
    N = theta_["N"]

    p_SI = 1.0 - jnp.exp(-Beta * I / N * dt)
    p_IR = 1.0 - jnp.exp(-mu_IR * dt)

    key_SI, key_IR = jax.random.split(key)
    dN_SI = jax.random.binomial(key_SI, n=S.astype(jnp.int32), p=p_SI)
    dN_IR = jax.random.binomial(key_IR, n=I.astype(jnp.int32), p=p_IR)
```

Basic Particle Filter I

In pypomp, the particle filter is implemented via the pfilter method. We must choose the number of particles to use by setting the J argument.

The pfilter method updates the model's results_history attribute with the results.

```
# Run a single particle filter
key = jax.random.key(42)
measSIR.pfilter(key=key, J=5000, reps=1)

# Access results from results_history
result = measSIR.results_history.last()
loglik = float(result.logLiks.values[0, 0])
print(f"Log-likelihood: {loglik:.4f}")
```

```
Log-likelihood: -131.4215
```

Basic Particle Filter II

We can run multiple particle filters to get an estimate of the Monte Carlo variability:

```
# Run 10 replicates of the particle filter
key = jax.random.key(652643293)
measSIR.pfilter(key=key, J=5000, reps=10)

# Get results
result = measSIR.results_history.last()
logliks = result.logLiks.values[0, :]  # All replicates for first theta
print(f"Log-likelihoods: {logliks}")
print(f"Mean: {np.mean(logliks):.4f}, SE: {np.std(logliks):.4f}")
```

```
Log-likelihoods: [-133.1564  -133.78828 -135.3216  -132.25987
 -130.68163 -134.23923 -132.26183 -134.54196]
Mean: -133.3060, SE: 1.3236
```

The logmeanexp Function

To combine multiple log-likelihood estimates, we use the `logmeanexp` function from `pypomp`, which computes:

$$\log\left(\frac{1}{n}\sum_{i=1}^{n} e^{x_i}\right)$$

in a numerically stable way.

```
# pypomp provides logmeanexp and logmeanexp_se
ll_est = pp.logmeanexp(logliks)
ll_se = pp.logmeanexp_se(logliks)
print(f"Log-likelihood estimate: {ll_est:.4f} (SE: {ll_se:.4f})")
```

```
Log-likelihood estimate: -132.4008 (SE: 0.7009)
```

Alternatively, use the `to_dataframe()` method which automatically applies `logmeanexp`:

```
# Get results as DataFrame with logmeanexp already applied
df = result.to_dataframe()
print(df)
```

Maximum Likelihood Estimation I

A maximum likelihood estimate (MLE) is:

$$\hat{\theta} = \arg\max_{\theta} \ell(\theta),$$

where $\arg\max_{\theta} g(\theta)$ means the value of θ at which the maximum of g is attained.

Maximum Likelihood Estimation II

Standard errors for the MLE — There are three main approaches:

1. **Fisher information**: Computationally quick but often unreliable for POMP models
2. **Profile likelihood estimation**: Generally preferable for POMP models
3. **Bootstrap/simulation study**: Most effort but can be the best approach

Confidence Intervals via Profile Likelihood I

Let $\theta = (\phi, \psi)$, where we want a confidence interval for ϕ.

The **profile log-likelihood** of ϕ is:

$$\ell^{\mathsf{profile}}(\phi) = \max_{\psi} \ell(\phi, \psi).$$

An approximate 95% confidence interval for ϕ is:

$$\left\{\phi : \ell(\hat{\theta}) - \ell^{\mathsf{profile}}(\phi) < 1.92\right\}.$$

This is known as a **profile likelihood confidence interval**. The cutoff 1.92 is derived from Wilks' theorem.

Visualizing the Likelihood Surface I

- If Θ is two-dimensional, then the surface $\ell(\theta)$ has features like a landscape.
- Local maxima of $\ell(\theta)$ are **peaks**.
- Local minima are **valleys**.
- Peaks may be separated by a valley or may be joined by a **ridge**.

Visualizing the Likelihood Surface II

Key features to notice:

- **Wedge-shaped relationships** between parameters are common in epidemiological models
- **Monte Carlo noise** in likelihood evaluation makes it hard to pick out exactly where the likelihood is maximized
- Nevertheless, major features of the likelihood surface are evident despite the noise

Computing Likelihood Slices I

A likelihood slice is a cross-section through the likelihood surface. Let's make slices in the β and μ_{IR} directions.

Computing Likelihood Slices II

```
def compute_likelihood_slice(param_name, param_values, base_theta,
                             J=5000, n_reps=3):
    """Compute log-likelihood for a slice of parameter values."""
    results = []
    for val in param_values:
        theta_test = base_theta.copy()
        theta_test[param_name] = val

        pomp_test = pp.Pomp(
            rinit=rinit, rproc=rproc, dmeas=dmeas, rmeas=rmeas,
            ys=ys, theta=theta_test, statenames=statenames,
            t0=0.0, nstep=7, accumvars=(3,), ydim=1, covars=None
        )

        key = jax.random.key(int(val * 1000))
        pomp_test.pfilter(key=key, J=J, reps=n_reps)

        pf_result = pomp_test.results_history.last()
        logliks_arr = pf_result.logLiks.values[0, :]

        results.append({
            param_name: val,
            'loglik': pp.logmeanexp(logliks_arr),
            'loglik_se': pp.logmeanexp_se(logliks_arr)
```

Two-Dimensional Likelihood Surface I

Two-Dimensional Likelihood Surface II

```
def compute_2d_likelihood_surface(beta_vals, mu_IR_vals, base_theta,
                                  J=2000, n_reps=2):
    """Compute 2D log-likelihood surface."""
    results = []
    for beta in beta_vals:
        for mu_IR in mu_IR_vals:
            theta_test = base_theta.copy()
            theta_test["Beta"] = beta
            theta_test["mu_IR"] = mu_IR

            pomp_test = pp.Pomp(
                rinit=rinit, rproc=rproc, dmeas=dmeas, rmeas=rmeas,
                ys=ys, theta=theta_test, statenames=statenames,
                t0=0.0, nstep=7, accumvars=(3,), ydim=1, covars=None
            )

            key = jax.random.key(int(beta * 100) + int(mu_IR * 1000))
            pomp_test.pfilter(key=key, J=J, reps=n_reps)

            pf_result = pomp_test.results_history.last()
            logliks_arr = pf_result.logLiks.values[0, :]

            results.append({
                'Beta': beta,
```

Maximizing the Particle Filter Likelihood I

- Likelihood maximization is key to profile intervals, likelihood ratio tests, and AIC, as well as computation of the MLE.
- An initial approach might be to use the particle filter log-likelihood estimate with a standard numerical optimizer (e.g., Nelder-Mead).
- In practice, this approach is unsatisfactory on all but the smallest POMP models.
- Standard numerical optimizers are not designed to maximize **noisy** and **computationally expensive** Monte Carlo functions.

Maximizing the Particle Filter Likelihood II

Trade-offs:

- If we use a deterministic optimizer and fix the RNG seed, the objective function becomes **jagged** (many small local knolls and pits).
- If we use a stochastic optimization algorithm, we can only obtain **estimates** of the MLE.

We'll present **iterated filtering** in the next lesson as a better approach.

Likelihood Ratio Tests for Nested Hypotheses I

Suppose we have two nested hypotheses:

- $H^{\langle 0 \rangle}$: $\theta \in \Theta^{\langle 0 \rangle}$ (dimension $D^{\langle 0 \rangle}$)
- $H^{\langle 1 \rangle}$: $\theta \in \Theta^{\langle 1 \rangle}$ (dimension $D^{\langle 1 \rangle}$)

where $\Theta^{\langle 0 \rangle} \subset \Theta^{\langle 1 \rangle}$.

Likelihood Ratio Tests for Nested Hypotheses II

Wilks' approximation: Under the null hypothesis $H^{\langle 0 \rangle}$:

$$\ell^{\langle 1 \rangle} - \ell^{\langle 0 \rangle} \approx \frac{1}{2}\chi^2_{D^{\langle 1 \rangle} - D^{\langle 0 \rangle}}$$

This can be used to construct a **likelihood ratio test**.

Akaike's Information Criterion (AIC) I

For non-nested hypotheses, we can compare likelihoods using AIC:

$$\text{AIC} = -2\ell(\hat{\theta}) + 2D$$

"Minus twice the maximized log-likelihood plus twice the number of parameters."

- Select the model with the **lowest AIC** score.
- AIC was derived as an approach to minimizing prediction error.
- Increasing parameters leads to overfitting which can decrease predictive skill.

Akaike's Information Criterion (AIC) II

Practical guidance:

- AIC is useful for selecting a model with reasonable predictive skill from a range of possibilities.
- View it as a procedure to select a reasonable predictive model, not as a formal hypothesis test.
- BIC provides a more severe penalty for complexity.

Exercise 3.1: Slices and Profiles

What is the difference between a likelihood **slice** and a **profile**? What is the consequence of this difference for the statistical interpretation of these plots? How should you decide whether to compute a profile or a slice?

Exercise 3.2: Cost of a Particle Filter Calculation I

- How much computer processing time does a particle filter take?
- How does this scale with the number of particles?

Form a conjecture based upon your understanding of the algorithm. Test your conjecture by running a sequence of particle filter operations, with increasing numbers of particles (J), measuring the time taken for each one. Plot and interpret your results.

Exercise 3.3: Log-likelihood Estimation I

Here are some desiderata for a Monte Carlo log-likelihood approximation:

- It should have low Monte Carlo bias and variance.
- It should be presented together with estimates of the bias and variance so that we know the extent of Monte Carlo uncertainty in our results.
- It should be computed in a length of time appropriate for the circumstances.

Exercise 3.3: Log-likelihood Estimation II

Set up a likelihood evaluation for the measles model, choosing the numbers of particles and replications so that your evaluation takes approximately one minute on your machine.

- Provide a Monte Carlo standard error for your estimate.
- Comment on the bias of your estimate.

Exercise 3.4: One-dimensional Likelihood Slice

Compute several likelihood slices in the η direction.

Exercise 3.5: Two-dimensional Likelihood Slice

Compute a slice of the likelihood in the β-η plane.

Summary I

1. The **likelihood function** is central to frequentist, Bayesian, and information-theoretic inference.
2. For POMP models, the likelihood involves a high-dimensional integral that cannot be computed analytically.
3. The **particle filter** provides an efficient Monte Carlo algorithm for computing the likelihood:
   - Alternates between prediction (simulation) and filtering (resampling)
   - Provides an unbiased estimate of the likelihood

Summary II

4. **Likelihood-based inference** provides tools for:
   - Maximum likelihood estimation
   - Profile likelihood confidence intervals
   - Likelihood ratio tests
   - Model comparison via AIC
5. The **geometry of the likelihood surface** reveals important features:
   - Wedge-shaped relationships between parameters
   - Monte Carlo noise affects optimization

References I

Fisher, R. A. (1922). On the mathematical foundations of theoretical statistics. *Philos Trans R Soc London A*, 222:309–368.