Lesson 3: Likelihood-based Inference for POMP Models

Aaron A. King   Edward L. Ionides   Kunyang He

Saturday, March 21, 2026

Objectives

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

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.

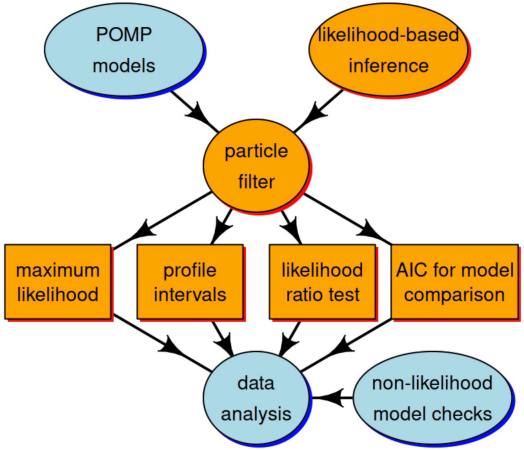


Figure 1: Schematic diagram. The Monte Carlo technique called the **particle filter** is central for connecting the higher-level ideas of POMP models and likelihood-based inference to the lower-level tasks involved in carrying out data analysis.

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

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 (Fisher, 1922) 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**:

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

the basis for modern frequentist, Bayesian, and information-theoretic inference.

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 $Y_{1:N} \sim f_{Y_{1:N}}(y_{1:N};\theta)$.

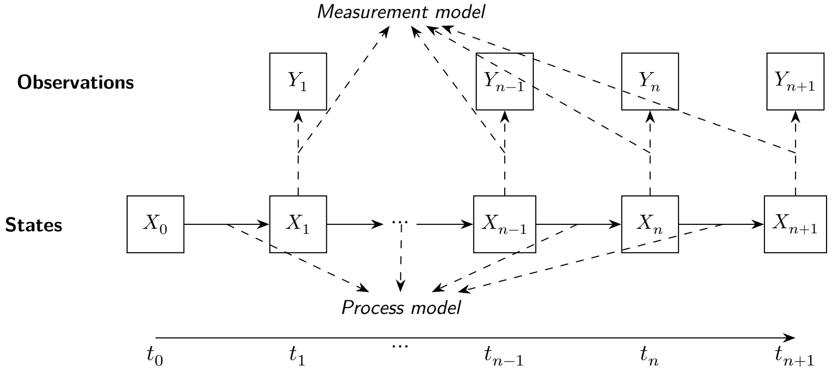
- For many dynamic models it is much more convenient to define the model via a procedure to **simulate** the random variable $Y_{1:N}$. This implicitly defines the corresponding density $f_{Y_{1:N}}(y_{1:N};\theta)$.
- For a complicated simulation procedure, it may be difficult or impossible to write 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

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.


Measurement model
Observations
Y_1
Y_{n-1}
Y_n
Y_{n+1}
States
X_0
X_1
...
X_{n-1}
X_n
X_{n+1}
Process model
t_0
t_1
...
t_{n-1}
t_n
t_{n+1}

The joint density factors as:

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

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

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

$$\mathcal{L}(\theta) = \int \left\{ \prod_{n=1}^{N} f_{Y_n|X_n}(y_n^*|x_n;\theta) \right\} f_{X_{0:N}}(x_{0:N};\theta)\, dx_{0:N}.$$

- 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 $X_{0:N} \sim f_{X_{0:N}}(x_{0:N};\theta)$.

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

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

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

We proceed by factorizing the likelihood differently:

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

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

The prediction formula (from Markov property):

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

The filtering formula (from Bayes' theorem):

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

This suggests we keep track of two 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 a **sequential Monte Carlo (SMC)** approach to recursively estimate these integrals.

Basic Particle Filter Algorithm

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:

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

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

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

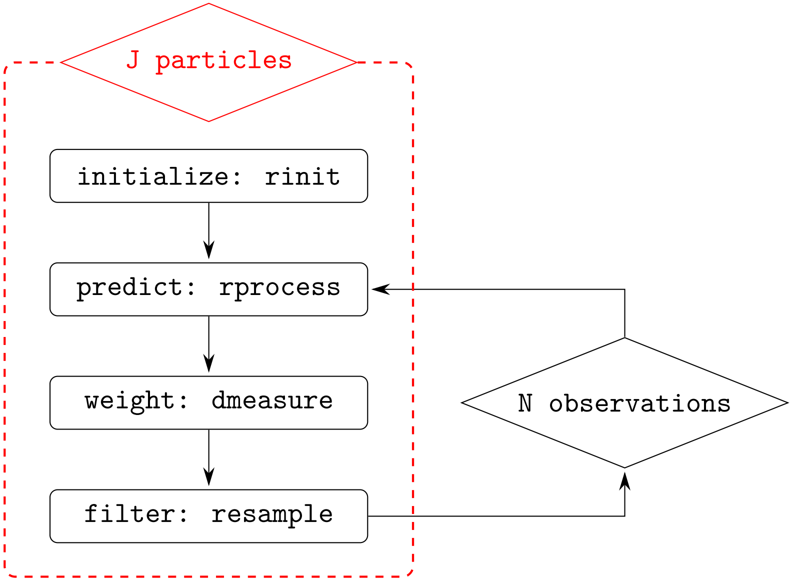
is approximated by:

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

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 a particle filter



Particle filter steps via `pypomp` model components

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.

Particle filtering in `pypomp`

- Run the `pfilter` method on a `Pomp` object.
- JAX will automatically parallelize across all the CPUs or GPUs it can access.

Set up Python

```
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
import os
import pickle

# run level 0 : quick, 1 : medium, 2 : slow
RL = 2

# manual control of cached using pickle
# remove the cache directory to force recomputation
cache_dir = "cache_" + str(RL)
os.makedirs(cache_dir, exist_ok=True)
```

Load Data and Build Model

```
meas = pd.read_csv("Measles_Consett_1948.csv",
    usecols=["week", "cases"],index_col="week")

meas.rename(columns={"week": "time",
                     "cases": "reports"})

#    .set_index("time")
#    .astype(float)
#)

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

SIR Model Components

```python
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}
```

```python
def rproc(X_, theta_, key, covars, t, dt):
    """Process simulator for SIR model."""
    S = jnp.asarray(X_["S"])
    I = jnp.asarray(X_["I"])
    R = jnp.asarray(X_["R"])
    H = jnp.asarray(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=jnp.int32(S), p=p_SI)
    dN_IR = jax.random.binomial(
        key_IR, n=jnp.int32(I), p=p_IR)

    return {"S": S - dN_SI,
            "I": I + dN_SI - dN_IR,
            "R": R + dN_IR,
            "H": H + dN_IR}
```

Negative binomial measurement model

- As discussed in Chapter 2, this allow for an overdispersed measurement distribution.
- The measurement log-density, `nbinom_logpmf`, is used from Chapter 2.
- We use a corresponding simulator, `pp.random.fast_approx_rnbinom` from `pypomp`.

```python
def dmeas(Y_, X_, theta_, covars, t):
    """Measurement density: log P(reports | H, rho, k)."""
    rho = theta_["rho"]
    k = theta_["k"]
    H = X_["H"]
    mu = rho * H
    return nbinom_logpmf(Y_["reports"], k, mu)

def rmeas(X_, theta_, key, covars, t):
    """Measurement simulator."""
    rho = theta_["rho"]
    k = theta_["k"]
    H = X_["H"]
    mu = rho * H
    reports = pp.random.fast_approx_rnbinom(key, k, mu=mu)
    return jnp.array([reports])
```

Create POMP Object

```
theta = {
    "Beta": 15.0,     # Transmission rate (per week)
    "mu_IR": 0.5,     # Recovery rate (per week)
    "N": 38000.0,     # Population size
    "eta": 0.06,      # Initial susceptible fraction
    "rho": 0.5,       # Reporting probability
    "k": 10.0         # Overdispersion parameter
}

statenames = ["S", "I", "R", "H"]

measSIR = pp.Pomp(
    rinit=rinit,
    rproc=rproc,
    dmeas=dmeas,
    rmeas=rmeas,
    ys=ys,
    theta=theta,
    statenames=statenames,
    t0=0.0,
    nstep=7,
    accumvars=("H",),
    ydim=1,
    covars=None
)
```

Basic Particle Filter

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.

```
key = jax.random.key(42)
measSIR.pfilter(key=key, J=[50,500,5000][RL], 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
```

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

```python
key = jax.random.key(652643293)
cache_file = cache_dir + "/pfilter-reps.pkl"
if os.path.exists(cache_file):
    with open(cache_file, 'rb') as f:
        result = pickle.load(f)
else:
    measSIR.pfilter(key=key,
        J=[50,500,5000][RL], reps=[2,6,10][RL])
    result = measSIR.results_history.last()
    with open(cache_file, 'wb') as f:
        pickle.dump(result,f)
logliks = result.logLiks.values[0, :]
print(f"Mean: {np.mean(logliks):.1f}",
      f"SE: {np.std(logliks):.1f}")
```

```
Mean: -133.3 SE: 1.3
```

Individual log-likelihoods:

```
Log-likelihoods:
-133.2 -133.8 -135.3 -132.3 -132.5
 -134.3 -130.7 -134.2 -132.3 -134.5
```

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.

```
ll_est = pp.logmeanexp(logliks)
ll_se = pp.logmeanexp_se(logliks)
print(f"Log-lik estimate: {ll_est:.4f}",
      f"(SE: {ll_se:.4f})")
```

```
Log-lik 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)
```

```
       logLik        se  Beta  mu_IR        N   eta  rho  
0 -132.400791  0.700865  15.0    0.5  38000.0  0.06  0.5  1
```

Maximum Likelihood Estimation

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.

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

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

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

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

An approximate 95% confidence interval for ϕ is:

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

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

Visualizing the Likelihood Surface

- 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**.

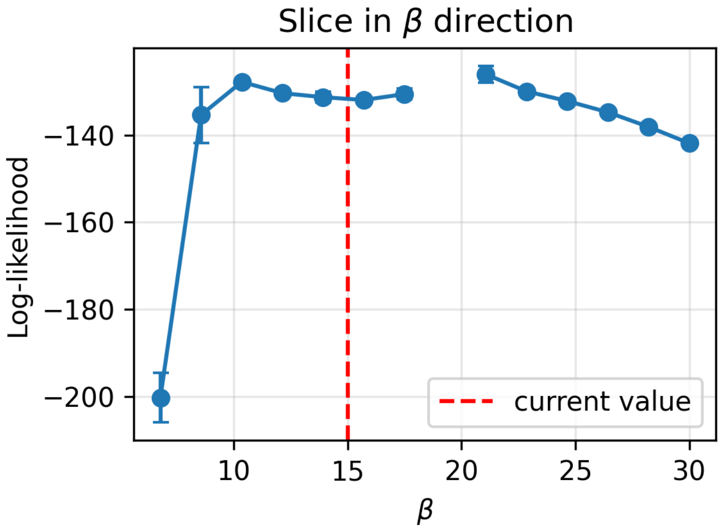
Key features to notice:

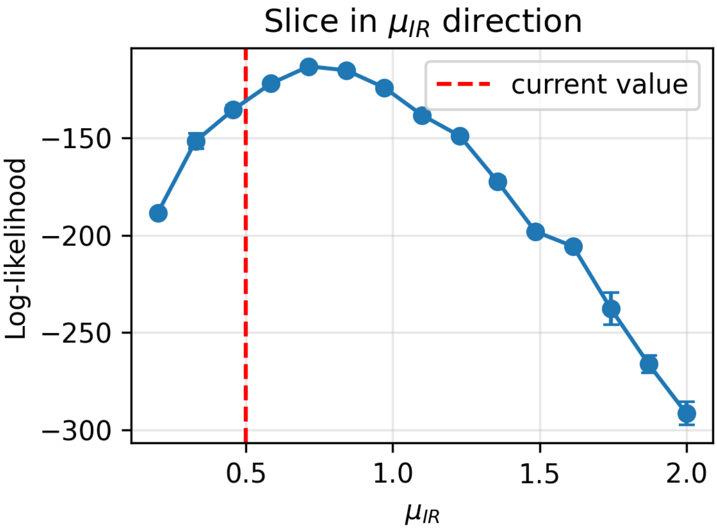
- **Wedge-shaped relationships** between parameters are common in epidemiological models
- **Monte Carlo noise** in likelihood evaluation makes it hard to pick out exactly where the likelihood is maximized
- Nevertheless, major features of the likelihood surface are evident despite the noise

Computing Likelihood Slices

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

```
beta_values = np.linspace(5, 30, [5,10,15][RL])
cache_file = cache_dir + "/beta-slice.pkl"
if os.path.exists(cache_file):
    with open(cache_file, 'rb') as f:
        beta_slice = pickle.load(f)
else:
    beta_slice = compute_likelihood_slice(
        "Beta", beta_values, theta,
        J=[10,500,5000][RL], n_reps=3)
    with open(cache_file, 'wb') as f:
        pickle.dump(beta_slice,f)
```

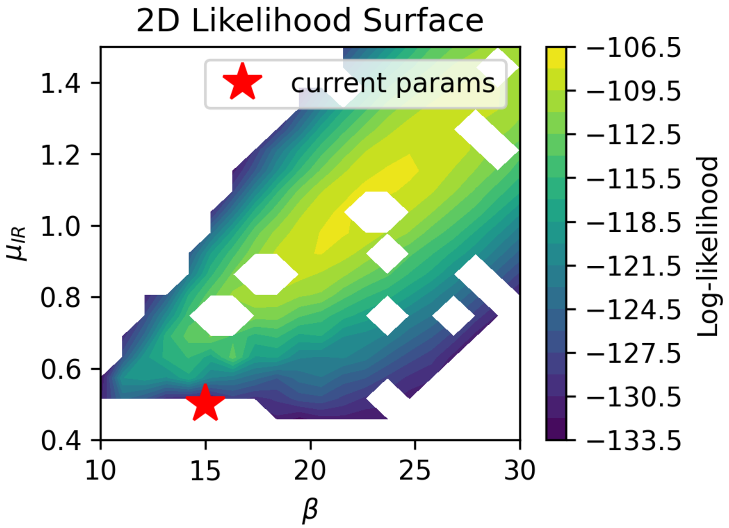

Slice in β direction
Log-likelihood
−140
−160
−180
−200
10
15
20
25
30
β
current value


Slice in μ_{IR} direction
current value
Log-likelihood
−150
−200
−250
−300
0.5
1.0
1.5
2.0
μ_{IR}

Two-Dimensional Likelihood Surface

```
beta_grid = np.linspace(10, 30, [3,10,20][RL])
mu_IR_grid = np.linspace(0.4, 1.5, [3,10,20][RL])

cache_file = cache_dir + "/2d_surface.pkl"
if os.path.exists(cache_file):
    with open(cache_file, 'rb') as f:
        surface_df = pickle.load(f)
else:
    surface_df = compute_2d_surface(
        beta_grid, mu_IR_grid, theta,
        J=[10,500,5000][RL], n_reps=2)
    with open(cache_file, 'wb') as f:
        pickle.dump(surface_df,f)
```


2D Likelihood Surface
current params
μ_{IR}
1.4
1.2
1.0
0.8
0.6
0.4
10
15
20
25
30
β
−106.5
−109.5
−112.5
−115.5
−118.5
−121.5
−124.5
−127.5
−130.5
−133.5
Log-likelihood

Maximizing the Particle Filter Likelihood

- 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 can struggle to maximize **noisy** and **computationally expensive** Monte Carlo functions.

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

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

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)

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

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

"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.

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

- 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

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.

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

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

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, e.g., **Wedge-shaped relationships between parameters**.

References I

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