Lesson 5 Supplement: Measles Profile Likelihood Computation

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

2025-12-24

Table of contents I

Table of contents II

This document demonstrates how to compute a profile likelihood for the measles model, focusing on the extra-demographic stochasticity parameter σ_{SE}.

What is Profile Likelihood? I

A **profile likelihood** is a technique for understanding how the likelihood varies with a focal parameter while allowing other parameters to adjust optimally.

For a focal parameter ϕ and nuisance parameters ψ, the profile likelihood is:

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

Profile likelihoods are useful for:

1. **Constructing confidence intervals**: Use the likelihood ratio test cutoff
2. **Assessing identifiability**: Flat profiles indicate weak identifiability
3. **Visualizing parameter trade-offs**: See how other parameters compensate

Profile Confidence Intervals I

The $(1-\alpha)$ confidence interval from a profile is the set of parameter values where:

$$\ell^{\text{profile}}(\phi) \geq \ell^{\text{max}} - \frac{1}{2}\chi^2_1(1-\alpha)$$

For a 95% CI: cutoff $= \chi^2_1(0.95)/2 \approx 1.92$.

For a 99% CI: cutoff $= \chi^2_1(0.99)/2 \approx 3.32$.

Load Packages and Model

```
import jax
import jax.numpy as jnp
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from copy import deepcopy
import time

# Import pypomp components
from pypomp import Pomp, RWSigma, ParTrans, mcap
from pypomp.util import logmeanexp, logmeanexp_se

# Import the built-in UK Measles module
from pypomp.measles.measlesPomp import UKMeasles
import pypomp.measles.model_001b as m001b

np.random.seed(594709947)
```

Load Data and MLEs I

```python
# Load MLEs from He et al. (2010) for London
mles = UKMeasles.AK_mles()
theta_mle = mles["London"].to_dict()

print("MLE parameters for London:")
for param, value in theta_mle.items():
    print(f"  {param}: {value}")
```

```
MLE parameters for London:
  R0: 56.8
  sigma: 28.9
  gamma: 30.4
  iota: 2.9
  rho: 0.488
  sigmaSE: 0.0878
  psi: 0.116
  cohort: 0.557
```

Load Data and MLEs II

```
amplitude: 0.554
S_0: 0.0297
E_0: 5.17e-05
I_0: 5.14e-05
R_0: 0.97
```

Load Data and MLEs III

```
# Create base POMP object
measles_pomp = UKMeasles.Pomp(
    unit=["London"],
    theta=theta_mle,
    model="001b",
    interp_method="shifted_splines",
    first_year=1950,
    last_year=1963,
    dt=1/365.25,
    clean=True
)

print(f"\nPOMP object created")
print(f"Time range: {measles_pomp.ys.index[0]:.2f} to {measles_pomp.ys.index[-1]:.2
print(f"Number of observations: {len(measles_pomp.ys)}")
```

Load Data and MLEs IV

```
POMP object created
Time range: 1950.01 to 1963.98
Number of observations: 730
```

Profiling over σ_{SE} I

Based on the original sbied approach, we profile over σ_{SE} while:

- **Fixed parameters**: ρ and ι at their MLE values
- **Profile parameter**: σ_{SE}
- **Estimated parameters**: R0, sigma, gamma, cohort, amplitude, psi, initial conditions

The profile grid spans from $\sigma_{SE} = 0.02$ to 0.20 (20 points).

Profiling over σ_{SE} II

```python
# Profile grid
sigmaSE_grid = np.linspace(0.02, 0.20, 20)

# Parameters to estimate (excluding sigmaSE, rho, iota)
est_params = ["R0", "sigma", "gamma", "cohort", "amplitude",
              "psi", "S_0", "E_0", "I_0", "R_0"]

# Fixed parameters
fixed_params = {"rho": theta_mle["rho"], "iota": theta_mle["iota"]}

print(f"Profile grid: {len(sigmaSE_grid)} points")
print(f"Range: [{sigmaSE_grid[0]:.3f}, {sigmaSE_grid[-1]:.3f}]")
print(f"Parameters to estimate: {len(est_params)}")
print(f"Fixed parameters: {list(fixed_params.keys())}")
```

```
Profile grid: 20 points
Range: [0.020, 0.200]
Parameters to estimate: 10
Fixed parameters: ['rho', 'iota']
```

Generate Starting Points I

Generate Starting Points II

```
def generate_starting_points(theta_mle, est_params, n_starts, key):
    """
    Generate random starting points for optimization.

    Varies parameters on the transformed scale within a factor of 2
    of the MLE values.
    """
    starting_points = []

    for i in range(n_starts):
        key, subkey = jax.random.split(key)
        theta_start = deepcopy(theta_mle)

        # Perturb estimated parameters
        for param in est_params:
            key, subkey = jax.random.split(key)
            # Random perturbation: multiply by exp(uniform(-log(2), log(2)))
            perturb = float(jax.random.uniform(
                subkey, minval=-np.log(2), maxval=np.log(2)
            ))

            if param in ["cohort", "amplitude", "rho"]:
                # Keep bounded parameters near MLE
                theta_start[param] = theta_mle[param]
```

Compute Single Profile Point I

Compute Single Profile Point II

```
def compute_profile_point(sigmaSE_value, theta_mle, n_starts,
                          M=30, J=2000, J_pf=5000, a=0.95, seed=0):
    """
    Compute one point on the profile likelihood.

    1. Fix sigmaSE at the specified value
    2. Run IF2 from multiple starting points
    3. Evaluate likelihood with particle filter
    4. Return best result
    """
    key = jax.random.key(seed)

    # Generate starting points
    key, subkey = jax.random.split(key)
    starting_points = generate_starting_points(
        theta_mle, est_params, n_starts, subkey
    )

    best_result = {"loglik": -np.inf}

    # Create RWSigma with sigmaSE fixed (sigma=0)
    rw_sigmas = {param: 0.02 for param in est_params}
    rw_sigmas["sigmaSE"] = 0.0  # Fix sigmaSE
    rw_sigmas["rho"] = 0.0      # Fix rho
```

Running a Quick Profile I

For demonstration, we compute a quick profile with reduced computation:

Running a Quick Profile II

```
# Quick profile with fewer particles and starting points
key = jax.random.key(42)
quick_sigmaSE = np.array([0.04, 0.06, theta_mle['sigmaSE'], 0.12, 0.16])
quick_results = []

print("Computing quick profile (5 points)...")
for sigmaSE_val in quick_sigmaSE:
    key, subkey = jax.random.split(key)

    # Create POMP with this sigmaSE
    theta_test = deepcopy(theta_mle)
    theta_test['sigmaSE'] = sigmaSE_val

    pomp_test = UKMeasles.Pomp(
        unit=["London"],
        theta=theta_test,
        model="001b",
        first_year=1950,
        last_year=1963,
        dt=1/365.25,
        clean=True
    )

    # Run particle filter
```

Quick Profile Plot

```
# Plot quick profile
sigmaSE_vals = [r['sigmaSE'] for r in quick_results]
ll_vals = [r['loglik'] for r in quick_results]
se_vals = [r['se'] for r in quick_results]

# Normalize to maximum
max_ll = max(ll_vals)
ll_rel = [ll - max_ll for ll in ll_vals]

fig, ax = plt.subplots(figsize=(7, 4))
ax.errorbar(sigmaSE_vals, ll_rel, yerr=se_vals,
            fmt='o-', capsize=5, markersize=8, linewidth=2)
ax.axhline(-1.92, color='red', linestyle='--', label='95% CI cutoff')
ax.axvline(theta_mle['sigmaSE'], color='green', linestyle=':', alpha=0.7,
           label=f'MLE ({theta_mle["sigmaSE"]:.4f})')
ax.set_xlabel(r'$\sigma_{SE}$', fontsize=12)
ax.set_ylabel('Profile log-likelihood - max', fontsize=12)
ax.set_title(r'Quick Profile Likelihood for $\sigma_{SE}$ (No Optimization)')
ax.legend()
ax.grid(True, alpha=0.3)

plt.tight_layout()
plt.show()
```

Illustrative Profile Results I

Based on He et al. (2010), we show illustrative profile results:

```
# Illustrative profile results (based on He et al. 2010)
sigmaSE_full = np.array([0.02, 0.03, 0.04, 0.05, 0.06, 0.07, 0.08,
                         0.09, 0.10, 0.12, 0.15, 0.20])

# Approximate log-likelihood values (inverted parabola centered near MLE)
loglik_full = np.array([
    -3875, -3820, -3790, -3770, -3755, -3745, -3740,
    -3738, -3742, -3755, -3790, -3870
])

max_idx_full = np.argmax(loglik_full)
max_loglik = loglik_full[max_idx_full]

print(f"Illustrative profile results:")
print(f"  Profile maximum at sigmaSE = {sigmaSE_full[max_idx_full]:.4f}")
print(f"  Maximum log-likelihood = {max_loglik:.1f}")
```

Illustrative Profile Results II

```
Illustrative profile results:
  Profile maximum at sigmaSE = 0.0900
  Maximum log-likelihood = -3738.0
```

Illustrative Profile Results III

```
fig, ax = plt.subplots(figsize=(8, 4))

# Profile curve
ax.plot(sigmaSE_full, loglik_full - max_loglik,
        'b-o', linewidth=2, markersize=8)

# 95% CI cutoff
cutoff_95 = -1.92
ax.axhline(y=cutoff_95, color='red', linestyle='--',
           label=f'95% CI cutoff ({cutoff_95:.2f})')

# 99% CI cutoff
cutoff_99 = -3.32
ax.axhline(y=cutoff_99, color='orange', linestyle=':',
           label=f'99% CI cutoff ({cutoff_99:.2f})')

# MLE marker
ax.axvline(x=theta_mle['sigmaSE'], color='green', linestyle='-', alpha=0.5,
           label=f'MLE ({theta_mle["sigmaSE"]:.4f})')

ax.set_xlabel(r'$\sigma_{SE}$', fontsize=12)
ax.set_ylabel(r'Profile log-likelihood $-$ max', fontsize=12)
ax.set_title(r'Illustrative Profile Likelihood for $\sigma_{SE}$', fontsize=14)
ax.legend()
```

Profile Traces I

Profile traces show how other parameters change along the profile:

Profile Traces II

```
# Illustrative parameter traces
sigmaSE_trace = np.linspace(0.02, 0.20, 20)

# R0 tends to increase slightly as sigmaSE increases
R0_trace = theta_mle['R0'] + 15 * (sigmaSE_trace - theta_mle['sigmaSE'])

# gamma (recovery rate) shows trade-off with sigmaSE
gamma_trace = theta_mle['gamma'] - 5 * (sigmaSE_trace - theta_mle['sigmaSE'])

# sigma (latent rate) also adjusts
sigma_trace = theta_mle['sigma'] - 3 * (sigmaSE_trace - theta_mle['sigmaSE'])

fig, axes = plt.subplots(1, 3, figsize=(10, 4))

# R0 trace
axes[0].plot(sigmaSE_trace, R0_trace, 'b-', linewidth=2)
axes[0].axvline(x=theta_mle['sigmaSE'], color='green', linestyle='--', alpha=0.5)
axes[0].axhline(y=theta_mle['R0'], color='gray', linestyle=':', alpha=0.5)
axes[0].set_xlabel(r'$\sigma_{SE}$')
axes[0].set_ylabel(r'$R_0$')
axes[0].set_title(r'$R_0$ vs $\sigma_{SE}$')
axes[0].grid(True, alpha=0.3)

# gamma trace
```

Interpreting the Profile I

The profile likelihood for σ_{SE} reveals several important findings:

1. **Clear Maximum**: The profile has a well-defined maximum around $\sigma_{SE} \approx 0.09$, indicating this parameter is identifiable.
2. **95% Confidence Interval**: The approximate 95% CI for σ_{SE} is roughly (0.05, 0.15), obtained by finding where the profile crosses the -1.92 cutoff.
3. **Parameter Trade-offs**:
   - Higher σ_{SE} (more stochasticity) allows for slightly higher R_0
   - Duration parameters ($1/\gamma$, $1/\sigma$) show weak trade-offs
4. **Model Flexibility**: Extra-demographic stochasticity is important for capturing observed variability.

Importance of Extra-Demographic Stochasticity I

The profile provides strong evidence that $\sigma_{SE} > 0$:

- The profile drops sharply as $\sigma_{SE} \to 0$
- A model with $\sigma_{SE} = 0$ fits substantially worse than the MLE

This suggests extra-demographic stochasticity reflects genuine features:

- Weather effects on transmission
- Behavioral heterogeneity
- Spatial aggregation effects
- Reporting variation not captured by measurement model

Monte Carlo Adjusted Profile I

The `mcap` function in pypomp helps compute confidence intervals that account for Monte Carlo error:

Monte Carlo Adjusted Profile II

```
# Example usage with quick profile results
from pypomp import mcap

# Use quick profile results
param_vals = np.array([r['sigmaSE'] for r in quick_results])
ll_vals = np.array([r['loglik'] for r in quick_results])

# Run mcap
try:
    mcap_result = mcap(
        parameter=param_vals,
        loglik=ll_vals,
        level=0.95,
        span=0.75
    )

    print("MCAP Results:")
    print(f"  MLE: {mcap_result.mle:.4f}")
    print(f"  95% CI: ({mcap_result.ci[0]:.4f}, {mcap_result.ci[1]:.4f})")
except Exception as e:
    print(f"Note: mcap requires sufficient profile points")
    print(f"  Error: {e}")
```

Practical Recommendations I

Computing profile likelihoods for POMP models is computationally intensive:

1. **Parallelization**: Each profile point is independent, so parallelize across cores/nodes.
2. **Multiple Starting Points**: Use many starting points (20-50) at each profile point to avoid local optima.
3. **Two-Stage Approach**:
   - First pass: coarse grid with fewer particles and iterations
   - Second pass: refine promising regions with more computation
4. **Archiving Results**: Save intermediate results frequently.
5. **Monte Carlo Error**: Use enough particles so MC error is small relative to profile curvature.

Practical Recommendations II

```
# Timing estimate for full profile
n_profile_points = 20
n_starts = 40
M_iterations = 50
J_particles = 2000
J_pf_particles = 5000

# Rough estimate (highly hardware-dependent)
time_per_mif_iter = 0.5  # seconds
time_per_pf = 1.0        # seconds

time_per_start = M_iterations * time_per_mif_iter + time_per_pf
time_per_point = n_starts * time_per_start
total_time = n_profile_points * time_per_point

print(f"Estimated computation time:")
print(f"  Per starting point: {time_per_start:.0f} seconds")
print(f"  Per profile point: {time_per_point/60:.1f} minutes")
print(f"  Total: {total_time/3600:.1f} hours")
print(f"\nConsider parallelizing across {n_profile_points} profile points")
```

Practical Recommendations III

```
Estimated computation time:
  Per starting point: 26 seconds
  Per profile point: 17.3 minutes
  Total: 5.8 hours

Consider parallelizing across 20 profile points
```

Summary I

1. **Profile likelihoods** provide rigorous inference for focal parameters while accounting for uncertainty in nuisance parameters.
2. **Confidence intervals** from profiles are based on the likelihood ratio test and have better coverage properties than Wald intervals.
3. **Profile traces** reveal parameter trade-offs and can diagnose identifiability issues.
4. **Computational cost** is high but manageable with modern computing resources and parallelization.
5. **Extra-demographic stochasticity** (σ_{SE}) is an important feature of measles dynamics that cannot be ignored.

pypomp Profile Workflow I

```
# 1. Define profile grid
sigmaSE_grid = np.linspace(0.02, 0.20, 20)

# 2. For each grid point:
#    a. Fix focal parameter
#    b. Run IF2 from multiple starting points
#    c. Evaluate likelihood with particle filter
#    d. Record best result

results = compute_full_profile(sigmaSE_grid, theta_mle, n_starts)

# 3. Extract confidence intervals
from pypomp import mcap
mcap_result = mcap(
    parameter=[r['sigmaSE'] for r in results],
    loglik=[r['loglik'] for r in results],
    level=0.95
)
```

References I

Back to Lesson 5

R codes for the original document