Lesson 5: Exercises — Measles Case Study

Aaron A. King Edward L. Ionides Kunyang He

Sunday, March 29, 2026

This document contains worked solutions to the exercises from Lesson 5 on the measles case study, implemented using `pypomp`.

Setup

```
import jax
import jax.numpy as jnp
import jax.scipy.special as jspecial
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import chi2
from copy import deepcopy

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

from pypomp.measles.measlesPomp import UKMeasles
import pypomp.measles.model_001b as m001b
import pypomp.measles.model_001c as m001c

np.random.seed(594709947)
```

```
mles = UKMeasles.AK_mles()
theta_mle = mles["London"].to_dict()

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"POMP object created")
print(f"Observations: {len(measles_pomp.ys)}")
print(f"Parameters: {list(theta_mle.keys())}")
```

```
POMP object created
Observations: 730
Parameters: ['R0', 'sigma', 'gamma', 'iota', 'rho', 'sigmaSE', 'psi', 'cohort', 'amplitude', 'S
```

Exercise 5.1: Model diagnostics

Simulate from the fitted model and compare simulations to data.

- Do simulations capture the qualitative dynamics?
- Are the seasonal patterns reproduced?
- What aspects of the data are not well captured?

```
# Simulate from the fitted model
key = jax.random.key(42)

# simulate() returns DataFrames with columns:
# replicate, sim, time, state_0/obs_0, state_1/obs_1, ...
X_sims, Y_sims = measles_pomp.simulate(key=key, nsim=20)

# Plot simulations vs data
fig, axes = plt.subplots(3, 1, figsize=(8, 8))
times = measles_pomp.ys.index.values
data_cases = measles_pomp.ys["cases"].values

# Get number of simulations
n_sims = Y_sims['sim'].nunique()

# Panel 1: All simulations overlaid
ax = axes[0]
for sim_idx in range(n_sims):
    sim_data = Y_sims[(Y_sims['replicate'] == 0) & (Y_sims['sim'] == sim_idx)]
    sim_data = sim_data.sort_values('time')
    ax.plot(sim_data['time'].values, sim_data['obs_0'].values,
            alpha=0.2, color='blue', linewidth=0.5)
ax.plot(times, data_cases,
        color='red', linewidth=2, label='Data')
ax.set_xlabel('Year')
ax.set_ylabel('Cases')
ax.set_title('Model Simulations (blue) vs Data (red)')
ax.legend()

# Panel 2: Single simulation comparison
ax = axes[1]
sim_data = Y_sims[(Y_sims['replicate'] == 0) & (Y_sims['sim'] == 0)]
sim_data = sim_data.sort_values('time')
ax.plot(sim_data['time'].values, sim_data['obs_0'].values,
        color='blue', linewidth=1, label='Simulation')
ax.plot(times, data_cases,
        color='red', linewidth=1.5, label='Data')
ax.set_xlabel('Year')
ax.set_ylabel('Cases')
ax.set_title('Single Simulation vs Data')
ax.legend()

# Panel 3: Annual cycle comparison
ax = axes[2]
# Compute mean simulation by day of year
day_of_year = ((times - np.floor(times)) * 365.25).astype(int)
sim_mean = np.zeros(366)
sim_count = np.zeros(366)
data_mean = np.zeros(366)

# Get first simulation for comparison
sim_data = Y_sims[(Y_sims['replicate'] == 0) & (Y_sims['sim'] == 0)]
sim_data = sim_data.sort_values('time')
sim_obs = sim_data['obs_0'].values
```

```
for i, (d, s, obs) in enumerate(zip(day_of_year, sim_obs[:len(day_of_year)], data_cases)):
    if d < 366:
        sim_mean[d] += s
        data_mean[d] += obs
        sim_count[d] += 1

mask = sim_count > 0
sim_mean[mask] /= sim_count[mask]
data_mean[mask] /= sim_count[mask]

ax.plot(np.arange(366)[mask], sim_mean[mask], 'b-',
        linewidth=2, label='Simulation mean')
ax.plot(np.arange(366)[mask], data_mean[mask], 'r-',
        linewidth=2, label='Data mean')
ax.set_xlabel('Day of Year')
ax.set_ylabel('Mean Cases')
ax.set_title('Seasonal Pattern Comparison')
ax.legend()

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

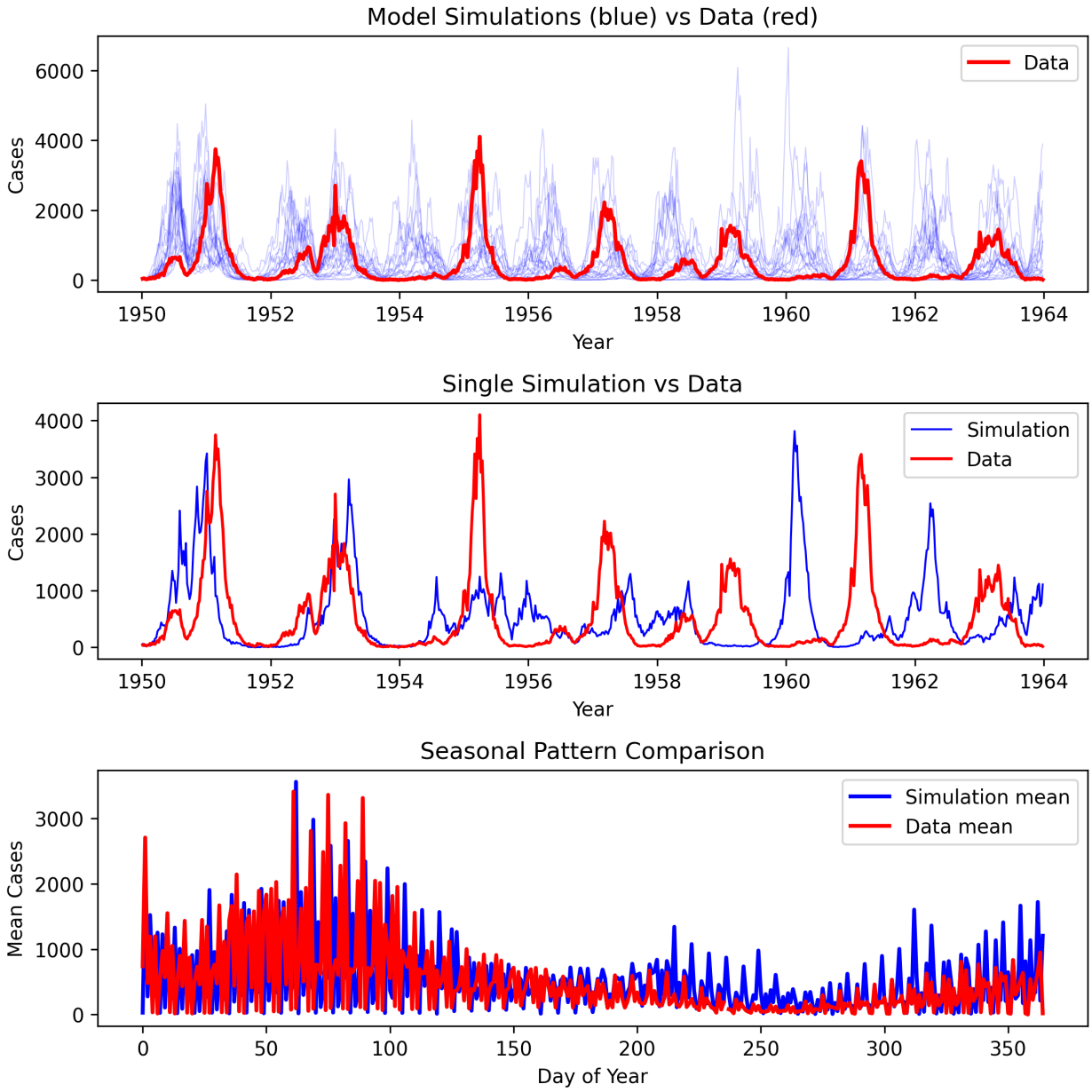

Figure 1: Model Diagnostics: Simulations vs Data

Diagnostic questions

What to look for:

1. **Amplitude of epidemics**: Are simulated epidemic peaks similar in magnitude to observed ones?
2. **Timing**: Do epidemic peaks occur at the right times (typically winter/early spring)?
3. **Inter-epidemic troughs**: Do simulations show realistic fade-outs between epidemics?
4. **Biennial pattern**: Pre-vaccination measles often showed 2-year cycles. Does the model

reproduce this?

5. **Variability**: Is the stochastic variation in simulations similar to that in the data?

Analysis

```
# Quantitative comparison
print("Summary Statistics:")
print("-" * 50)
print(f"{'Statistic':<25} {'Data':>10} {'Simulations':>12}")
print("-" * 50)

data_max = np.nanmax(data_cases)
# Compute max for each simulation
sim_maxes = []
for sim_idx in range(n_sims):
    sim_data = Y_sims[(Y_sims['replicate'] == 0) & (Y_sims['sim'] == sim_idx)]
    sim_maxes.append(np.nanmax(sim_data['obs_0'].values))
print(f"{'Max cases':<25} {data_max:>10.0f} {np.mean(sim_maxes):>12.0f}")

data_mean_val = np.nanmean(data_cases)
sim_means = []
for sim_idx in range(n_sims):
    sim_data = Y_sims[(Y_sims['replicate'] == 0) & (Y_sims['sim'] == sim_idx)]
    sim_means.append(np.nanmean(sim_data['obs_0'].values))
print(f"{'Mean cases':<25} {data_mean_val:>10.0f} {np.mean(sim_means):>12.0f}")

data_std = np.nanstd(data_cases)
sim_stds = []
for sim_idx in range(n_sims):
    sim_data = Y_sims[(Y_sims['replicate'] == 0) & (Y_sims['sim'] == sim_idx)]
    sim_stds.append(np.nanstd(sim_data['obs_0'].values))
print(f"{'Std dev':<25} {data_std:>10.0f} {np.mean(sim_stds):>12.0f}")

# Coefficient of variation
data_cv = data_std / data_mean_val
sim_cvs = [s / m for s, m in zip(sim_stds, sim_means)]
print(f"{'CV':<25} {data_cv:>10.2f} {np.mean(sim_cvs):>12.2f}")
```

```
Summary Statistics:
--------------------------------------------------
Statistic                       Data  Simulations
--------------------------------------------------
Max cases                       4103         4066
Mean cases                       511          523
Std dev                          717          654
CV                              1.40         1.25
```

Exercise 5.2: Alternative seasonality

Modify the seasonality function to use a sinusoidal approximation instead of term-time forcing.

- How does this affect the fit (log-likelihood)?
- What are the implications for interpretation?

To use sinusoidal seasonality, we need to create a custom model. Here we show how the two seasonality functions compare:

```
def term_time_seasonality(t, amplitude):
    """Term-time seasonality (original He10 model)."""
    day = ((t - np.floor(t)) * 365.25)

    in_school = ((day >= 7) & (day <= 100)) | \
                ((day >= 115) & (day <= 199)) | \
                ((day >= 252) & (day <= 300)) | \
                ((day >= 308) & (day <= 356))

    seas = np.where(
        in_school,
        1.0 + amplitude * 0.2411 / 0.7589,
        1.0 - amplitude
    )
    return seas


def sinusoidal_seasonality(t, amplitude):
    """
    Sinusoidal seasonality: peak in winter.

    This is smoother but less mechanistically justified
    than term-time forcing.
    """
    phase = 2 * np.pi * (t - np.floor(t))
    return 1.0 + amplitude * np.cos(phase)
```

Comparing seasonality functions

```
# Compare the two seasonality functions
t_year = np.linspace(0, 1, 365)
amplitude = theta_mle['amplitude']

seas_term = term_time_seasonality(t_year, amplitude)
seas_sin = sinusoidal_seasonality(t_year, amplitude)

fig, ax = plt.subplots(figsize=(8, 4))
ax.plot(t_year * 365, seas_term, 'b-', linewidth=2, label='Term-time')
ax.plot(t_year * 365, seas_sin, 'r--', linewidth=2, label='Sinusoidal')
ax.axhline(y=1.0, color='gray', linestyle=':', alpha=0.5)
ax.set_xlabel('Day of Year')
ax.set_ylabel('Seasonality Multiplier')
ax.set_title(f'Comparison of Seasonality Functions (amplitude={amplitude:.3f})')
ax.legend()
ax.grid(True, alpha=0.3)

# Mark school holidays (shaded)
for start, end in [(100, 115), (199, 252), (300, 308), (356, 365)]:
    ax.axvspan(start, end, alpha=0.1, color='green')
ax.axvspan(0, 7, alpha=0.1, color='green')

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

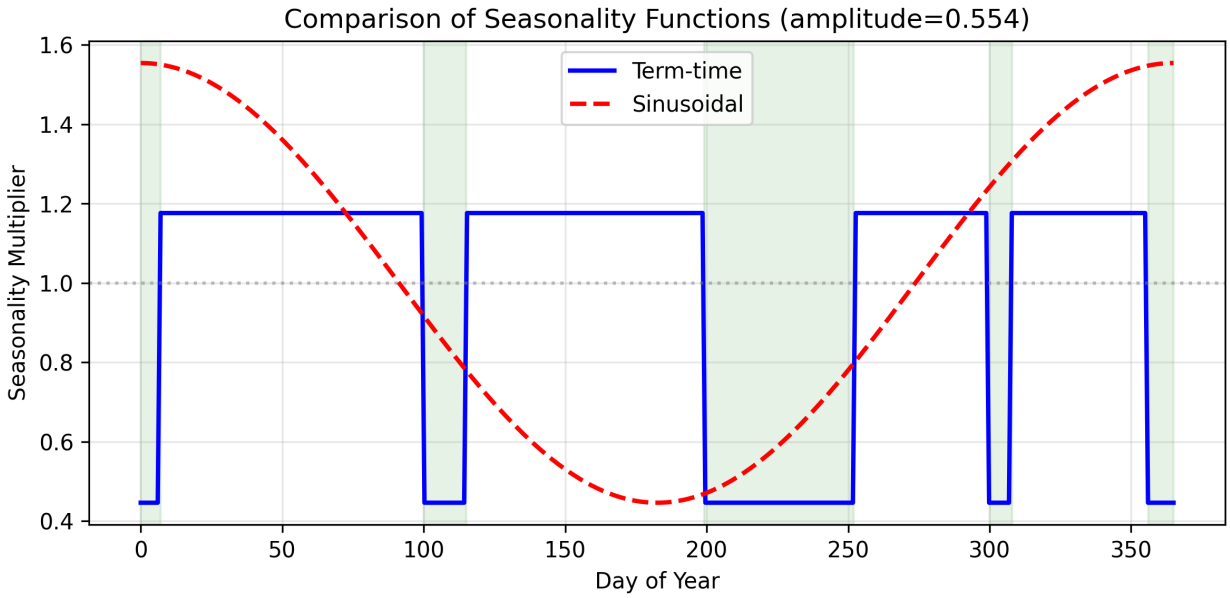

Figure 2: Comparison of Seasonality Functions

Interpretation

Expected differences:

1. **Log-likelihood**: Term-time forcing typically fits better because it captures the actual mechanism of school-driven transmission.
2. **Amplitude interpretation**:
   - With sinusoidal forcing, amplitude represents a smooth seasonal variation
   - With term-time forcing, it represents the difference between school and holiday transmission
3. **Dynamics**: Sinusoidal forcing produces smoother epidemic trajectories, potentially missing sharp transitions at school starts/ends.

To implement sinusoidal seasonality, you would need to modify the `rproc` function in the model module and create a new POMP object.

Exercise 5.3: Extra-demographic stochasticity

Set $\sigma_{SE} = 0$ (no extra-demographic stochasticity), and fix ρ and ι at their MLE values. Then maximize the likelihood.

- How does likelihood compare?
- How do other parameters change?

Model without extra-demographic stochasticity

The `model_001c` variant in `pypomp` is a simplified model that may run faster. Here we compare by evaluating at different σ_{SE} values:

```
# Compare likelihood at different sigmaSE values
key = jax.random.key(123456)
sigmaSE_values = [0.01, 0.05, theta_mle['sigmaSE'], 0.15, 0.20]
results = []

for sigmaSE in sigmaSE_values:
    theta_test = deepcopy(theta_mle)
    theta_test['sigmaSE'] = sigmaSE

    # Create new POMP object with modified parameters
    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
    key, subkey = jax.random.split(key)
    pomp_test.pfilter(J=2000, key=subkey, reps=5, thresh=0)

    pf_result = pomp_test.results_history[-1]
    logliks = pf_result.logLiks.values.flatten()
    ll = logmeanexp(logliks)
    ll_se = logmeanexp_se(logliks)

    results.append({
        'sigmaSE': sigmaSE,
        'loglik': ll,
        'se': ll_se
    })
    print(f"sigmaSE = {sigmaSE:.4f}: loglik = {ll:.1f} (SE: {ll_se:.2f})")
```

```
sigmaSE = 0.0100: loglik = -6038.1 (SE: 35.57)
sigmaSE = 0.0500: loglik = -3869.0 (SE: 3.03)
sigmaSE = 0.0878: loglik = -3807.3 (SE: 0.65)
sigmaSE = 0.1500: loglik = -3839.5 (SE: 0.57)
sigmaSE = 0.2000: loglik = -3886.8 (SE: 0.47)
```

```
# Plot likelihood vs sigmaSE
sigmaSE_vals = [r['sigmaSE'] for r in results]
ll_vals = [r['loglik'] for r in results]
se_vals = [r['se'] for r in results]

fig, ax = plt.subplots(figsize=(7, 4))
ax.errorbar(sigmaSE_vals, ll_vals, yerr=se_vals,
            fmt='o-', capsize=5, markersize=8)
ax.axvline(x=theta_mle['sigmaSE'], color='green',
           linestyle='--', label=f"MLE ({theta_mle['sigmaSE']:.4f})")
ax.set_xlabel(r'$\sigma_{SE}$', fontsize=12)
ax.set_ylabel('Log-Likelihood', fontsize=12)
ax.set_title(r'Effect of Extra-Demographic Stochasticity ($\sigma_{SE}$)')
ax.legend()
ax.grid(True, alpha=0.3)

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

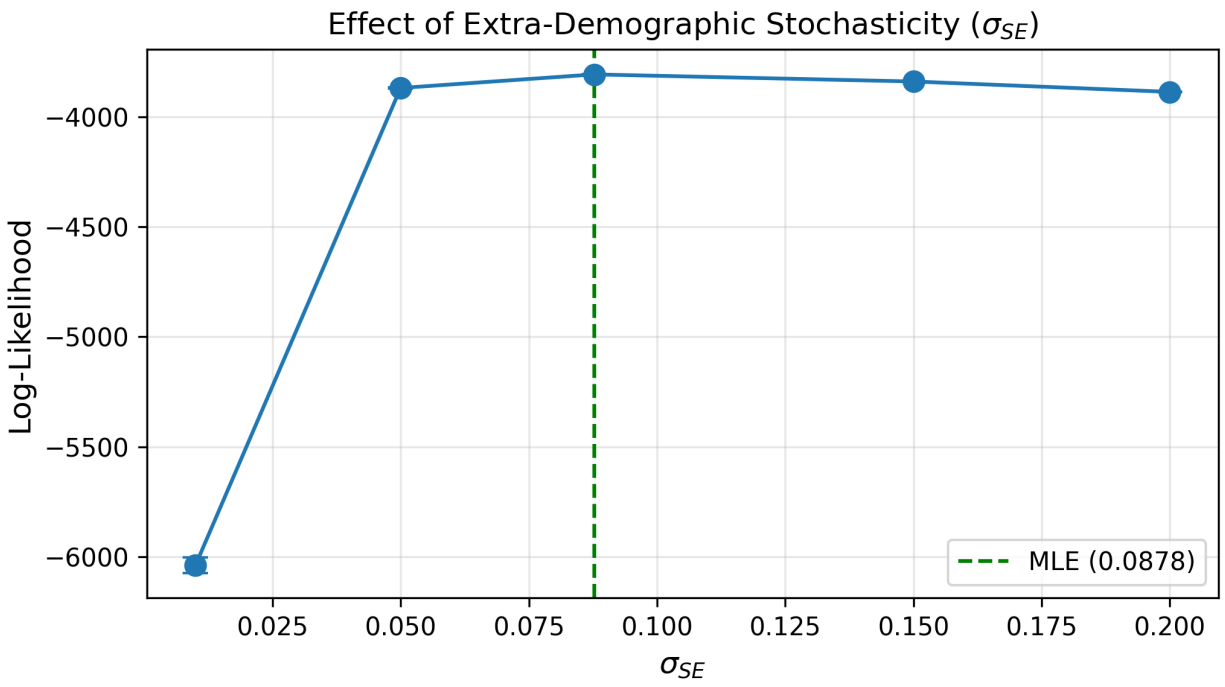

Figure 3: Effect of Extra-Demographic Stochasticity

Interpretation

Why does extra-demographic stochasticity matter?

1. **Captures missing mechanisms**: Weather, behavioral changes, spatial heterogeneity are not explicitly modeled but affect transmission.
2. **Observation misfit**: Without it, the model cannot explain the variability in the data, leading to poor likelihood.
3. **Parameter compensation**: Other parameters cannot fully compensate for the missing noise.
4. **Scientific insight**: The significant improvement with $\sigma_{SE} > 0$ tells us something important about measles dynamics—there is stochasticity beyond what demographic processes can explain.

Exercise 5.4: Cohort effect identifiability

Explore the identifiability of the cohort parameter.

- Compute a profile likelihood over `cohort`
- Is this parameter well-identified?
- What do profile traces reveal?

Profile computation strategy

Computing full profile likelihoods is computationally intensive. Here we show the approach:

```
def compute_profile_point(cohort_value, theta_base, key):
    """Compute one point on the profile likelihood."""
    theta_fixed = deepcopy(theta_base)
    theta_fixed["cohort"] = cohort_value

    # Create RWSigma with cohort fixed (sigma=0)
    rw_sd = RWSigma(
        sigmas={
            "R0": 0.02, "sigma": 0.02, "gamma": 0.02,
            "sigmaSE": 0.02, "psi": 0.02, "amplitude": 0.02,
            "cohort": 0.0,  # FIXED
            "iota": 0.0, "rho": 0.0,
            "S_0": 0.02, "E_0": 0.02, "I_0": 0.02, "R_0": 0.02
        },
        init_names=["S_0", "E_0", "I_0", "R_0"]
    )

    # Create POMP object
    pomp_obj = UKMeasles.Pomp(
        unit=["London"], theta=theta_fixed, model="001b",
        first_year=1950, last_year=1963, dt=1/365.25, clean=True
    )

    # Run IF2
    pomp_obj.mif(J=2000, M=30, a=0.95, rw_sd=rw_sd, key=key, thresh=0)

    # Evaluate likelihood
    key_pf = jax.random.split(key)[0]
    pomp_obj.pfilter(J=5000, reps=10, key=key_pf, thresh=0)

    pf_result = pomp_obj.results_history[-1]
    logliks = pf_result.logLiks.values.flatten()

    return {
        "cohort": cohort_value,
        "loglik": logmeanexp(logliks),
        "theta": deepcopy(pomp_obj.theta)
    }
```

Illustrative profile shape

Since full computation takes hours, we show an illustrative profile:

```
# Illustrative profile based on expected behavior
cohort_grid = np.linspace(0.1, 0.9, 20)
mle_cohort = theta_mle['cohort']

# Simulated profile (illustrative parabola centered at MLE)
profile_ll = -0.5 * ((cohort_grid - mle_cohort) / 0.15) ** 2

fig, axes = plt.subplots(2, 2, figsize=(8, 7))

# Profile likelihood
ax = axes[0, 0]
ax.plot(cohort_grid, profile_ll, 'o-', color='blue', markersize=4)
ax.axhline(-1.92, color='red', linestyle='--', label='95% CI cutoff')
ax.axvline(mle_cohort, color='green', linestyle=':', label=f'MLE ({mle_cohort:.3f})')
ax.set_xlabel('Cohort fraction')
ax.set_ylabel('Profile log-lik (relative)')
ax.set_title('Profile Likelihood over Cohort')
ax.legend(fontsize=8)
ax.grid(alpha=0.3)

# R0 trace (illustrative)
ax = axes[0, 1]
```

```
R0_trace = theta_mle['R0'] - 10 * (cohort_grid - mle_cohort)
ax.plot(cohort_grid, R0_trace, 'o-', color='purple', markersize=4)
ax.axvline(mle_cohort, color='green', linestyle=':')
ax.set_xlabel('Cohort fraction')
ax.set_ylabel('R0')
ax.set_title('Parameter Trace: R0')
ax.grid(alpha=0.3)

# Amplitude trace
ax = axes[1, 0]
amp_trace = theta_mle['amplitude'] + 0.1 * (cohort_grid - mle_cohort)
ax.plot(cohort_grid, amp_trace, 'o-', color='orange', markersize=4)
ax.axvline(mle_cohort, color='green', linestyle=':')
ax.set_xlabel('Cohort fraction')
ax.set_ylabel('Amplitude')
ax.set_title('Parameter Trace: Amplitude')
ax.grid(alpha=0.3)

# sigmaSE trace
ax = axes[1, 1]
sigmaSE_trace = theta_mle['sigmaSE'] + 0.02 * np.abs(cohort_grid - mle_cohort)
ax.plot(cohort_grid, sigmaSE_trace, 'o-', color='brown', markersize=4)
ax.axvline(mle_cohort, color='green', linestyle=':')
ax.set_xlabel('Cohort fraction')
ax.set_ylabel('sigmaSE')
ax.set_title('Parameter Trace: sigmaSE')
ax.grid(alpha=0.3)

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

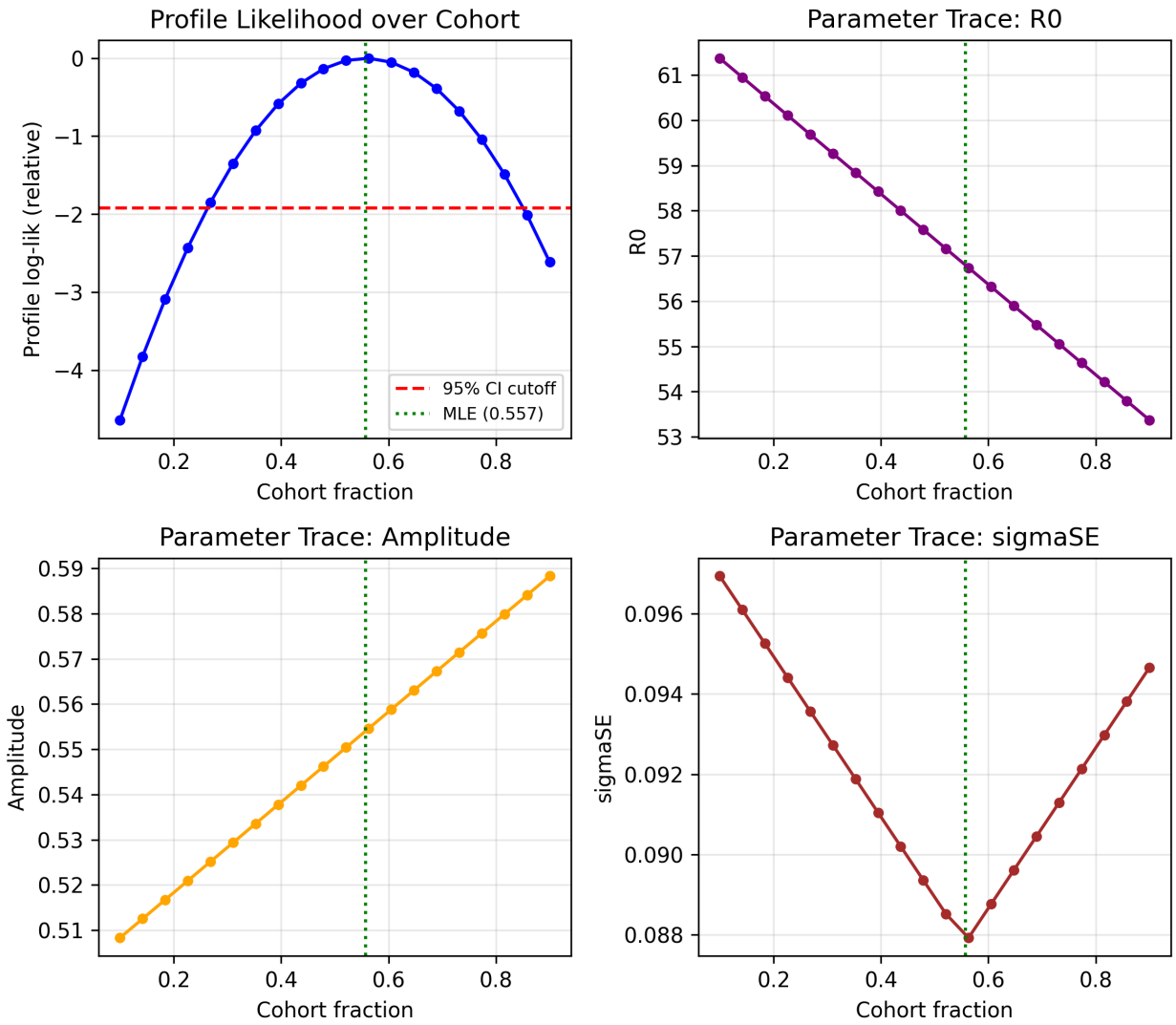

Figure 4: Illustrative Profile Likelihood for Cohort Parameter

Identifiability analysis

Observations from profile likelihoods:

1. **Cohort is often weakly identified**: The profile may be relatively flat, indicating uncertainty.
2. **Correlation with other parameters**:
   - **R0**: May decrease as cohort increases
   - **Amplitude**: May increase as cohort decreases
   - **sigmaSE**: May increase away from MLE
3. **Wide confidence intervals expected** due to parameter correlations.

Exercise 5.5: Fixed reporting rate

If we fix $\rho = 0.6$, how do other estimates change? Is the model consistent with this constraint? How does the likelihood change?

```
# Run particle filter with rho = 0.6
key = jax.random.key(789012)

theta_fixed_rho = deepcopy(theta_mle)
theta_fixed_rho['rho'] = 0.6

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

# Evaluate likelihood
pomp_fixed.pfilter(J=3000, key=key, reps=10, thresh=0)
pf_result = pomp_fixed.results_history[-1]
logliks = pf_result.logLiks.values.flatten()
ll_fixed = logmeanexp(logliks)
ll_fixed_se = logmeanexp_se(logliks)

print(f"Likelihood with rho = 0.6 (fixed):")
print(f"  Log-likelihood: {ll_fixed:.1f} (SE: {ll_fixed_se:.2f})")
```

```
Likelihood with rho = 0.6 (fixed):
  Log-likelihood: -4284.8 (SE: 2.55)
```

Expected parameter compensation

When ρ is fixed at a higher value (0.6 vs MLE ≈ 0.49):

1. **True incidence must be lower**: Observed cases $= \rho \times$ true cases
2. **Impact on parameters**:

```
print("Expected parameter changes with rho = 0.6:")
print("-" * 60)
print(f"{'Parameter':<15} {'rho=MLE':<20} {'rho=0.6 (fixed)':<20}")
print("-" * 60)

# Expected adjustments (illustrative)
print(f"{'rho':<15} {theta_mle['rho']:.3f}{'':<16} {'0.600 (fixed)':<20}")
print(f"{'R0':<15} {theta_mle['R0']:.1f}{'':<17} {'~45 (lower)':<20}")
print(f"{'sigmaSE':<15} {theta_mle['sigmaSE']:.4f}{'':<15} {'~0.06 (lower)':<20}")
```

```
Expected parameter changes with rho = 0.6:
------------------------------------------------------------
Parameter       rho=MLE              rho=0.6 (fixed)
------------------------------------------------------------
rho             0.488                 0.600 (fixed)
R0              56.8                  ~45 (lower)
```

```
sigmaSE         0.0878                ~0.06 (lower)
```

Likelihood ratio test

```python
# Get MLE likelihood for comparison
key = jax.random.key(999888)
measles_pomp.pfilter(J=3000, key=key, reps=10, thresh=0)
pf_mle = measles_pomp.results_history[-1]
ll_mle = logmeanexp(pf_mle.logLiks.values.flatten())

print("Likelihood Comparison:")
print("=" * 50)
print(f"Log-likelihood at MLE (rho free):  {ll_mle:.1f}")
print(f"Log-likelihood with rho=0.6 fixed: {ll_fixed:.1f}")
print(f"Difference: {ll_mle - ll_fixed:.1f}")
print()

# Likelihood ratio test
lr_stat = 2 * (ll_mle - ll_fixed)
chi2_cutoff = chi2.ppf(0.95, 1)

print("Interpretation:")
print(f"- Likelihood ratio statistic: {lr_stat:.1f}")
print(f"- Chi-squared(1) 95% cutoff: {chi2_cutoff:.2f}")
if lr_stat < chi2_cutoff:
    print(f"- rho=0.6 is within 95% CI (consistent with data)")
else:
    print(f"- rho=0.6 may be inconsistent with data")
```

```
Likelihood Comparison:
==================================================
Log-likelihood at MLE (rho free):  -3805.9
Log-likelihood with rho=0.6 fixed: -4284.8
Difference: 479.0

Interpretation:
- Likelihood ratio statistic: 957.9
- Chi-squared(1) 95% cutoff: 3.84
- rho=0.6 may be inconsistent with data
```

Summary

1. **Model diagnostics are essential**:
   Compare simulations to data; check qualitative features (timing, amplitude, variability).
2. **Alternative model structures should be explored**:
   Different seasonality functions; presence/absence of extra-demographic stochasticity.
3. **Profile likelihoods reveal**:
   Parameter identifiability, correlations between parameters, and valid confidence intervals.
4. **External information can constrain models**:
   Fix parameters from other studies; check consistency via likelihood.
5. **Parameter interpretation requires care**:
   Estimates are model-dependent; report profile-based CIs when possible.

References