Lesson 6: Case Study - Polio in Wisconsin

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

2025-12-24

Table of contents I

Table of contents II

Table of contents III

This lesson presents a case study of polio in Wisconsin, demonstrating the use of covariates in **pypomp** and likelihood-based inference for partially observed Markov process models.

Learning Objectives

1. Demonstrate the use of covariates in **pypomp** to add demographic data (birth rates and total population) and seasonality to an epidemiological model.
2. Show how partially observed Markov process (POMP) models and methods can be used to understand transmission dynamics of polio.
3. Practice maximizing the likelihood for such models. How to set up a global search for a maximum likelihood estimate. How to assess whether a search has been successful.
4. Provide a workflow that can be adapted to related data analysis tasks.

What are Covariates? I

- Suppose our time series of primary interest is $y_{1:N}$.
- A *covariate* time series is an additional time series $z_{1:N}$ which is used to help explain $y_{1:N}$.
- When we talk about covariates, it is often implicit that we think of $z_{1:N}$ as a measure of an *external forcing* to the system producing $y_{1:N}$. This means that the process generating the data $z_{1:N}$ affects the process generating $y_{1:N}$, but not vice versa.

What are Covariates? II

- For example, the weather might affect human health, but human health has negligible effect on weather: weather is an external forcing to human health processes.
- When the process leading to $z_{1:N}$ is not external to the system generating it, we must be alert to the possibility of *reverse causation* and *confounding variables*.

Including Covariates in the General POMP Framework I

The general POMP modeling framework allows essentially arbitrary modeling of covariates.

Recall that a POMP model is specified by defining, for $n = 1:N$,

$$\begin{array}{l} f_{X_0}(x_0;\theta), \\ f_{X_n|X_{n-1}}(x_n|x_{n-1};\theta), \\ f_{Y_n|X_n}(y_n|x_n;\theta). \end{array}$$

Including Covariates in the General POMP Framework II

The possibility of a general dependence on n includes the possibility that there is some covariate time series $z_{0:N}$ such that

$$\begin{array}{lcl} f_{X_0}(x_0;\theta) & = & f_{X_0}(x_0;\theta,z_0) \\ f_{X_n|X_{n-1}}(x_n|x_{n-1};\theta) & = & f_{X_n|X_{n-1}}(x_n|x_{n-1};\theta,z_n), \\ f_{Y_n|X_n}(y_n|x_n;\theta) & = & f_{Y_n|X_n}(y_n|x_n;\theta,z_n). \end{array}$$

Seasonality in a POMP Model I

- One specific choice of covariates is to construct $z_{0:N}$ so that it fluctuates periodically, once per year. This allows *seasonality* to enter the POMP model in whatever way is appropriate for the system under investigation.
- All that remains is to hypothesize what is a reasonable way to include covariates for your system, and to fit the resulting model.
- Now we can evaluate and maximize the log-likelihood, we can construct AIC or likelihood ratio tests to see if the covariate helps describe the data.
- This also lets us compare alternative ways the covariates might enter the process model and/or the measurement model.

Covariates in pypomp I

- **pypomp** provides facilities for including covariates in a Pomp object.
- Covariates entered via the `covars` argument to `Pomp` are automatically interpolated to match observation times and passed to the `rinit`, `rproc`, `dmeas`, and `rmeas` functions.
- We see this in practice in the following epidemiological model, which has population census, birth data and seasonality as covariates.

Background I

- The massive global polio eradication initiative (GPEI) has brought polio from a major global disease to the brink of extinction.
- Finishing this task is proving hard, and improved understanding of polio ecology might assist.
- Martinez-Bakker et al. (2015) investigated this using extensive state level pre-vaccination era data in USA.
- We will follow the approach of Martinez-Bakker et al. (2015) for one state (Wisconsin). In the context of their model, we can quantify seasonality of transmission, the role of the birth rate in explaining the transmission dynamics, and the persistence mechanism of polio.

Background II

- Martinez-Bakker et al. (2015) carried out this analysis for all 48 contiguous states and District of Columbia, and their data and code are publicly available.
- The data we study, in `polio_wisconsin.csv`, consist of:
  - `cases`: the monthly reported polio cases
  - `births`: the monthly recorded births
  - `pop`: the annual census
  - `time`: date in years

Loading the Data I

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

# For reproducibility
np.random.seed(594709947)

# Load the data
# Note: You can download from https://kingaa.github.io/sbied/polio/polio_wisconsin.c
data = pd.read_csv("polio_wisconsin.csv", comment='#')
print(data.head())
```

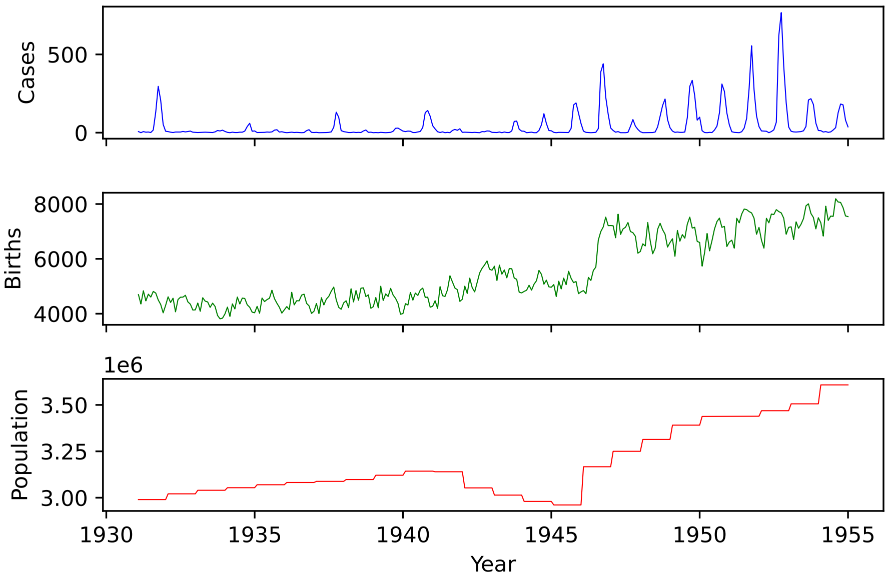
Loading the Data II

```
          time  cases  births      pop
0  1931.083333      7    4698  2990000
1  1931.166667      0    4354  2990000
2  1931.250000      7    4836  2990000
3  1931.333333      3    4468  2990000
4  1931.416667      4    4712  2990000
```

Visualizing the Data I

```
fig, axes = plt.subplots(3, 1, figsize=(6, 4), sharex=True)

axes[0].plot(data['time'], data['cases'], 'b-', linewidth=0.5)
axes[0].set_ylabel('Cases')

axes[1].plot(data['time'], data['births'], 'g-', linewidth=0.5)
axes[1].set_ylabel('Births')

axes[2].plot(data['time'], data['pop'], 'r-', linewidth=0.5)
axes[2].set_ylabel('Population')
axes[2].set_xlabel('Year')

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

Visualizing the Data II



Compartment Model I

We use the compartment model of Martinez-Bakker et al. (2015):

- Compartments representing susceptible babies in each of six one-month birth cohorts (S^B_1,...,S^B_6)
- Susceptible older individuals (S^O)
- Infected babies (I^B)
- Infected older individuals (I^O)
- Recovered with lifelong immunity (R)

Compartment Model II

The state vector of the disease transmission model consists of numbers of individuals in each compartment at each time:

$$X(t) = \big(S_1^B(t), ..., S_6^B(t), I^B(t), I^O(t), R(t)\big).$$

- Babies under six months are modeled as fully protected from symptomatic poliomyelitis.
- Older infections lead to reported cases (usually paralysis) at a rate ρ.

Force of Infection I

- Duration of infection is comparable to the one-month reporting aggregation, so a discrete time model may be appropriate.
- Martinez-Bakker et al. (2015) fitted monthly reported cases, May 1932 through January 1953, so we set $t_n = 1932 + (4+n)/12$ and

$$X_n = X(t_n) = (S^B_{1,n}, ..., S^B_{6,n}, I^B_n, I^O_n, R_n).$$

Force of Infection II

- The mean force of infection, in units of yr^{-1}, is modeled as

$$\bar{\lambda}_n = \left(\beta_n \frac{I^O_n + I^B_n}{P_n} + \psi\right)$$

where P_n is census population interpolated to time t_n and seasonality of transmission is modeled as

$$\beta_n = \exp\left\{\sum_{k=1}^{K} b_k \xi_k(t_n)\right\},$$

with $\{\xi_k(t), k = 1, \dots, K\}$ a periodic B-spline basis with $K = 6$.

- P_n and $\xi_k(t_n)$ are *covariate time series*.

Stochastic Force of Infection I

- The force of infection has a stochastic perturbation,

$$\lambda_n = \bar{\lambda}_n \epsilon_n,$$

  where ϵ_n is a Gamma random variable with mean 1 and variance $\sigma^2_{\mathrm{env}} + \sigma^2_{\mathrm{dem}}/\bar{\lambda}_n$.
- These two terms capture variation on the environmental and demographic scales, respectively.
- All compartments suffer a mortality rate, set at $\delta = 1/60\,\mathrm{yr}^{-1}$.

Stochastic Force of Infection II

- Within each month, all susceptible individuals are modeled as having exposure to constant competing hazards of mortality and polio infection. The chance of remaining in the susceptible population when exposed to these hazards for one month is therefore

$$p_n = \exp\big\{-(\delta + \lambda_n)/12\big\},$$

  with the chance of polio infection being

$$q_n = (1 - p_n)\lambda_n/(\lambda_n + \delta).$$

State Variables and Parameters I

```
# State variable names
statenames = ["SB1", "SB2", "SB3", "SB4", "SB5", "SB6", "IB", "SO", "IO"]

# Initial time
t0 = 1932 + 4/12

# Number of B-spline basis functions for seasonality
K = 6

# We do not explicitly code R, since it is defined implicitly as the
# total population minus the sum of the other compartments.
```

State Variables and Parameters II

```python
# Regular parameters (RPs) - to be estimated
rp_names = ["b1", "b2", "b3", "b4", "b5", "b6",
            "psi", "rho", "tau", "sigma_dem", "sigma_env"]

# Initial value parameters (IVPs) - to be estimated
ivp_names = ["S0_0", "I0_0"]

# Fixed parameters (FPs)
fp_names = ["delta", "SB1_0", "SB2_0", "SB3_0", "SB4_0", "SB5_0", "SB6_0"]

paramnames = rp_names + ivp_names + fp_names
```

Loading Covariates from R-generated File I

The covariate file `covar1.csv` is pre-generated from R/pomp to ensure consistency with the original analysis. It contains births (B), population (P), and periodic B-spline basis functions (xi1-xi6).

Loading Covariates from R-generated File II

```python
# Load covariates from R-generated file
# This file is generated by pomp to match the original analysis
cov_raw = pd.read_csv("covar1.csv")

# The file has variable names in first column, times as column headers
# We need to transpose it
first_col = cov_raw.columns[0]
covars = (cov_raw.rename(columns={first_col: "var"})
                 .set_index("var")
                 .T)

# Convert index to float (time values)
covars.index = covars.index.astype(float)
covars.index.name = "time"
covars = covars.sort_index()

print(f"Covariate columns: {list(covars.columns)}")
print(f"Time range: {covars.index[0]:.4f} to {covars.index[-1]:.4f}")
print(covars.head())
```

Loading Covariates from R-generated File III

```
Covariate columns: ['B', 'xi1', 'xi2', 'xi3', 'xi4', 'xi5', 'x
Time range: 1931.0833 to 1955.0000
var                 B           xi1       xi2       xi3
time
1931.083333  4698.0  4.791667e-01  0.479167  0.020833  0.0000
1931.166667  4354.0  1.666667e-01  0.666667  0.166667  1.2461
1931.250000  4836.0  2.083333e-02  0.479167  0.479167  2.0833
1931.333333  4468.0  1.246112e-33  0.166667  0.666667  1.6666
1931.416667  4712.0  0.000000e+00  0.020833  0.479167  4.7916

var                 xi6             P
time
1931.083333  0.020833  2.993754e+06
1931.166667  0.000000  2.996197e+06
1931.250000  0.000000  2.998629e+06
```

Loading Covariates from R-generated File IV

```
1931.333333  0.000000  3.001045e+06
1931.416667  0.000000  3.003439e+06
```

Fixed Parameters and Starting Values I

Fixed Parameters and Starting Values II

```python
# Find initial births for SB1_0 through SB6_0 from covariate table
def get_initial_births(covars_df, t0_val):
    """Extract initial birth values from covariate table."""
    idx0 = np.argmin(np.abs(covars_df.index.values - t0_val))
    B_series = covars_df["B"].values
    out = []
    for k_back in range(0, 6):
        j = max(0, idx0 - k_back)
        out.append(float(B_series[j]))
    return out

SB0_list = get_initial_births(covars, t0)

fixed_params = {
    'delta': 1/60,  # Mortality rate (per year)
    'SB1_0': SB0_list[0],
    'SB2_0': SB0_list[1],
    'SB3_0': SB0_list[2],
    'SB4_0': SB0_list[3],
    'SB5_0': SB0_list[4],
    'SB6_0': SB0_list[5],
}

print("Fixed parameters:")
```

Process Model (rproc) I

Process Model (rproc) II

```python
def rproc(X_, theta_, key, covars, t, dt):
    """
    Process model for polio transmission.

    This follows the Martinez-Bakker et al. (2015) model structure.
    """
    # Extract parameters
    b1, b2, b3, b4, b5, b6 = theta_['b1'], theta_['b2'], theta_['b3'], \
                             theta_['b4'], theta_['b5'], theta_['b6']
    psi = theta_['psi']
    sigma_dem = theta_['sigma_dem']
    sigma_env = theta_['sigma_env']
    delta = theta_['delta']

    # Extract states
    SB1, SB2, SB3, SB4, SB5, SB6 = X_['SB1'], X_['SB2'], X_['SB3'], \
                                   X_['SB4'], X_['SB5'], X_['SB6']
    IB, SO, IO = X_['IB'], X_['SO'], X_['IO']

    # Extract covariates (from R-generated covar1.csv)
    B = covars['B']
    P = covars['P']
    xi1, xi2, xi3 = covars['xi1'], covars['xi2'], covars['xi3']
    xi4, xi5, xi6 = covars['xi4'], covars['xi5'], covars['xi6']
```

Measurement Model I

Measurement Model II

```
def dmeas(Y_, X_, theta_, covars, t):
    """
    Measurement density: discretized truncated normal with numerical stability.
    """
    rho = theta_['rho']
    tau = theta_['tau']
    IO = X_['IO']
    cases = Y_['cases']

    mu = rho * IO
    # Add base variance (1.0) to prevent distribution from being too narrow
    sd = jnp.sqrt((tau * IO)**2 + jnp.maximum(mu, 0.0) + 1.0)

    def _cdf(z):
        return 0.5 * (1.0 + jax.scipy.special.erf((z - mu) / (jnp.sqrt(2.0) * sd)))

    tol = 1e-18
    prob = jax.lax.cond(
        cases > 0.0,
        lambda _: _cdf(cases + 0.5) - _cdf(cases - 0.5) + tol,
        lambda _: _cdf(0.5) + tol,
        operand=None
    )
```

Initial State Distribution I

```python
def rinit(theta_, key, covars, t0):
    """
    Initial state distribution.
    """
    P = covars['P']

    return {
        'SB1': theta_['SB1_0'],
        'SB2': theta_['SB2_0'],
        'SB3': theta_['SB3_0'],
        'SB4': theta_['SB4_0'],
        'SB5': theta_['SB5_0'],
        'SB6': theta_['SB6_0'],
        'IB': 0.0,
        'IO': theta_['IO_0'] * P,
        'SO': theta_['SO_0'] * P
    }
```

Constructing the Pomp Object I

Constructing the Pomp Object II

```
from pypomp import Pomp

# Filter data to the analysis period
data_filtered = data[(data['time'] > t0 + 0.01) & (data['time'] < 1953 + 1/12 + 0.0

# Create observations DataFrame
ys = data_filtered[['cases']].copy()
ys.index = data_filtered['time']

# Filter covariates to match observation times
ys = ys.loc[ys.index.intersection(covars.index)]

# Create pomp object
polio = Pomp(
    ys=ys,
    theta=params_guess,
    rinit=rinit,
    rproc=rproc,
    dmeas=dmeas,
    rmeas=rmeas,
    covars=covars,
    statenames=statenames,
    t0=t0,
    nstep=1,
```

Setting Run Levels to Control Computation Time I

```
run_level = 2

Np = [100, 1000, 5000][run_level - 1]
Nmif = [10, 100, 200][run_level - 1]
Nreps_eval = [2, 10, 20][run_level - 1]
Nreps_local = [10, 20, 40][run_level - 1]
Nreps_global = [10, 20, 100][run_level - 1]
Nsim = [50, 100, 500][run_level - 1]

print(f"Run level: {run_level}")
print(f"Np={Np}, Nmif={Nmif}, Nreps_eval={Nreps_eval}")
```

```
Run level: 2
Np=1000, Nmif=100, Nreps_eval=10
```

Evaluating Likelihood at Starting Parameters I

Evaluating Likelihood at Starting Parameters II

```
params_rpomp = {
    'b1': 3.0, 'b2': 0.0, 'b3': 1.5, 'b4': 6.0, 'b5': 5.0, 'b6': 3.0,
    'psi': 0.002, 'rho': 0.01, 'tau': 0.001,
    'sigma_dem': 0.04, 'sigma_env': 0.5,
    'SO_0': 0.12, 'IO_0': 0.001,
    'delta': 1/60,  # 0.01666667
    'SB1_0': 4069.0, 'SB2_0': 4565.0, 'SB3_0': 4410.0,
    'SB4_0': 4616.0, 'SB5_0': 4305.0, 'SB6_0': 4032.0
}

polio.theta = params_rpomp

from pypomp.util import logmeanexp, logmeanexp_se

key = jax.random.key(3899882)
polio.pfilter(J=Np, key=key, reps=20, thresh=0)

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

L1 = logmeanexp(logliks)
L1_se = logmeanexp_se(logliks)

print(f"pypomp Log-likelihood: {L1:.2f} (SE: {L1_se:.2f})")
```

Investigating Local Persistence I

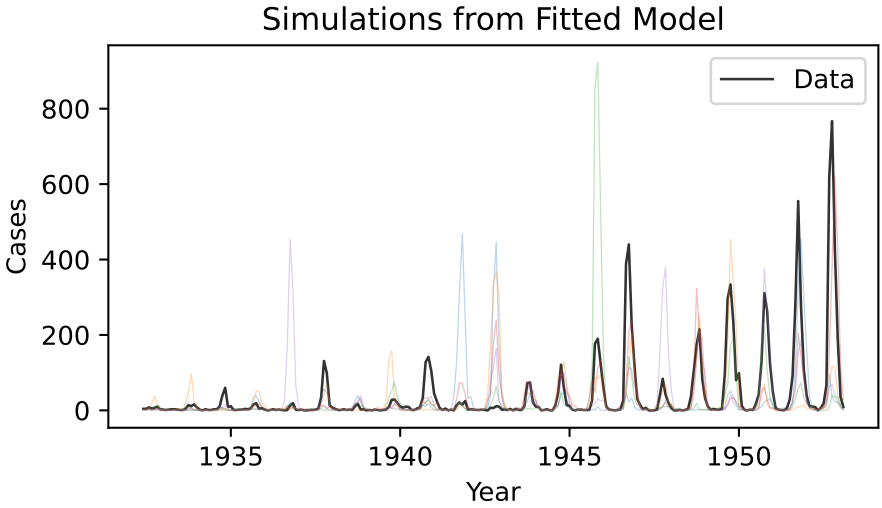
```
key = jax.random.key(1643079359)
X_sims, Y_sims = polio.simulate(key=key, nsim=Nsim)

print(f"Simulated {Nsim} trajectories")
```

```
Simulated 100 trajectories
```

Analyzing Persistence I

```
fig, ax = plt.subplots(figsize=(5, 3))
ax.plot(ys.index, ys['cases'], 'k-', linewidth=1, label='Data', alpha=0.8)

for i in range(min(5, Nsim)):
    sim_data = Y_sims[Y_sims['sim'] == i]
    ax.plot(sim_data['time'], sim_data['obs_0'], alpha=0.3, linewidth=0.5)

ax.set_xlabel('Year')
ax.set_ylabel('Cases')
ax.set_title('Simulations from Fitted Model')
ax.legend()
plt.tight_layout()
plt.show()
```

Analyzing Persistence II



IF2 Random Walk Standard Deviations I

```
from pypomp import RWSigma

rw_sd = RWSigma(
    sigmas={
        'b1': 0.02, 'b2': 0.02, 'b3': 0.02,
        'b4': 0.02, 'b5': 0.02, 'b6': 0.02,
        'psi': 0.02, 'rho': 0.02, 'tau': 0.02,
        'sigma_dem': 0.02, 'sigma_env': 0.02,
        'IO_0': 0.2, 'SO_0': 0.2,
        'delta': 0.0,
        'SB1_0': 0.0, 'SB2_0': 0.0, 'SB3_0': 0.0,
        'SB4_0': 0.0, 'SB5_0': 0.0, 'SB6_0': 0.0
    },
    init_names=['IO_0', 'SO_0']
)
```

Running IF2 (Local Search) I

```python
key = jax.random.key(942098028)
np.random.seed(12345)
n_starts = min(5, Nreps_local)
theta_list = [params_guess.copy() for _ in range(n_starts)]

polio.mif(J=Np, M=Nmif, key=key, rw_sd=rw_sd, a=0.5, theta=theta_list)

mif_result = polio.results_history[-1]
print(f"MIF completed")

# MIF result stores final parameters, not logLiks directly
# We need to run pfilter to evaluate the likelihood
print(f"Number of parameter sets after MIF: {len(polio.theta)}")
```

```
MIF completed
Number of parameter sets after MIF: 5
```

Evaluating the MIF Results I

```
# Run pfilter to evaluate log-likelihoods at the MIF-optimized parameters
polio.pfilter(J=Np, key=jax.random.key(12345), reps=Nreps_eval, thresh=0)

pf_result = polio.results_history[-1]
logliks = pf_result.logLiks.values

# Get mean loglik for each parameter set
mean_logliks = logliks.mean(axis=1)
best_idx = np.argmax(mean_logliks)

ll_mif = logmeanexp(logliks[best_idx, :])
ll_mif_se = logmeanexp_se(logliks[best_idx, :])

print(f"Log-likelihood after local search: {ll_mif:.2f} (SE: {ll_mif_se:.2f})")
print(f"Best parameter set index: {best_idx}")
```

```
Log-likelihood after local search: -1024.36 (SE: 0.50)
Best parameter set index: 4
```

Setting Up Global Search I

```
box = {
    'b1': (-2, 8), 'b2': (-2, 8), 'b3': (-2, 8),
    'b4': (-2, 8), 'b5': (-2, 8), 'b6': (-2, 8),
    'psi': (0, 0.1), 'rho': (0, 0.1), 'tau': (0, 0.1),
    'sigma_dem': (0, 0.5), 'sigma_env': (0, 1),
    'S0_0': (0, 1), 'I0_0': (0, 0.01)
}
```

Running Global Search I

```python
def random_params_from_box(box, n, fixed_params):
    params_list = []
    for _ in range(n):
        params = {}
        for name, (lo, hi) in box.items():
            params[name] = np.random.uniform(lo, hi)
        params.update(fixed_params)
        params_list.append(params)
    return params_list

np.random.seed(833102018)
starts = random_params_from_box(box, min(5, Nreps_global), fixed_params)
print(f"Generated {len(starts)} random starting points")
```

```
Generated 5 random starting points
```

Running Global Search II

```python
key = jax.random.key(71449038)
polio.theta = starts
polio.mif(J=Np, M=Nmif, key=key, rw_sd=rw_sd, a=0.5)

# Evaluate likelihoods at the optimized parameters
polio.pfilter(J=Np, key=jax.random.key(99999), reps=Nreps_eval, thresh=0)

pf_result = polio.results_history[-1]
logliks = pf_result.logLiks.values

results = []
for i, theta_i in enumerate(polio.theta):
    ll = logliks[i, :].mean()
    results.append({'logLik': ll, **theta_i})
    print(f"  Search {i+1}: logLik = {ll:.2f}")

results_df = pd.DataFrame(results)
print(f"\nBest log-likelihood: {results_df['logLik'].max():.2f}")
```

Running Global Search III

```
  Search 1: logLik = -1262.75
  Search 2: logLik = -1251.39
  Search 3: logLik = -951.16
  Search 4: logLik = -1193.55
  Search 5: logLik = -1162.76

Best log-likelihood: -951.16
```

Analyzing Global Search Results I

Analyzing Global Search Results II

```
fig, axes = plt.subplots(2, 2, figsize=(5, 4))

good_results = results_df[results_df['logLik'] > results_df['logLik'].max() - 20]

axes[0,0].scatter(good_results['rho'], good_results['logLik'], s=10)
axes[0,0].set_xlabel('rho')
axes[0,0].set_ylabel('logLik')

axes[0,1].scatter(good_results['psi'], good_results['logLik'], s=10)
axes[0,1].set_xlabel('psi')
axes[0,1].set_ylabel('logLik')

axes[1,0].scatter(good_results['sigma_dem'], good_results['logLik'], s=10)
axes[1,0].set_xlabel('sigma_dem')
axes[1,0].set_ylabel('logLik')

axes[1,1].scatter(good_results['sigma_env'], good_results['logLik'], s=10)
axes[1,1].set_xlabel('sigma_env')
axes[1,1].set_ylabel('logLik')

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

IID Negative Binomial Benchmark I

```python
from scipy.optimize import minimize
from scipy.stats import nbinom

def nb_negloglik(params, cases):
    size = np.exp(params[0])
    prob = 1 / (1 + np.exp(-params[1]))
    return -np.sum(nbinom.logpmf(cases.astype(int), size, prob))

cases = ys['cases'].values
result = minimize(nb_negloglik, [0, -5], args=(cases,))
nb_loglik = -result.fun

print(f"Negative binomial (IID) log-likelihood: {nb_loglik:.1f}")
```

```
Negative binomial (IID) log-likelihood: -1036.2
```

ARMA Benchmark I

```
from statsmodels.tsa.arima.model import ARIMA

log_y = np.log(cases + 1)
arma_model = ARIMA(log_y, order=(2, 0, 2), seasonal_order=(1, 0, 1, 12))
arma_fit = arma_model.fit()

arma_loglik = arma_fit.llf - np.sum(log_y)

print(f"ARMA(2,0,2)x(1,0,1)_12 log-likelihood: {arma_loglik:.1f}")
```

```
ARMA(2,0,2)x(1,0,1)_12 log-likelihood: -824.3
```

Profile Over Reporting Rate ρ I

```
profile_pts = [3, 5, 30][run_level - 1]
rho_values = np.linspace(0.01, 0.025, profile_pts)
print(f"Profiling over {profile_pts} values of rho")

rw_sd_profile = RWSigma(
    sigmas={
        'b1': 0.02, 'b2': 0.02, 'b3': 0.02,
        'b4': 0.02, 'b5': 0.02, 'b6': 0.02,
        'psi': 0.02, 'rho': 0.0,  # Fixed!
        'tau': 0.02, 'sigma_dem': 0.02, 'sigma_env': 0.02,
        'I0_0': 0.2, 'S0_0': 0.2,
        'delta': 0.0,
        'SB1_0': 0.0, 'SB2_0': 0.0, 'SB3_0': 0.0,
        'SB4_0': 0.0, 'SB5_0': 0.0, 'SB6_0': 0.0
    },
    init_names=['I0_0', 'S0_0']
)

for rho_val in rho_values[:3]:
    print(f"  Would profile at rho = {rho_val:.4f}")
```

Profile Over Reporting Rate ρ II

```
Profiling over 5 values of rho
  Would profile at rho = 0.0100
  Would profile at rho = 0.0138
  Would profile at rho = 0.0175
```

What We Learned I

1. **Covariates** can be included naturally in POMP models to incorporate external forcing like birth rates, population, and seasonality.
2. **Seasonal transmission** is modeled using periodic B-spline basis functions, allowing flexible seasonality patterns.
3. **Global search** from diverse starting points is essential to have confidence in MLEs for complex models.
4. **Profile likelihood** provides rigorous uncertainty quantification for individual parameters.
5. **Benchmark comparisons** (IID, ARMA) help interpret whether mechanistic models are performing adequately.

pypomp API Key Points I

1. RWSigma requires **ALL parameters** in sigmas dict - use 0.0 for fixed params
2. rmeas must return a JAX array with shape (ydim,), e.g., jnp.array([value])
3. Use nstep=1 for discrete-time models
4. Multiple starting points: pass theta=[list_of_dicts] to mif()
5. After mif(), run pfilter() to evaluate log-likelihoods
6. Access pfilter results via pf_result.logLiks.values

Exercise 1: Initial Values I

When carrying out parameter estimation for dynamic systems, we need to specify beginning values for both the dynamic system (in the state space) and the parameters (in the parameter space).

Tasks:

1. Discuss issues in specifying and inferring initial conditions, with particular reference to this polio example.
2. Suggest a possible improvement in the treatment of initial conditions here, code it up and make some preliminary assessment of its effectiveness.
3. How will you decide if it is a substantial improvement?

Exercise 2: Parameter Estimation Using Randomized Starting Values I

Think about possible improvements on the assignment of randomized starting values for the parameter estimation searches.

Tasks:

1. Propose and try out a modification of the procedure.
2. Does it make a difference?

Acknowledgments I

- This lesson is prepared for the Simulation-based Inference for Epidemiological Dynamics module at the Summer Institute in Statistics and Modeling in Infectious Diseases (SISMID).
- The materials build on previous versions of this course and related courses.

References

Martinez-Bakker, M., King, A. A., and Rohani, P. (2015). Unraveling the transmission ecology of polio. *PLoS Biol*, 13(6):e1002172.