Lesson 6: Exercises - Polio Case Study

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 contains worked solutions to the exercises from Lesson 6 on the polio case study, implemented using **pypomp**.

Import Packages

```
import jax
import jax.numpy as jnp
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import minimize
from scipy.stats import nbinom, chi2, qmc
from copy import deepcopy

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

np.random.seed(594709947)
```

Load Data and Construct Model I

```
# Load the data
data = pd.read_csv("polio_wisconsin.csv", comment='#')

# Constants
K = 6
t0 = 1932 + 4/12

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

print(f"Data loaded: {len(data)} rows")
print(f"Time range: {data['time'].min():.2f} to {data['time'].max():.2f}")
```

```
Data loaded: 288 rows
Time range: 1931.08 to 1955.00
```

Load Data and Construct Model II

```
# Load covariates from R-generated file (covar1.csv)
# This ensures consistency with the R/pomp analysis
cov_raw = pd.read_csv("covar1.csv")

# Transpose: variable names in first column, times as column headers
first_col = cov_raw.columns[0]
covars = (cov_raw.rename(columns={first_col: "var"})
                 .set_index("var")
                 .T)

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}")
```

```
Covariate columns: ['B', 'xi1', 'xi2', 'xi3', 'xi4', 'xi5', 'x
Time range: 1931.0833 to 1955.0000
```

Load Data and Construct Model III

```
# Fixed parameters from covariate table
def get_initial_births(covars_df, t0_val):
    idx0 = np.argmin(np.abs(covars_df.index.values - t0_val))
    B_series = covars_df["B"].values
    return [float(B_series[max(0, idx0 - k)]) for k in range(6)]

SB0_list = get_initial_births(covars, t0)

fixed_params = {
    'delta': 1/60,
    '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],
}

# Starting parameters
params_guess = {
    '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,
    'S0_0': 0.12, 'I0_0': 0.001,
    **fixed_params
}
```

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?

Discussion of Initial Conditions I

```
print("Current initial condition approach:")
print("")
print("1. SB1_0 through SB6_0: Fixed from birth data (covar1.csv)")
print("   - Assumed known from vital statistics")
print("   - Reasonable since births are well-recorded")
print("")
print("2. SO_0, IO_0: Estimated as fractions of population P")
print("   - SO_0   0.12 means 12% susceptible")
print("   - IO_0   0.001 means 0.1% infected")
print("")
print("Issues:")
print("- SO_0 and IO_0 may be poorly identified")
print("- Initial conditions far from t0 may not matter much")
print("- Correlation with other parameters (e.g., rho)")
```

Discussion of Initial Conditions II

Current initial condition approach:

1. SB1_0 through SB6_0: Fixed from birth data (covar1.csv)
   - Assumed known from vital statistics
   - Reasonable since births are well-recorded

2. SO_0, IO_0: Estimated as fractions of population P
   - SO_0 0.12 means 12% susceptible
   - IO_0 0.001 means 0.1% infected

Issues:
- SO_0 and IO_0 may be poorly identified
- Initial conditions far from t0 may not matter much
- Correlation with other parameters (e.g., rho)

Improvement: Estimate More Initial States I

```python
# Alternative: Also estimate initial baby susceptibles as fractions
def rinit_v2(theta_, key, covars, t0):
    """
    Modified rinit: SB values scaled by a survival fraction.
    """
    P = covars['P']
    s_factor = theta_.get('s_init', 1.0)

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

print("Alternative: Add 's_init' parameter to scale baby susceptibles")
```

Improvement: Estimate More Initial States II

```
Alternative: Add 's_init' parameter to scale baby susceptibles
```

Comparing Initial Condition Treatments I

```
print("How to assess if modification is substantial:")
print("")
print("1. Same number of parameters → Direct likelihood comparison")
print("   - If modification gives same number of params,")
print("     compare maximized log-likelihoods directly")
print("")
print("2. Different number of parameters → Use AIC")
print("   - AIC = -2 * logLik + 2 * k")
print("   - Lower AIC is better")
print("   - Penalizes additional parameters")
print("")
print("3. Profile likelihood")
print("   - Check if new params are identifiable")
print("   - Examine confidence intervals")
```

Comparing Initial Condition Treatments II

How to assess if modification is substantial:

1. Same number of parameters → Direct likelihood comparison
   - If modification gives same number of params, compare maximized log-likelihoods directly

2. Different number of parameters → Use AIC
   - AIC = -2 * logLik + 2 * k
   - Lower AIC is better
   - Penalizes additional parameters

3. Profile likelihood
   - Check if new params are identifiable
   - Examine confidence intervals

Exercise 2: 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?

Latin Hypercube Sampling I

Latin Hypercube Sampling II

```python
def lhs_params_from_box(box, n, fixed_params):
    """
    Generate n parameter sets using Latin Hypercube Sampling.
    Better space coverage than simple random sampling.
    """
    param_names = list(box.keys())
    d = len(param_names)

    sampler = qmc.LatinHypercube(d=d, seed=42)
    sample = sampler.random(n=n)

    params_list = []
    for i in range(n):
        params = {}
        for j, name in enumerate(param_names):
            lo, hi = box[name]
            params[name] = lo + sample[i, j] * (hi - lo)
        params.update(fixed_params)
        params_list.append(params)

    return params_list

box = {
    'b1': (-2, 8), 'b2': (-2, 8), 'b3': (-2, 8),
```

Comparing Sampling Methods I

Comparing Sampling Methods II

```
def uniform_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(42)
uniform_starts = uniform_params_from_box(box, 20, fixed_params)
lhs_starts = lhs_params_from_box(box, 20, fixed_params)

fig, axes = plt.subplots(1, 2, figsize=(5, 2.5))

uni_rho = [p['rho'] for p in uniform_starts]
uni_psi = [p['psi'] for p in uniform_starts]
axes[0].scatter(uni_rho, uni_psi, s=20)
axes[0].set_xlabel('rho')
axes[0].set_ylabel('psi')
axes[0].set_title('Uniform Random')

lhs_rho = [p['rho'] for p in lhs_starts]
```

Does It Make a Difference? I

```
print("Comparison of sampling strategies:")
print("")
print("1. COVERAGE:")
print("   - LHS guarantees even coverage in each dimension")
print("   - Random sampling can have gaps and clusters")
print("")
print("2. EFFICIENCY:")
print("   - LHS often finds good regions with fewer samples")
print("   - Especially valuable in high-dimensional spaces")
print("")
print("3. REPRODUCIBILITY:")
print("   - LHS with fixed seed gives consistent coverage")
print("   - Random sampling varies between runs")
```

Does It Make a Difference? II

Comparison of sampling strategies:

1. COVERAGE:
   - LHS guarantees even coverage in each dimension
   - Random sampling can have gaps and clusters

2. EFFICIENCY:
   - LHS often finds good regions with fewer samples
   - Especially valuable in high-dimensional spaces

3. REPRODUCIBILITY:
   - LHS with fixed seed gives consistent coverage
   - Random sampling varies between runs

Exercise 3: Demography I

The model assumes that individuals remain in each monthly age class for one month and then move to the next.

Tasks:

1. In what way is this demographic model idealized?
2. Propose an improvement.
3. How would you decide if the improvement is valuable for understanding polio transmission?

Model Idealizations I

```
print("Idealizations in the demographic model:")
print("")
print("1. DISCRETE TIME:")
print("   - Assumes exact 1-month duration in each class")
print("   - Reality: continuous aging, exponential waiting times")
print("")
print("2. PERFECT SYNCHRONY:")
print("   - All births at start of month")
print("   - All transitions at month boundaries")
print("")
print("3. UNIFORM MORTALITY:")
print("   - Same death rate delta for all ages")
print("   - Reality: infant mortality higher than adults")
```

Model Idealizations II

Idealizations in the demographic model:

1. DISCRETE TIME:
   - Assumes exact 1-month duration in each class
   - Reality: continuous aging, exponential waiting times

2. PERFECT SYNCHRONY:
   - All births at start of month
   - All transitions at month boundaries

3. UNIFORM MORTALITY:
   - Same death rate delta for all ages
   - Reality: infant mortality higher than adults

Proposed Improvement: Gamma-distributed Duration I

```
print("Improvement: Erlang-distributed durations")
print("")
print("Instead of fixed 1-month duration, use:")
print("- Multiple sub-stages per age class")
print("- Each sub-stage has exponential waiting time")
print("- Total duration follows Erlang (Gamma) distribution")
print("")
print("Trade-off:")
print("- More realistic waiting times")
print("- But: more state variables, slower computation")
```

```
Improvement: Erlang-distributed durations

Instead of fixed 1-month duration, use:
- Multiple sub-stages per age class
- Each sub-stage has exponential waiting time
- Total duration follows Erlang (Gamma) distribution
```

Proposed Improvement: Gamma-distributed Duration II

```
Trade-off:
- More realistic waiting times
- But: more state variables, slower computation
```

Exercise 4: Diagnosing Filtering Problems I

Discuss the potential problems that may have arisen in the fitting of this model.

Tasks:

1. Check conditional log-likelihoods for outliers
2. Check effective sample sizes
3. Check parameter convergence using traces
4. Assess whether cooling fraction was appropriate

Setup for Diagnostics I

```
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']
)

key = jax.random.key(12345)
polio.mif(J=Np, M=Nmif, key=key, rw_sd=rw_sd, a=0.5)
mif_result = polio.results_history[-1]

polio.pfilter(J=Np, key=jax.random.key(99), reps=1, thresh=0, CLL=True, ESS=True)
pf_result = polio.results_history[-1]
print("Diagnostics computed")
```

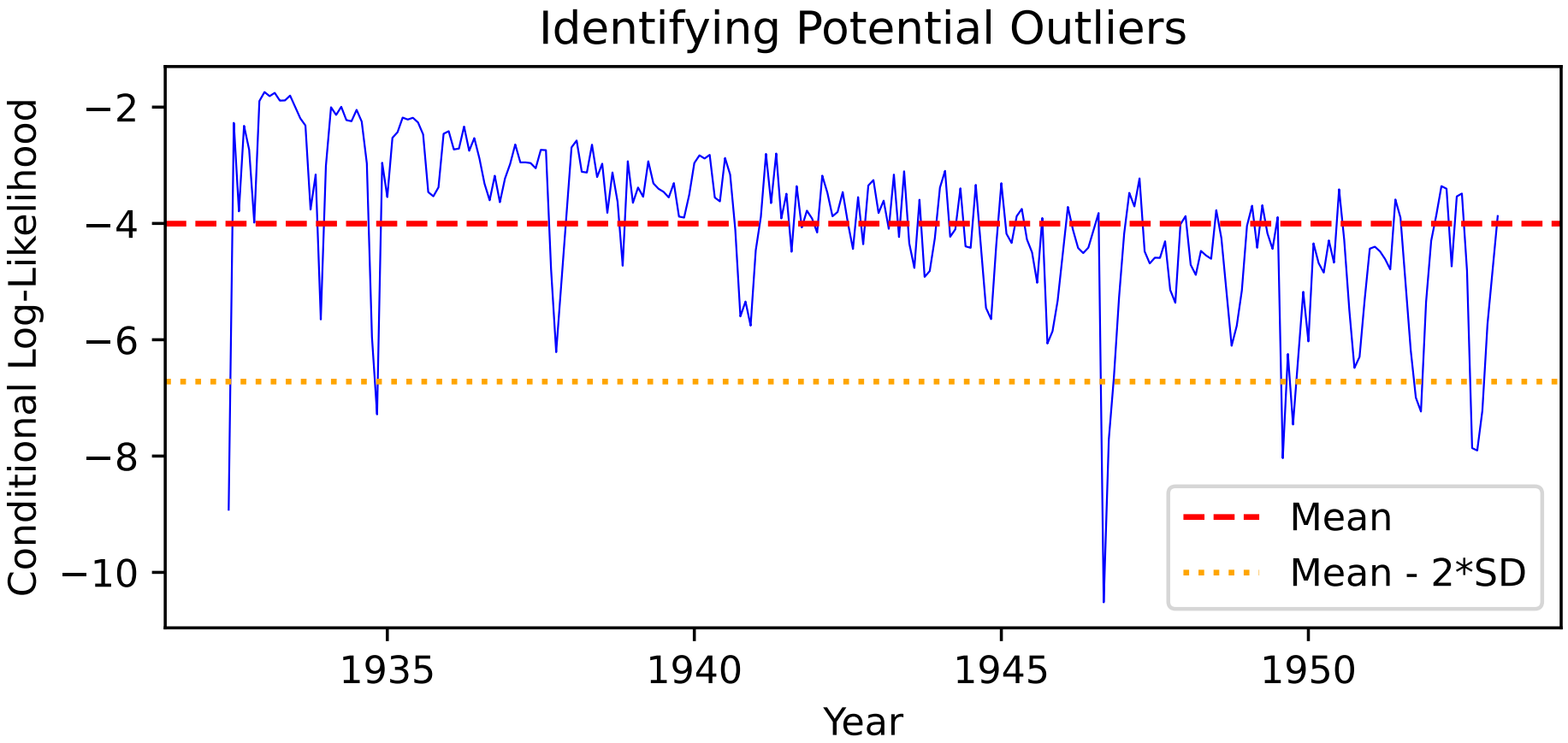
Setup for Diagnostics II

```
Diagnostics computed
```

Conditional Log-Likelihoods I

```
if hasattr(pf_result, 'CLL') and pf_result.CLL is not None:
    fig, ax = plt.subplots(figsize=(6, 3))
    times = polio.ys.index.values
    cll = pf_result.CLL.values[0, 0, :]

    ax.plot(times, cll, 'b-', linewidth=0.5)
    ax.axhline(y=np.mean(cll), color='r', linestyle='--', label='Mean')
    ax.axhline(y=np.mean(cll) - 2*np.std(cll), color='orange',
               linestyle=':', label='Mean - 2*SD')
    ax.set_xlabel('Year')
    ax.set_ylabel('Conditional Log-Likelihood')
    ax.set_title('Identifying Potential Outliers')
    ax.legend()
    plt.tight_layout()
    plt.show()

    threshold = np.mean(cll) - 3*np.std(cll)
    outliers = times[cll < threshold]
    print(f"Potential outlier times: {outliers}")
else:
    print("CLL not computed - run pfilter with CLL=True")
```

Conditional Log-Likelihoods II

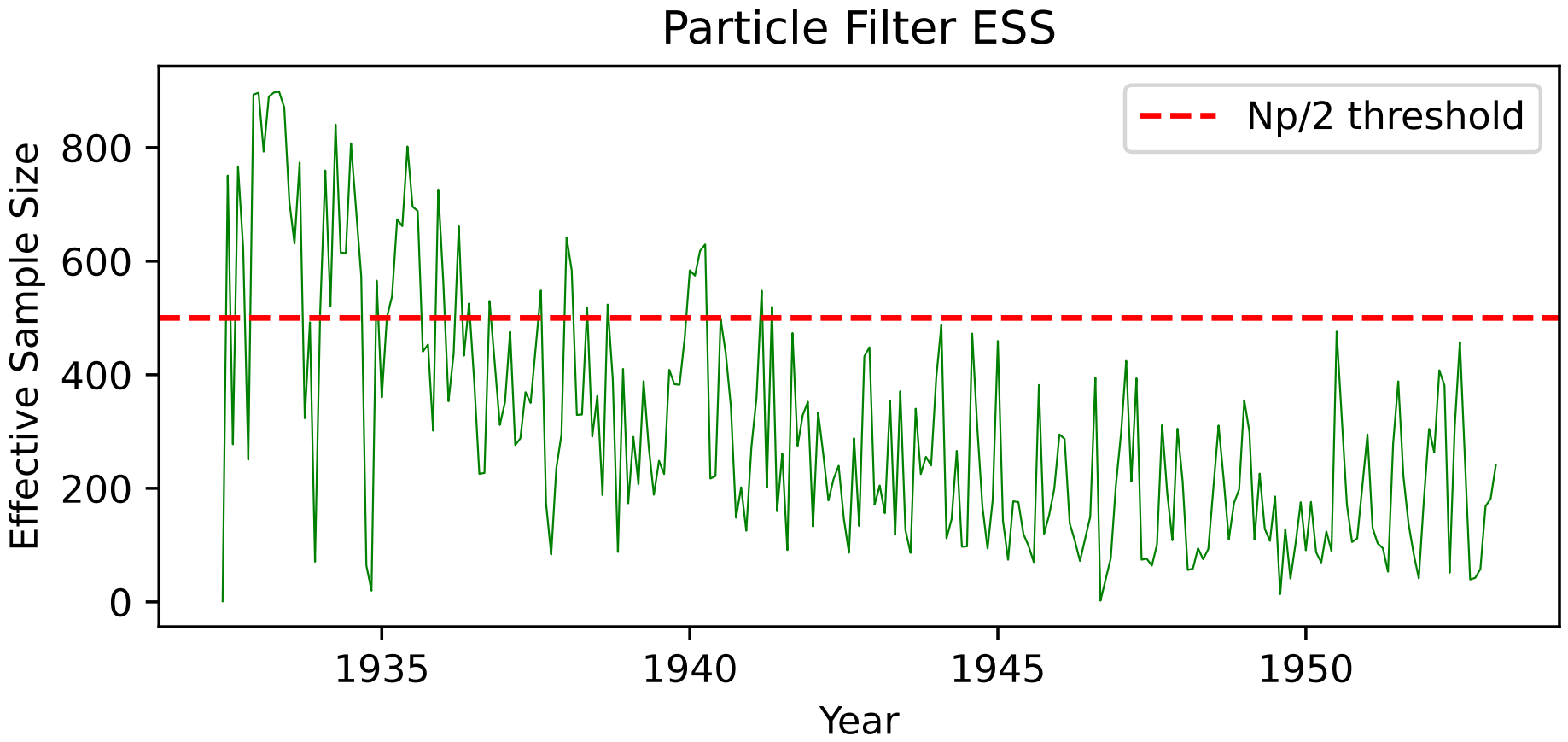



```
Potential outlier times: [1932.41666667 1946.66666667]
```

Effective Sample Size I

```
if hasattr(pf_result, 'ESS') and pf_result.ESS is not None:
    fig, ax = plt.subplots(figsize=(6, 3))
    times = polio.ys.index.values
    ess = pf_result.ESS.values[0, 0, :]
    ax.plot(times, ess, 'g-', linewidth=0.5)
    ax.axhline(y=Np/2, color='r', linestyle='--', label='Np/2 threshold')
    ax.set_xlabel('Year')
    ax.set_ylabel('Effective Sample Size')
    ax.set_title('Particle Filter ESS')
    ax.legend()
    plt.tight_layout()
    plt.show()

    low_ess = np.sum(ess < Np/10)
    print(f"Times with ESS < Np/10: {low_ess} out of {len(ess)}")
else:
    print("ESS not computed - run pfilter with ESS=True")
```

Effective Sample Size II

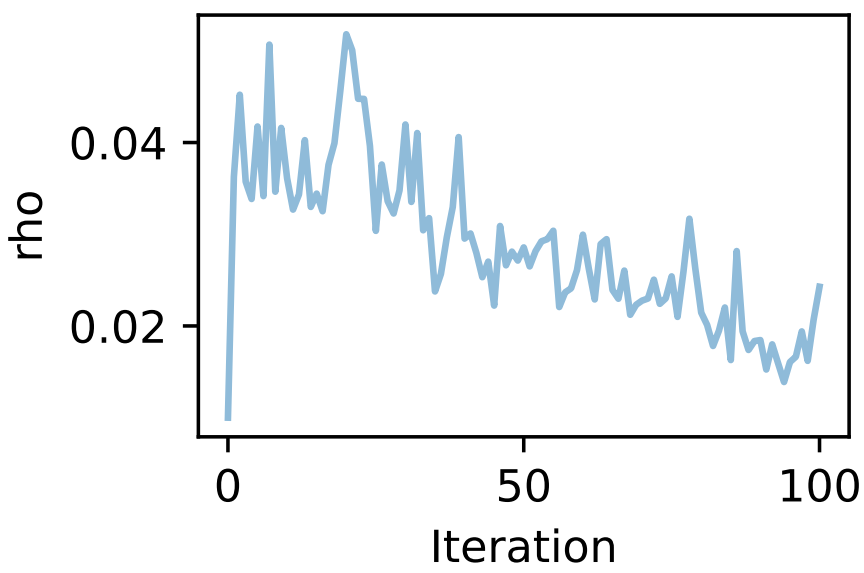


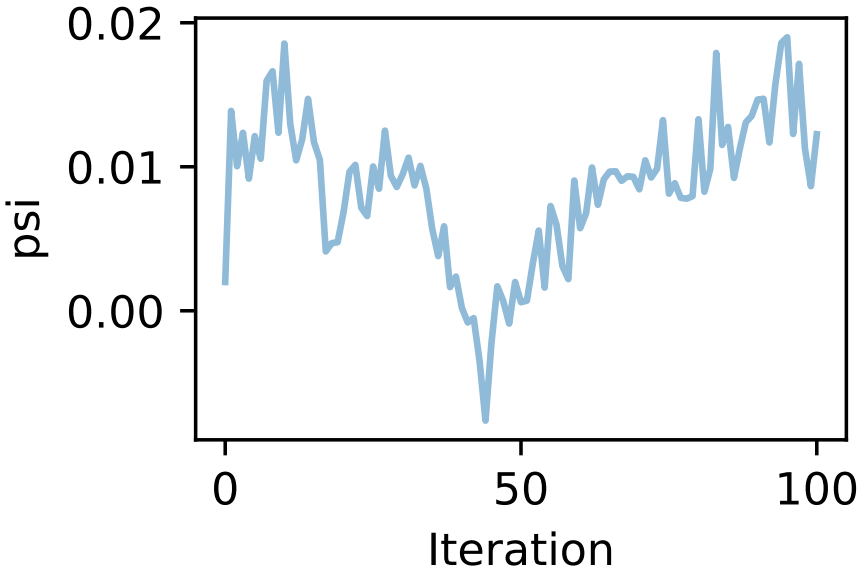

```
Times with ESS < Np/10: 41 out of 249
```

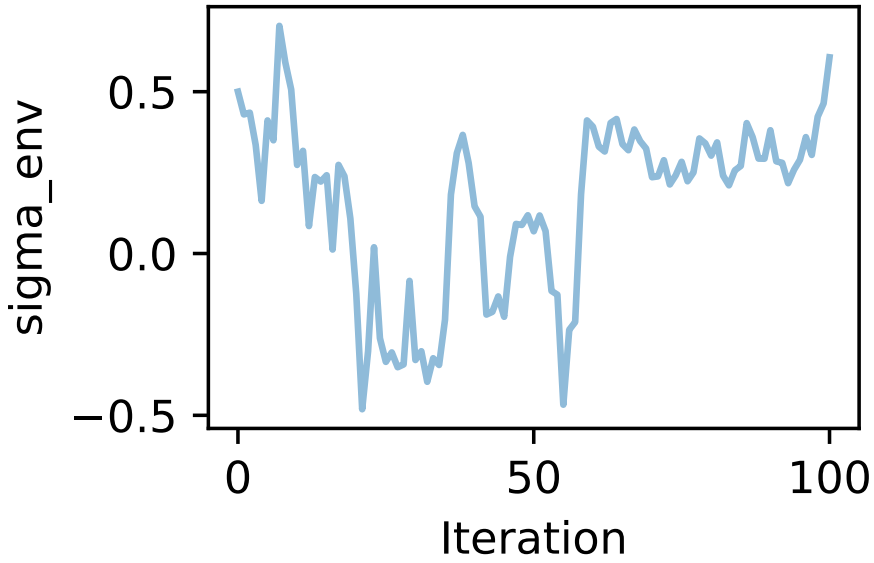
Parameter Convergence I

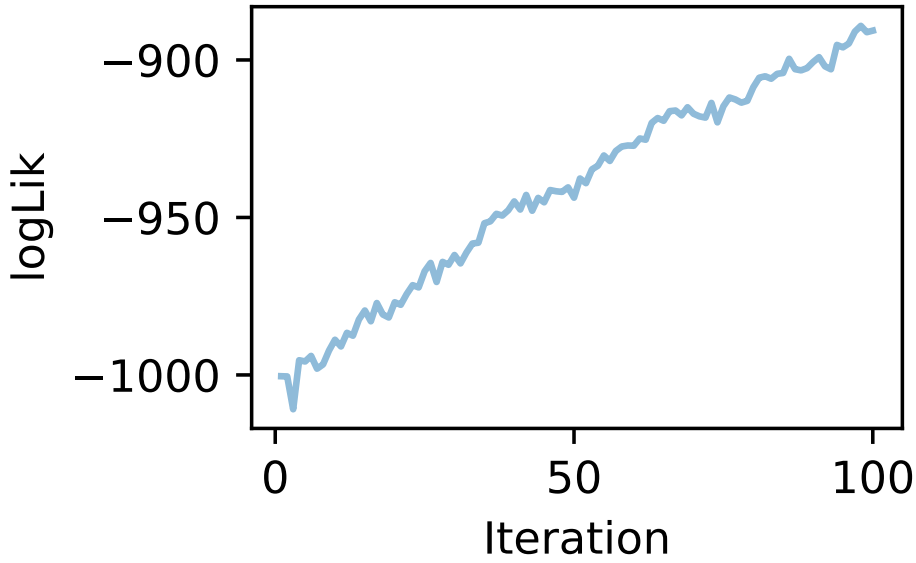
```
if hasattr(mif_result, 'traces_da') and mif_result.traces_da is not None:
    traces_da = mif_result.traces_da

    params_to_plot = ['rho', 'psi', 'sigma_env', 'logLik']
    variables = traces_da.coords['variable'].values

    fig, axes = plt.subplots(2, 2, figsize=(6, 4))
    for ax, param in zip(axes.flatten(), params_to_plot):
        if param in variables:
            for rep in range(traces_da.sizes['replicate']):
                trace = traces_da.isel(replicate=rep).sel(variable=param).values
                ax.plot(trace, alpha=0.5)
            ax.set_xlabel('Iteration')
            ax.set_ylabel(param)
    plt.tight_layout()
    plt.show()
else:
    print("Traces not available")
```

Parameter Convergence II









Cooling Fraction Assessment I

```
print("Cooling fraction assessment:")
print("")
print("The cooling parameter 'a' controls how quickly perturbations decrease.")
print("")
print("Signs of a being too small:")
print("  - Parameter traces flatten early (freeze)")
print("  - Log-likelihood plateaus prematurely")
print("")
print("Signs of a being too large:")
print("  - Parameters still varying at final iteration")
print("  - Need very many iterations to converge")
print("")
print(f"Current setting: a = 0.5")
```

Cooling Fraction Assessment II

```
Cooling fraction assessment:

The cooling parameter 'a' controls how quickly perturbations c

Signs of a being too small:
  - Parameter traces flatten early (freeze)
  - Log-likelihood plateaus prematurely

Signs of a being too large:
  - Parameters still varying at final iteration
  - Need very many iterations to converge

Current setting: a = 0.5
```

Exercise 5: Choosing Algorithmic Parameters I

Suppose you have a 10-minute search with `Np`, `Nmif`, and `Reps`. How would you adjust for a 2-hour search?

Scaling Strategy I

Scaling Strategy II

```
current = {'Np': 1000, 'Nmif': 100, 'Nstarts': 10}
time_factor = 120 / 10  # 12x

print(f"Time scaling factor: {time_factor}x")
print("")

strategies = {
    'Balanced': {
        'Np': int(current['Np'] * time_factor**(1/3)),
        'Nmif': int(current['Nmif'] * time_factor**(1/3)),
        'Nstarts': int(current['Nstarts'] * time_factor**(1/3))
    },
    'More particles': {
        'Np': int(current['Np'] * time_factor**(1/2)),
        'Nmif': current['Nmif'],
        'Nstarts': int(current['Nstarts'] * time_factor**(1/2))
    },
    'More searches': {
        'Np': current['Np'],
        'Nmif': current['Nmif'],
        'Nstarts': int(current['Nstarts'] * time_factor)
    }
}
```

Two-Stage Search Implementation I

```
print("Two-stage search (recommended for 2-hour budget):")
print("")
print("Stage 1 (exploration): 1 hour")
stage1 = {'Np': 1000, 'Nmif': 50, 'Nstarts': 50}
print(f"  Np={stage1['Np']}, Nmif={stage1['Nmif']}, Nstarts={stage1['Nstarts']}")
print(f"  Goal: Find promising regions")
print("")

print("Stage 2 (refinement): 1 hour")
stage2 = {'Np': 2000, 'Nmif': 100, 'Nstarts': 10}
print(f"  Np={stage2['Np']}, Nmif={stage2['Nmif']}, Nstarts={stage2['Nstarts']}")
print(f"  Goal: Refine top results from Stage 1")
```

```
Two-stage search (recommended for 2-hour budget):

Stage 1 (exploration): 1 hour
  Np=1000, Nmif=50, Nstarts=50
  Goal: Find promising regions
```

Two-Stage Search Implementation II

```
Stage 2 (refinement): 1 hour
  Np=2000, Nmif=100, Nstarts=10
  Goal: Refine top results from Stage 1
```

Summary I

Exercise 1 (Initial Values):

- Initial conditions can affect likelihood but may be weakly identified
- Modifications should be compared via maximized log-likelihood
- No new parameters $\rightarrow$ direct likelihood comparison

Exercise 2 (Starting Values):

- LHS provides better coverage than uniform random
- Informed priors based on biological knowledge can help
- Multiple strategies should be compared empirically

Summary (continued) I

Exercise 3 (Demography):

- Model makes simplifying assumptions about age structure
- Continuous time with gamma-distributed durations could be more realistic
- Key: check if conclusions are robust to modifications

Exercise 4 (Diagnostics):

- Monitor ESS and conditional log-likelihoods for problems
- Check parameter traces for convergence using `traces_da`
- Adjust cooling fraction if traces freeze too early/late

Exercise 5 (Algorithmic Parameters):

- More starting points generally more valuable than longer runs
- Two-stage search: explore first, refine later

Summary (continued) II

- Monitor diagnostics to guide parameter choices

Key pypomp API 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. Use `polio.prune(n=k)` to keep top k parameter sets
6. Access traces via `mif_result.traces_da` (xarray DataArray with dimensions `('replicate', 'iteration', 'variable')`)

Acknowledgments I

- This lesson is prepared for the Simulation-based Inference for Epidemiological Dynamics module at SISMID.
- The materials build on previous versions of this course and related courses.

References