Lesson 2: Exercises on Simulation of Stochastic Dynamic Models

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

This document contains worked solutions to the exercises from Lesson 2 on simulation of stochastic dynamic models, implemented using `pypomp`.

Setup

Import Required Packages

```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 io
import requests
```

Load the Consett Measles Data

```python
# Download and prepare data
meas = (pd.read_csv(
          "https://kingaa.github.io/sbied/stochsim/Measles_Consett_1948.csv")
          .loc[:, ["week", "cases"]]
          .rename(columns={"week": "time", "cases": "reports"})
          .set_index("time")
          .astype(float))

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

Define Helper Functions

```python
def nbinom_logpmf(x, k, mu):
    """Log PMF of NegBin(k, mu) that is robust when mu == 0."""
    x = jnp.asarray(x)
    k = jnp.asarray(k)
    mu = jnp.asarray(mu)
    # Handle mu == 0 separately
    logp_zero = jnp.where(x == 0, 0.0, -jnp.inf)
```

```
    safe_mu = jnp.where(mu == 0.0, 1.0, mu)   # Dummy value, ignored
    core = (jax.scipy.special.gammaln(k + x)
            - jax.scipy.special.gammaln(k)
            - jax.scipy.special.gammaln(x + 1)
            + k * jnp.log(k / (k + safe_mu))
            + x * jnp.log(safe_mu / (k + safe_mu)))
    return jnp.where(mu == 0.0, logp_zero, core)

def rnbinom(key, k, mu):
    """Sample from NegBin(k, mu) via Gamma-Poisson mixture."""
    key_g, key_p = jax.random.split(key)
    lam = jax.random.gamma(key_g, k) * (mu / k)
    return jax.random.poisson(key_p, lam)
```

Exercise 2.1: Little o Notation

Problem Statement

What is the link between little o notation and the derivative?
Explain why

$$f(x + \delta) = f(x) + \delta\, g(x) + o(\delta)$$

is the same statement as

$$\frac{df}{dx} = g(x).$$

What considerations might help you choose which of these notations to use?

Solution

We combine the equation
$$f(x + \delta) = f(x) + \delta\, g(x) + o(\delta),$$

with the definition that $h(\delta) = o(\delta)$ means

$$\lim_{\delta \to 0} \frac{h(\delta)}{\delta} = 0.$$

Rearranging, we see that
$$\lim_{\delta \to 0} \frac{f(x + \delta) - f(x)}{\delta} = g(x)$$

which, by definition, implies that $df(x)/dx = g(x)$.

Key Insight: Little-o notation is compact, and so is useful for simplifying complex expressions.

When to use which notation:

- Use derivative notation ($\frac{df}{dx} = g(x)$) when working with deterministic, continuous dynamics

- Use little-o notation when working with stochastic processes, particularly Markov chains, where discrete transitions happen over small time intervals δ

Exercise 2.2: Euler versus Gillespie

Problem Statement

A widely used exact simulation method for continuous-time Markov chains is Gillespie's algorithm. We do not emphasize it here. **Why?**

When would you prefer an implementation of Gillespie's algorithm to an Euler solution?

Solution

Why we don't emphasize Gillespie's algorithm:

For reasons explained in the lesson, scientific conclusions may not hinge on the extent to which numerical approximations agree with exact solutions of the equations defining a continuous-time model.

For constant-rate compartmental models, the Gillespie algorithm gives an **exact** solution at the expense of additional computation time. We may on occasion want an exact simulator, and in that case Gillespie can be used.

When to prefer Gillespie's algorithm:

1. When population sizes are **small** (Gillespie is exact but slow for large populations)
2. When event rates are **low** (fewer events to simulate)
3. When **exact** stochastic realizations are scientifically important
4. When you need to avoid **discretization artifacts** from Euler methods

When to prefer Euler methods:

1. **Large populations**: Gillespie becomes impractically slow
2. **Environmental stochasticity**: Euler handles this naturally, Gillespie does not
3. **Observation times**: Euler naturally aligns with fixed observation intervals
4. **Computational efficiency**: Euler is generally faster for practical applications
5. **Tau-leaping**: A variant that bridges both approaches

Numerically, Gillespie's algorithm is often approximated using so-called tau-leaping methods, which are closely related to Euler's approach.

Exercise 2.3: Explore the SIR Model

Problem Statement

Fiddle with the parameters to see if you can find a model for which the data are a more plausible realization.

Baseline SIR Model Setup

First, let's set up the baseline SIR model from the lesson:

```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  # Accumulator for true incidence
    return {"S": S0, "I": I0, "R": R0, "H": H0}

def rproc(X_, theta_, key, covars, t, dt):
    """Process simulator for SIR model with Euler-binomial scheme."""
    S, I, R, H = X_["S"], X_["I"], X_["R"], 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)

    S_new = S - dN_SI
    I_new = I + dN_SI - dN_IR
    R_new = R + dN_IR
    H_new = H + dN_IR

    return {"S": S_new, "I": I_new, "R": R_new, "H": H_new}

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: sample reports ~ NegBin(k, rho*H)."""
    rho = theta_["rho"]
    k = theta_["k"]
    H = X_["H"]
    mu = rho * H
    reports = rnbinom(key, k, mu)
    return jnp.array([reports])
```

Helper Function for Simulation and Plotting

```python
def run_simulation_and_plot(theta, title="SIR simulation", n_sims=20):
    """Create SIR model, run simulations, and plot results."""

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

    sir_obj = 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
    )

    key = jax.random.key(42)
    X_sims, Y_sims = sir_obj.simulate(key=key, nsim=n_sims)

    sim_df = Y_sims.pivot_table(
        index="time",
        columns="sim",
        values="obs_0"
    )
    sim_df.columns = [f"sim_{i+1}" for i in range(n_sims)]

    fig, ax = plt.subplots(figsize=(4, 3))
    ys["reports"].plot(ax=ax, lw=2, color="black", label="observed")
    sim_df.plot(ax=ax, lw=0.6, alpha=0.5, legend=False, color="red")
    ax.set(
        xlabel="Week of 1948",
        ylabel="Reported measles cases",
        title=title,
    )
    ax.grid(alpha=0.15)
    plt.tight_layout()
    plt.show()

    return sir_obj
```

Attempt 1: Original Parameters

The original parameters from the lesson produce simulations where the outbreak is too small and dies out too quickly:

```
theta_original = {
    "Beta": 7.5,      # Transmission rate (per week)
    "mu_IR": 0.5,     # Recovery rate (per week)
    "N": 38000.0,     # Population size
    "eta": 0.03,      # Initial susceptible fraction
    "rho": 0.5,       # Reporting probability
    "k": 10.0         # Overdispersion parameter
}

_ = run_simulation_and_plot(theta_original, "Original parameters (Beta=7.5)")
```
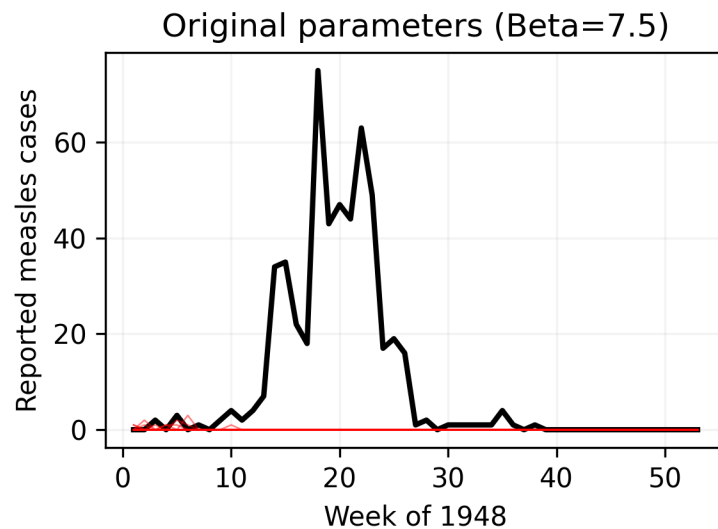

Attempt 2: Increase Force of Infection

Let's try increasing β to increase the overall incidence:

```
theta_test1 = {
    "Beta": 25.0,     # Increased transmission rate
    "mu_IR": 0.5,
    "N": 38000.0,
    "eta": 0.03,
    "rho": 0.5,
    "k": 10.0
}

_ = run_simulation_and_plot(theta_test1, "Increased Beta=25")
```
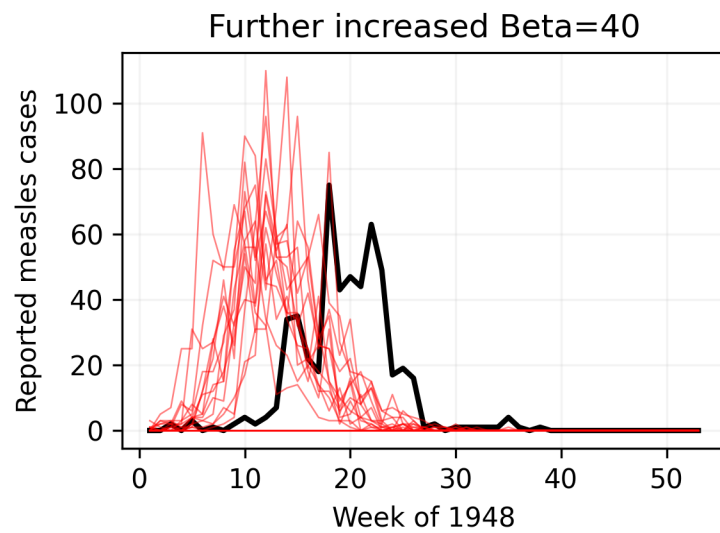
Increased Beta=25
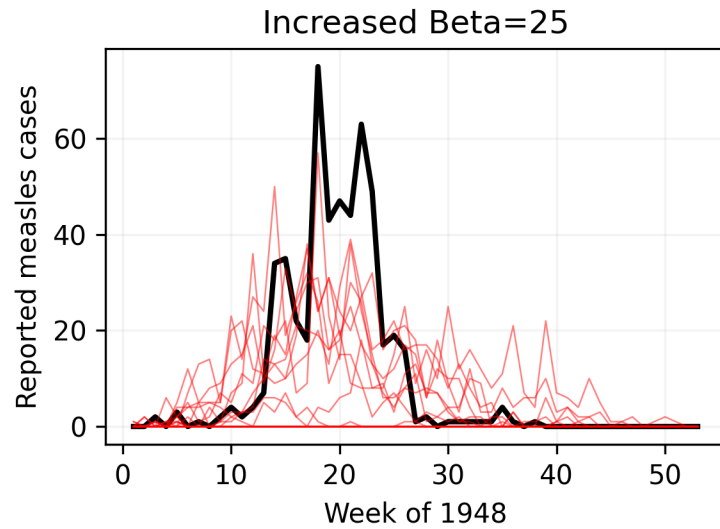
This produces larger outbreaks, but they may still be too short.

Attempt 3: Further Increase Beta

```
theta_test2 = {
    "Beta": 40.0,      # Further increased transmission rate
    "mu_IR": 0.5,
    "N": 38000.0,
    "eta": 0.03,
    "rho": 0.5,
    "k": 10.0
}

_ = run_simulation_and_plot(theta_test2, "Further increased Beta=40")
```
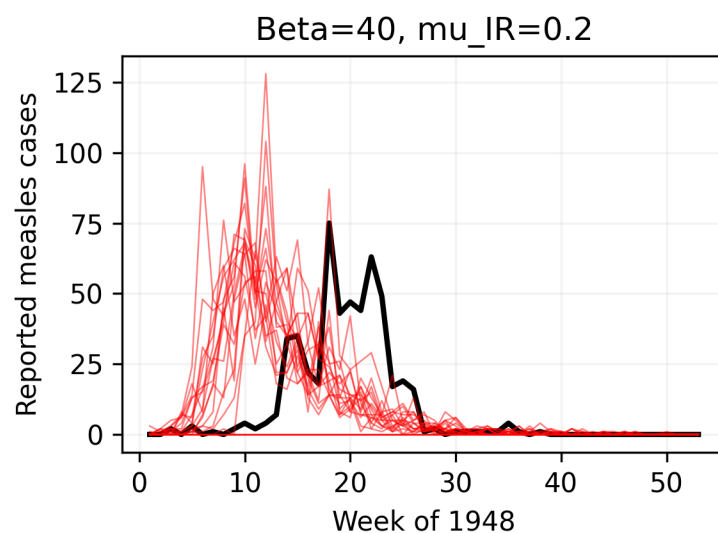

Further increased Beta=40

Attempt 4: Adjust Recovery Rate

Let's try decreasing μ_{IR} (longer infectious period):

```
theta_test3 = {
    "Beta": 40.0,
    "mu_IR": 0.2,       # Decreased recovery rate (longer infectious period)
    "N": 38000.0,
    "eta": 0.03,
    "rho": 0.5,
    "k": 10.0
}


_ = run_simulation_and_plot(theta_test3, "Beta=40, mu_IR=0.2")
```
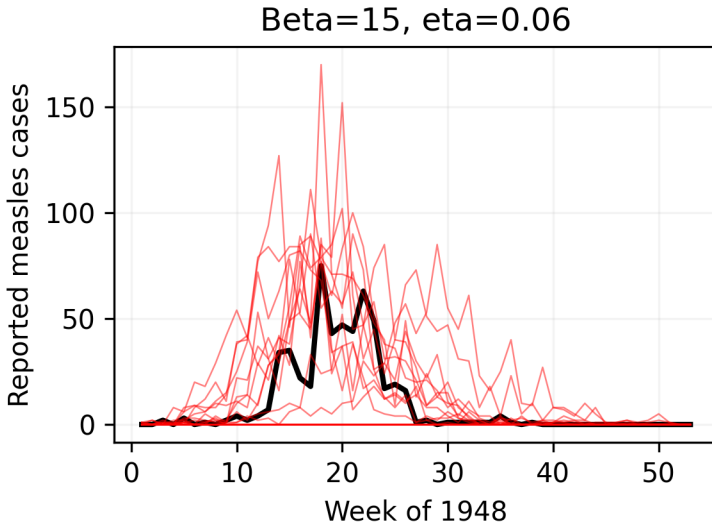

Attempt 5: Increase Susceptible Fraction

Let's try increasing the initial susceptible fraction η:

```
theta_test4 = {
    "Beta": 15.0,
    "mu_IR": 0.5,
    "N": 38000.0,
    "eta": 0.06,        # Doubled susceptible fraction
    "rho": 0.5,
    "k": 10.0
}


_ = run_simulation_and_plot(theta_test4, "Beta=15, eta=0.06")
```

Beta=15, eta=0.06

Summary of Exploration

Different parameter combinations affect the outbreak dynamics:

Parameter	Effect of Increasing
β (Beta)	Faster spread, higher peak, shorter duration
μ_{IR}	Faster recovery, shorter outbreaks
η	More susceptibles, larger outbreaks
ρ	More cases reported (observation only)
k	Less overdispersion in observations

Finding parameters that match the data well requires systematic fitting methods (covered in later lessons).

Exercise 2.4: The SEIR Model

Problem Statement

Below is a diagram of the so-called SEIR model. This differs from the SIR model in that infected individuals must pass a period of latency before becoming infectious.

$$ \boxed{S} \xrightarrow{\mu_{SE}} \boxed{E} \xrightarrow{\mu_{EI}} \boxed{I} \xrightarrow{\mu_{IR}} \boxed{R} $$
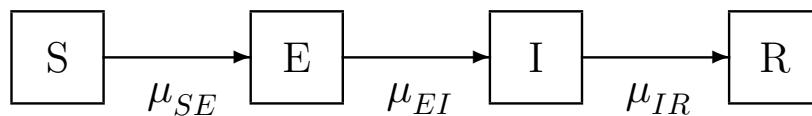
Task: Modify the codes above to construct a `Pomp` object containing the Consett measles data and an SEIR model. Perform simulations as above and adjust parameters to get a sense of whether improvement is possible by including a latent period.

SEIR Model Components

First, we define the initial state simulator for the SEIR model:

```python
def seir_rinit(theta_, key, covars, t0):
    """Initial state simulator for SEIR model."""
    N = theta_["N"]
    eta = theta_["eta"]
    S0 = jnp.round(N * eta)
    E0 = 0.0   # No initial exposed individuals
    I0 = 1.0   # Start with one infectious
    R0 = jnp.round(N * (1 - eta)) - 1.0
    H0 = 0.0   # Accumulator for true incidence
    return {"S": S0, "E": E0, "I": I0, "R": R0, "H": H0}
```

Now the process simulator, which includes the additional E (exposed) compartment:

```python
def seir_rproc(X_, theta_, key, covars, t, dt):
    """Process simulator for SEIR model with Euler-binomial scheme."""
    # Unpack state
    S, E, I, R, H = X_["S"], X_["E"], X_["I"], X_["R"], X_["H"]

    # Unpack parameters
    Beta = theta_["Beta"]
    mu_EI = theta_["mu_EI"]  # Rate of leaving E (1/latent period)
    mu_IR = theta_["mu_IR"]  # Rate of leaving I (1/infectious period)
    N = theta_["N"]

    # Transition probabilities (exponential form)
    p_SE = 1.0 - jnp.exp(-Beta * I / N * dt)  # S -> E
    p_EI = 1.0 - jnp.exp(-mu_EI * dt)          # E -> I
    p_IR = 1.0 - jnp.exp(-mu_IR * dt)          # I -> R

    # Draw transitions
    key_SE, key_EI, key_IR = jax.random.split(key, 3)
    dN_SE = jax.random.binomial(key_SE, n=jnp.int32(S), p=p_SE)
    dN_EI = jax.random.binomial(key_EI, n=jnp.int32(E), p=p_EI)
    dN_IR = jax.random.binomial(key_IR, n=jnp.int32(I), p=p_IR)

    # Update state
    S_new = S - dN_SE
    E_new = E + dN_SE - dN_EI
    I_new = I + dN_EI - dN_IR
    R_new = R + dN_IR
    H_new = H + dN_IR  # Track recoveries (proxy for new cases)

    return {"S": S_new, "E": E_new, "I": I_new, "R": R_new, "H": H_new}
```

The measurement model remains the same as for SIR:

```python
def seir_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 seir_rmeas(X_, theta_, key, covars, t):
    """Measurement simulator: sample reports ~ NegBin(k, rho*H)."""
    rho = theta_["rho"]
    k = theta_["k"]
    H = X_["H"]
    mu = rho * H
    reports = rnbinom(key, k, mu)
    return jnp.array([reports])
```

Helper Function for SEIR Simulations

```python
def run_seir_simulation_and_plot(theta, title="SEIR simulation", n_sims=20):
    """Create SEIR model, run simulations, and plot results."""

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

    seir_obj = pp.Pomp(
        rinit=seir_rinit,
        rproc=seir_rproc,
        dmeas=seir_dmeas,
        rmeas=seir_rmeas,
        ys=ys,
        theta=theta,
        statenames=statenames,
        t0=0.0,
        nstep=7,                  # 7 sub-steps per week (daily steps)
        accumvars=("H",),      # Accumulator variable
        ydim=1,
        covars=None
    )

    key = jax.random.key(42)
    X_sims, Y_sims = seir_obj.simulate(key=key, nsim=n_sims)

    sim_df = Y_sims.pivot_table(
        index="time",
        columns="sim",
        values="obs_0"
    )
```

```
    sim_df.columns = [f"sim_{i+1}" for i in range(n_sims)]

    fig, ax = plt.subplots(figsize=(4, 3))
    ys["reports"].plot(ax=ax, lw=2, color="black", label="observed")
    sim_df.plot(ax=ax, lw=0.6, alpha=0.5, legend=False, color="blue")
    ax.set(
        xlabel="Week of 1948",
        ylabel="Reported measles cases",
        title=title,
    )
    ax.grid(alpha=0.15)
    plt.tight_layout()
    plt.show()

    return seir_obj
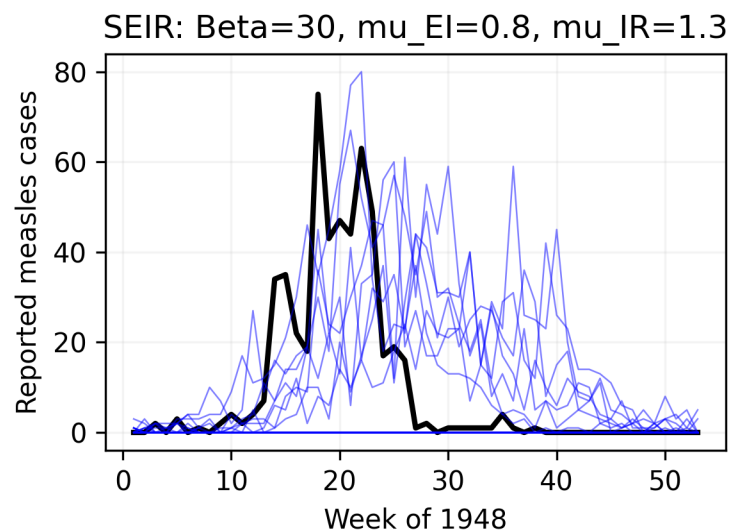```

## SEIR Simulation: Attempt 1

```
theta_seir1 = {
    "Beta": 30.0,   # Transmission rate
    "mu_EI": 0.8,   # 1/latent period (~ 1.25 weeks latent)
    "mu_IR": 1.3,   # 1/infectious period (~ 0.77 weeks infectious)
    "N": 38000.0,
    "eta": 0.06,
    "rho": 0.5,
    "k": 10.0
}

_ = run_seir_simulation_and_plot(theta_seir1, "SEIR: Beta=30, mu_EI=0.8, mu_IR=1.3")
```
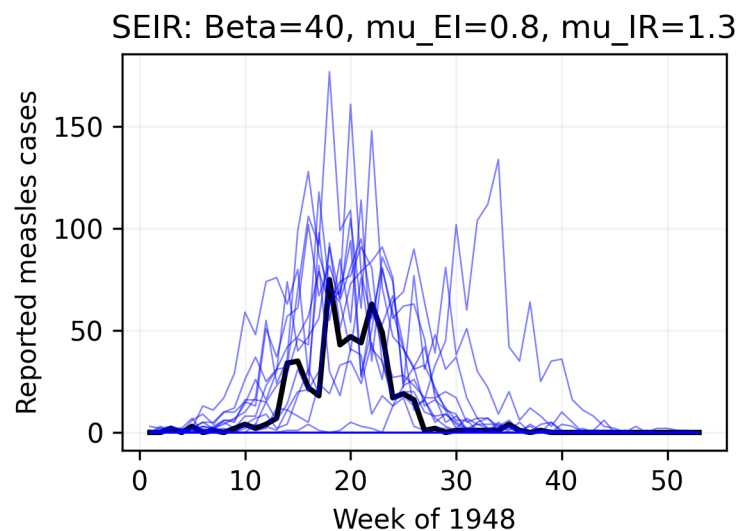


SEIR: Beta=30, mu_EI=0.8, mu_IR=1.3

## SEIR Simulation: Attempt 2

```
theta_seir2 = {
    "Beta": 40.0,      # Increased transmission
    "mu_EI": 0.8,
    "mu_IR": 1.3,
    "N": 38000.0,
    "eta": 0.06,
    "rho": 0.5,
    "k": 10.0
}

_ = run_seir_simulation_and_plot(theta_seir2, "SEIR: Beta=40, mu_EI=0.8, mu_IR=1.3")
```


SEIR: Beta=40, mu_EI=0.8, mu_IR=1.3

Comparison: SIR vs SEIR

The SEIR model introduces a **latent period** (time in the E compartment before becoming infectious), which can affect:

1. **Outbreak timing**: Delayed onset due to latent period
2. **Peak timing**: Later peak as infections take longer to develop
3. **Outbreak shape**: More gradual rise and fall

For measles, realistic parameters might be:

- Latent period ($1/\mu_{EI}$): ~8-13 days
- Infectious period ($1/\mu_{IR}$): ~4-8 days

The SEIR model provides a more biologically realistic representation of measles epidemiology.

Summary

Key Takeaways

1. **Little-o notation** provides a compact way to express derivatives and is especially useful for continuous-time Markov chains

2. **Euler vs Gillespie**: Euler methods are preferred for computational efficiency and handling large populations; Gillespie provides exact simulations but is slower

3. **Parameter exploration**: Finding good parameters requires systematic experimentation and ultimately formal fitting methods

4. **SEIR vs SIR**: The SEIR model's latent period can better capture the epidemiology of diseases like measles

References

References