# Lesson 7: Case Study - Forecasting Ebola

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

2025-12-24

# Table of contents I

# Table of contents II

This lesson presents a case study of forecasting Ebola, demonstrating the use of diagnostic probes for model criticism and forecasting methods based on POMP models.

# Learning Objectives

1. To explore the use of POMP models in the context of an outbreak of an emerging infectious disease.
2. To demonstrate the use of diagnostic probes for model criticism.
3. To illustrate some forecasting methods based on POMP models.
4. To provide an example that can be modified to apply similar approaches to other outbreaks of emerging infectious diseases.

This lesson follows King et al. (2015), all codes for which are available on datadryad.org.

# An Emerging Infectious Disease Outbreak I

Let's situate ourselves at the beginning of October 2014. The WHO situation report contained data on the number of cases in each of Guinea, Sierra Leone, and Liberia. Key questions included:

1. How fast will the outbreak unfold?
2. How large will it ultimately prove?
3. What interventions will be most effective?

# An Emerging Infectious Disease Outbreak II

As is to be expected in the case of a fast-moving outbreak of a novel pathogen in an underdeveloped country, the answers to these questions were sought in a context far from ideal:

- Case ascertainment is difficult and the case definition itself may be evolving.
- Surveillance effort is changing on the same timescale as the outbreak itself.
- The public health and behavioral response to the outbreak is rapidly changing.

# Best Practices I

- The King et al. (2015) paper focused critical attention on the economical and therefore common practice of fitting deterministic transmission models to cumulative incidence data.
- Specifically, King et al. (2015) showed how this practice easily leads to overconfident prediction that, worryingly, can mask their own presence.

# Best Practices II

- The paper recommended the use of POMP models, for several reasons:
  - Such models can accommodate a wide range of hypothetical forms.
  - They can be readily fit to incidence data, especially during the exponential growth phase of an outbreak.
  - Stochastic models afford a more explicit treatment of uncertainty.
  - POMP models come with a number of diagnostic approaches built-in, which can be used to assess model misspecification.

# Situation-Report Data I

The data we focus on here are from the WHO Situation Report of 1 October 2014. Supplementing these data are population estimates for the three countries.

```
import jax
import jax.numpy as jnp
import jax.random as jr
import jax.scipy as jsp
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from pypomp import Pomp
from pypomp.util import logmeanexp, logmeanexp_se
from scipy.stats import chi2

np.random.seed(594709947)

# Load the data
dat = pd.read_csv("https://kingaa.github.io/sbied/ebola/ebola_data.csv")
print(dat.head(10))
```

# Situation-Report Data II

| | week | date | country | cases |
|---|---|---|---|---|
| 0 | 1 | 2014-01-05 | Guinea | 2.244 |
| 1 | 2 | 2014-01-12 | Guinea | 2.244 |
| 2 | 3 | 2014-01-19 | Guinea | 0.073 |
| 3 | 4 | 2014-01-26 | Guinea | 5.717 |
| 4 | 5 | 2014-02-02 | Guinea | 3.954 |
| 5 | 6 | 2014-02-09 | Guinea | 5.444 |
| 6 | 7 | 2014-02-16 | Guinea | 3.274 |
| 7 | 8 | 2014-02-23 | Guinea | 5.762 |
| 8 | 9 | 2014-03-02 | Guinea | 7.615 |
| 9 | 10 | 2014-03-09 | Guinea | 7.615 |

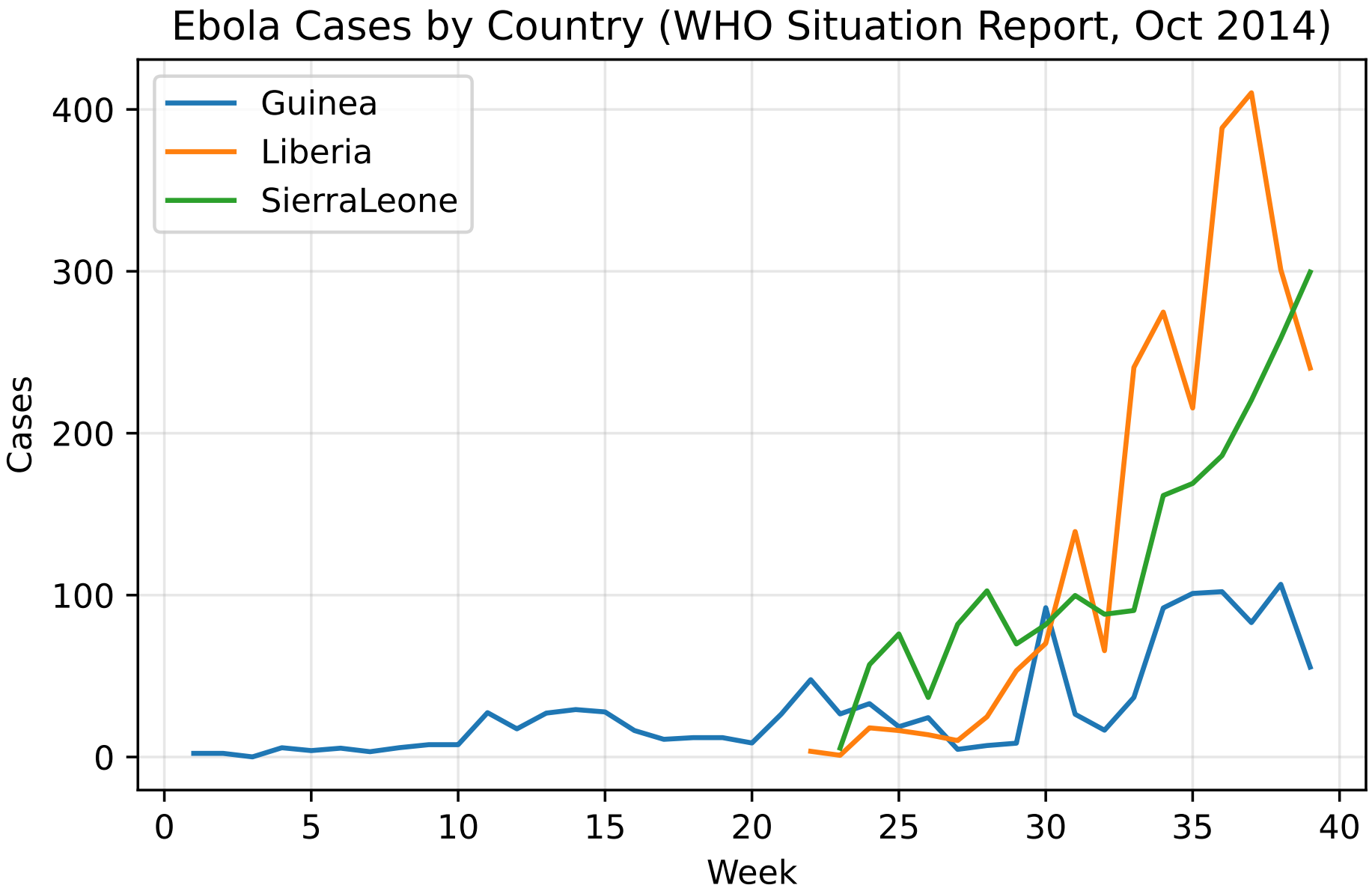
# Situation-Report Data III

```
# Population sizes (2014 census)
populations = {
    "Guinea": 10628972.0,
    "Liberia": 4092310.0,
    "SierraLeone": 6190280.0
}
```

# Situation-Report Data IV

```
# Plot the data
fig, ax = plt.subplots(figsize=(6, 4))
for country in dat['country'].unique():
    country_data = dat[dat['country'] == country]
    ax.plot(country_data['week'], country_data['cases'],
            label=country, linewidth=1.5)
ax.set_xlabel('Week')
ax.set_ylabel('Cases')
ax.set_title('Ebola Cases by Country (WHO Situation Report, Oct 2014)')
ax.legend()
ax.grid(alpha=0.3)
plt.tight_layout()
plt.show()
```

# Situation-Report Data V



# SEIR Model with Gamma-Distributed Latent Period I

- Many of the early modeling efforts used variants on the simple SEIR model.
- Here, we'll focus on a variant that attempts a more careful description of the duration of the latent period.
- Specifically, this model assumes that the amount of time an infection remains latent is

$$\text{LP} \sim \text{Gamma}(m, \frac{1}{m\,\alpha}),$$

  where $m$ is an integer.
- This means that the latent period has expectation $1/\alpha$ and variance $1/(m\,\alpha)$. In this document, we'll fix $m = 3$.

# SEIR Model with Gamma-Distributed Latent Period II

- We implement Gamma distributions using the so-called *linear chain trick*.

The model structure is:

$$S \rightarrow E_1 \rightarrow E_2 \rightarrow E_3 \rightarrow I \rightarrow R/\text{dead}$$

# SEIR Model with Gamma-Distributed Latent Period III

The observations are modeled as a negative binomial process conditional on the number of infections. That is, if $C_t$ are the reported cases at week $t$ and $H_t$ is the true incidence, then we postulate that $C_t|H_t$ is negative binomial with

$$\mathbb{E}\left[C_t|H_t\right] = \rho\, H_t \quad \text{and} \quad \mathrm{Var}\left[C_t|H_t\right] = \rho\, H_t\,(1 + k\,\rho\, H_t).$$

The negative binomial process allows for overdispersion in the counts. This overdispersion is controlled by parameter $k$. When $k \to 0$, the distribution reduces to Poisson.

# Building the Model in pypomp I

```
# Number of exposed stages (for linear chain trick)
nstageE = 3

# Euler integration timestep
timestep = 0.1
```

# Building the Model in pypomp II

```
def rinit(theta_, key, covars, t0=None):
    """Initial state distribution."""
    N = theta_["N"]
    S_0 = theta_["S_0"]
    E_0 = theta_["E_0"]
    I_0 = theta_["I_0"]
    R_0 = theta_["R_0"]

    m = N / (S_0 + E_0 + I_0 + R_0)
    S = jnp.rint(m * S_0)
    E_each = jnp.rint(m * E_0 / nstageE)
    I = jnp.rint(m * I_0)
    R = jnp.rint(m * R_0)

    result = {"S": S, "I": I, "R": R, "N_EI": 0.0, "N_IR": 0.0}
    for i in range(nstageE):
        result[f"E{i+1}"] = E_each

    return result
```

# Building the Model in pypomp III

```
def rproc(X_, theta_, key, covars, t=None, dt=None):
    """Process model: SEIR with Erlang-distributed exposed period."""
    N = theta_["N"]
    R0 = theta_["R0"]
    alpha = theta_["alpha"]
    gamma = theta_["gamma"]

    S = X_["S"]
    E = jnp.array([X_[f"E{i+1}"] for i in range(nstageE)])
    I = X_["I"]
    R = X_["R"]
    N_EI_prev = X_["N_EI"]
    N_IR_prev = X_["N_IR"]

    # Transmission rate
    beta = R0 * gamma
    lam = beta * I / N

    # Transition probabilities (clipped to valid range)
    pS = jnp.clip(1.0 - jnp.exp(-lam * timestep), 0.0, 1.0)
    pE = jnp.clip(1.0 - jnp.exp(-nstageE * alpha * timestep), 0.0, 1.0)
    pI = jnp.clip(1.0 - jnp.exp(-gamma * timestep), 0.0, 1.0)

    # Split keys for binomial draws
```

# Parameter Estimates I

- King et al. (2015) estimated parameters for this model for each country.
- A Latin hypercube design was used to initiate a large number of iterated filtering runs.
- Profile likelihoods were computed for each country against the parameters $k$ (the measurement model overdispersion) and $\mathcal{R}_0$ (the basic reproductive ratio).
- Full details are given on the datadryad.org site.

# Parameter Estimates II

```
# Display loaded MLE parameters for all countries
profs = pd.read_csv("https://kingaa.github.io/sbied/ebola/ebola_profiles.csv")

for country in ["Guinea", "SierraLeone", "Liberia"]:
    best = profs[profs["country"] == country].copy()
    best = best.loc[best["loglik"].idxmax()]
    print(f"\n{country} MLE parameters:")
    print(f"  R0: {best['R0']:.3f}")
    print(f"  alpha: {best['alpha']:.3f}")
    print(f"  gamma: {best['gamma']:.3f}")
    print(f"  rho: {best['rho']:.3f}")
    print(f"  k: {best['k']:.3f}")
    print(f"  loglik: {best['loglik']:.2f}")
```

# Parameter Estimates III

```
Guinea MLE parameters:
  R0: 1.215
  alpha: 0.620
  gamma: 1.005
  rho: 0.200
  k: 0.374
  loglik: -152.24

SierraLeone MLE parameters:
  R0: 1.322
  alpha: 0.620
  gamma: 1.005
  rho: 0.200
  k: 0.038
```

# Parameter Estimates IV

```
  loglik: -77.71

Liberia MLE parameters:
  R0: 1.905
  alpha: 0.620
  gamma: 1.005
  rho: 0.200
  k: 0.236
  loglik: -88.10
```

# Diagnostics or Model Criticism I

- Parameter estimation is the process of finding the parameters that are "best", in some sense, for a given model, from among the set of those that make sense for that model.
- Model selection, likewise, aims at identifying the "best" model, in some sense, from among a set of candidates.
- One can do both of these things more or less well, but no matter how carefully they are done, the best of a bad set of models is still bad.

# Diagnostics or Model Criticism II

- Let's investigate the model here, at its maximum-likelihood parameters, to see if we can identify problems.
- The guiding principle in this is that, if the model is "good", then the data are a plausible realization of that model.
- Therefore, we can compare the data directly against model simulations.
- Moreover, we can quantify the agreement between simulations and data in any way we like.

# Diagnostics or Model Criticism III

- Any statistic, or set of statistics, that can be applied to the data can also be applied to simulations.
- Shortcomings of the model should manifest themselves as discrepancies between the model-predicted distribution of such statistics and their value on the data.
- **pypomp** provides tools to facilitate this process through simulation-based diagnostics.

# Model Simulations I

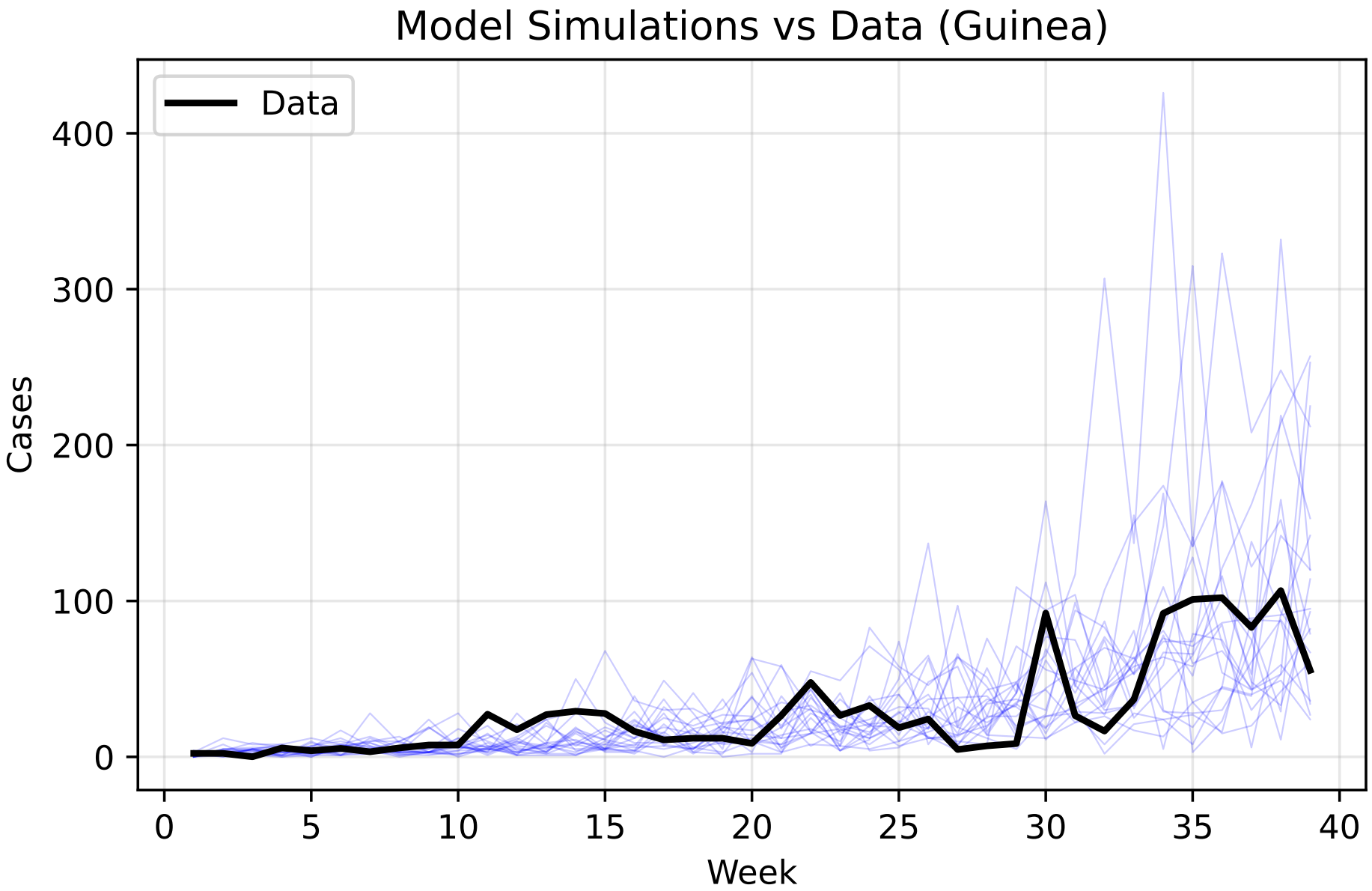
```
# Simulate from the model at MLE
key = jax.random.key(1234567)
X_sims, Y_sims = gin.simulate(key=key, nsim=100)

print(f"Simulated {100} trajectories")
```

```
Simulated 100 trajectories
```

# Model Simulations II

```
# Plot simulations vs data
fig, ax = plt.subplots(figsize=(6, 4))

# Plot simulations
for i in range(min(20, 100)):
    sim_data = Y_sims[Y_sims['sim'] == i]
    ax.plot(sim_data['time'], sim_data['obs_0'],
            alpha=0.2, color='blue', linewidth=0.5)

# Plot actual data
ax.plot(dat_g['week'], dat_g['cases'],
        'k-', linewidth=2, label='Data')

ax.set_xlabel('Week')
ax.set_ylabel('Cases')
ax.set_title('Model Simulations vs Data (Guinea)')
ax.legend()
ax.grid(alpha=0.3)
plt.tight_layout()
plt.show()
```

# Model Simulations III



# Diagnostic Probes I

- Does the data look like it could have come from the model?
  - The simulations appear to be growing a bit more quickly than the data.
- Let's try to quantify this:
  - First, we'll write a function that estimates the exponential growth rate by linear regression.
  - Then, we'll apply it to the data and to simulations.

# Diagnostic Probes II

```
def growth_rate_probe(y):
    """Estimate exponential growth rate from case data."""
    y = np.array(y).flatten()
    # Handle zeros by adding small constant
    y_safe = np.maximum(y, 0.5)
    log_y = np.log(y_safe)

    # Linear regression on log data
    t = np.arange(len(y))
    mask = ~np.isnan(log_y) & ~np.isinf(log_y)
    if mask.sum() < 2:
        return np.nan

    # Simple linear regression
    t_m = t[mask]
    y_m = log_y[mask]
    slope = np.cov(t_m, y_m)[0, 1] / np.var(t_m)
    return slope

def residual_sd_probe(y):
    """Compute SD of residuals from exponential trend."""
    y = np.array(y).flatten()
    y_safe = np.maximum(y, 0.5)
    log_y = np.log(y_safe)
```

# Interpretation I

- Do these results bear out our suspicion that the model and data differ in terms of growth rate?
- The simulations also appear to be more highly variable around the trend than do the data.
- Let's also look more carefully at the distribution of values about the trend using additional probes.

# Forecasting and Forecasting Uncertainty I

- To this point in the course, we've focused on using POMP models to answer scientific questions, i.e., to compare alternative hypothetical explanations for the data in hand.
- Of course, we can also use them to make forecasts.

# Forecasting and Forecasting Uncertainty II

- A set of key issues surrounds quantifying the forecast uncertainty.
- This arises from four sources:
  1. measurement error
  2. process noise
  3. parametric uncertainty
  4. structural uncertainty
- Here, we'll explore how we can account for the first three of these in making forecasts for the Sierra Leone outbreak.

# Parameter Uncertainty I

We take an *empirical Bayes* approach.

First, we set up a collection of parameter vectors in a neighborhood of the maximum likelihood estimate containing the region of high likelihood.

# Parameter Uncertainty II

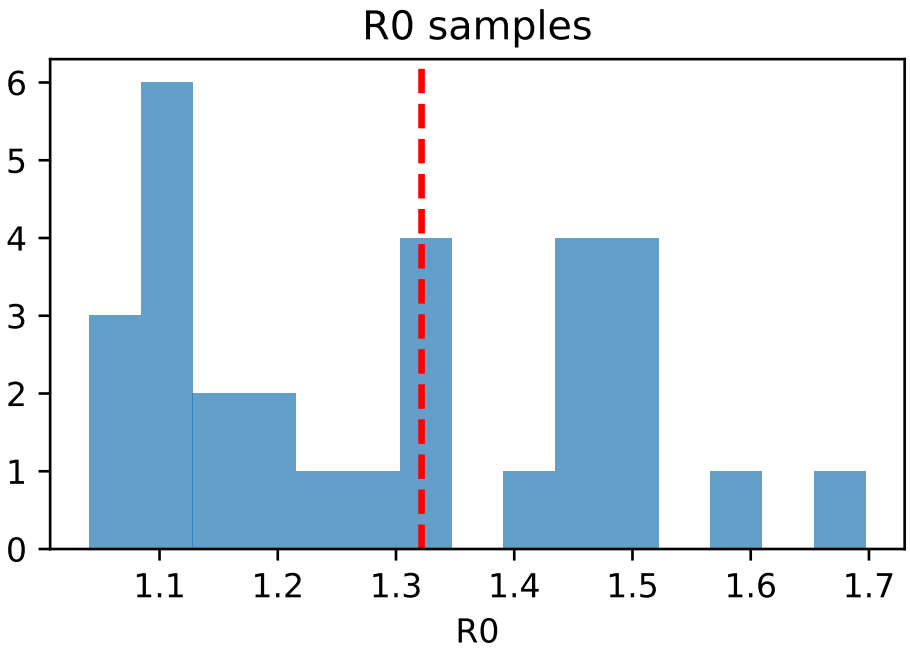
```python
# Generate parameter samples around MLE
def sample_params_near_mle(base_theta, n_samples=50, scale=0.1):
    """Sample parameters in neighborhood of MLE."""
    params_list = []
    for _ in range(n_samples):
        params = base_theta.copy()
        for key in ["R0", "rho", "k"]:
            # Perturb on log scale
            params[key] = base_theta[key] * np.exp(np.random.normal(0, scale))
        params_list.append(params)
    return params_list

# Sample parameters for Sierra Leone
np.random.seed(887851050)
param_samples = sample_params_near_mle(sle.theta[0],
                                       n_samples=30, scale=0.15)
print(f"Generated {len(param_samples)} parameter samples")
```

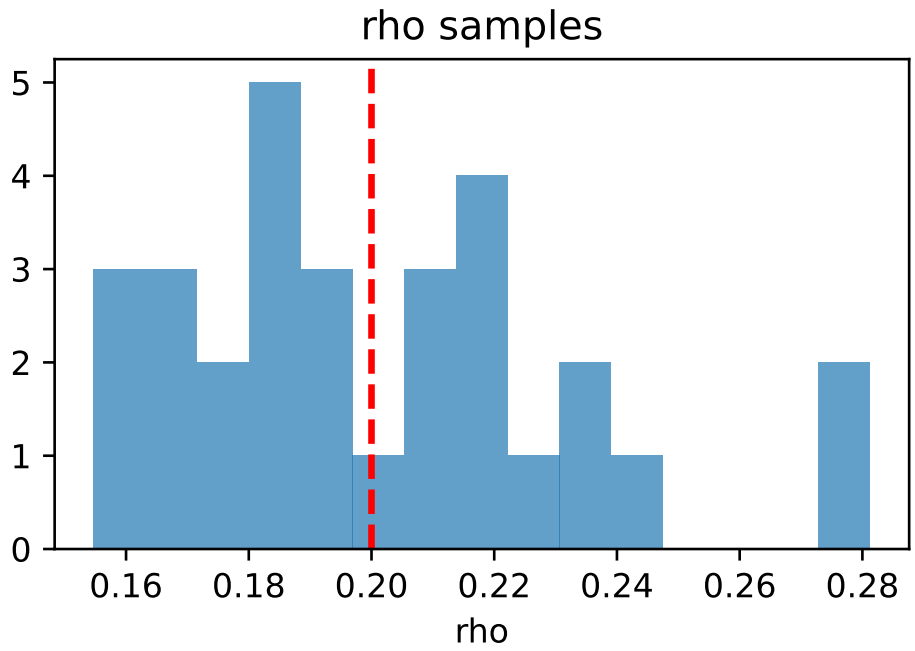
```
Generated 30 parameter samples
```

# Parameter Uncertainty III

```
# Visualize parameter distribution
R0_vals = [p["R0"] for p in param_samples]
rho_vals = [p["rho"] for p in param_samples]

fig, axes = plt.subplots(1, 2, figsize=(8, 3))
axes[0].hist(R0_vals, bins=15, alpha=0.7)
axes[0].axvline(sle.theta[0]["R0"], color='r',
                linewidth=2, linestyle='--')
axes[0].set_xlabel('R0')
axes[0].set_title('R0 samples')

axes[1].hist(rho_vals, bins=15, alpha=0.7)
axes[1].axvline(sle.theta[0]["rho"], color='r',
                linewidth=2, linestyle='--')
axes[1].set_xlabel('rho')
axes[1].set_title('rho samples')

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

# Parameter Uncertainty IV



# Process Noise and Measurement Error I

Next, we carry out a particle filter at each parameter vector, which gives us estimates of both the likelihood and the filter distribution at that parameter value.

# Process Noise and Measurement Error II

```
# For each parameter set, run particle filter and extract final states
# Then simulate forecasts

def run_forecast(pomp_obj, params, n_particles=500,
                 forecast_weeks=12, seed=12345):
    """Run particle filter and generate forecasts."""
    # Update parameters
    pomp_obj.theta = [params]

    # Run particle filter
    key = jax.random.key(seed)
    pomp_obj.pfilter(J=n_particles, key=key, reps=1, thresh=0)

    pf_result = pomp_obj.results_history[-1]
    loglik = pf_result.logLiks.values.flatten()[0]

    # Simulate forecasts from final state
    X_sims, Y_sims = pomp_obj.simulate(key=key, nsim=10)

    return loglik, Y_sims
```

# Process Noise and Measurement Error III

```
# Run forecasts for a subset of parameter samples
forecast_results = []
n_forecast = min(10, len(param_samples))  # Limit for demo

for i, params in enumerate(param_samples[:n_forecast]):
    loglik, sims = run_forecast(sle, params, n_particles=200,
                                seed=i * 1000)
    forecast_results.append({
        'params': params,
        'loglik': loglik,
        'sims': sims
    })
    print(f"Forecast {i+1}: loglik = {loglik:.2f}, R0 = {params['R0']:.3f}")
```

# Process Noise and Measurement Error IV

```
Forecast 1: loglik = -136.28, R0 = 1.497
Forecast 2: loglik = -93.57, R0 = 1.164
Forecast 3: loglik = -82.50, R0 = 1.202
Forecast 4: loglik = -90.92, R0 = 1.448
Forecast 5: loglik = -170.08, R0 = 1.040
Forecast 6: loglik = -145.88, R0 = 1.119
Forecast 7: loglik = -79.89, R0 = 1.308
Forecast 8: loglik = -77.09, R0 = 1.320
Forecast 9: loglik = -105.28, R0 = 1.458
Forecast 10: loglik = -88.28, R0 = 1.438
```

# Computing Forecast Quantiles I

We give these prediction distributions weights proportional to the estimated likelihoods of the parameter vectors.

```
# Compute weights from log-likelihoods
logliks = np.array([r['loglik'] for r in forecast_results])
# Subtract max for numerical stability
weights = np.exp(logliks - np.max(logliks))
weights = weights / weights.sum()

# Effective sample size
ess = 1 / np.sum(weights**2)
print(f"Effective sample size: {ess:.1f}")
print(f"Weights range: [{weights.min():.4f}, {weights.max():.4f}]")
```

```
Effective sample size: 1.1
Weights range: [0.0000, 0.9386]
```

# Computing Forecast Quantiles II

```
# Combine forecasts with weights
# Plot weighted forecast envelope

fig, ax = plt.subplots(figsize=(6, 4))

# Plot data
ax.plot(dat_s['week'], dat_s['cases'], 'k-',
        linewidth=2, label='Data')

# Plot weighted simulations (clip alpha to valid range)
for i, result in enumerate(forecast_results):
    sims = result['sims']
    for j in range(min(5, 10)):
        sim_data = sims[sims['sim'] == j]
        # Clip alpha to [0, 1] range
        alpha_val = np.clip(weights[i] * 3, 0.05, 1.0)
        ax.plot(sim_data['time'], sim_data['obs_0'],
                alpha=alpha_val, color='blue', linewidth=0.5)

ax.set_xlabel('Week')
ax.set_ylabel('Cases')
ax.set_title('Ebola Forecasts for Sierra Leone')
ax.legend()
ax.grid(alpha=0.3)
```

# Exercise 1: The Sierra Leone Outbreak

Apply probes to investigate the extent to which the SEIR model above is an adequate description of the data from the Sierra Leone outbreak.

- Have a look at the probes provided with **pypomp** (growth rate, residual SD, autocorrelation).
- Try also to come up with some informative probes of your own.
- Discuss the implications of your findings.

# Exercise 2: Decomposing the Uncertainty

As we have discussed, the uncertainty shown in the forecasts above has three sources: parameter uncertainty, process noise, and measurement error.

Show how you can break the total uncertainty into these three components. Produce plots similar to that above showing each of the components.

# Summary I

1. **POMP models** are well-suited for modeling emerging infectious disease outbreaks, providing a framework for both inference and forecasting.
2. **Model criticism** through simulation-based diagnostics is essential. We compare data against model simulations using probe statistics.
3. **Forecasting uncertainty** has multiple sources:
    - Measurement error
    - Process noise
    - Parametric uncertainty
    - Structural uncertainty
4. **Empirical Bayes** approaches allow us to incorporate parameter uncertainty into forecasts by sampling from the high-likelihood region.

# 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

King, A. A., Domenech de Cellès, M., Magpantay, F. M. G., and Rohani, P. (2015). Avoidable errors in the modelling of outbreaks of emerging pathogens, with special reference to Ebola. *Proc R Soc Lond B*, 282(1806):20150347.