Chapter 6: Case Study – Forecasting Ebola

Source code at pypomp/tutorials/sbied/chapter6

Aaron A. King Edward L. Ionides Kunyang He

Compiled on April 9, 2026

Introduction

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.

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

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?

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

- King et al. (2015) focused critical attention on the simple 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.

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.

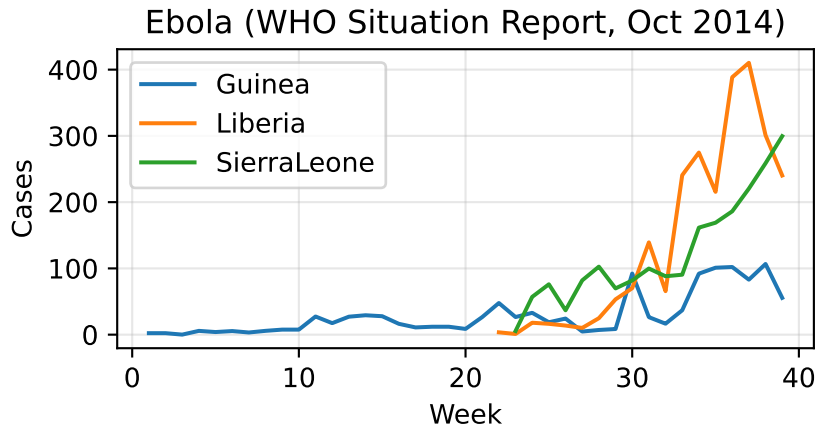
Situation-report data

```
dat = pd.read_csv("ebola_data.csv")
print(dat.head(10))
```

```
   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
```

Known population size

```
populations = {
    "Guinea": 10628972.0,
    "Liberia": 4092310.0,
    "SierraLeone": 6190280.0,
}
```



SEIR model with gamma-distributed latent period

- 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 $1/(m\,\alpha^2)$. In this document, we'll fix $m = 3$.

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

The model structure is:

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

The equations are solved numerically by Euler's method.

```
nstageE = 3
timestep = 0.1
```

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

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

```
def rinit(theta_, key, covars, t0):
    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
```

`rproc` deals with multiple E compartments:

```
def rproc(X_, theta_, key, covars, t, dt):
    N = theta_["N"]
    R0 = theta_["R0"]
    alpha = theta_["alpha"]
    gamma = theta_["gamma"]

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

    beta = R0 * gamma
    lam = beta * I / N

    pS = 1.0 - jnp.exp(-lam * timestep)
    pE = 1.0 - jnp.exp(
        -nstageE * alpha * timestep)
    pI = 1.0 - jnp.exp(-gamma * timestep)

    keys = jax.random.split(key, 2 + nstageE)
```

```
    transS = fast_approx_rbinom(
        keys[0], jnp.float32(S), pS)

    transE = []
    for i in range(nstageE):
        transEi = fast_approx_rbinom(
            keys[1 + i],
            jnp.float32(jnp.maximum(E[i], 0)),
            pE)
        transE.append(transEi)
    transE = jnp.stack(transE)

    transI = fast_approx_rbinom(
        keys[-1], jnp.float32(I), pI)

    S_new = S - transS
    E_new = jnp.zeros(nstageE)
    E_new = E_new.at[0].set(
        E[0] + transS - transE[0])
    for i in range(1, nstageE):
        E_new = E_new.at[i].set(
            E[i] + transE[i-1] - transE[i])
    I_new = I + transE[-1] - transI
    R_new = R + transI

    result = {
        "S": S_new, "I": I_new, "R": R_new,
        "N_EI": N_EI + transE[-1],
        "N_IR": N_IR + transI,
    }
    for i in range(nstageE):
        result[f"E{i+1}"] = E_new[i]
    return result
```

```
def dmeas(Y_, X_, theta_, covars, t):
    rho = theta_["rho"]
    k = theta_["k"]
    N_EI = jnp.maximum(X_["N_EI"], 0.0)
    mu = rho * N_EI
    cases = jnp.rint(Y_["cases"])
    return jnp.where(
        k > 0,
        nbinom_logpmf(cases, 1.0 / k, mu),
        jnp.where(
```

```
            mu > 0,
            cases * jnp.log(mu) - mu
            - jax.scipy.special.gammaln(cases + 1),
            jnp.where(cases == 0, 0.0, -jnp.inf)))
```

```
def rmeas(X_, theta_, key, covars, t):
    rho = theta_["rho"]
    k = theta_["k"]
    N_EI = jnp.maximum(X_["N_EI"], 0.0)
    mu = rho * N_EI
    k1, k2 = jax.random.split(key)
    r = 1.0 / jnp.maximum(k, 1e-10)
    lam = fast_approx_rgamma(k1, r) * (mu / r)
    y = fast_approx_rpoisson(k2, lam)
    return jnp.array([y])
```

Building the pomp object

We define a function `python build_ebola_model(country)` that builds a POMP model corresponding to `country`.

- Parameter values correspond to the MLE of a previous search.
- Model components are constructed above.

```
Models built for Guinea, Sierra Leone, and Liberia
```

Parameter estimates

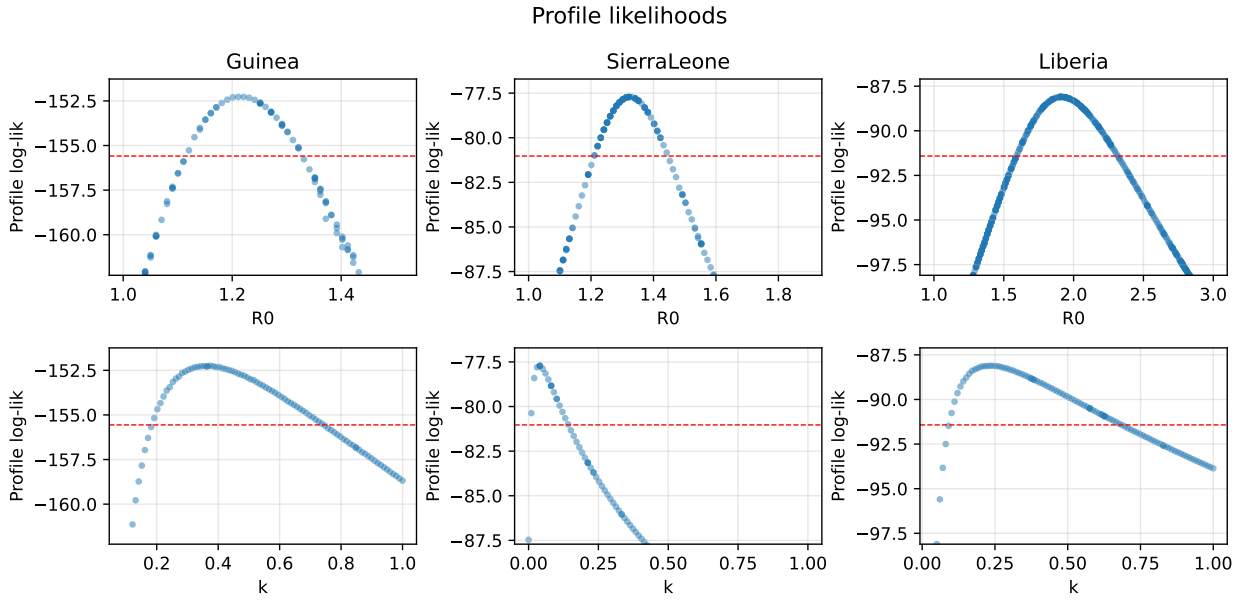
- King et al. (2015) estimated parameters for this model for each country.
- A Latin hypercube design was used to initiate a large number of iterated filtering runs.
- Profile likelihoods were computed for each country against the parameters k (the measurement model overdispersion) and $\mathcal{R}_0$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

Pre-computed MLEs

```
profs = pd.read_csv("ebola_profiles.csv")
```

```
Guinea MLE:
  R0=1.215  alpha=0.620  gamma=1.005
  rho=0.200  k=0.374  loglik=-152.2

SierraLeone MLE:
  R0=1.322  alpha=0.620  gamma=1.005
  rho=0.200  k=0.038  loglik=-77.7
```

```
Liberia MLE:
  R0=1.905  alpha=0.620  gamma=1.005
  rho=0.200  k=0.236  loglik=-88.1
```



Diagnostics or model criticism

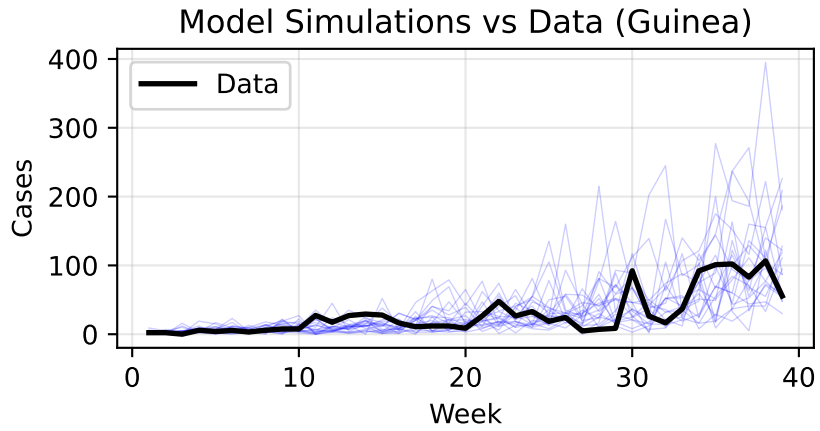
- Parameter estimation finds the "best" parameters for a given model. Model selection identifies the "best" model from among candidates. But the best of a bad set of models is still bad.
- The guiding principle: if the model is "good", the data are a plausible realization of that model.
- Any statistic that can be applied to the data can also be applied to simulations. Shortcomings of the model should manifest as discrepancies.
- `pypomp` provides a `probe` method to facilitate this process.

Model simulations

```
cache_file = cache_dir + "/sim-diagnostics.pkl"

key_chain, key_sim = jax.random.split(key_chain)
if os.path.exists(cache_file):
    with open(cache_file, 'rb') as f:
        X_sims, Y_sims = pickle.load(f)
else:
    X_sims, Y_sims = gin.simulate(
        key=key_sim, nsim=200)
    with open(cache_file, 'wb') as f:
```

```
        pickle.dump((X_sims, Y_sims), f)

print(f"Simulated {200} trajectories")
```

```
Simulated 200 trajectories
```



Diagnostic probes

- 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 quantify this using the `probe` method in `pypomp`.

```
def growth_rate(df):
    """Exponential growth rate via linear
    regression on log(cases)."""
    y = df.iloc[:, 0].values
    y_safe = np.maximum(y, 0.5)
    log_y = np.log(y_safe)
    t = np.arange(len(y))
    mask = np.isfinite(log_y)
    if mask.sum() < 2:
        return np.nan
    t_m, y_m = t[mask], log_y[mask]
    return float(np.cov(t_m, y_m)[0, 1]
                 / np.var(t_m))
```

```
def residual_sd(df):
    """SD of residuals from exponential trend."""
```

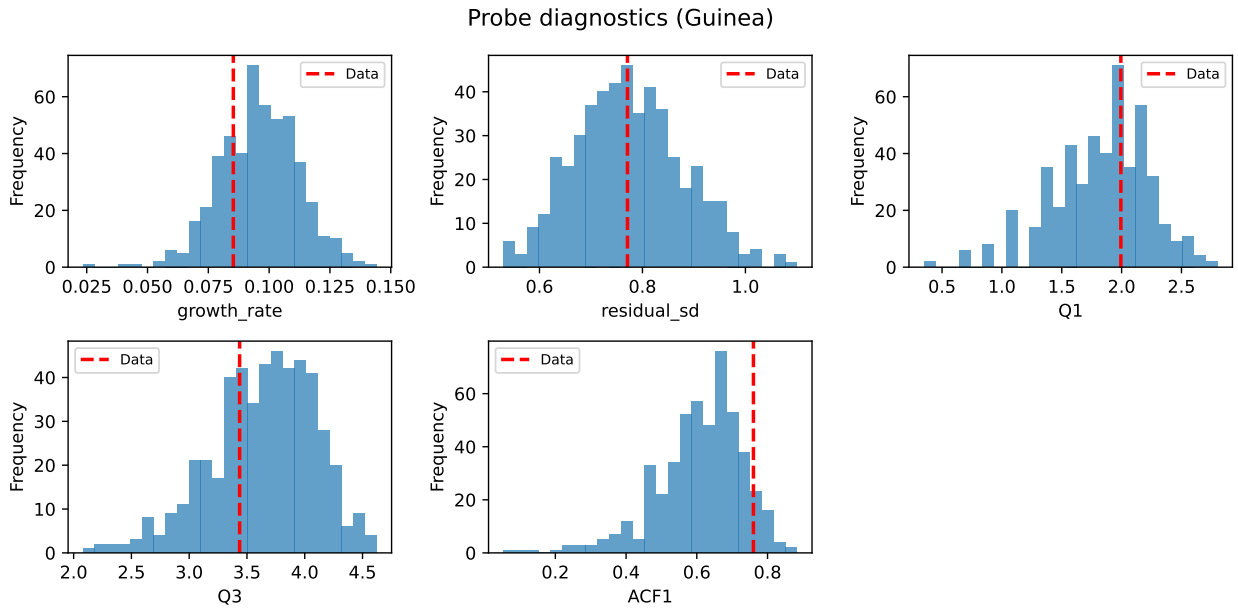
```
    y = df.iloc[:, 0].values
    y_safe = np.maximum(y, 0.5)
    log_y = np.log(y_safe)
    t = np.arange(len(y))
    mask = np.isfinite(log_y)
    if mask.sum() < 2:
        return np.nan
    t_m, y_m = t[mask], log_y[mask]
    slope = np.cov(t_m, y_m)[0,1] / np.var(t_m)
    intercept = y_m.mean() - slope * t_m.mean()
    resid = y_m - (slope * t_m + intercept)
    return float(np.std(resid))
```

```
def quantile_1(df):
    """1st quartile of log cases."""
    y = np.maximum(df.iloc[:, 0].values, 0.5)
    return float(np.quantile(np.log(y), 0.25))

def quantile_3(df):
    """3rd quartile of log cases."""
    y = np.maximum(df.iloc[:, 0].values, 0.5)
    return float(np.quantile(np.log(y), 0.75))

def acf_1(df):
    """Lag-1 autocorrelation of log cases."""
    y = np.maximum(df.iloc[:, 0].values, 0.5)
    x = np.log(y)
    x = x - x.mean()
    if np.var(x) == 0:
        return 0.0
    return float(
        np.corrcoef(x[:-1], x[1:])[0, 1])
```

```
probes = {
    "growth_rate": growth_rate,
    "residual_sd": residual_sd,
    "Q1": quantile_1,
    "Q3": quantile_3,
    "ACF1": acf_1,
}

cache_file = cache_dir + "/probe-results.pkl"
```

```
if os.path.exists(cache_file):
    with open(cache_file, 'rb') as f:
        probe_df = pickle.load(f)
else:
    key = jax.random.key(887851050)
    probe_df = gin.probe(
        probes=probes, nsim=500, key=key)
    with open(cache_file, 'wb') as f:
        pickle.dump(probe_df, f)

# Separate data and simulation values
data_vals = probe_df[
    probe_df['is_real_data'] == True]
sim_vals = probe_df[
    probe_df['is_real_data'] == False]

print("Probe results:")
for name in probes:
    dv = float(data_vals[
        data_vals['probe']==name]['value'].iloc[0])
    sv = sim_vals[
        sim_vals['probe']==name]['value']
    pval = float(np.mean(sv <= dv))
    print(f"  {name}: data={dv:.4f}  "
          f"sim_mean={sv.mean():.4f}  "
          f"p={pval:.3f}")
```

```
Probe results:
  growth_rate: data=0.0853  sim_mean=0.0959  p=0.258
  residual_sd: data=0.7709  sim_mean=0.7749  p=0.500
  Q1: data=1.9938  sim_mean=1.8356  p=0.610
  Q3: data=3.4364  sim_mean=3.6384  p=0.312
  ACF1: data=0.7601  sim_mean=0.6092  p=0.916
```



- 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 using POMP models

- 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.

Forecasting Ebola: an empirical Bayes approach

To incorporate **parameter uncertainty**, we use **empirical Bayes**.

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

```
np.random.seed(887851050)
base_theta = sle.theta[0]
```

```
n_forecast = 50
param_samples = []
for _ in range(n_forecast):
    th = dict(base_theta)
    for p in ["R0", "rho", "k"]:
        th[p] = float(base_theta[p]) \
            * np.exp(np.random.normal(0, 0.15))
    param_samples.append(th)
```

Process noise and measurement error

To incorporate **dynamic noise and measurement error** we use a particle filter at each parameter vector, which gives us estimates of both the likelihood and the filter distribution at that parameter value.

```
sle_forecast = pp.Pomp(
    rinit=rinit, rproc=rproc,
    dmeas=dmeas, rmeas=rmeas,
    ys=sle.ys, theta=param_samples,
    t0=sle.t0, dt=timestep, ydim=1,
    covars=None,
    statenames=sle.statenames,
    accumvars=("N_EI", "N_IR"))
```

- We set `theta` to an array of parameters, which will trigger simulations at each parameter vector.

```
cache_file = cache_dir + "/forecast-pfilter.pkl"
if os.path.exists(cache_file):
    with open(cache_file, 'rb') as f:
        pf_cache = pickle.load(f)
    logliks = pf_cache['logliks']
    fm_values = pf_cache['fm_values']
else:
    key = jax.random.key(12345)
    sle_forecast.pfilter(
        J=1000, key=key, reps=5,
        filter_mean=True)
    pf = sle_forecast.results_history.last()

    logliks = np.array([
        float(pp.logmeanexp(
            pf.logLiks.values[j, :]))
        for j in range(n_forecast)])
    fm_values = np.array(pf.filter_mean.values)
    with open(cache_file, 'wb') as f:
        pickle.dump({'logliks': logliks,
                     'fm_values': fm_values}, f)
```

```
print(f"Log-likelihoods range: "
      f"[{logliks.min():.1f}, "
      f"{logliks.max():.1f}]")
```

```
Log-likelihoods range: [-236.0, -77.0]
```

We extract the filtered state means at the end of the data for use as initial conditions for the forecasts.

```
# filter_mean shape: (theta, replicate, time, state)
statenames_list = sle.statenames

# Average over replicates, take last time point
forecast_thetas = []
for j in range(n_forecast):
    th = param_samples[j].copy()
    # Mean filtered state across reps at last time
    mean_state = fm_values[j, :, -1, :].mean(axis=0)
    for k_idx, name in enumerate(statenames_list):
        th[f"{name}_init"] = float(mean_state[k_idx])
    # Reset accumulators
    th["N_EI_init"] = 0.0
    th["N_IR_init"] = 0.0
    forecast_thetas.append(th)
```

We simulate forward from the filtered states to generate forecasts.

- For the forecast, we set `rinit` to copy the previous filter state, saved in the parameter vector.
- We set the initial time `t0` to be the time of the last observation.
- The input observations, `ys`, are needed only for their time index, which will be the time at which the simulations are recorded.
- See the source code for full details.

```
Forecast: 6000 rows, 50 param sets × 10 sims × 12 weeks
```

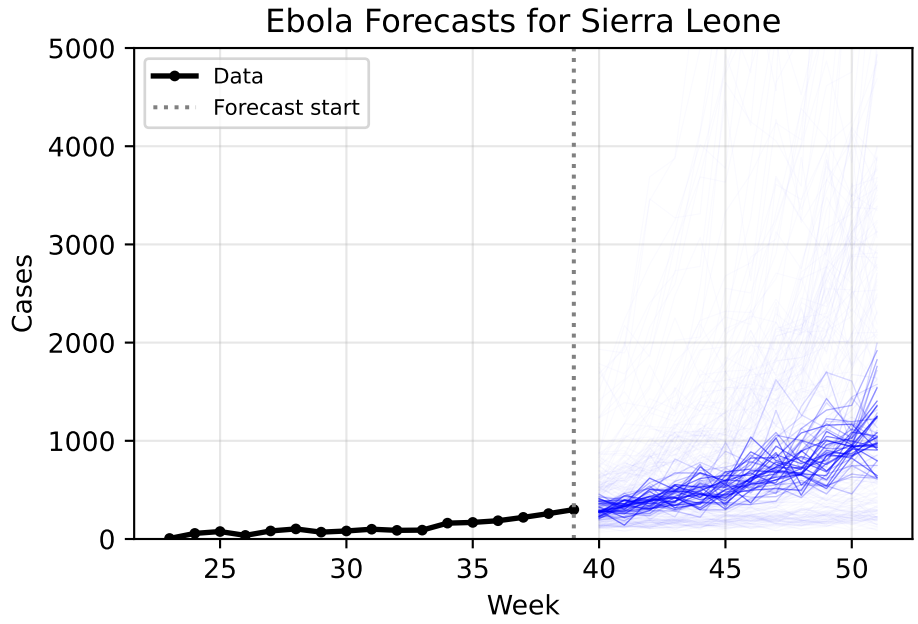
Computing forecast quantiles

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

```
weights = np.exp(logliks - logliks.max())
weights = weights / weights.sum()
```

```
ess = 1.0 / np.sum(weights**2)
print(f"Effective sample size: {ess:.1f}")
```

```
Effective sample size: 6.1
```



Exercises

Exercise 6.1. *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` (`probe` method). Try also to come up with some informative probes of your own. Discuss the implications of your findings.

Exercise 6.2. *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

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 (probes) is essential. We compare data against model simulations using probe statistics.
3. **Forecasting uncertainty** has multiple sources: measurement error, process noise, parametric uncertainty, and structural uncertainty.

4. **Empirical Bayes** approaches allow us to incorporate parameter uncertainty into forecasts by sampling from the high-likelihood region and weighting by likelihood.

License and acknowledgments

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.