Flexible Simulation and Calibration of Compartmental Epidemiological Models

Shreya Mukherjee

ARTPARK, IISc

July 4, 2025

Overview

Motivation

- Study the spread and control of infectious diseases using compartmental models (e.g., SIR, SEIR).
- Epidemiological models help understand transmission dynamics, predict outbreaks, and guide policy.
- Most existing implementations are rigid or hardcoded for specific models.
- Need a general and flexible modeling framework using ODEs.
- Enable experimentation with custom model structures, simulation settings, and inference techniques.
- Support synthetic data generation, noise injection, and parameter fitting for testing robustness.

Project Goals

- Standard models (SIR, SEIR) often lack modularity or flexibility. We develop a configurable and extensible framework to support arbitrary compartmental models via YAML config or class-based definitions.
- Design modular code for loading model definitions, running simulations, injecting noise, and generating plots.
- Support easy addition/removal of vital dynamics and other extensions like waning immunity.
- Calibrate model parameters using synthetic or real epidemiological data. We add synthetic noise to simulate real-world uncertainty and test robustness.
- Fit parameters using deterministic optimizers: BFGS, Nelder–Mead, L-BFGS-B. We implement Bayesian parameter inference via Markov Chain Monte Carlo (MCMC) using `emcee`.
- Provide visual diagnostics such as trajectory plots, corner plots (MCMC), and loss landscapes. It supports partial observation and subsampling of compartments during fitting.

Code Architecture

- **config.yaml**: Defines model structure (compartments, parameters, transitions).
- **model.py**: Contains `Population` and `CompartmentalModel` classes to build ODE systems dynamically.
- **main.py**: Orchestrates the workflow by loading model, simulating, adding noise, and parameter calibration.
- **calibration.py**: Implements parameter fitting using BFGS, L-BFGS-B, Nelder-Mead, and optional MCMC.
- **plotting.py**: Provides plotting utilities for simulations and fitted results.
- **data/**: Stores synthetic or real datasets used for fitting.
- **plots/**: Stores generated output plots and visualizations.
- **CLI interface**: Enabled via `argparse` for flexible command-line execution.

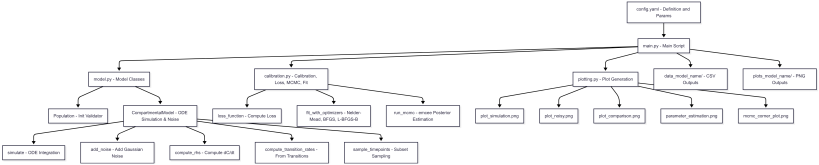
Mermaid Diagram (Concept)

Module-Level Diagram:

- Config $\rightarrow$ Parameters
- model.py $\rightarrow$ Class definitions
- main.py $\rightarrow$ Execution driver
- Output $\rightarrow$ Plot results

Architecture Diagram (Mermaid)

- Shows interaction between config, model logic, and simulation script.
- Highlights modular design: flexible, reusable components.



CompartmentalModel Class

- Parses transition rules from YAML configuration.
- Dynamically constructs and evaluates ODE right-hand side (RHS).
- Simulates compartment trajectories using `scipy.integrate.odeint`.
- Supports time-varying parameters and vital dynamics.
- Easily extendable to new compartments or custom transitions.
- Interoperable with calibration and plotting modules.

Example Transition Definition (YAML)

```
transitions:
  S -> E: beta * S * I / N
  E -> I: sigma * E
  I -> R: gamma * I

with vital dynamics:
     -> S: mu * N
    S -> : mu * S
    E -> : mu * E
    I -> : mu * I
    R -> : mu * R
```

Adding Noise

- Add Gaussian noise to clean simulation data.
- Simulates real-world measurement error.
- Controlled using `noise_std` in config.
- Injected independently into each compartment (e.g., S, I, R).

Noise Function

```python
def add_noise(self, data, std):
    return data + np.random.normal(0, std, data.shape)
```

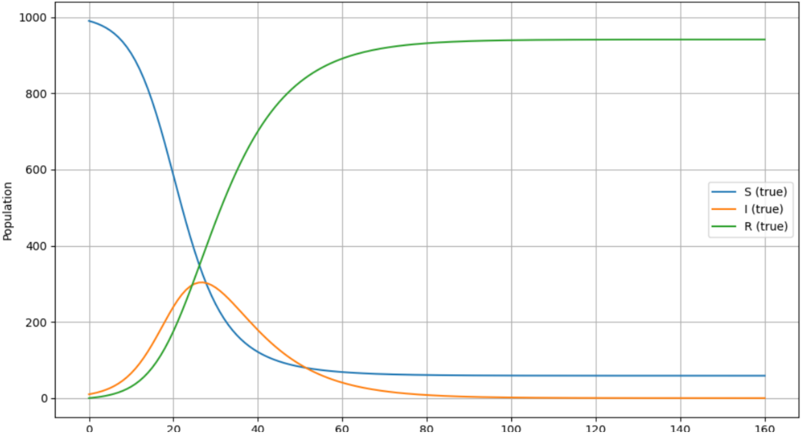
Subset Sampling

- Randomly select time points from noisy data.
- Fit model parameters using only selected subset.
- Mimics limited or sparse real-world data availability.
- Controlled via `subset_fraction` in the config file.

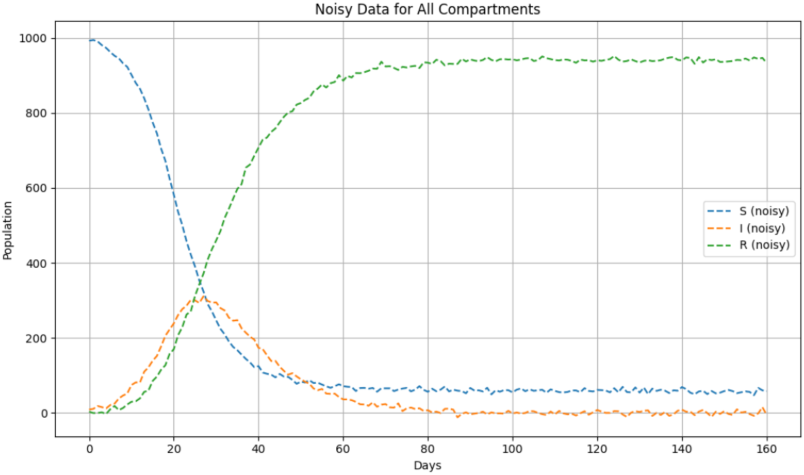
Optimizers Used

- **Nelder-Mead**: Derivative-free.
- **BFGS**: Gradient-based.
- **L-BFGS-B**: BFGS with bounds.
- All optimizers minimize a loss function (e.g., squared error between simulation and data).
- Implemented via `scipy.optimize.minimize()`.

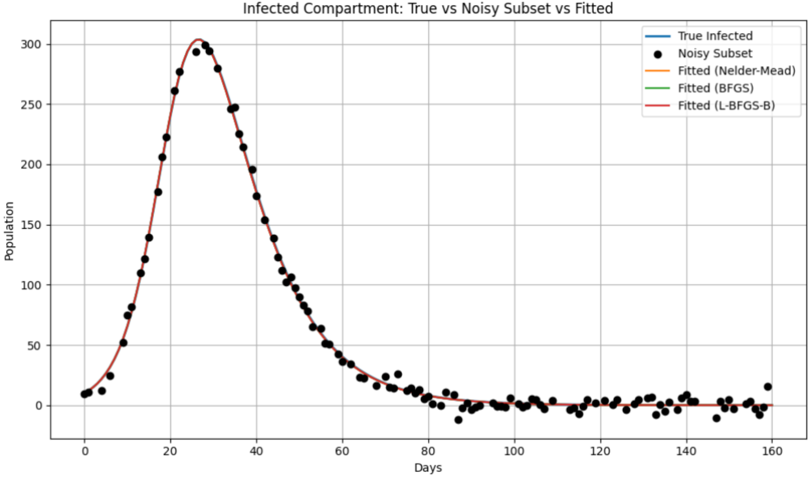
True Simulation (SIR)



Noisy Simulation (SIR)



Fitted vs True (SIR)



Fitted Parameters

- Nelder-Mead: $\hat{\beta} = 0.3008$, $\hat{\gamma} = 0.1003$
- BFGS: $\hat{\beta} = 0.3008$, $\hat{\gamma} = 0.1003$
- L-BFGS-B: $\hat{\beta} = 0.3008$, $\hat{\gamma} = 0.1003$

Conclusion

- Framework allows easy definition and simulation of models.
- Flexible support for noise and fitting.
- Can extend to more complex models or real data.

Future Work

- Add support for time-varying parameters (e.g., seasonality, interventions).
- Integrate real-world datasets for calibration and validation.
- Extend framework to support spatial or network-based epidemic models.

Thank You!