DATA+AI SUMMIT EUROPE

FORMERLY SPARK+AI SUMMIT

Bayesian Modeling of the Temporal Dynamics of COVID-19 using PyMC3

Srijith Rajamohan, Ph.D.

Senior Developer Advocate (Data Science)

#DataTeams #DataAISummit

Agenda

Overview

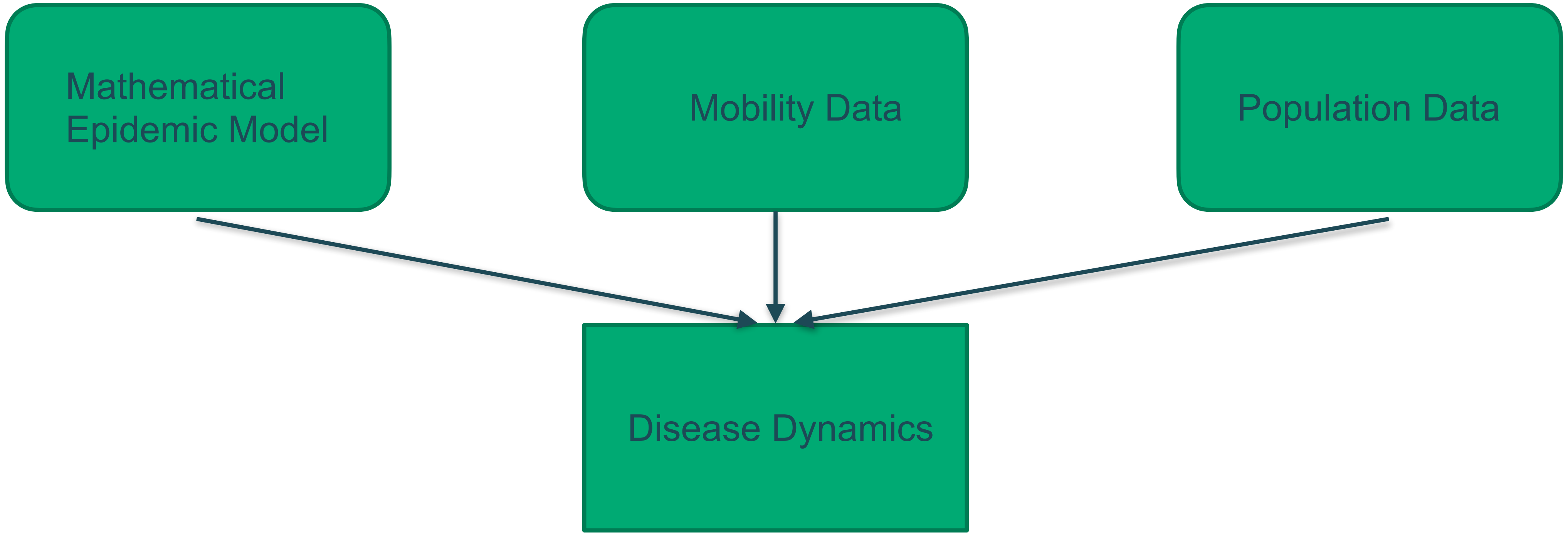
- Compartmental models for COVID-19
- The Data
- The SIR Model
- Bayesian Inference for ODEs with PyMC3
- Inference Workflow on Databricks
- Acknowledgements

Compartmental Models for Temporal Dynamics

Julia notebook – https://github.com/sjster/Epidemic

- Set of Ordinary Differential Equations (ODEs) for closed populations (no movement)
  - Model Disease propagation in homogeneous compartments
  - Fundamental assumptions may not hold in large populations
  - Vital statistics such as the number of births and deaths may not be included here
- Various compartments depicting stages of disease propagation
  - Susceptible Infected Recovered (SIR)
  - Susceptible Infected Recovered Susceptible (SIRS)
  - Susceptible Exposed Infected Recovered (SEIR)
  - Susceptible Exposed Infected Recovered Dead (SEIRD)
  - SIDARTHE (https://www.nature.com/articles/s41591-020-0883-7)

Real-world Epidemic Modeling (Spatio-Temporal Dynamics)



GLEAM

- GLEAM divides the population into grid cells (25km x 25km)
- Models mobility of people in time steps
  - Global mobility - Airports as transportation hubs (Airline traffic data from IATA)
  - Local mobility - Short-range commuting of population between urban centers
- Use stochastic mathematical models to characterize the disease
- Make millions of simulations to make predictions

The Data

- Get the COVID case data from the Johns Hopkins University CSSE Github page
  - https://github.com/CSSEGISandData/COVID-19/tree/master/csse_covid_19_data/csse_covid_19_time_series
- Confirmed cases
  - https://raw.githubusercontent.com/CSSEGISandData/COVID-19/master/csse_covid_19_data/csse_covid_19_time_series/time_series_covid19_confirmed_global.csv
- Number of deaths
  - https://raw.githubusercontent.com/CSSEGISandData/COVID-19/master/csse_covid_19_data/csse_covid_19_time_series/time_series_covid19_deaths_global.csv

The SIR Model

SIR Equations

1. $$\frac{dS}{dt} = -\lambda \frac{SI}{N}$$

2. $$\frac{dI}{dt} = \lambda \frac{SI}{N} - \mu I$$

3. $$\frac{dR}{dt} = f\mu I$$

Outline

- S + I + R = N (Total population)
- S(0), I(0), R(0) are initial conditions
  - I(0) is known
  - S(0) is calculated from above
- λ is the rate of infection
- μ is the rate of recovery
- The fraction of people who recover is 'f' but we set that to 1 here
- We have I(t), which is our observation
- Use Bayesian inference to estimate λ, μ

The SIR Model Parameters

- λ is the disease transmission coefficient
  - This depends on the number of interactions in unit time with infectious people
  - This in turn depends on the number of infectious people in the population
  - λ = contact rate x transmission probability
- The force of infection or risk at any time 't' is defined as $\lambda \frac{I_t}{N}$
- μ is the fraction of recovery that happens in unit time
  - μ^{-1} is hence the mean recovery time
- The 'basic reproduction number' R_0 is the average number of secondary cases produced by a single primary case (Examples https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6002118/)
  - $R_0 = \frac{\lambda}{\mu}$ (assuming S_0 is close to 1), $R_0 > 1$ results in proliferation of the disease
- If we vaccinate a fraction 'p' of the population to get $(1 - p)R_0 < 1$, we can halt the spread of the disease

The SIRS Model

SIRS Equations

1. $$\frac{dS}{dt} = -\lambda\frac{SI}{N} + \gamma R$$

2. $$\frac{dI}{dt} = \lambda\frac{SI}{N} - \mu I$$

3. $$\frac{dR}{dt} = \mu I - \gamma R$$

Outline

- Most likely a better low-fidelity model for COVID-19
- No lifetime immunity from infection
- λ, μ are the same
- γ is the rate at which immunity is lost and the population moves back to the susceptible pool

Temporal discretization of SIR

First order discretization

1. $$(S_t - S_{t-1})/\Delta t = -\lambda \frac{SI}{N}$$

2. $$(I_t - I_{t-1})/\Delta t = \lambda \frac{SI}{N} - \mu I$$

3. $$(R_t - R_{t-1})/\Delta t = \mu I$$

Second order discretization

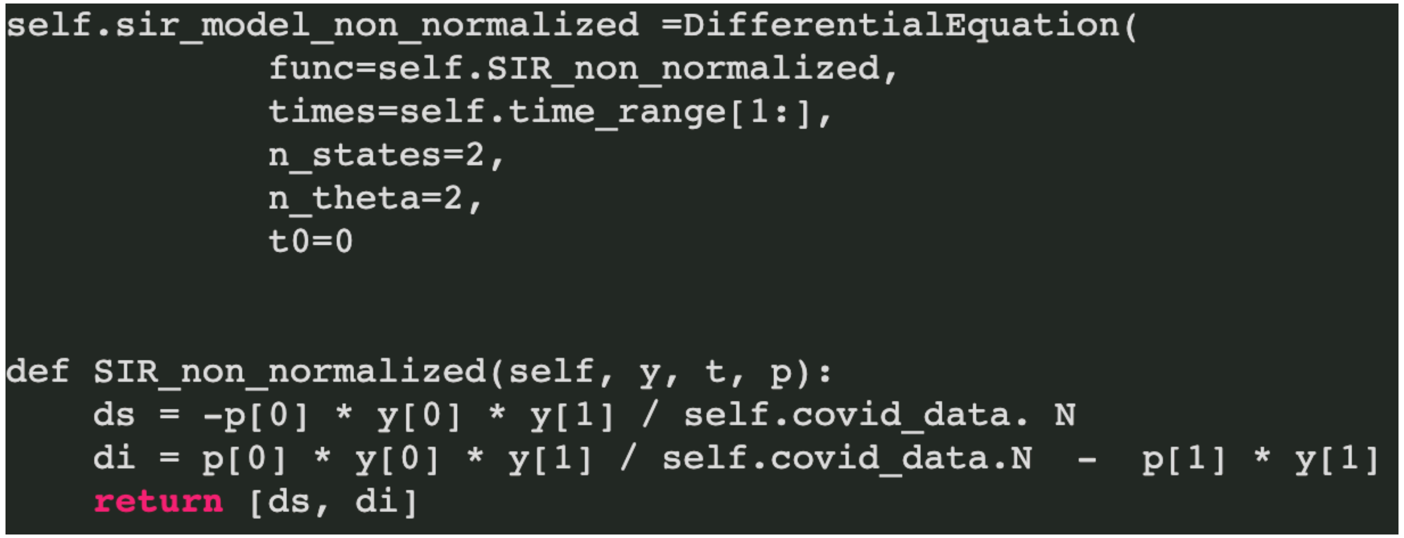
$$S_t = (4 - \frac{2\Delta t \lambda I}{N})\frac{S_{t-1}}{3} - \frac{S_{t-2}}{3}$$

$$I_t = (\frac{2\Delta t \lambda S_{t-1}}{N} - 2\Delta t \mu + 4)I_{t-1} - \frac{I_{t-2}}{3}$$

$$R_t = \frac{2\Delta t \mu I_{t-1} + 4R_{t-1} - R_{t-2}}{3}$$

The DifferentialEquation method in PyMC3

- PyMC3 has an ODE module
- Use the DifferentialEquation method from the ODE module
- Cons: Tends to be slow
- Faster: the ‘sunode’ module in PyMC3
  - E.g. 5.4 mins vs 16s for 100 samples and 100 tuning samples, 20 time points
- No U-Turn Sampler (NUTS)is the default algorithm, Metropolis (not recommended) is faster but less accurate

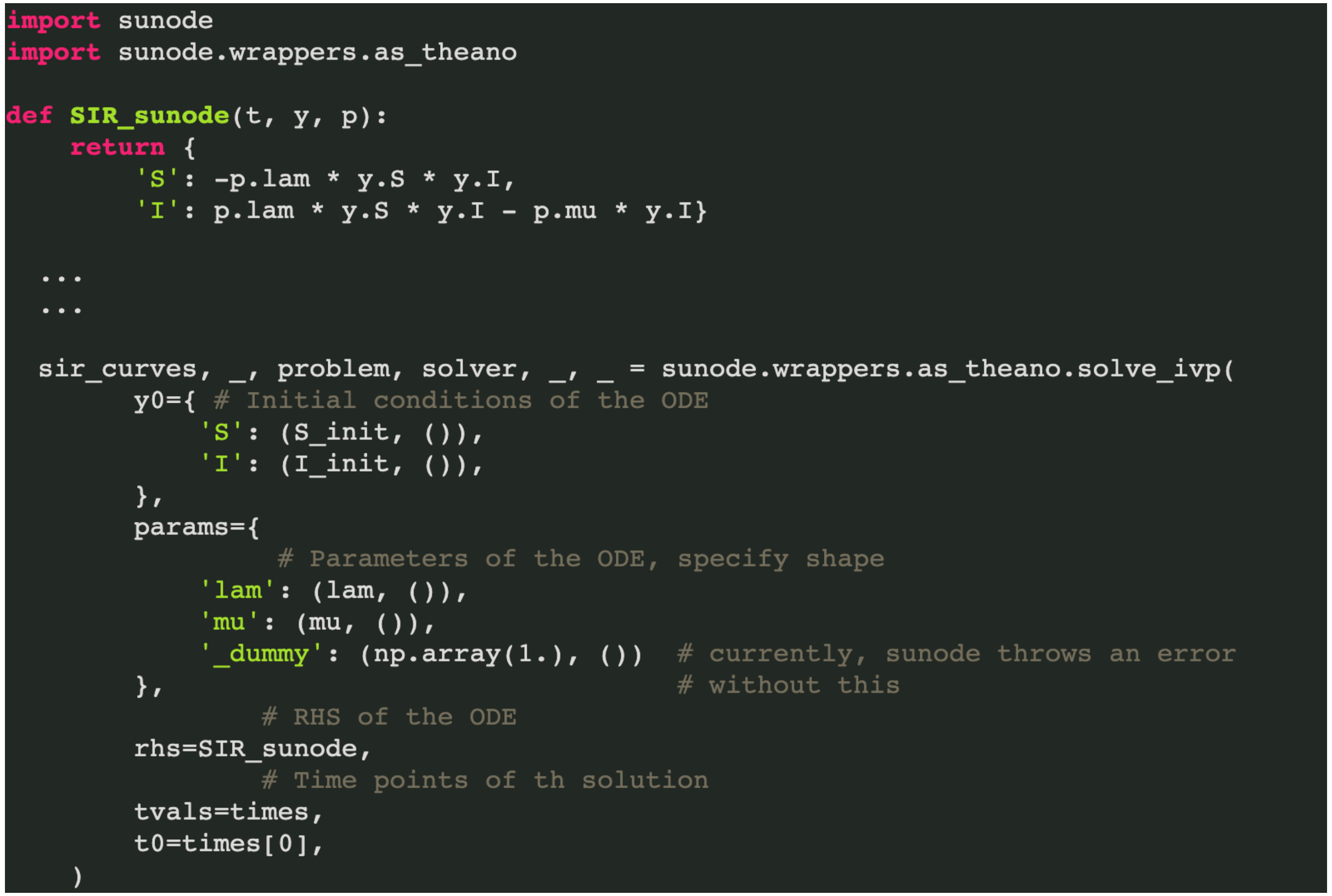


```
self.sir_model_non_normalized =DifferentialEquation(
            func=self.SIR_non_normalized,
            times=self.time_range[1:],
            n_states=2,
            n_theta=2,
            t0=0


def SIR_non_normalized(self, y, t, p):
    ds = -p[0] * y[0] * y[1] / self.covid_data. N
    di = p[0] * y[0] * y[1] / self.covid_data.N  -  p[1] * y[1]
    return [ds, di]
```

Sunode Module for Solving ODEs

RHS

ODE(IVP) Solver

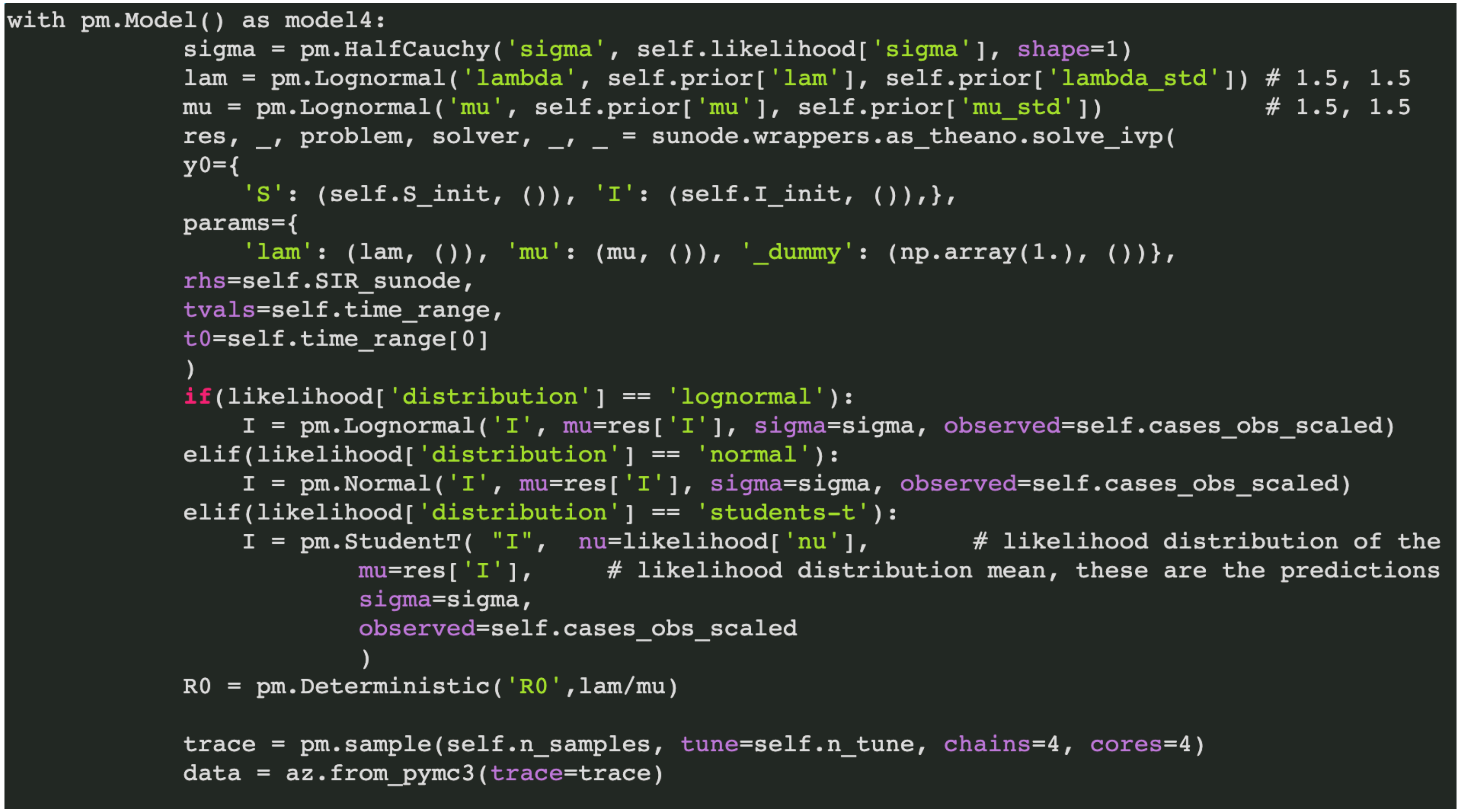


```
import sunode
import sunode.wrappers.as_theano

def SIR_sunode(t, y, p):
    return {
        'S': -p.lam * y.S * y.I,
        'I': p.lam * y.S * y.I - p.mu * y.I}

  ...
  ...

  sir_curves, _, problem, solver, _, _ = sunode.wrappers.as_theano.solve_ivp(
        y0={ # Initial conditions of the ODE
            'S': (S_init, ()),
            'I': (I_init, ()),
        },
        params={
                 # Parameters of the ODE, specify shape
            'lam': (lam, ()),
            'mu': (mu, ()),
            '_dummy': (np.array(1.), ())  # currently, sunode throws an error
        },                                # without this
                # RHS of the ODE
        rhs=SIR_sunode,
                # Time points of th solution
        tvals=times,
        t0=times[0],
    )
```

The Inference Process for an SIR model

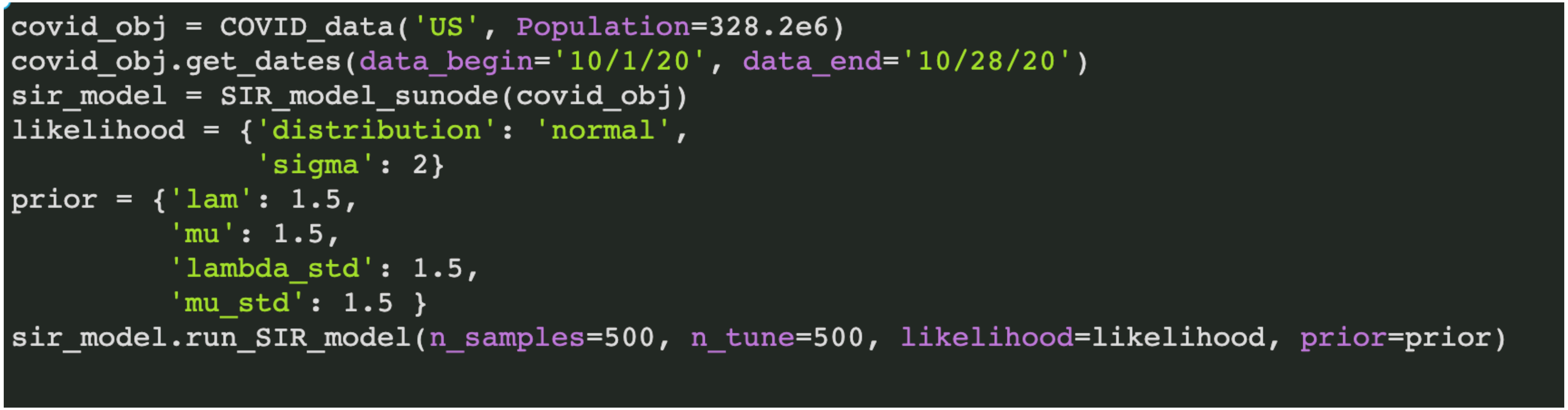
- Select reasonable priors for the λ, μ disease parameters
  - Lognormal is a reasonable prior
  - Mean parameter should be close to what we expect these parameters to be
- Data likelihood should have high fidelity (domain expertise!)
  - Normal distribution
  - Lognormal distribution
  - Student's t-distribution
- Get Susceptible (S) and Infectious (I) numbers from the ODE solver
- Sample for values of λ, μ

Inference with PyMC3



```
with pm.Model() as model4:
            sigma = pm.HalfCauchy('sigma', self.likelihood['sigma'], shape=1)
            lam = pm.Lognormal('lambda', self.prior['lam'], self.prior['lambda_std']) # 1.5, 1.5
            mu = pm.Lognormal('mu', self.prior['mu'], self.prior['mu_std'])           # 1.5, 1.5
            res, _, problem, solver, _, _ = sunode.wrappers.as_theano.solve_ivp(
            y0={
                'S': (self.S_init, ()), 'I': (self.I_init, ()),},
            params={
                'lam': (lam, ()), 'mu': (mu, ()), '_dummy': (np.array(1.), ())},
            rhs=self.SIR_sunode,
            tvals=self.time_range,
            t0=self.time_range[0]
            )
            if(likelihood['distribution'] == 'lognormal'):
                I = pm.Lognormal('I', mu=res['I'], sigma=sigma, observed=self.cases_obs_scaled)
            elif(likelihood['distribution'] == 'normal'):
                I = pm.Normal('I', mu=res['I'], sigma=sigma, observed=self.cases_obs_scaled)
            elif(likelihood['distribution'] == 'students-t'):
                I = pm.StudentT( "I",  nu=likelihood['nu'],       # likelihood distribution of the
                        mu=res['I'],     # likelihood distribution mean, these are the predictions
                        sigma=sigma,
                        observed=self.cases_obs_scaled
                        )
            R0 = pm.Deterministic('R0',lam/mu)

            trace = pm.sample(self.n_samples, tune=self.n_tune, chains=4, cores=4)
            data = az.from_pymc3(trace=trace)
```

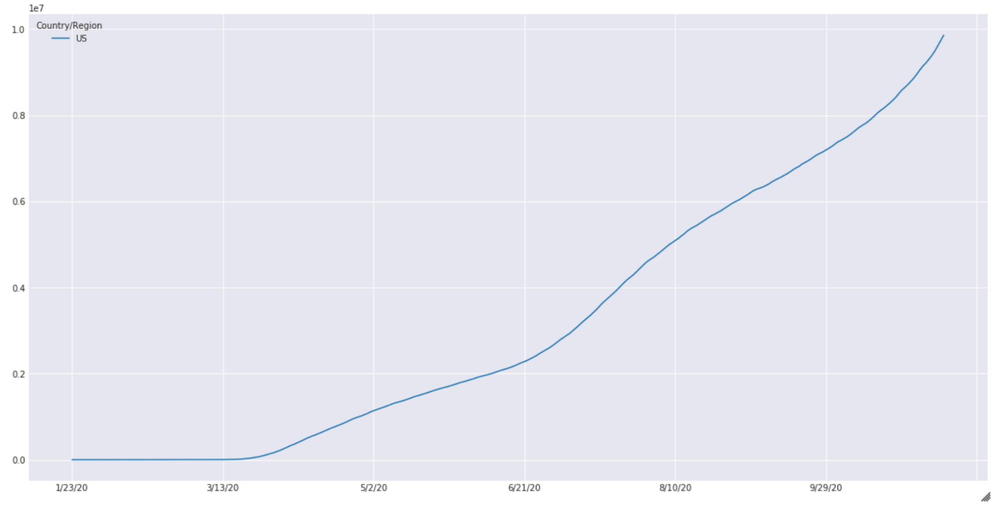
Inference Workflow on Databricks



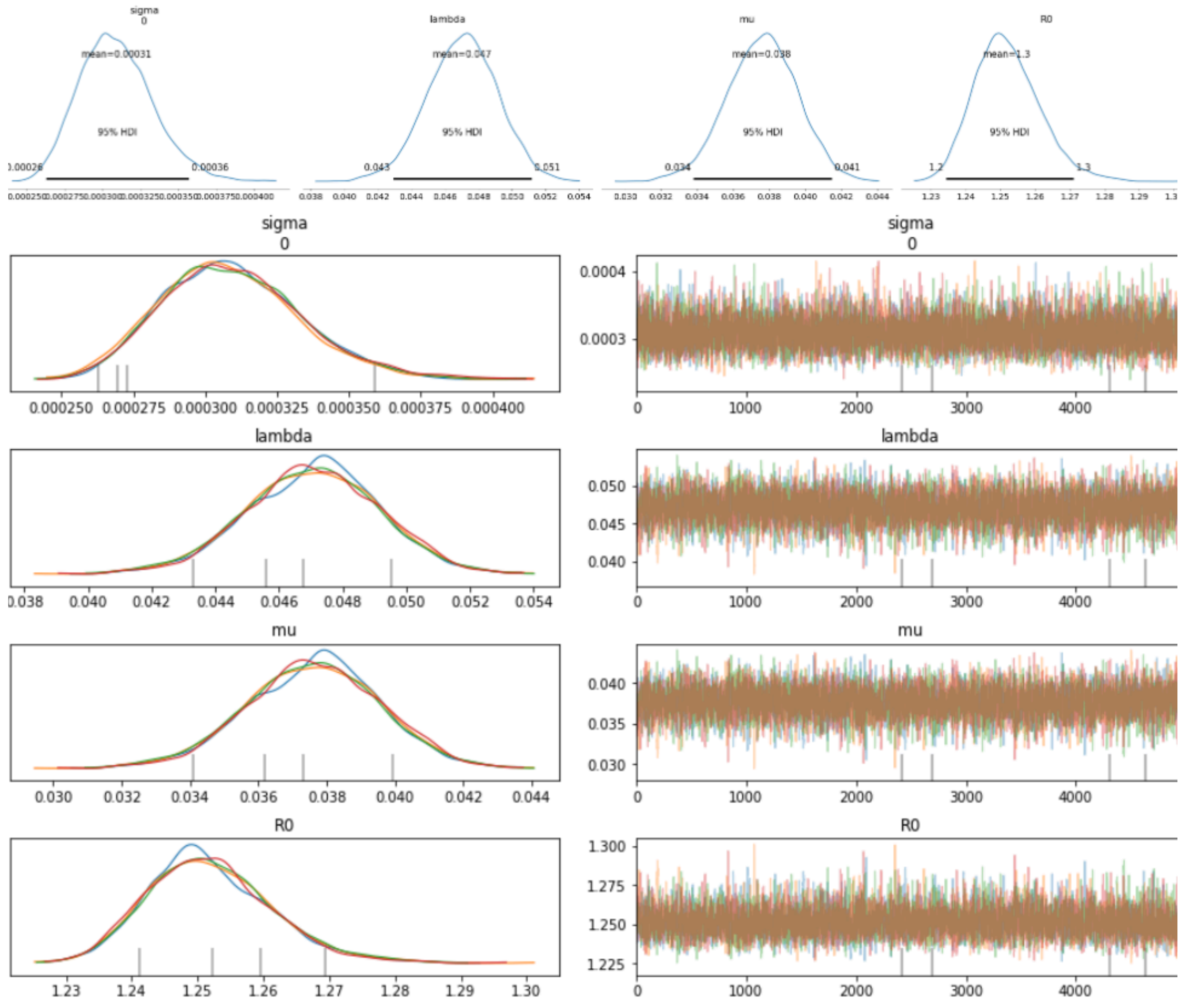
```
covid_obj = COVID_data('US', Population=328.2e6)
covid_obj.get_dates(data_begin='10/1/20', data_end='10/28/20')
sir_model = SIR_model_sunode(covid_obj)
likelihood = {'distribution': 'normal',
              'sigma': 2}
prior = {'lam': 1.5,
         'mu': 1.5,
         'lambda_std': 1.5,
         'mu_std': 1.5 }
sir_model.run_SIR_model(n_samples=500, n_tune=500, likelihood=likelihood, prior=prior)
```

Automate this process on Databricks for a number of combination of parameters and priors

Inference with PyMC3



Number of Infections in the US



Posterior with Highest Density Interval

Notes

Some guidelines

- At least 5000 samples and 1000 samples for tuning
- Lambda of 1.5 and mu of 1.5
- Sigma of 2
- Sample from 3 chains at least
- Set 'target_accept' to > 0.85
- Sample in parallel with cores=n
- Inspect trace for convergence
- Limited time-samples have an impact on inference accuracy
- Normalize your data – large values are not good for convergence

Debugging your model

- theano.printing.Print(STRING)(VAR)
- Pass 'testval' as a test value to stochastic variables
- Model.check_test_point()
- step = pm.Metropolis() for quick debugging – rougher posterior but much faster
- If the sampling is slow, check your prior and likelihood distributions

Acknowledgements

- The work by the Priesemann Group
  - https://github.com/Priesemann-Group/covid_bayesian_mcmc
- Demetri Pananos work on the PyMC3 page
  - https://docs.pymc.io/notebooks/ODE_API_introduction.html

Thank you!