week8

Working on estimation of parameters in compartementalized state space models using Hamiltonian Monte Carlo. Starting with a simple SEIR model, we will use the NUTS algorithm to estimate the parameters of the model.

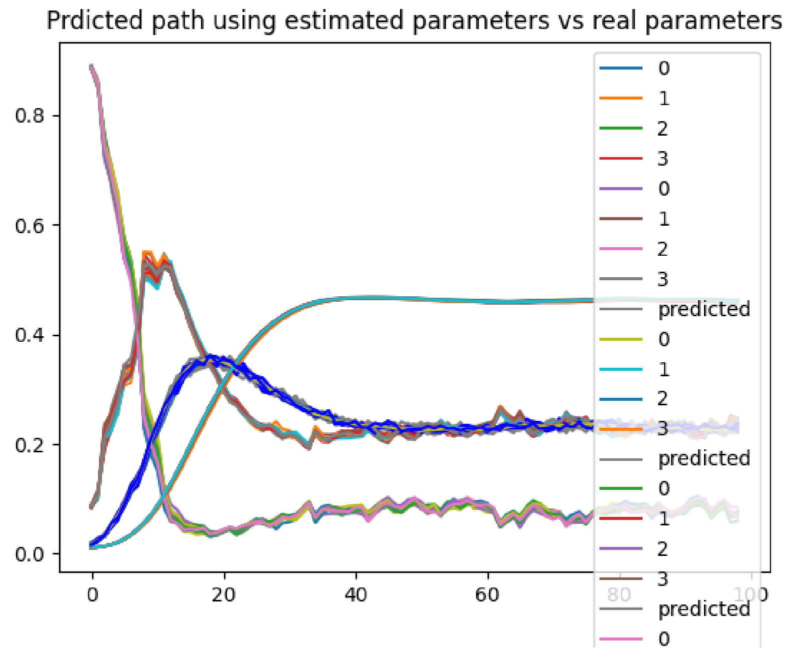
Goals for this week: - Run the model with ClimateHealthTimeSeries data (X) - Inlcude Mosquito compartments in the model - Use data from multiple locations - Use weather data on finer resolution than health data (i.e. daily weather data and monthly health data)

test_estimate_single_parameter

Samples data from a simple SEIR model and estimates the beta parameter using NUTS. Includes a time varying temperature parameter.

```
real_params = {'beta': 0.5}
sample, log_prob, reconstruct_state = simple_model()
T = 100
temperature = np.random.normal(1, 1, T - 1)
observed = sample(T, jax.random.PRNGKey(100), real_params, temperature)
init_diffs = np.random.normal(0, 1, (T - 1))
init_dict = {'logits_array': init_diffs} | {'beta': 0.0}
lp = log_prob(observed, temperature)
samples = nuts_sample(lp, jax.random.PRNGKey(0),
                      init_dict, 100, 100)

fig = model_evaluation_plot(sample, real_params, samples, temperature)
return show(fig)
```



test_run_with_climate_health_data

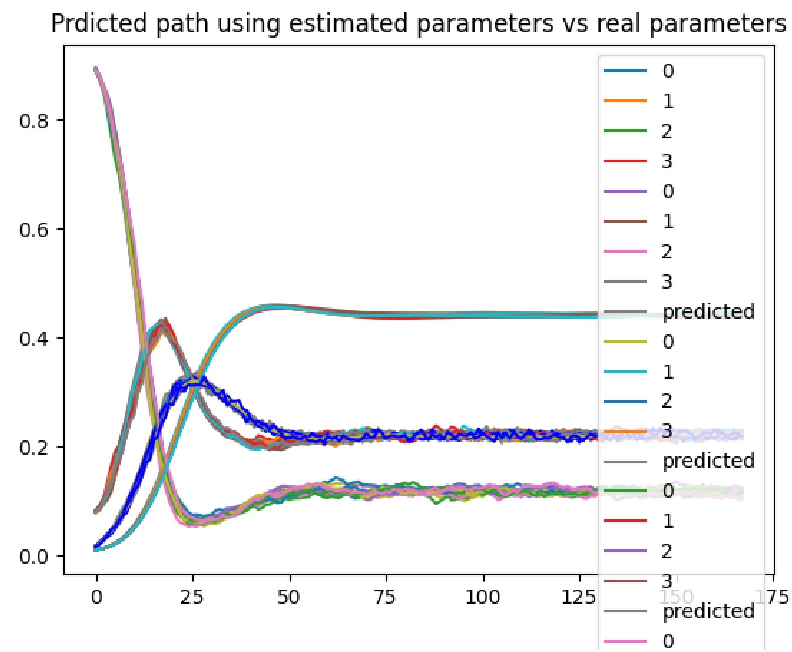
Runs the model with ClimateHealthTimeSeries data - Load climate data from file - Simulate health data using model - Run model on climatehealth data set - Plot model evaluation

```
climate_data = ClimateData.from_csv(EXAMPLE_DATA_PATH / 'climate_data.csv')
T = len(climate_data) + 1
sample, log_prob, reconstruct_state = simple_model()
real_params = {'beta': 0.001}

class Simulator:
    def simulate(self, climate_data: ClimateData):
        samples = sample(T, jax.random.PRNGKey(0), real_params, climate_data.max_temperature
        return HealthData(time_period=climate_data.time_period, disease_cases=samples)

health_data = Simulator().simulate(climate_data)
init_diffs = np.random.normal(0, 1, (T - 1))
init_dict = {'logits_array': init_diffs} | {'beta': 0.0}
lp = log_prob(health_data.disease_cases, climate_data.max_temperature)
samples = nuts_sample(lp, jax.random.PRNGKey(0),
                      init_dict, 50, 500)
fig = model_evaluation_plot(sample, real_params, samples, climate_data.max_temperature)
```

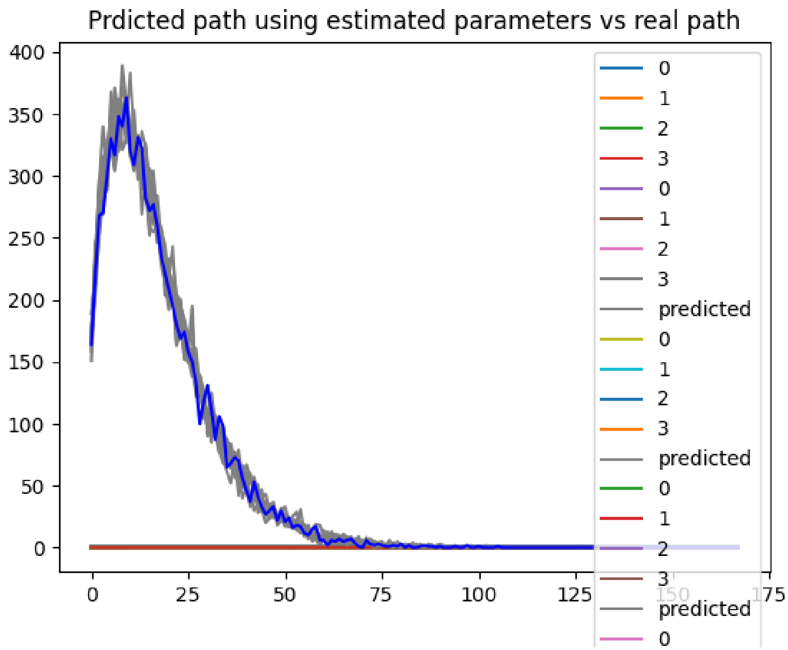
```python
return show(fig)
```



test_simplified_interface

Automate boilerplate code

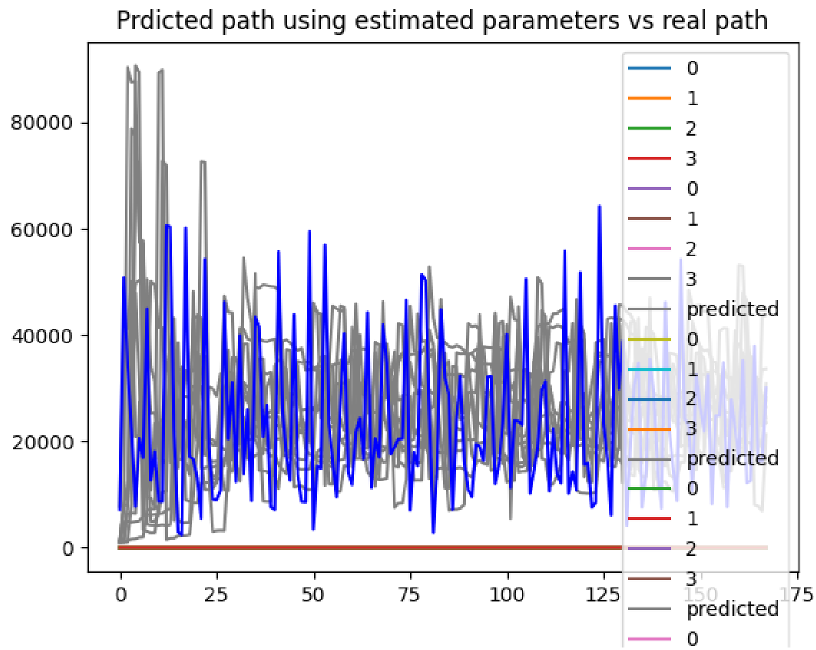
```python
model = lambda: (simple_model(), (['beta'], None))
figure = check_model_capacity(model)
return show(figure)
```



test_mored_advanced_model

Check model_capacity for full SEIR model. Needs more warmup samples to converge. For parameter sets without internal equilibrium it might not converge properly. That is not an issue When introducing priors should have favouring an internal equilibrium.

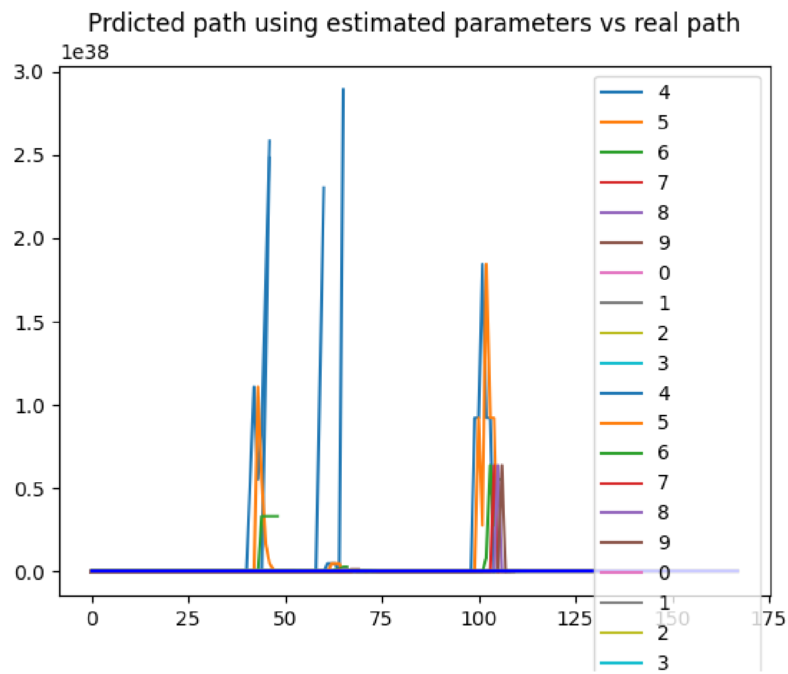
```
model = seir_model
return show(check_model_capacity(model, n_warmup_samples=500))
```



test_mosquito_human_model

Check model capacity when mosquito populations are mediators from weather to disease Seems to be some issues with the identifiability Checking subcases

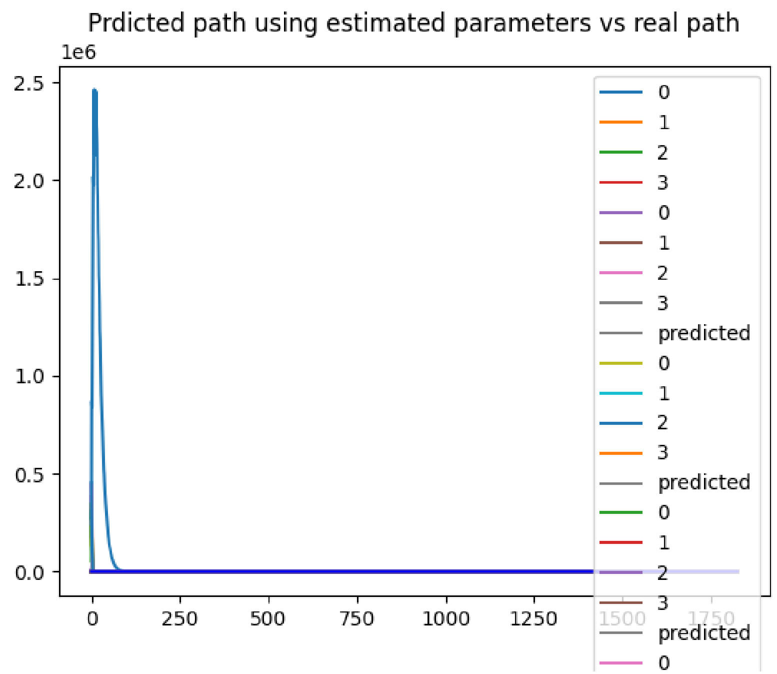
```
model = mosquito_model
return show(check_model_capacity(model))
```



test__mosquito__model

Simple mosquito model dependent on weather data. Does not converge since depdending on paramters population can explode or die out, where the probability of observing the scenario from one gets to be == 0 on the computer. Need to restrict the parameter space sufficiently to ensure convergence.

```
fig = check_model_capacity(pure_mosquito_model, n_warmup_samples=10,
                           data_filename=EXAMPLE_DATA_PATH / 'climate_data_daily.csv')
return show(fig)
```



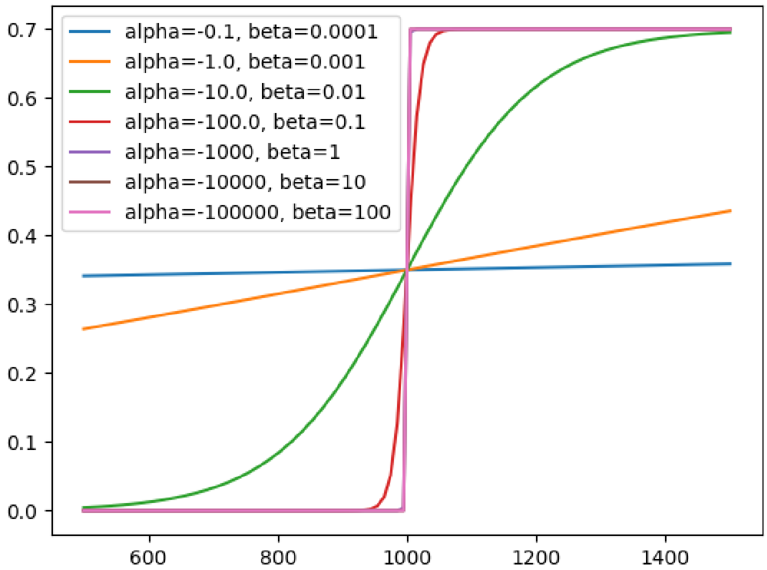
test_investigate_carrying_capacity_function

Death rate is modelled as a function of the population size. Need to find good parameters for that model. Trying to find a parametrization that gives a cc of 1000.

a+bx = 0 a+b*1000 = 0 a = -1000b

Looks like alpha=-10, beta=0.01 seems good

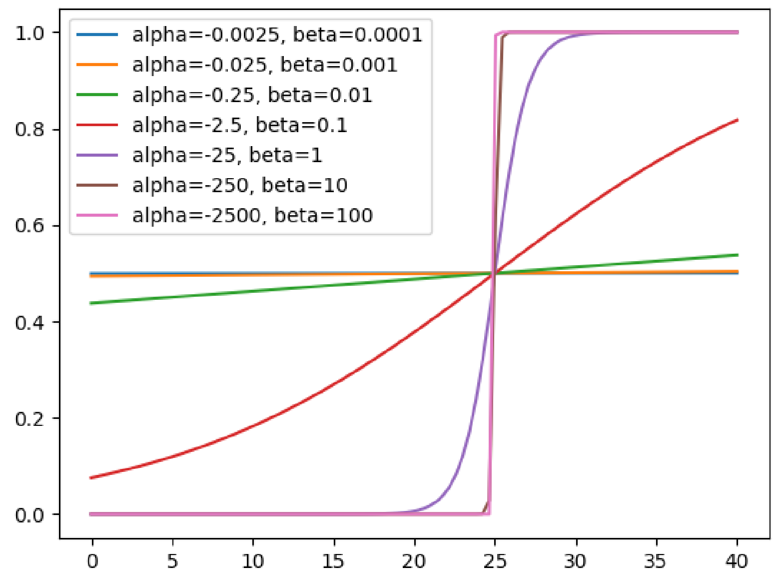
```
x = np.linspace(500, 1500, 100)
for beta in [0.0001, 0.001, 0.01, 0.1, 1, 10, 100]:
    alpha = -1000 * beta
    death_rates = get_death_rate(alpha, beta, x)
    plt.plot(x, death_rates, label=f'alpha={alpha}, beta={beta}')
plt.legend()
return show(plt.gcf())
```



test_investigate_temperature_dependency

Investigate the temperature dependency of the mosquito population. We want the maturation rate to rize with temperature around 20-30 degrees. alpha+beta*25 = 0 alpha = -beta*25

```
x = np.linspace(0, 40, 100)
for beta in [0.0001, 0.001, 0.01, 0.1, 1, 10, 100]:
    alpha = -25 * beta
    maturation_rate = get_maturation_rate_by_temp(dict(temp_base=alpha, temp_dependency=beta
    plt.plot(x, maturation_rate, label=f'alpha={alpha}, beta={beta}')
plt.legend()
return show(plt.gcf())
```



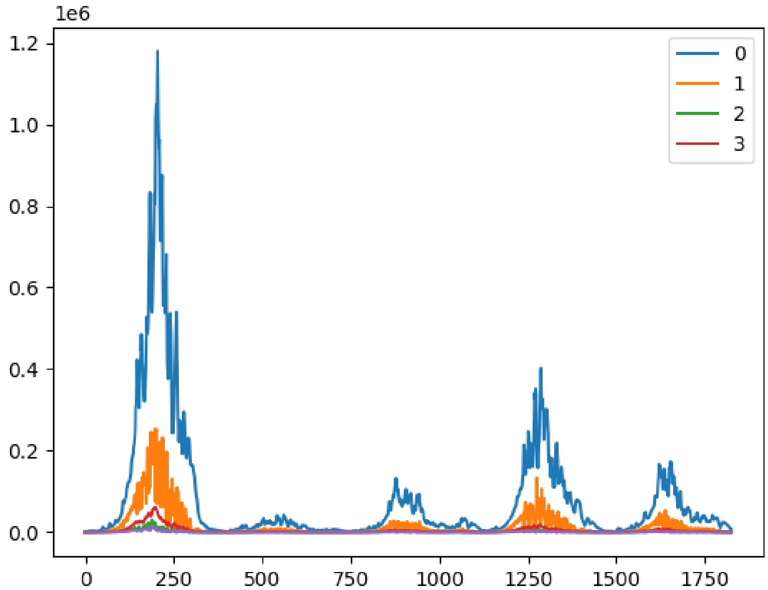
test__explore__parameter__space__for__mosquito

Set the parameters to reasonable values and check if we get a stable model. This was possible with centrally parameterized model, so need to find a similar set of parameters for the non-central model.

```
(sample, log_prob, reconstruct_state), (param_names, n_states) = pure_mosquito_model()
climate_data = ClimateData.from_csv(EXAMPLE_DATA_PATH / 'climate_data_daily.csv')[:365 * 5]
real_params = {
    'temp_base': -30.,
    'temp_dependency': 1.,
    'lo_pupae_maturation': logit(0.33),
    'logscale': np.log(0.1),
    'mosquito_death_logit': logit(0.1),
    'carry_beta': 0.01,  # Verified
    'carry_alpha': -10,  # Verified
    'log_eggrate': jnp.log(10),
    'lo_rate': 0.
}

simulator = get_simulator(sample, real_params)
mosquito_data = simulator.simulate(climate_data)
plt.plot(mosquito_data.disease_cases)
```

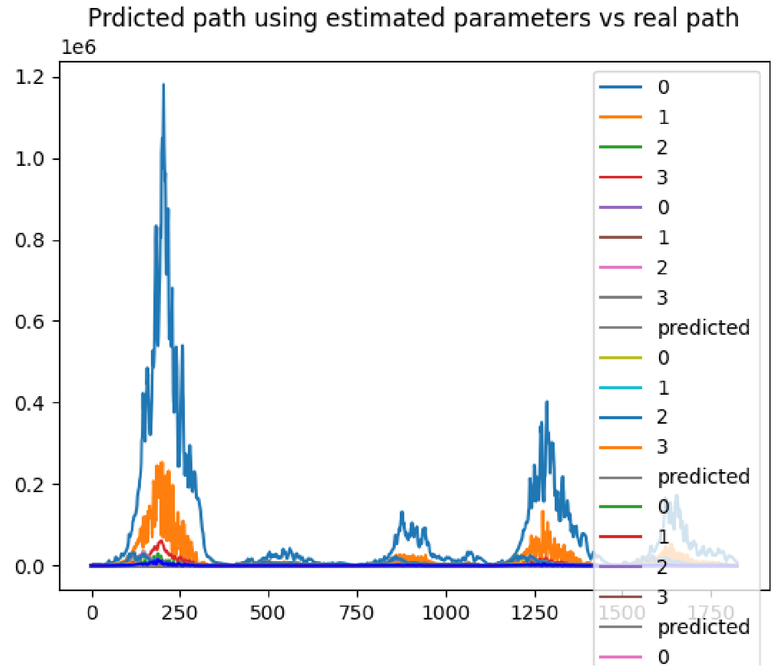
```
return show(plt.gcf())
```



test_estimate_good_mosquito_parameters

Try estimating parameters for a well-behaved mosquito model. For now this works when the parameters are set to reasonable values (i.e. true ones). It is worth to note that it still takes time to converge since the initial states proposed are not good (i.e. they are not likely states given the parameters). Maybe it's a good strategy to use the initial parameters to propose initial states. It is also quite slow now so might need some speedups before applying to the real data.

```
mosquito_data, climate_data, real_params = get_parameterized_mosquito_model()
(sample, log_prob, reconstruct_state), (param_names, n_states) = pure_mosquito_model()
health_data = mosquito_data
sampler = SimpleSampler(jax.random.PRNGKey(0),
                        log_prob, sample,
                        param_names, n_states,
                        n_warmup_samples=500)
data_set = ClimateHealthTimeSeries.combine(health_data, climate_data)
sampler.train(data_set, init_values=real_params)
fig = prediction_plot(health_data, sampler, climate_data, 10)
return show(fig)
# diag_plot = sampler.diagnostic_plot(real_params)
# return show(diag_plot)
```



Prdicted path using estimated parameters vs real path
1e6
1.2
1.0
0.8
0.6
0.4
0.2
0.0
0
250
500
750
1000
1250
1500
1750
0
1
2
3
0
1
2
3
predicted
0
1
2
3
predicted
0
1
2
3
predicted
0