week9

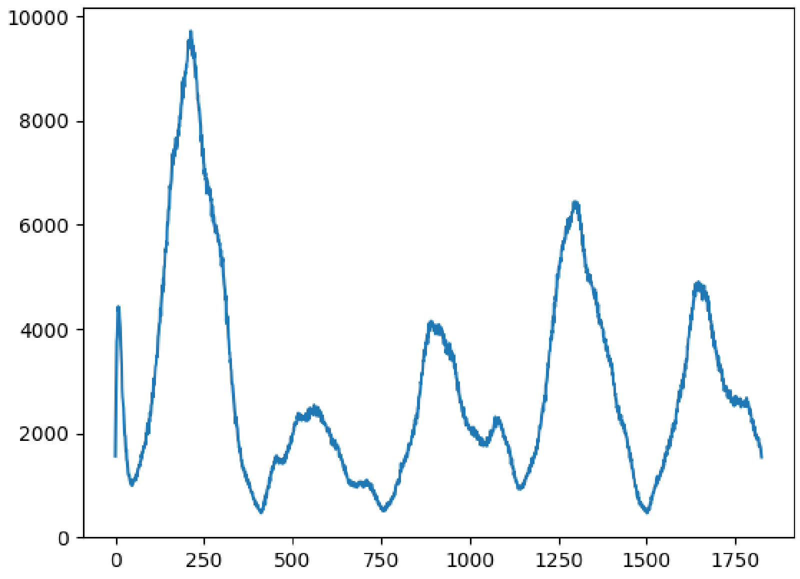
Should get the mosquito model incorporated into the human model. With that and different scale for health and climate data, it is ready to use on real data.

- 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_human_mosquito_model

Find a good parameter space for human/mosquito model

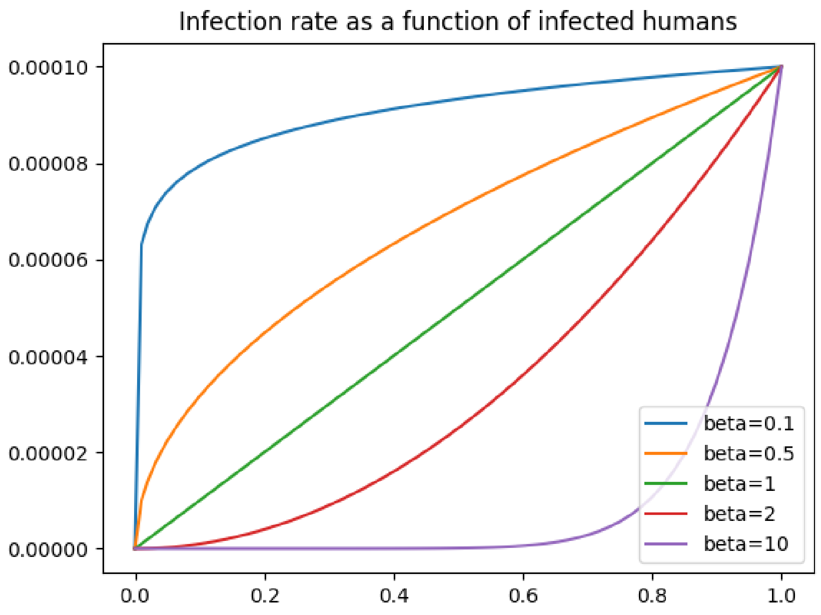
```
(sample, log_prob, reconstruct_state, sample_diffs), (param_names, n_states) = full_mode
print(param_names)
climate_data = ClimateData.from_csv(EXAMPLE_DATA_PATH / 'climate_data_daily.csv')[:365 
simulator = get_simulator(sample, param_names)
mosquito_data = simulator.simulate(climate_data)
plt.plot(mosquito_data.disease_cases)
return show(plt.gcf())
```



test__investigate__infection__rate__function

Should be quite low for 0 infected humans and rise approximately linearly with the number of infected humans.

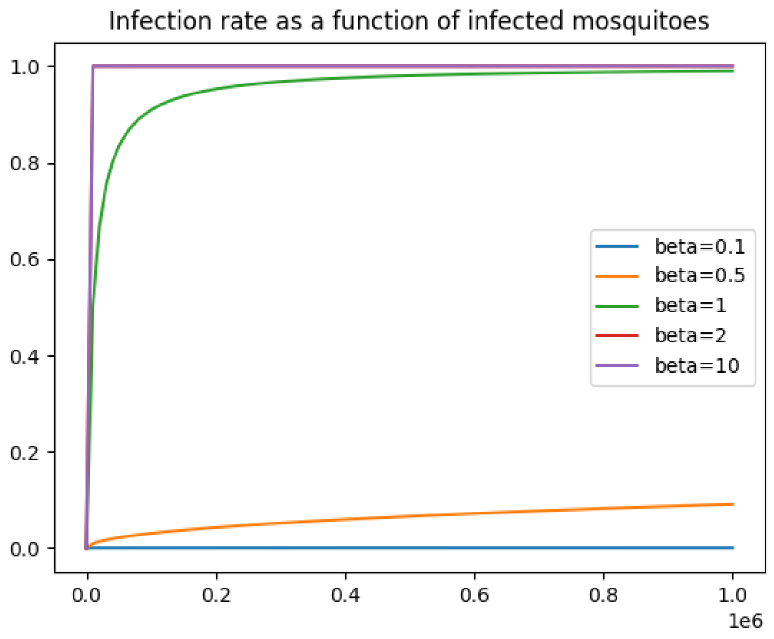
```
alpha = logit(0.0001)
x = np.linspace(0, 1, 100)
for beta in [0.1, 0.5, 1, 2, 10]:
    plt.plot(x, expit(alpha + beta * np.log(x)), label=f'beta={beta}')
plt.legend()
plt.title('Infection rate as a function of infected humans')
return show(plt.gcf())
```



test__ivestigate__human__inection__rate__function

Should be quite low for 0 infected mosquitoes and rise approximately linearly with the number of infected mosquitoes.

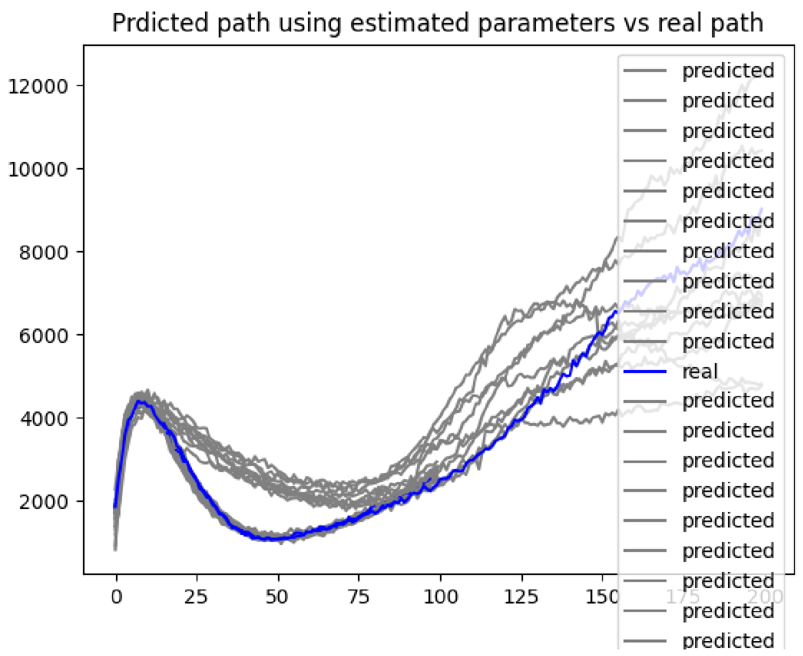
```
alpha = logit(0.0001)
x = np.linspace(0, 1000000, 100)
for beta in [0.1, 0.5, 1, 2, 10]:
    plt.plot(x, expit(alpha + beta * np.log(x)), label=f'beta={beta}')
plt.legend()
plt.title('Infection rate as a function of infected mosquitoes')
return show(plt.gcf())
```



test_human_mosquito_model_estimation

Try to estimate parameters for a well parameterized full model. This is not really estimating since we feed the true parameters to the model, but it is a good test of the capacity of the model to fit the data.

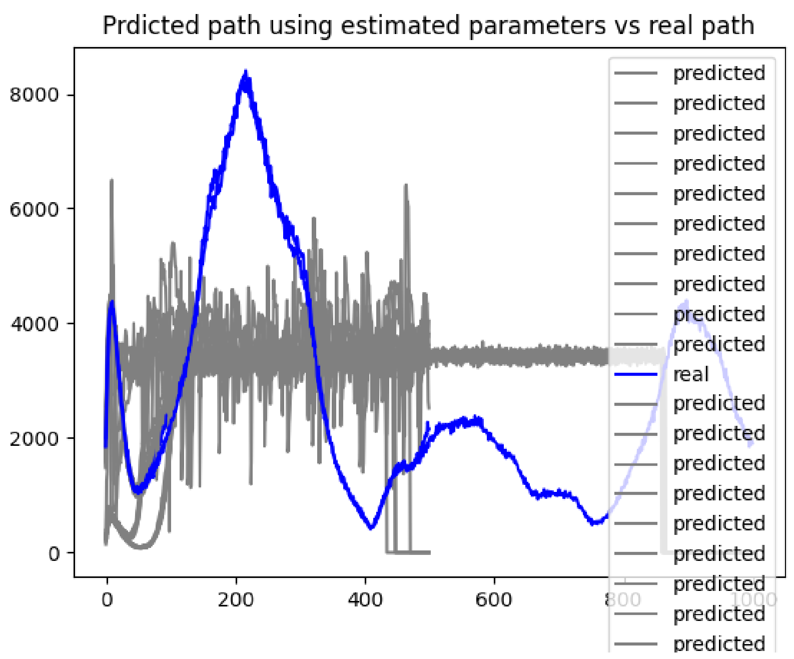
```
(sample, log_prob, reconstruct_state, sample_diffs), (real_params, n_states) = full_model()
for T in [100, 200]:
    fig = evaluate_human_mosq_model(log_prob, n_states, real_params, sample, sample_diffs, T
return show(fig)
```



test_human_mosquito_model_estimation_more_Ts

Try to estimate parameters for a well parameterized full model. This is not really estimating since we feed the true parameters to the model, but it is a good test of the capacity of the model to fit the data. It seems that for large T are some numerical issues. The dependency on the first diff might be too big (too many steps), and gradient calculation becomes too unstable. Maybe we should break the dependencies by sampling actual states instead of diffs at fixed time intervals.

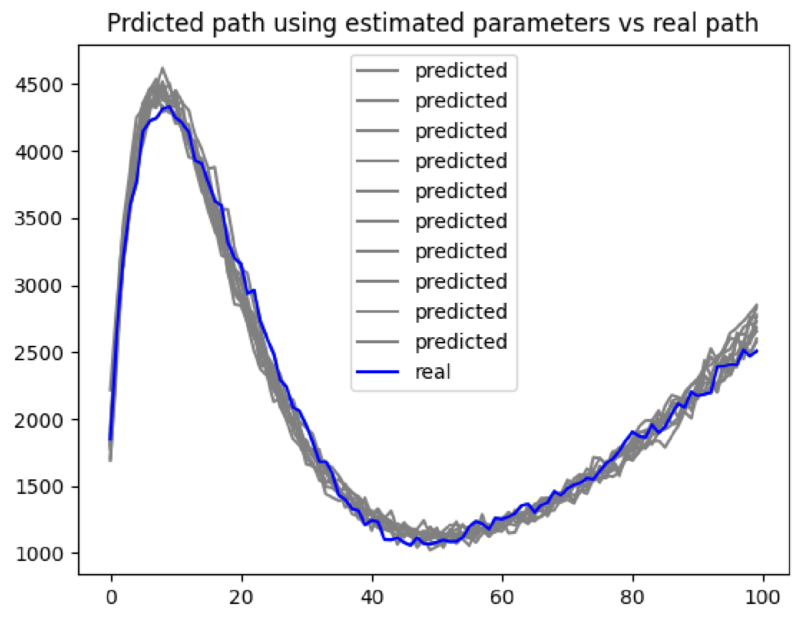
```
(sample, log_prob, reconstruct_state, sample_diffs), (real_params, n_states) = full_model()
for T in [10, 100, 500, 1000]:
    fig = evaluate_human_mosq_model(log_prob, n_states, real_params, sample, sample_diffs, T
return show(fig)
```



test_refactor_mosquito_model

Needs some refactoring before making the hybrid model. Time to use classes

```
T = 100
model_spec = MosquitoModelSpec
model = DiffModel(model_spec)
simulator = SimpleSimulator.from_model(model)
climate_data = ClimateData.from_csv(EXAMPLE_DATA_PATH / 'climate_data_daily.csv')[:T]
health_data = simulator.simulate(climate_data)
plt.plot(health_data.disease_cases);
plt.show()
data_set = ClimateHealthTimeSeries.combine(health_data, climate_data)
sampler = SimpleSampler.from_model(model, jax.random.PRNGKey(0))
sampler.train(data_set,
              init_values=model_spec.good_params | {
                  'init_diffs': model.sample_diffs(transition_key=jax.random.PRNGKey(10000)
                                                   params=model_spec.good_params,
                                                   exogenous=climate_data.max_temperature)}
return show(prediction_plot(health_data, sampler, climate_data, 10))
```



test__scan__with__fixed__values

Find out how to use scan over 2d array with init values

```
init_values = jnp.array([0.9, 0.7, 0.2])
diffs = jnp.array([[0.1, 0.3, 0.1], [0.2, 0.1, 0.1], [0.1, 0.1, 0.1]])

def transition(state, diff):
    new_state = state + diff
    return new_state, new_state

val = jax.lax.scan(transition, init_values, diffs)[1]
print(val)
```

test__scan__with__fixed__values__3d

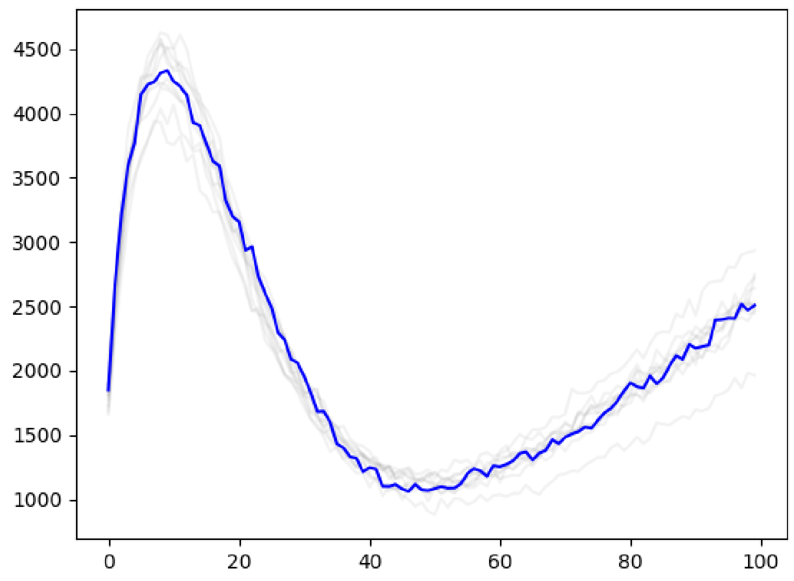
Find out how to use scan over 2d array with init values, n_states=2, n_fixed=3, T=9

```
init_values = jnp.array([[0.9, 0.1],
                         [0.7, 0.1],
                         [0.7, 0.2]])
diffs = jnp.array([[[0.1, 0.2], [0.3, 0.4], [0.1, 0.3], [0.5, 0.5]],
                   [[0.2, 0.2], [0.1, 0.1], [0.1, 0.1], [0.5, 0.5]],
                   [[0.1, 0.1], [0.1, 0.1], [0.1, 0.1], [0.5, 0.5]]])
diffs = jnp.swapaxes(diffs, 0, 1)

def transition(state, diff):
    new_state = state + diff
    return new_state, new_state

val = jax.lax.scan(transition, init_values, diffs)[1]
val = jnp.swapaxes(val, 0, 1).reshape(-1, init_values.shape[-1])
print(val)
```

test__hybrid__central__noncentral__model

This seems to maybe work, but requires alot of warmup samples to get the right parameters. Might need to speed up the sampling, but promising results.

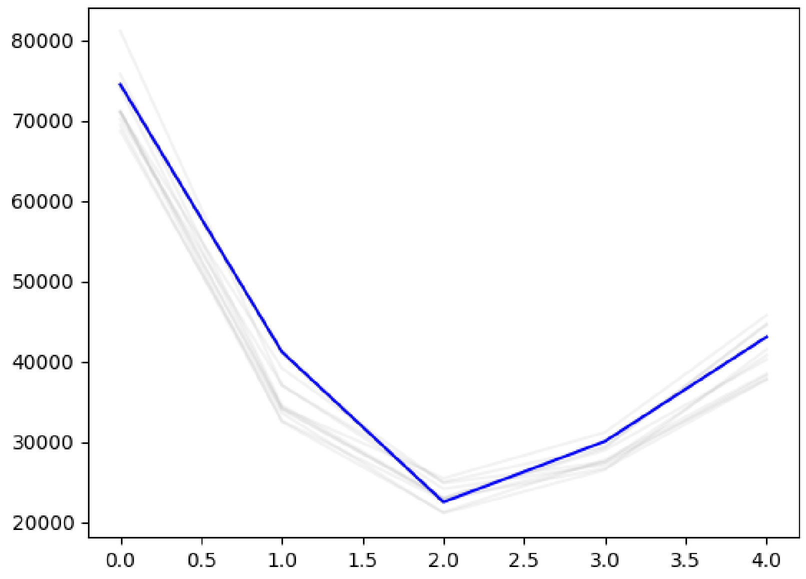
```
    return check_hybrid_model_capacity(T=100, n_warmup_samples=200, n_samples=100)
    # (sample, log_prob, reconstruct_state, sample_diffs), (real_params, n_states) = simple_
```



test__multilevel__model

Test functionality with monthly disease observations and daily weather data

```
T = 100
return check_hybrid_model_capacity(T=T, periods_lengths=jnp.full(T//20, 20), n_warmup_sample
```



test_speedup_transitions

Check if we can get significant speedup by leveraging jax.jit: Did not manage to speed it up. Will try to reparameterize ot make state reconstruction faster.

```
T = 400
model_spec = MosquitoModelSpec
model = HybridModel(model_spec)
climate_data = ClimateData.from_csv(EXAMPLE_DATA_PATH / 'climate_data_daily.csv')[:T]
diffs = model.sample_diffs(transition_key=jax.random.PRNGKey(10000),
                           params=model_spec.good_params,
                           exogenous=climate_data.max_temperature)
transformed_states = jnp.array([model_spec.init_state] * (T // 100))
t = time.time()
model.recontstruct_state(diffs, transformed_states, params=model_spec.good_params)
print(time.time() - t) #0.20 seconds
# Did not manage to speed it up with jit
```