Deterministic SIR Model with Stochastic Cases, with Normalising Flow

```r
library(dplyr, warn.conflicts = FALSE)
library(ggplot2)
library(patchwork)
library(tidyr)
library(torch)
library(torchflow)

theme_set(theme_minimal())
```

Stochastic SIR model

Forward simulation

The following function simulates the stochastic SIR model. The model is defined by the following parameters:

- N is the total population size.
- D is the duration of the infection.
- R_0 is the basic reproduction number.
- $\gamma = 1/D$ is the rate of recovery.
- $\beta = R_0\gamma$ is the rate of infection.
- ϵ is the rate of new infections from outside the population.
- α is the rate of reporting cases.
- S_1, I_1, R_1 are the initial number of susceptible, infected, and recovered individuals

At time t, the number of new infections is given by the binomial distribution:

$$I_t \sim \text{Binomial}(S_{t-1}, \lambda_t)$$

where $\lambda_t = 1 - \exp(-\beta I_{t-1}/N - \epsilon)$. The parameter ϵ is the rate of new infections from outside the population, which ensures that an epidemic can reappear even after it is extinct.

The number of new recoveries is given by the binomial distribution:

$$R_t \sim \text{Binomial}(I_{t-1}, \gamma)$$

where γ is the rate of recovery.

The number of new cases is given by the binomial distribution:

$$C_t \sim \text{Binomial}(I_t, \alpha)$$

```r
forward_simulate <- function(
  T,
  N,
  alpha,
  beta,
  gamma,
```

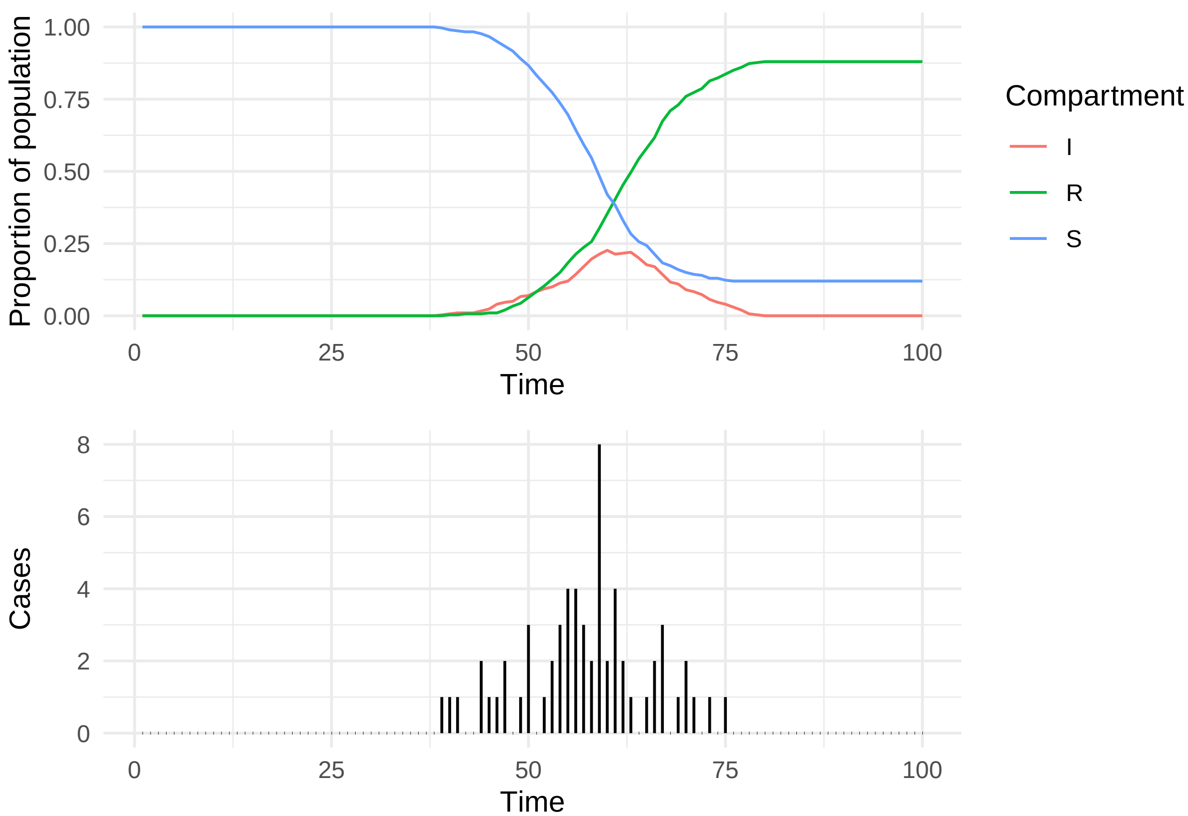
```r
  epsilon,
  S_1,
  I_1,
  R_1
) {
  n <- length(alpha)

  S <- matrix(0, n, T)
  I <- matrix(0, n, T)
  R <- matrix(0, n, T)
  cases <- matrix(0, n, T)

  S[, 1] <- round(N * S_1)
  I[, 1] <- round(N * I_1)
  R[, 1] <- round(N * R_1)

  for (t in 2 : T) {
    lambda <- 1 - exp(-beta * I[, t - 1] / N - epsilon)
    new_infections <- rbinom(n, S[, t - 1], lambda)
    new_recoveries <- rbinom(n, I[, t - 1], gamma)
    new_cases <- rbinom(n, new_infections, alpha)

    S[, t] <- S[, t - 1] - new_infections
    I[, t] <- I[, t - 1] + new_infections - new_recoveries
    R[, t] <- R[, t - 1] + new_recoveries
    cases[, t] <- new_cases
  }

  list(
    N = N,
    time = 1 : T,
    S = S / N,
    I = I / N,
    R = R / N,
    cases = cases / N
  )
}
```

We normalise S, I, R, and cases to be between 0 and 1, a proportion of the total population.

Here is an example of a forward simulation:

```r
set.seed(20250720)
example_simulation <- forward_simulate(
  T = 100,
  N = 300,
  alpha = 0.25,
  beta = 0.5,
  gamma = 0.2,
  epsilon = 0.0001,
  S_1 = 1,
  I_1 = 0,
  R_1 = 0
)
```

```r
example_simulation_df <- tibble(
  time = 1 : 100,
  S = example_simulation$S[1, ],
  I = example_simulation$I[1, ],
  R = example_simulation$R[1, ],
  cases = example_simulation$N * example_simulation$cases[1, ]
)

wrap_plots(
  example_simulation_df %>%
    select(-cases) %>%
    pivot_longer(-time, names_to = 'variable', values_to = 'value') %>%
    ggplot(aes(x = time)) +
      geom_line(
        mapping = aes(y = value, colour = variable)
      ) +
      labs(x = 'Time', y = 'Proportion of population', colour = 'Compartment'),
  ggplot(example_simulation_df, aes(x = time, xend = time, y = 0, yend = cases)) +
    geom_segment() +
    labs(x = 'Time', y = 'Cases'),
  ncol = 1
)
```



Here we generate from a prior distribution as follows:

- R_0 is uniformly distributed between 1.01 and 3.
- D is uniformly distributed between 2 and 6.

- ϵ is uniformly distributed between 1e-4 and 2 * 1e-4.
- α is uniformly distributed between 0.2 and 0.3.

Once these are generated, we simulate a single trajectory of the model.

```r
generate <- function(
  n,
  T,
  N,
  R_0_min = 1.01,
  R_0_max = 3,
  D_min = 2,
  D_max = 6,
  epsilon_min = 1e-4,
  epsilon_max = 2 * 1e-4,
  alpha_min = 0.2,
  alpha_max = 0.3
) {
  R_0 <- runif(n, R_0_min, R_0_max)
  D <- runif(n, D_min, D_max)
  epsilon <- runif(n, epsilon_min, epsilon_max)
  alpha <- runif(n, alpha_min, alpha_max)
  gamma <- 1 / D
  beta <- R_0 * gamma

  simulation <- forward_simulate(
    T,
    N,
    alpha,
    beta,
    gamma,
    epsilon,
    1,
    0,
    0
  )

  c(simulation, list(
    R_0 = R_0,
    D = D,
    alpha = alpha,
    beta = beta,
    gamma = gamma,
    epsilon = epsilon
  ))
}
```

We generate four example datasets:

```r
T <- 100
N <- 300

n_examples <- 4
example_data <- generate(n_examples, T = T, N = N)
str(example_data)
```

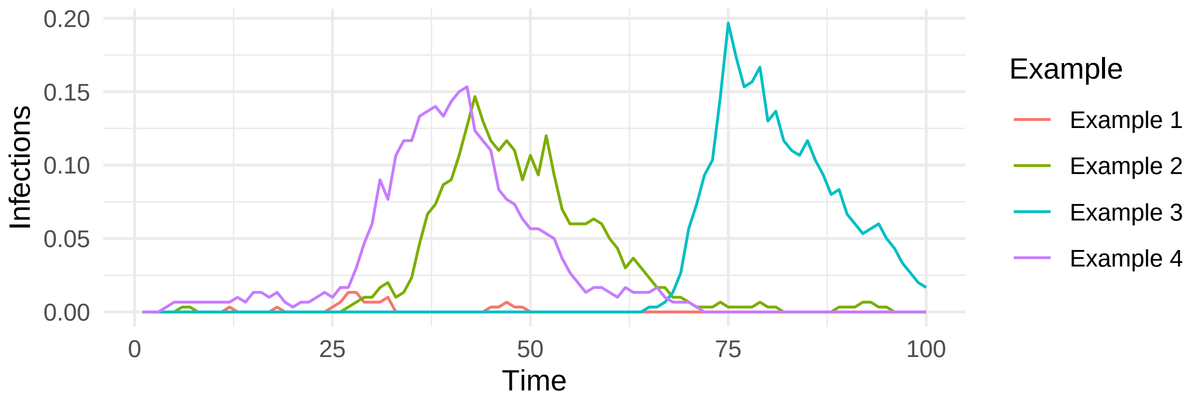
```
## List of 12
##  $ N      : num 300
##  $ time   : int [1:100] 1 2 3 4 5 6 7 8 9 10 ...
##  $ S      : num [1:4, 1:100] 1 1 1 1 1 1 1 1 1 1 ...
##  $ I      : num [1:4, 1:100] 0 0 0 0 0 0 0 0 0 0 ...
##  $ R      : num [1:4, 1:100] 0 0 0 0 0 0 0 0 0 0 ...
##  $ cases  : num [1:4, 1:100] 0 0 0 0 0 0 0 0 0 0 ...
##  $ R_0    : num [1:4] 1.06 1.96 2.47 1.69
##  $ D      : num [1:4] 2.4 3.16 3.54 3.66
##  $ alpha  : num [1:4] 0.21 0.207 0.251 0.24
##  $ beta   : num [1:4] 0.442 0.622 0.697 0.461
##  $ gamma  : num [1:4] 0.416 0.316 0.282 0.273
##  $ epsilon: num [1:4] 0.000145 0.000177 0.000121 0.000168
```

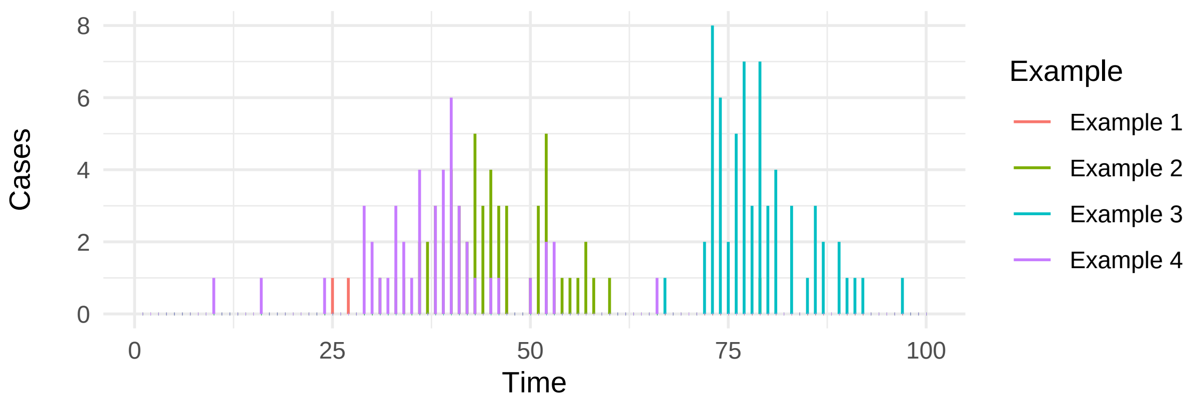
```r
print(example_data[c('gamma', 'beta', 'alpha', 'epsilon')])
```

```
## $gamma
## [1] 0.4161333 0.3164321 0.2821377 0.2731598
##
## $beta
## [1] 0.4420780 0.6216277 0.6970645 0.4612615
##
## $alpha
## [1] 0.2098382 0.2072343 0.2512811 0.2403872
##
## $epsilon
## [1] 0.0001453925 0.0001765887 0.0001211146 0.0001678262
```

We can plot the results:

```r
example_data_df <- tibble(
  time = rep(1 : T, each = n_examples),
  example = rep(sprintf('Example %d', 1 : n_examples), T),
  S = as.vector(example_data$S),
  I = as.vector(example_data$I),
  R = as.vector(example_data$R),
  cases = as.vector(example_data$N * example_data$cases)
)

wrap_plots(
  ggplot(example_data_df, aes(x = time, y = I, colour = example)) +
    geom_line() +
    labs(x = 'Time', y = 'Infections', colour = 'Example'),
  ggplot(example_data_df, aes(x = time, xend = time, y = 0, yend = cases, colour = example)) +
    geom_segment() +
    labs(x = 'Time', y = 'Cases', colour = 'Example'),
  ncol = 1
)
```





Normalising flow

The following code define a normalising flow model that takes as input the number of cases up to time t. It then approximates the posterior distribution of the $(\alpha, \beta, \gamma, \epsilon, S_t, I_t)$ parameters. More precisely, it approximates a transformation of the parameters:

- $\log \alpha$
- $\log \beta$
- $\log \gamma$
- $\log \epsilon$
- $\text{logit}(S_t)$, where S_t is clamped to be between 1e-5 and 1 - 1e-5
- $\text{logit}(I_t)$, where I_t is clamped to be between 1e-5 and 1 - 1e-5

The network that summarises the data into a fixed dimensional space is a Gated Recurrent Unit (GRU). This is a type of recurrent neural network that is designed to process sequential data. It is constructed such that, after all GRU layers have been applied, the output at time t is only a function of the data up to time t. This allows us to input varying sequence lengths.

```
device <- if (cuda_is_available()) torch_device("cuda") else torch_device("cpu")

summary_gru <- nn_module(
  "summary_gru",
  initialize = function(n_input_size, n_layers, n_output_size) {
    self$n_layers <- n_layers
    self$gru <- nn_gru(n_input_size, n_output_size, n_layers, batch_first = TRUE)
  },
```

```
  forward = function(conditioning) {
    t <- conditioning[, 1, drop = FALSE]
    x <- conditioning[, 2 : conditioning$shape[2], drop = FALSE]

    t_int <- t$to(dtype = torch_int64())
    x_unsqueezed <- x$unsqueeze(-1)

    # h_all contains the hidden state of the GRU at each time point
    h_all <- self$gru(x_unsqueezed)[[1]]

    # h_last is the hidden state at the last time point
    t_int_big <- t_int$unsqueeze(2)$expand(c(-1, -1, h_all$shape[3]))
    h_last <- torch_gather(h_all, 2, t_int_big)
    h_last$squeeze(2)
  }
)

n_summary <- 128L
summary_model <- summary_gru(n_input_size = 1, n_layers = 2, n_output_size = n_summary)
summary_model$to(device = device)

n_params <- 6L
flow_model <- nn_sequential_conditional_flow(
  nn_affine_coupling_block(n_params, n_summary),
  nn_permutation_flow(n_params),
  nn_affine_coupling_block(n_params, n_summary),
  nn_permutation_flow(n_params),
  nn_affine_coupling_block(n_params, n_summary),
  nn_permutation_flow(n_params),
  nn_affine_coupling_block(n_params, n_summary),
  nn_permutation_flow(n_params),
  nn_affine_coupling_block(n_params, n_summary)
)
flow_model$to(device = device)

summarizing_flow_model <- nn_summarizing_conditional_flow(summary_model, flow_model)
summarizing_flow_model$to(device = device)
```

Then we define a function to generate batches of data for training and testing:

```
logit <- function(x) {
  log(x / (1 - x))
}

expit <- function(y) {
  exp(y) / (1 + exp(y))
}

clamp <- function(x, min, max) {
  pmax(pmin(x, max), min)
}

epsilon <- 1e-5
```

```r
generate_conditional_samples <- function(...) {
  n <- 1024L
  data <- generate(n, T = T, N = N)
  t <- sample.int(T, n, replace = TRUE)

  list(
    target = torch_tensor(cbind(
      log(data$alpha),
      log(data$beta),
      log(data$gamma),
      log(data$epsilon),
      logit(clamp(data$S[cbind(seq_len(n), t)], epsilon, 1 - epsilon)),
      logit(clamp(data$I[cbind(seq_len(n), t)], epsilon, 1 - epsilon))
    ), device = device),
    conditioning = torch_tensor(
      cbind(t, data$cases),
      device = device
    ),
    latent = torch_stack(list(
      data$S,
      data$I,
      data$R
    ), dim = -1)$to(device = device)
  )
}
```

We can calculate the initial test loss:

```r
test_set <- generate_conditional_samples()

test_loss <- forward_kl_loss(summarizing_flow_model(test_set$target, test_set$conditioning))
cat('Initial test loss:', test_loss$item(), '\n')
```

```
## Initial test loss: 109.8265
```

Now we train the model.

```r
run_training <- function(n_epochs, lr) {
  system.time(train_conditional_flow(
    summarizing_flow_model,
    generate_conditional_samples,
    n_epochs = n_epochs,
    batch_size = 256L,
    lr = lr,
    after_epoch = function(epoch, ...) {
      if (epoch %% 64L == 0) {
        test_loss <- forward_kl_loss(summarizing_flow_model(test_set$target, test_set$conditioning))
        cat('Epoch', epoch, '| test loss:', test_loss$item(), '\n')
      }
    },
    verbose = interactive()
  ))
}

if (!file.exists('summarizing_flow_model.pt')) {
```

```r
  run_training(256L, 0.001)
  run_training(1024L, 0.0001)
  run_training(512L, 0.00001)
  torch_save(summarizing_flow_model, 'summarizing_flow_model.pt')
} else {
  summarizing_flow_model <- torch_load('summarizing_flow_model.pt')
  summarizing_flow_model$to(device = device)
}
```

The training is complete.

Results

Let's generate some samples from the approximate posterior.

We will generate samples from the posterior for four entries of the test set.

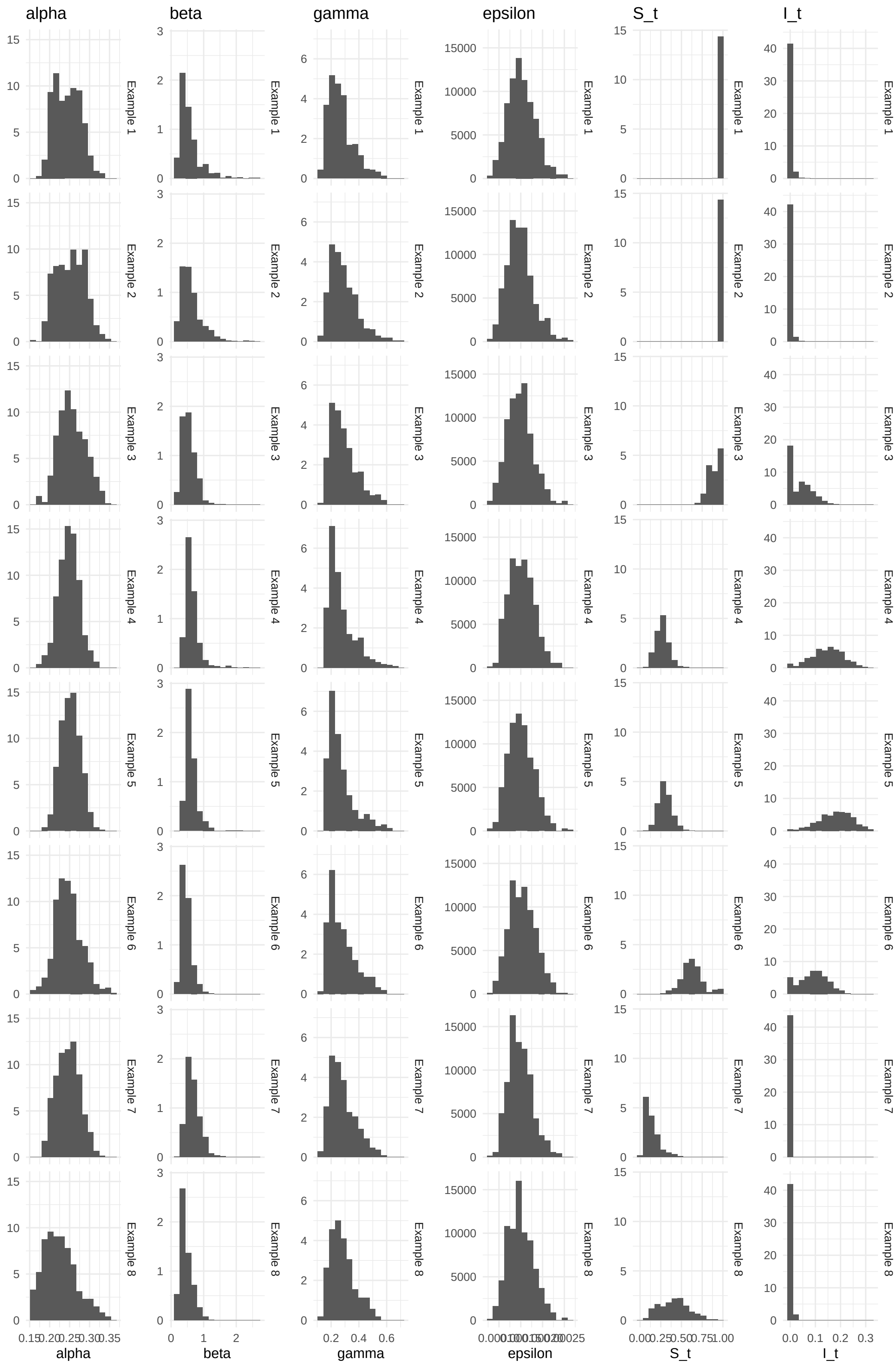
```r
sample_dataset <- function(n_samples, dataset) {
  n_entries <- dataset$conditioning$shape[1]
  samples_tensor <- generate_from_conditional_flow(
    summarizing_flow_model,
    n_samples,
    dataset$conditioning
  )
  samples <- as_array(samples_tensor)
  dimnames(samples) <- list(
    NULL,
    NULL,
    c('alpha', 'beta', 'gamma', 'epsilon', 'S_t', 'I_t')
  )
  samples[, , 'alpha'] <- exp(samples[, , 'alpha'])
  samples[, , 'beta'] <- exp(samples[, , 'beta'])
  samples[, , 'gamma'] <- exp(samples[, , 'gamma'])
  samples[, , 'epsilon'] <- exp(samples[, , 'epsilon'])
  samples[, , 'S_t'] <- expit(samples[, , 'S_t'])
  samples[, , 'I_t'] <- expit(samples[, , 'I_t'])

  forward_samples <- lapply(seq_len(n_entries), function(i) {
    test_t <- as_array(dataset$conditioning[i, 1])[1]
    S_1 <- samples[, i, 'S_t']
    I_1 <- samples[, i, 'I_t']
    R_1 <- 1 - S_1 - I_1
    simulation <- forward_simulate(
      T = T - test_t + 1L,
      N = N,
      alpha = samples[, i, 'alpha'],
      beta = samples[, i, 'beta'],
      gamma = samples[, i, 'gamma'],
      epsilon = samples[, i, 'epsilon'],
      S_1 = S_1,
      I_1 = I_1,
      R_1 = R_1
    )
    simulation$time <- test_t : T
```

```
    simulation
  })

  target <- as_array(dataset$target)
  colnames(target) <- c('alpha', 'beta', 'gamma', 'epsilon', 'S', 'I')
  target[, 'alpha'] <- exp(target[, 'alpha'])
  target[, 'beta'] <- exp(target[, 'beta'])
  target[, 'gamma'] <- exp(target[, 'gamma'])
  target[, 'epsilon'] <- exp(target[, 'epsilon'])
  target[, 'S'] <- expit(target[, 'S'])
  target[, 'I'] <- expit(target[, 'I'])

  t <- as_array(dataset$conditioning[, 1])
  cases <- as_array(dataset$conditioning[, 2 : 101])
  latent <- as_array(dataset$latent)

  list(
    samples = samples,
    forward_samples = forward_samples,
    target = target,
    cases = cases,
    latent = latent,
    t = t
  )
}

test_t <- as_array(test_set$conditioning[, 1])
example_indices <- c(
  sample(which(test_t < 25), 1),
  sample(which(test_t < 25), 1),
  sample(which(test_t >= 25 & test_t < 50), 1),
  sample(which(test_t >= 25 & test_t < 50), 1),
  sample(which(test_t >= 50 & test_t < 75), 1),
  sample(which(test_t >= 50 & test_t < 75), 1),
  sample(which(test_t >= 75), 1),
  sample(which(test_t >= 75), 1)
)
example_dataset <- list(
  target = test_set$target[example_indices, ],
  conditioning = test_set$conditioning[example_indices, , drop = FALSE],
  latent = test_set$latent[example_indices, , , drop = FALSE]
)

n_samples <- 512L
example_samples <- sample_dataset(n_samples, example_dataset)
n_examples <- length(example_indices)
```

Now we will plot the samples and the true values. First, histograms (red lines are the true values):

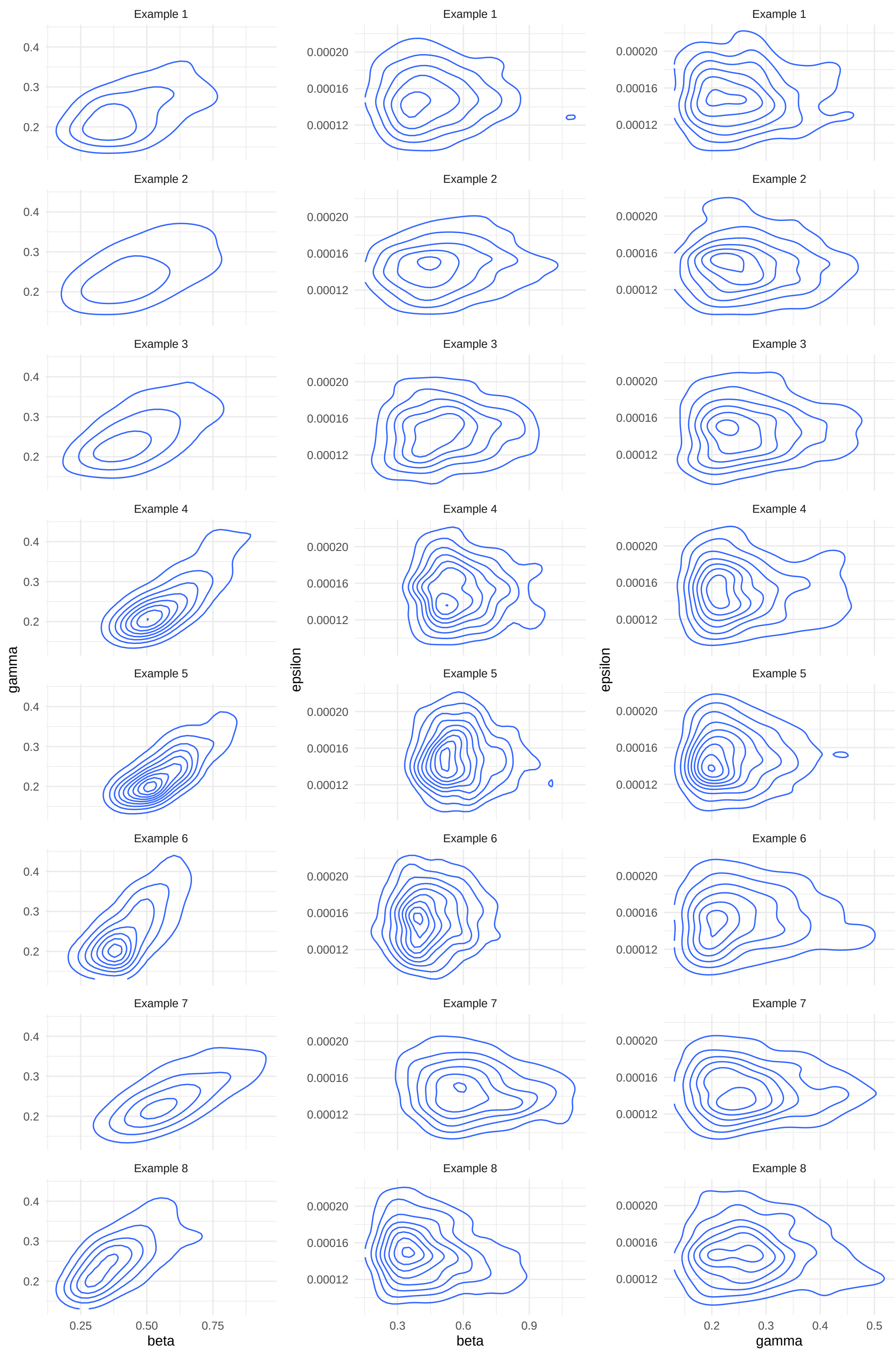
```
wrap_plots(lapply(dimnames(example_samples$samples)[[3]], function(param_i) {
  example_samples_df_i <- tibble(
    sample = rep(1 : n_samples, n_examples),
    example = rep(sprintf('Example %d', 1 : n_examples), each = n_samples),
    value = as.vector(example_samples$samples[, , param_i])
```

```r
  )

  example_samples_df_i <- example_samples_df_i %>%
    filter(
      value > quantile(value, 0.001),
      value < quantile(value, 0.999)
    )

  ggplot(example_samples_df_i, aes(value, after_stat(density))) +
    geom_histogram(bins = 15) +
    facet_grid(example ~ .) +
    ggtitle(param_i) +
    labs(x = param_i, y = NULL)
}), nrow = 1)
```


alpha
beta
gamma
epsilon
S_t
I_t
Example 1
Example 2
Example 3
Example 4
Example 5
Example 6
Example 7
Example 8
0.15 0.20 0.25 0.30 0.35
alpha
0 1 2
beta
0.2 0.4 0.6
gamma
epsilon
0.00 0.25 0.50 0.75 1.00
S_t
0.0 0.1 0.2 0.3
I_t

Then pairwise plots for the dynamical parameters (red lines are the true values):

```
example_samples_pairwise <- tibble(
  sample = rep(1 : n_samples, n_examples),
  example = rep(sprintf('Example %d', 1 : n_examples), each = n_samples)
)
for (param_i in dimnames(example_samples$samples)[[3]]) {
  example_samples_pairwise[[param_i]] <- as.vector(example_samples$samples[, , param_i])
}

make_pairwise_plot <- function(param_i, param_j, name_i, name_j) {
  ggplot(
    example_samples_pairwise %>%
      filter(
        {{ param_i }} > quantile({{ param_i }}, 0.001),
        {{ param_i }} < quantile({{ param_i }}, 0.999),
        {{ param_j }} > quantile({{ param_j }}, 0.001),
        {{ param_j }} < quantile({{ param_j }}, 0.999)
      ),
    aes({{ param_i }}, {{ param_j }})
  ) +
    geom_density_2d(bins = 10) +
    facet_wrap(~ example, ncol = 1) +
    labs(x = name_i, y = name_j)
}

wrap_plots(
  make_pairwise_plot(beta, gamma, 'beta', 'gamma'),
  make_pairwise_plot(beta, epsilon, 'beta', 'epsilon'),
  make_pairwise_plot(gamma, epsilon, 'gamma', 'epsilon'),
  nrow = 1
)
```


Example 1
Example 2
Example 3
Example 4
Example 5
Example 6
Example 7
Example 8
gamma
epsilon
beta
0.25
0.50
0.75
0.3
0.6
0.9
0.2
0.3
0.4
0.5
0.00020
0.00016
0.00012

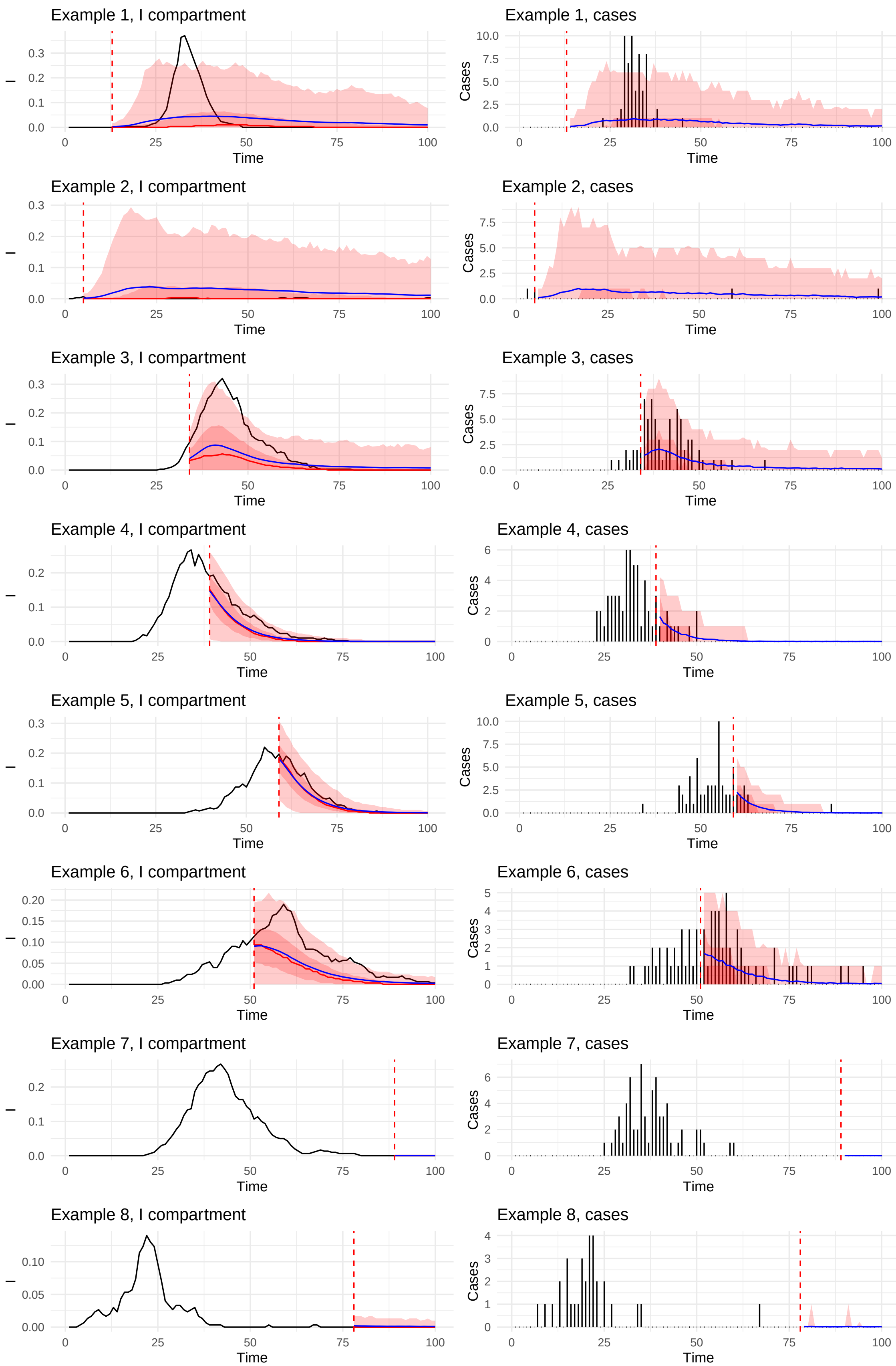
Then we plot the true and forecasted trajectories for the I compartment and for the cases. Here,

- The black line is the true trajectory / cases.
- For the I compartment, the blue line is the posterior mean, while the red line is the posterior median.
- For the I compartment, the two ribbons span 25th to 75th and the 2.5th to 97.5th percentiles.
- For the cases, the blue line is the posterior mean, while the ribbons span from 0 to the the 50th, 0 to the 75th, and 0 to the 97.5th percentiles.

```
plot_forward_samples <- function(samples) {
  n_examples <- length(samples$forward_samples)
  wrap_plots(lapply(seq_len(n_examples), function(i) {
    forward_samples_i <- samples$forward_samples[[i]]

    forward_mean_I <- apply(forward_samples_i$I, 2, mean)
    forward_quantiles_I <- apply(forward_samples_i$I, 2, quantile, c(0.025, 0.25, 0.5, 0.75, 0.975))
    latent_I_df <- tibble(
      time = 1 : T,
      I = samples$latent[i, , 2]
    )
    forward_I_df <- tibble(
      time = forward_samples_i$time,
      I_mean = forward_mean_I,
      I_q025 = forward_quantiles_I[1, ],
      I_q250 = forward_quantiles_I[2, ],
      I_q500 = forward_quantiles_I[3, ],
      I_q750 = forward_quantiles_I[4, ],
      I_q975 = forward_quantiles_I[5, ]
    )

    forward_mean_cases <- apply(forward_samples_i$cases, 2, mean)
    forward_quantiles_cases <- apply(forward_samples_i$cases, 2, quantile, c(0.5, 0.75, 0.975))
    cases_df <- tibble(
      time = 1 : T,
      cases = N * samples$cases[i, ]
    )
    forward_cases_df <- tibble(
      time = forward_samples_i$time,
      cases_mean = N * forward_mean_cases,
      cases_q500 = N * forward_quantiles_cases[1, ],
      cases_q750 = N * forward_quantiles_cases[2, ],
      cases_q975 = N * forward_quantiles_cases[3, ]
    ) %>%
      filter(time > samples$t[i])

    wrap_plots(
      ggplot(mapping= aes(time)) +
        geom_line(
          data = latent_I_df,
          mapping = aes(time, I)
        ) +
        geom_ribbon(
          data = forward_I_df,
          mapping = aes(ymin = I_q025, ymax = I_q975),
          alpha = 0.2,
          fill = 'red'
```

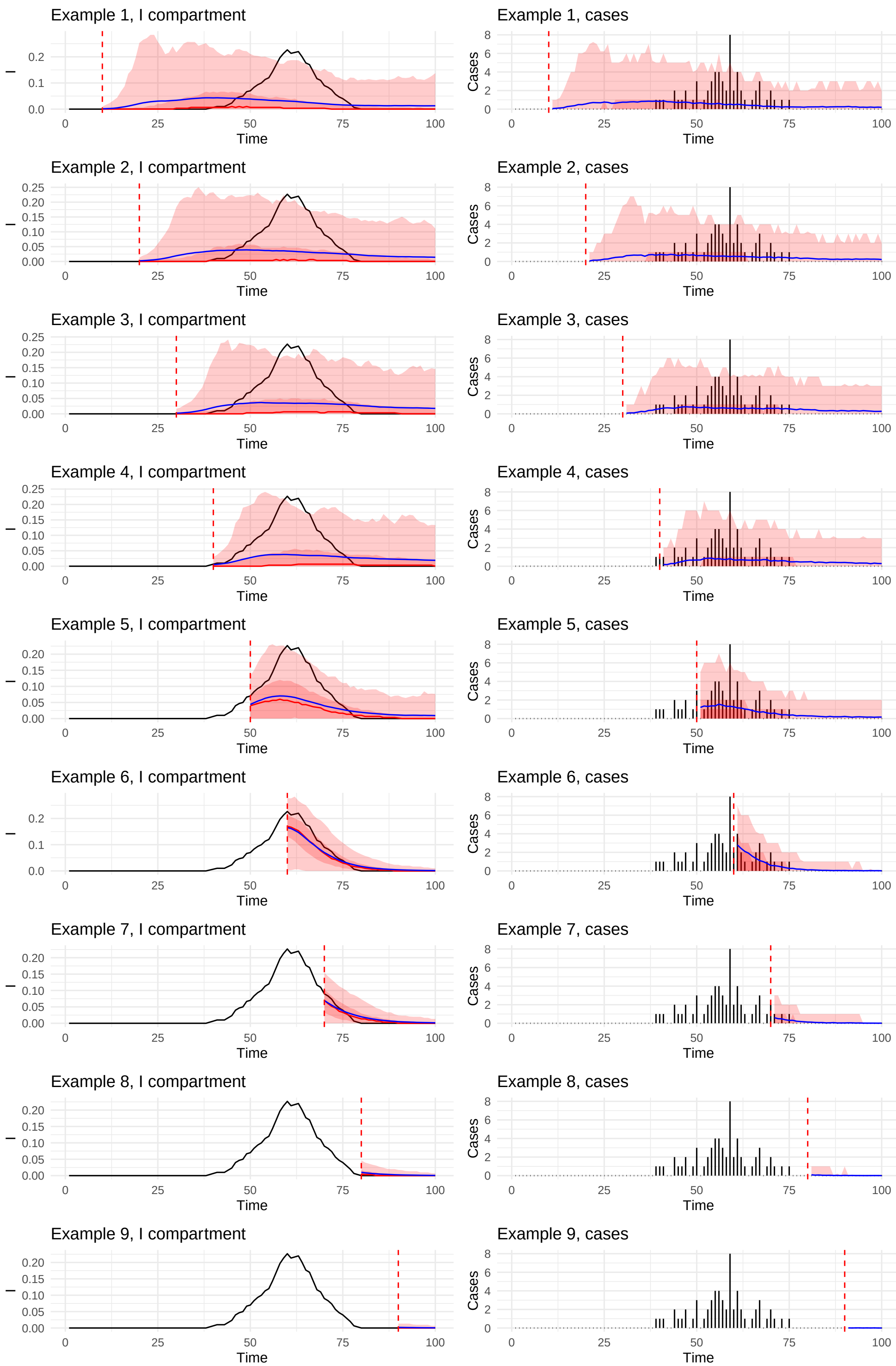
```
    ) +
    geom_ribbon(
      data = forward_I_df,
      mapping = aes(ymin = I_q250, ymax = I_q750),
      alpha = 0.2,
      fill = 'red'
    ) +
    geom_line(
      data = forward_I_df,
      mapping = aes(time, I_q500),
      colour = 'red'
    ) +
    geom_line(
      data = forward_I_df,
      mapping = aes(time, I_mean),
      colour = 'blue'
    ) +
    geom_vline(
      xintercept = samples$t[i],
      colour = 'red',
      linetype = 'dashed'
    ) +
    labs(x = 'Time', y = 'I') +
    ggtitle(sprintf('Example %d, I compartment', i)),
  ggplot(mapping = aes(time)) +
    geom_segment(
      data = cases_df,
      mapping = aes(x = time, xend = time, y = 0, yend = cases)
    ) +
    geom_ribbon(
      data = forward_cases_df,
      mapping = aes(ymin = 0, ymax = cases_q500),
      alpha = 0.2,
      fill = 'red'
    ) +
    geom_ribbon(
      data = forward_cases_df,
      mapping = aes(ymin = 0, ymax = cases_q750),
      alpha = 0.2,
      fill = 'red'
    ) +
    geom_ribbon(
      data = forward_cases_df,
      mapping = aes(ymin = 0, ymax = cases_q975),
      alpha = 0.2,
      fill = 'red'
    ) +
    geom_line(
      data = forward_cases_df,
      mapping = aes(time, cases_mean),
      colour = 'blue'
    ) +
    geom_vline(
```

```r
          xintercept = samples$t[i],
          colour = 'red',
          linetype = 'dashed'
        ) +
        labs(x = 'Time', y = 'Cases') +
        ggtitle(sprintf('Example %d, cases', i)),
      nrow = 1
    )
  }), ncol = 1)
}

plot_forward_samples(example_samples)
```


Example 1, I compartment
Example 1, cases
Example 2, I compartment
Example 2, cases
Example 3, I compartment
Example 3, cases
Example 4, I compartment
Example 4, cases
Example 5, I compartment
Example 5, cases
Example 6, I compartment
Example 6, cases
Example 7, I compartment
Example 7, cases
Example 8, I compartment
Example 8, cases
I
Cases
Time

Let's also focus on the single example from the start of this document and step through some forecasts.

```
single_example_times <- seq(10, 90, by = 10)
single_example <- list(
  target = torch_tensor(do.call(rbind, lapply(single_example_times, function(t) {
    c(
      log(0.25),
      log(0.5),
      log(0.2),
      log(0.0001),
      logit(clamp(example_simulation$S[, t], epsilon, 1 - epsilon)),
      logit(clamp(example_simulation$I[, t], epsilon, 1 - epsilon))
    )
  })), device = device),
  conditioning = torch_tensor(do.call(rbind, lapply(single_example_times, function(t) {
    c(t, example_simulation$cases[1, ])
  })), device = device),
  latent = torch_tensor(abind::abind(lapply(single_example_times, function(t) {
    cbind(example_simulation$S[1, ], example_simulation$I[1, ], example_simulation$R[1, ])
  }), along = 0), device = device)
)
single_example_samples <- sample_dataset(512L, single_example)
n_single_examples <- length(single_example_times)

plot_forward_samples(single_example_samples)
```


Example 1, I compartment
Example 1, cases
Example 2, I compartment
Example 2, cases
Example 3, I compartment
Example 3, cases
Example 4, I compartment
Example 4, cases
Example 5, I compartment
Example 5, cases
Example 6, I compartment
Example 6, cases
Example 7, I compartment
Example 7, cases
Example 8, I compartment
Example 8, cases
Example 9, I compartment
Example 9, cases
Time
Cases