SM2 / SC2 Project

Using ABC/SIR to Model the Spread of Influenza in a Boarding School

Grace Yan, Ollie Baker and Daniel Gardner

For this group project, we investigated the problem of intractable likelihoods using Approximate Bayesian Computation. Such simulation based methods are best applied to this case where we are working with a model such that we are able to simulate results easily from it, yet have no analytical form of the likelihood available. This is exactly the case in epidemiology models, where we are unable to use methods such as maximum likelihood estimation to estimate the infection parameters given data.

Data-set

We begin by importing the dataset we chose for this project: the `bsflu` dataset from the package `pomp`. This dataset records a 1978 Influenza outbreak in a boy's boarding school.

```
library(pomp)
library(Rcpp)
library(sitmo)
library(ggplot2)
data(bsflu)
```

The dataset tallies infection information over a period of 14 days, in a boarding school of 763 students.

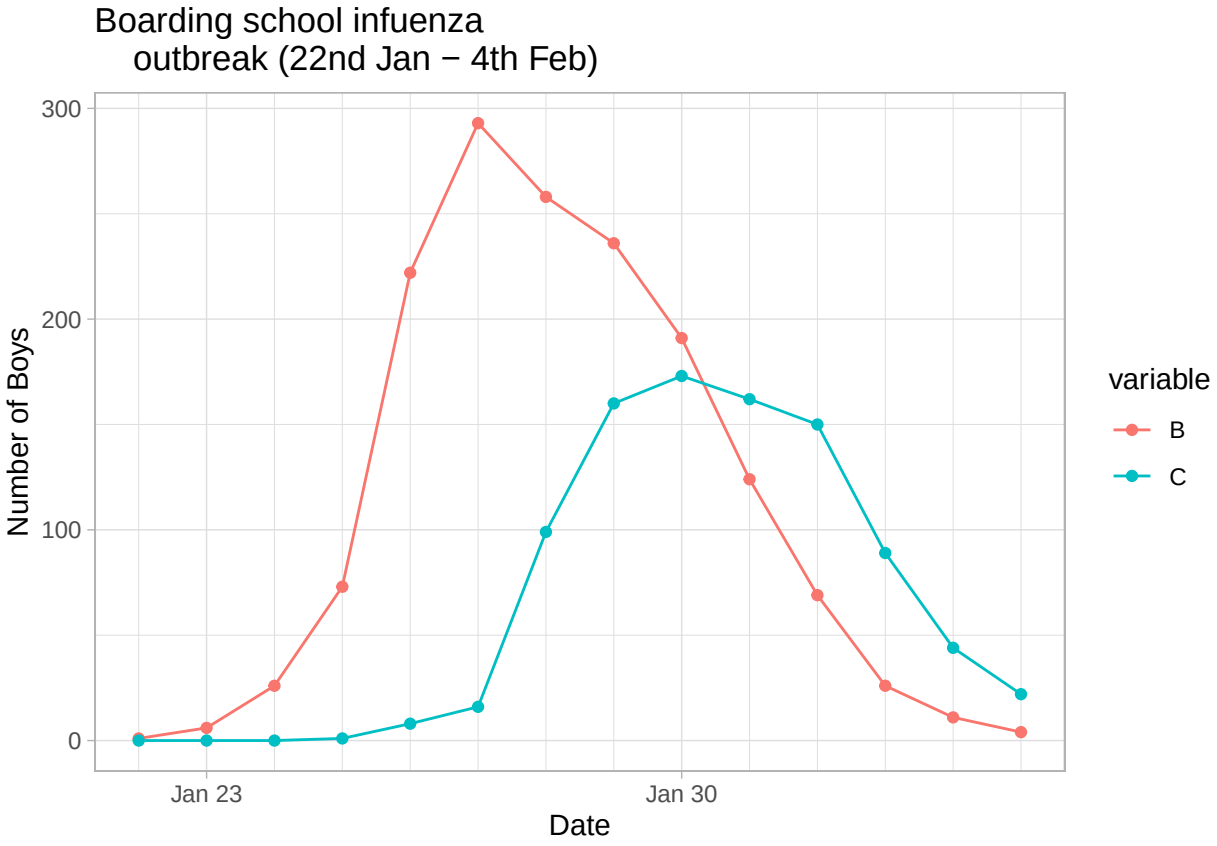
```
head(bsflu)
```

```
##         date   B  C day
## 1 1978-01-22   1  0   1
## 2 1978-01-23   6  0   2
## 3 1978-01-24  26  0   3
## 4 1978-01-25  73  1   4
## 5 1978-01-26 222  8   5
## 6 1978-01-27 293 16   6
```

The column `B` contains the number of students who are bedridden with the flu on a given day (i.e. classes as 'infected'). C contains the number of students who are *convalescent* (i.e. not infected but yet unable to return to class).

```
library(tidyr)
library(ggplot2)


bsflu |>
    gather(variable,value,-date,-day) |>
    ggplot(aes(x=date,y=value,color=variable))+
    geom_line()+
    geom_point()+
    labs(y="Number of Boys",x="Date",title="Boarding school infuenza
```

```
    outbreak (22nd Jan - 4th Feb)")+
  scale_x_date(date_minor_breaks = "1 day",date_labels = "%b %d")+
    theme_light()
```



Model

In order to model disease data, we will use the well-studied SIR model. This model models the number of people in three states: Susceptible, Infected, and Recovered. The model is defined by the following system of differential equations:

$$\frac{dS}{dt} = -\beta SI$$
$$\frac{dI}{dt} = \beta SI - \gamma I$$
$$\frac{dR}{dt} = \gamma I$$

Where S is the proportion of susceptible individuals, I is the proportion of infected individuals, and R is the proportion of recovered individuals. β is the rate of infection, and γ is the rate of recovery.

With such a definition, we can translate the column `B` in the `bsflu` data directly to the variable I simply by dividing `B` by the total number of students ($N = 763$). Unfortunately, the column `C` has no analogy in the model, as it acts as a confusing 'between recovery' state that cannot be grouped in with either I or R. Therefore going forward, we will primarily be using the column `B` as the observed data in our SIR model estimate.

Approximate Bayesian computation is a simulation based approach, and will require many individual computations. In the interest of speed therefore, we will implement the SIR model in C++, then use RCPP to call the C++ code from R.

```
sourceCpp(code = "
  #include <Rcpp.h>
  using namespace Rcpp;

  // [[Rcpp::export]]
  NumericVector SIR(NumericVector s, NumericVector i, NumericVector r,
  double s0, double i0, double r0, double beta, double gamma) {
    s[0] = s0;
    i[0] = i0;
    r[0] = r0;

    for (int t = 1; t < s.size(); t++) {
      s[t] = s[t-1] - beta * s[t-1] * i[t-1];
      i[t] = i[t-1] + beta * s[t-1] * i[t-1] - gamma * i[t-1];
      r[t] = r[t-1] + gamma * i[t-1];
    }

    return i;
  }
")
```

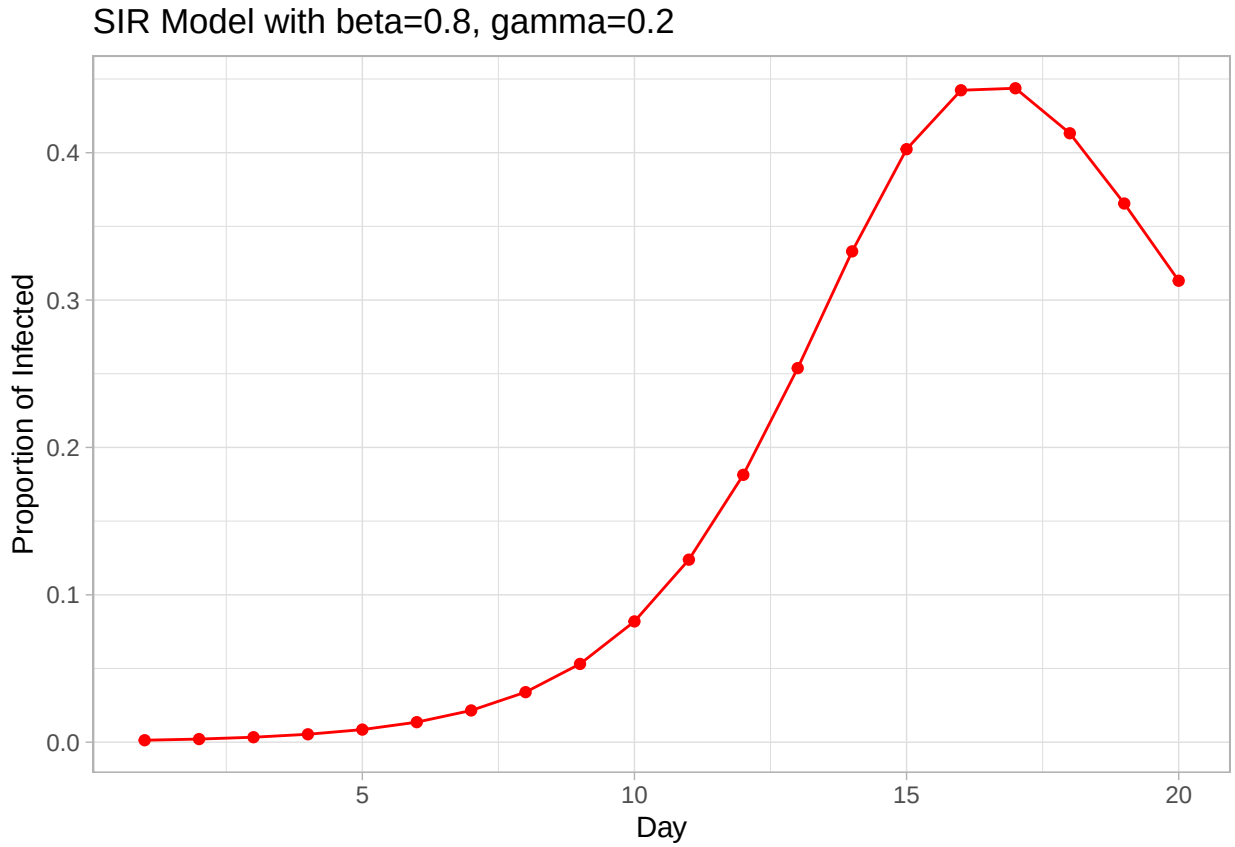
Below is a demonstration of the SIR model above, with $\beta = 0.8$ and $\gamma = 0.2$.

```
s <- numeric(20)
i <- numeric(20)
r <- numeric(20)

sir <- SIR(s, i, r, 1 - 1/763, 1/763, 0, 0.8, 0.2)

# convert SIR to a data frame
sir <- data.frame(day = 1:20, sir = sir)
print(sir)
```

```
##    day         sir
## 1    1 0.001310616
## 2    2 0.002095611
## 3    3 0.003349026
## 4    4 0.005347643
## 5    5 0.008527571
## 6    6 0.013569424
## 7    7 0.021518985
## 8    8 0.033942171
## 9    9 0.053083221
## 10  10 0.081917372
## 11  11 0.123828512
## 12  12 0.181407697
## 13  13 0.253810306
## 14  14 0.333041786
## 15  15 0.402372183
```

```
## 16  16 0.442376936
## 17  17 0.443721280
## 18  18 0.413185769
## 19  19 0.365510796
## 20  20 0.313113502
```

```
ggplot(aes(x=day,y=sir), data = sir)+
  geom_line(color="red")+
  geom_point(color="red")+
  labs(y="Proportion of Infected",x="Day",
  title="SIR Model with beta=0.8, gamma=0.2")+theme_light()
```



ABC Definition

Consider first the general case of intractable likelihood: having a model $f(.|\theta)$ with intractable likelihood $l(\theta|.)$ and parameter θ.

For ABC, we first repeatedly generate samples of θ from a prior distribution $\theta \sim \pi(.)$. Then for each generated value of θ, we input this into our effectively 'black box' model to get simulated data $\tilde{y}(\theta) \sim f(.|\theta)$.

We then need to define some distance metric D between the observed data y and simulated data $\tilde{y}(\theta)$, only accepting the proposed θ if this distance falls below a defined tolerance value ϵ.

Even a simple rejection sampling algorithm such as this can be shown to produce samples $\{\theta_1, ..., \theta_M\}$ (M being the number of accepted values) that are samples from the joint distribution:

$$\pi_\epsilon(\theta, \tilde{y}|y) = \frac{\pi(\theta) f(\tilde{y}|\theta) \mathbb{I}(\tilde{y} \in A)}{\int_A \int_\Theta \pi(\theta) f(\tilde{y}|\theta) d\tilde{y} d\theta}$$

Where $\mathbb{I}$ is the indicator function, Θ is the support of θ, and A is the acceptance region defined by D, y, and ϵ. Then given a suitable choice of tolerance value, this can produce an approximation to the posterior distribution of θ [1].

$$\pi_\epsilon(\theta|y) = \int_A \pi_\epsilon(\theta, \tilde{y}|y)d\tilde{y} \approx \pi(\theta|y)$$

Clearly here much of the resulting estimate relies on our choice of tolerance parameter ϵ and distance metric D, both of which will be looked at later. Something else to consider is that in practice the distance metric is applied to a *summary statistic* of the data rather than the raw data, in order to reduce dimensionality. This can be anything from the mean $\bar{y}$ and empirical quantiles, to more complex statistics such as kernels or auxiliary parameters. These methods will be looked at near the tail end of our investigation.

ABC Implementation

For the distance metric within ABC, we will need to compare the simulated data to the observed data. However as noted in the Data-set section, only the column B can be used. Hence we will compare the B column of the dataset to the number of infected individuals I in our SIR model. Similar to the approach of [2], we will use the mean squared error between the proportions of infected individuals in the observed data and the simulated data as our distance metric, and compare it to the other distance metric used; the absolute error between the proportion of infected individuals on the final day of the observed data and the simulated data.

The following code implements the ABC/SIR algorithm in C++.

```
sourceCpp(code='
#include <Rcpp.h>
#include <RcppParallel.h>
#include <omp.h>
#include <sitmo.h>
#include <cmath>
using namespace Rcpp;

// [[Rcpp::depends(RcppParallel)]]
// [[Rcpp::depends(sitmo)]]
// [[Rcpp::plugins(openmp)]]

// Function to simulate data from SIR model

// [[Rcpp::export]]
double unif_sitmo(int seed) {
  uint32_t coreseed = static_cast<uint32_t>(seed);
  sitmo::prng eng(coreseed);
  double mx = sitmo::prng::max();
  double x = eng() / mx;
  return x;
}

double max_double(double* x){

  double m = 0;
  for (int i=0; i<20; i++) {
    if(x[i] > m){
      m = x[i];
    }
```

```
  }
  return m;
}

double calc_dist_serial(double* x_sim, NumericVector x) {
  double total;
  for (int i=0; i<20; i++) {
    total += pow(x[i]-x_sim[i],2);
  }
  return total;
}

double mean_std_dist(double* x_sim, NumericVector x, NumericVector w) {
  double mean;
  double sd;
  double s;
  for (int i=0;i < 20; i++){
    s += x_sim[i];
  }
  for (int i=0;i < 20; i++){
    mean += (i+1)*x_sim[i]/s;
  }
  for (int i=0;i < 20; i++){
    sd += pow(i+1-mean,2)*x_sim[i]/s;
  }

  sd = sqrt(sd);

  double total = w[0]*pow(mean-13.14,2)/pow(13.14,2)+
  w[1]*pow(sd-2.19,2)/pow(2.19,2)+w[2]*pow(max_double(x_sim)-
  max(x),2)/(pow(max(x),2));

  return total;

}

// [[Rcpp::export]]
NumericMatrix ABC(int n, double eps, int p, NumericVector x, int ncores, int metric,
NumericVector w)
{

  NumericMatrix accepted_samples(n, p);
  int count = 0;
  double dist;

  #pragma omp parallel num_threads(ncores)
  {
    double theta_sim[2];
    #pragma omp for
    for (int i=0; i<n; i++) {

      #pragma omp critical
      {
```

```
      theta_sim[0] = 3*unif_sitmo(i);
      theta_sim[1] = unif_sitmo(i+n);
      }

      double S[20];
      double I[20];
      double R[20];

      S[0] = 762.0/763.0;
      I[0] = 1.0/763.0;
      R[0] = 0.0;

      for (int t = 1; t < 20; t++) {
        S[t] = S[t-1] - theta_sim[0] * S[t-1] * I[t-1];
        I[t] = I[t-1] + theta_sim[0] * S[t-1] * I[t-1] - theta_sim[1] * I[t-1];
        R[t] = R[t-1] + theta_sim[1] * I[t-1];
      }

      #pragma omp critical
      {
      if (metric == 1) {
        dist = calc_dist_serial(I, x);
      } else if (metric == 2) {
        dist = pow(I[19]-x[19], 2);
      } else if (metric == 3) {
        dist = mean_std_dist(I, x, w);
      }
      if (dist < eps) {
        accepted_samples(i, 0) = theta_sim[0];
        accepted_samples(i, 1) = theta_sim[1];
        count++;
      }
      }
    }
  }

  std::cout << "Acceptance rate: " << (double)count / n << std::endl;
  return accepted_samples;

}

')
```

The function `ABC` takes in the number of samples `n`, the tolerance `eps`, the number of parameters `p`, the observed data `x`, and the number of cores to use `ncores`. It then returns a matrix of accepted samples. It is important to note that we should normalise the `x` vector before inputting it into the ABC function, as the SIR model uses proportions rather than raw numbers.

To make the code run faster, we use OpenMP to parallelise it. More specifically, we parallelise the 'for' loop in the ABC algorithm. This is fine as the iterations are independent, meaning that it does not matter what order they are implemented in (unlike a Metropolis-Hastings algorithm, for example, where each iteration depends on the result of the previous iteration). With parallelisation, it is necessary to ensure thread safety when handling multiple threads to prevent race conditions (when multiple threads access shared data concurrently without proper synchronisation, e.g. two threads trying to modify the same variable

simultaneously), incorrect results or crashes. Since all the threads need to access the `accepted_samples` vector, and since R and Rcpp's API are not thread safe in general, we decided to put the acceptance step inside a critical section. The `count` and `dist` variables are also shared among the threads, so the parts of the code that update these variables are also put inside the critical section.

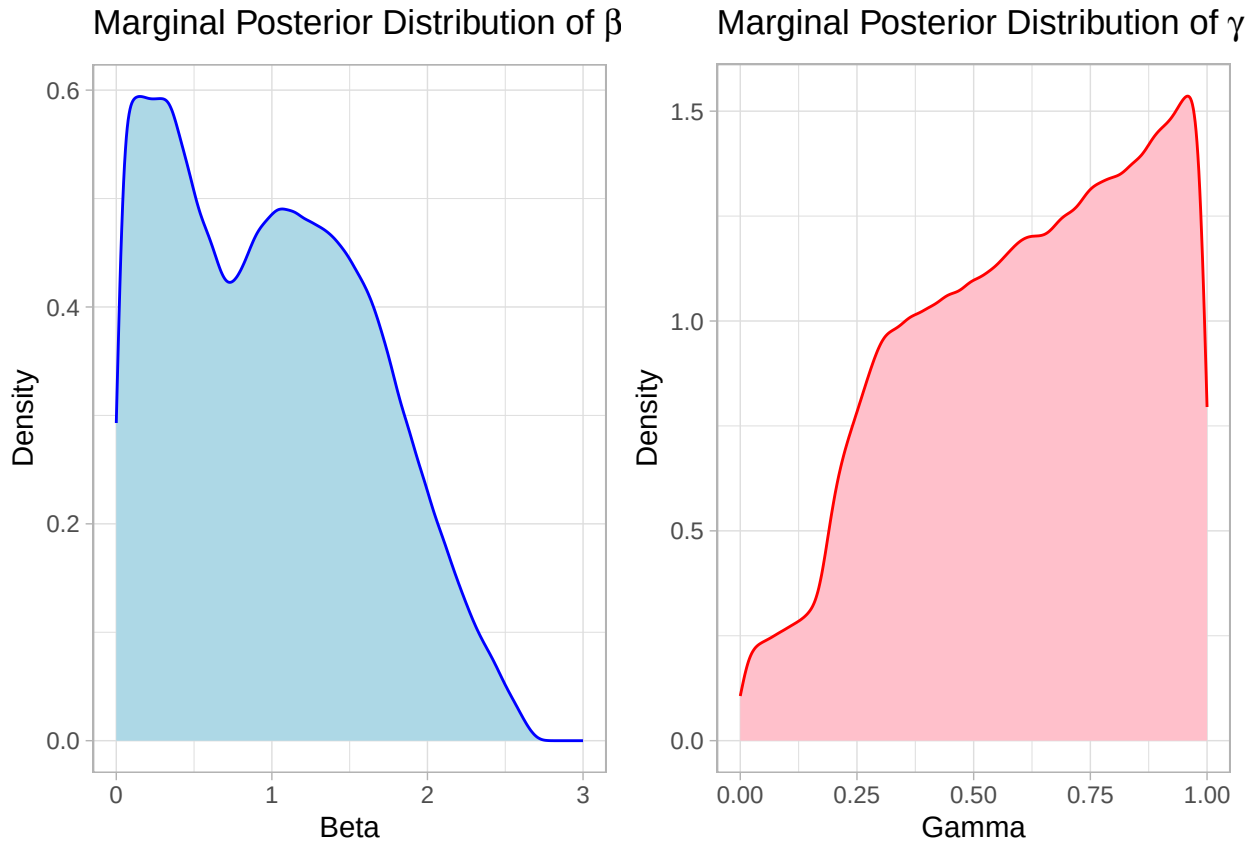
Comparison of Metrics

As an example, we can run the ABC algorithm with 10^7 samples, a tolerance of 0.8, and 2 parameters, using the mean squared error initially as the distance metric.

```
x <- c(rep(1/763,5), 2/763, bsflu$B/763)
accepted_samples <- ABC(1e6, .8, 2, x, 4, 1, NA)

# remove zeros
accepted_samples <- accepted_samples[!rowSums(accepted_samples==0),]
```

We can then plot the marginal posterior distributions of the parameters β and γ.

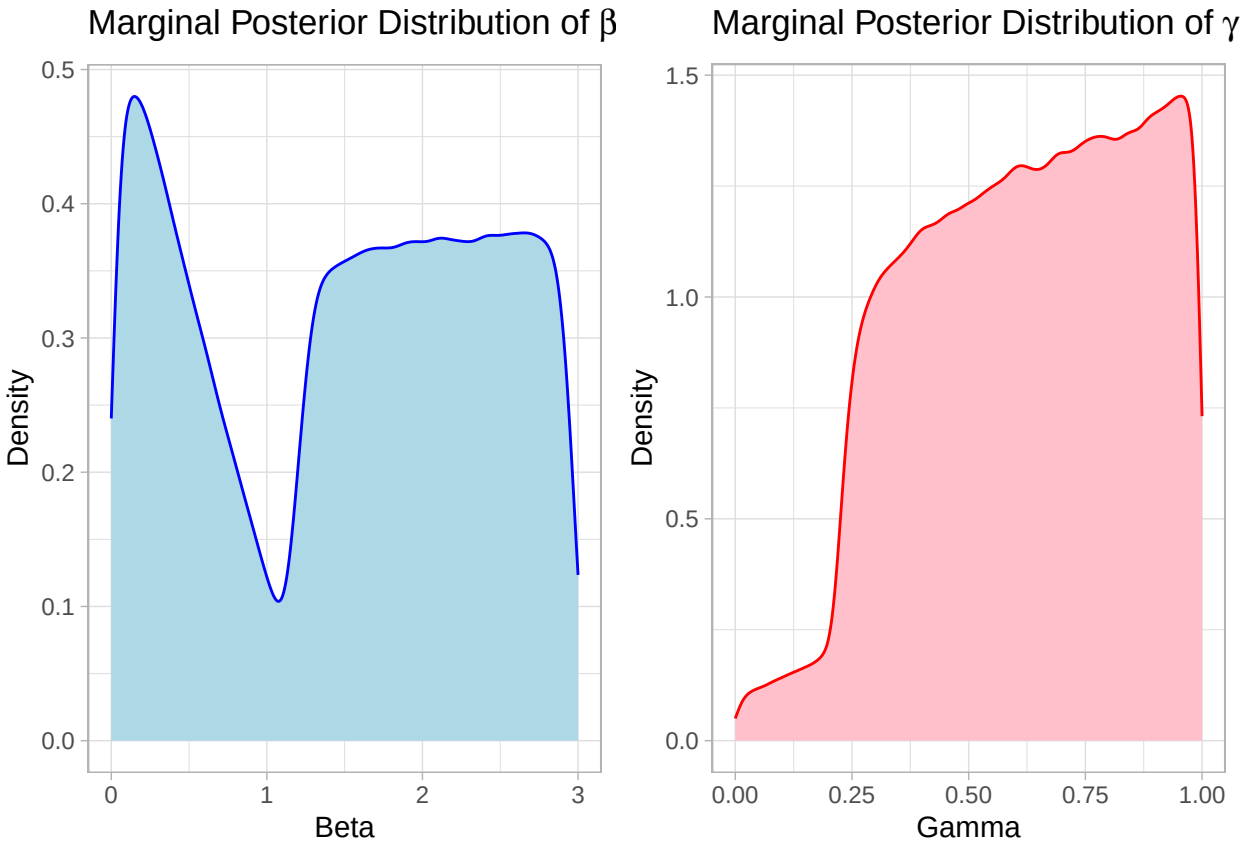
```
library(cowplot)
# plot the density histograms
example_beta_plot<-ggplot(aes(x=accepted_samples[,1]),
  data = as.data.frame(accepted_samples))+
  geom_density(fill="lightblue",color="blue")+
  xlim(c(0,3))+
  labs(x="Beta",y="Density",
  title=expression(paste("Marginal Posterior Distribution of ",
                         beta)))+
  theme_light()

example_gamma_plot<-ggplot(aes(x=accepted_samples[,2]), data = as.data.frame(accepted_samples))+
  geom_density(fill="pink",color="red")+
  xlim(c(0,1))+
  labs(x="Gamma",y="Density",
  title=expression(paste("Marginal Posterior Distribution of ",
  gamma)))+
  theme_light()

plot_grid(example_beta_plot, example_gamma_plot)
```



Now we can compare these posterior distributions to the distributions obtained when we use the absolute error between the proportion of infected individuals on the final day of the observed data and the simulated data as the distance metric.

```
accepted_samples2 <- ABC(1e6, 0.001, 2, x, 4, 2,NA)
accepted_samples2 <- accepted_samples2[!rowSums(accepted_samples2==0),]
```

```
abs_beta_plot<-ggplot(aes(x=accepted_samples2[,1]),
data = as.data.frame(accepted_samples2))+
  geom_density(fill="lightblue",color="blue")+
  labs(x="Beta",y="Density",
  title=expression(paste("Marginal Posterior Distribution of ",
  beta)))+
  theme_light()

abs_gamma_plot<-ggplot(aes(x=accepted_samples2[,2]),
data = as.data.frame(accepted_samples2))+
  geom_density(fill="pink",color="red")+
  labs(x="Gamma",y="Density",
  title=expression(paste("Marginal Posterior Distribution of ",
  gamma)))+
  theme_light()

plot_grid(abs_beta_plot,abs_gamma_plot)
```



Acceptance Rates

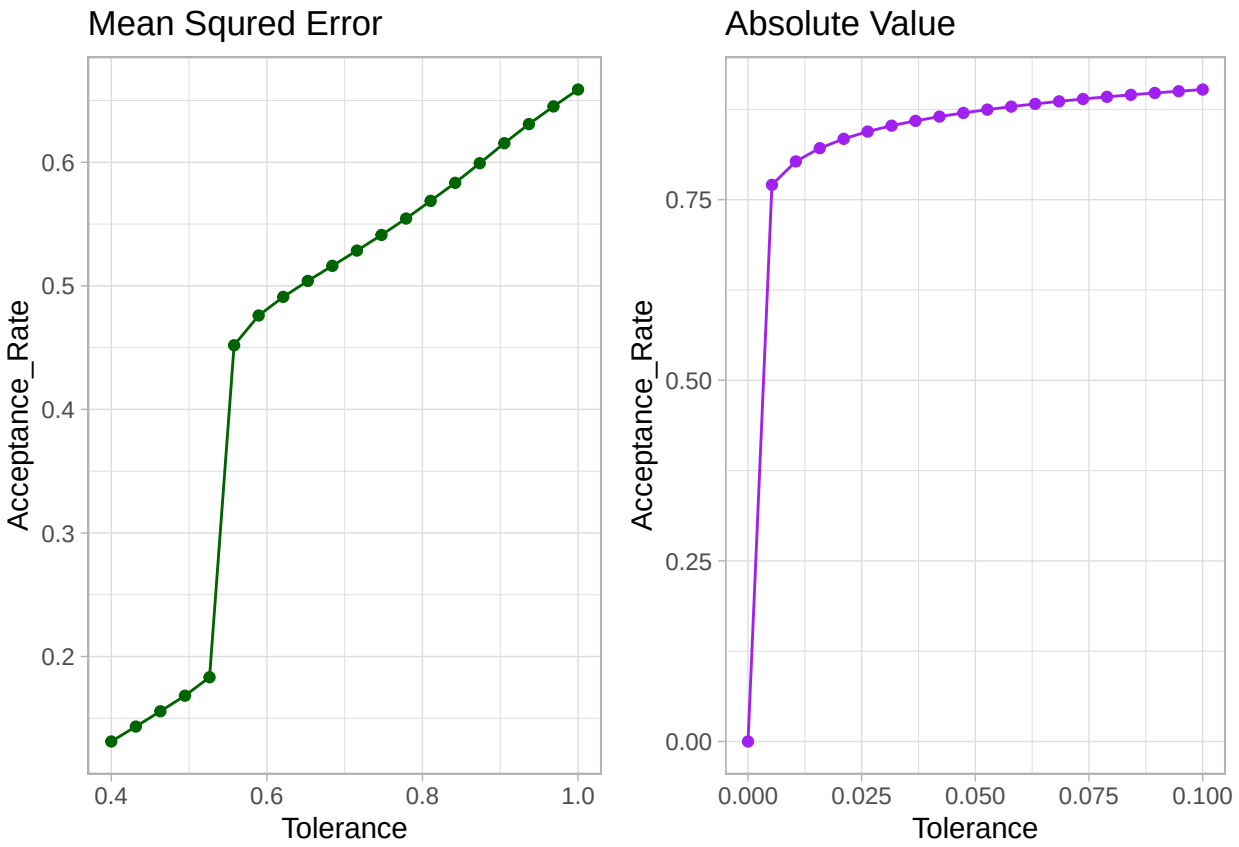
We can plot the acceptance rates for the two distance metrics as a function of the tolerance value.

```
eps <- seq(0.4, 1, length.out=20)
acceptance_rates <- numeric(length(eps))
for (i in 1:length(eps)) {
  accepted_samples <- ABC(1e6, eps[i], 2, x, 4, 1,NA)
  acceptance_rates[i] <- sum(accepted_samples[,1] != 0) / nrow(accepted_samples)
}
```

```
eps2 <- seq(0, 0.1, length.out=20)
acceptance_rates2 <- numeric(length(eps2))
for (i in 1:length(eps2)) {
  accepted_samples2 <- ABC(1e6, eps2[i], 2, x, 4, 2,NA)
    acceptance_rates2[i] <- sum(accepted_samples2[,1] != 0)
}
acceptance_rates2 <- acceptance_rates2 / nrow(accepted_samples)
```

```
msedata<-data.frame(Tolerance=eps,Acceptance_Rate=acceptance_rates)
absdata<-data.frame(Tolerance=eps2, Acceptance_Rate=acceptance_rates2)
mse_plot<-ggplot(msedata,aes(x=Tolerance,y=Acceptance_Rate))+
  geom_line(col='darkgreen') + geom_point(col='darkgreen')+
  theme_light() + ggtitle("Mean Squred Error")
```

```
abs_plot<-ggplot(absdata,aes(x=Tolerance,y=Acceptance_Rate))+
  geom_line(col='purple') + geom_point(col='purple')+
  theme_light() + ggtitle("Absolute Value")
plot_grid(mse_plot,abs_plot)
```

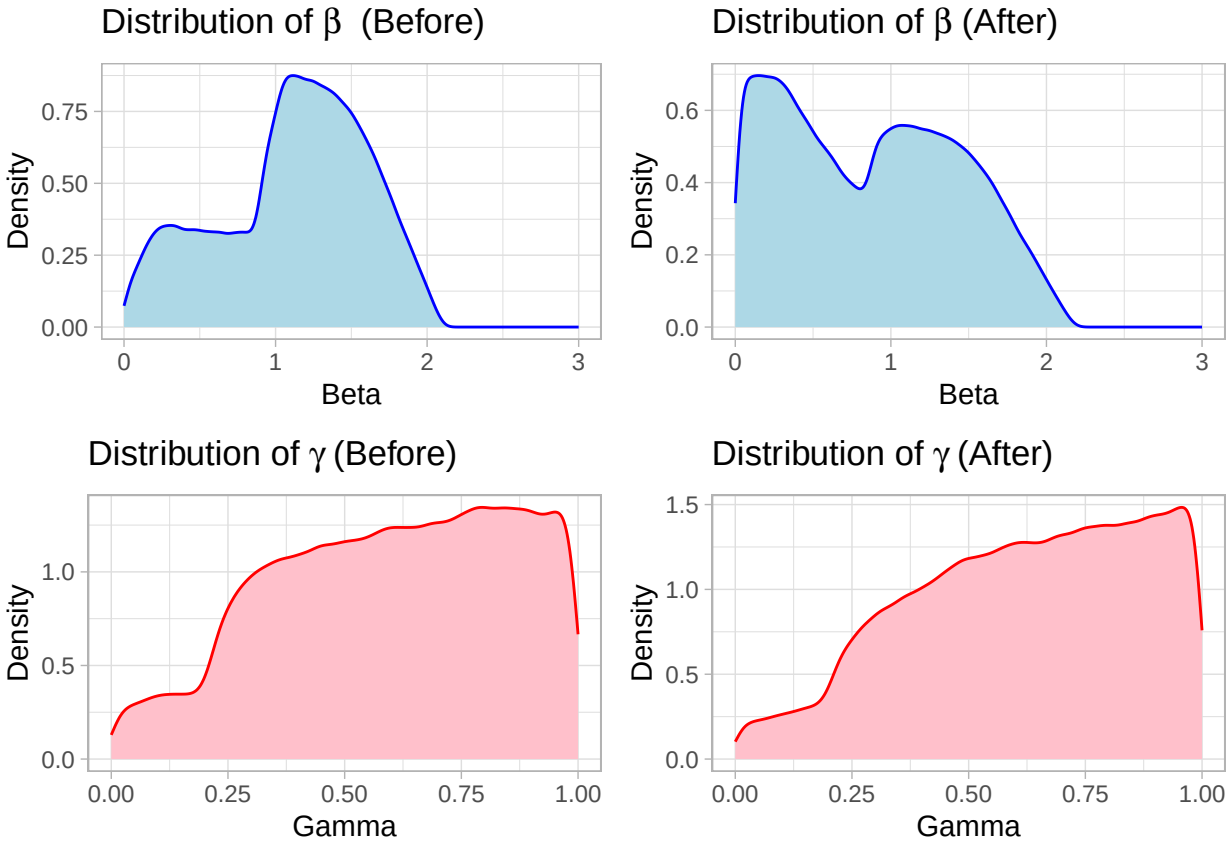


We see that there is a sharp transition in the acceptance rate in the mean squared error plot. Below we plot the marginal posterior distribution of β and γ at a tolerance before and after the transition.

```
accepted_samples_before <- ABC(1e6, 0.55, 2, x, 4, 1,NA)
accepted_samples_before <- accepted_samples_before[!rowSums(accepted_samples_before==0),]
accepted_samples_after <- ABC(1e6, 0.6, 2, x, 4, 1,NA)
accepted_samples_after <- accepted_samples_after[!rowSums(accepted_samples_after==0),]
```

```
before_beta_plot<-ggplot(aes(x=accepted_samples_before[,1]),
data = as.data.frame(accepted_samples_before))+
  geom_density(fill="lightblue",color="blue")+
  xlim(c(0,3))+
  labs(x="Beta",y="Density",
  title=expression(paste("Distribution of ", beta,"  (Before)")))+theme_light()

before_gamma_plot<-ggplot(aes(x=accepted_samples_before[,2]),
data = as.data.frame(accepted_samples_before))+
  geom_density(fill="pink",color="red")+
  xlim(c(0,1))+
  labs(x="Gamma",y="Density",
```

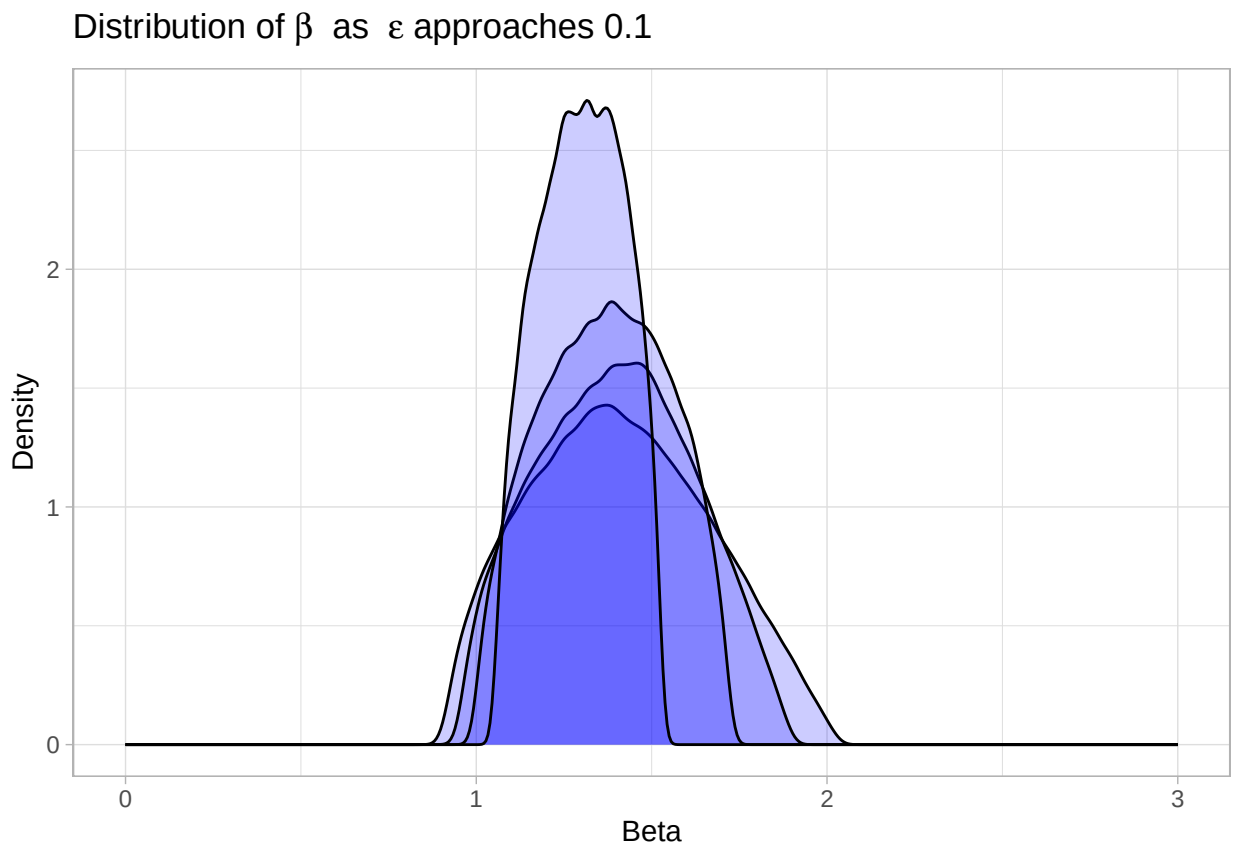
```
  title=expression(paste("Distribution of ", gamma," (Before)")))+theme_light()

after_beta_plot<-ggplot(aes(x=accepted_samples_after[,1]),
data = as.data.frame(accepted_samples_after))+
  geom_density(fill="lightblue",color="blue")+
  xlim(c(0,3))+
  labs(x="Beta",y="Density",
  title=expression(paste("Distribution of ", beta, " (After)")))+theme_light()

after_gamma_plot<-ggplot(aes(x=accepted_samples_after[,2]),
data = as.data.frame(accepted_samples_after))+
  geom_density(fill="pink",color="red")+
  xlim(c(0,1))+
  labs(x="Gamma",y="Density",
  title=expression(paste("Distribution of ", gamma, " (After)")))+theme_light()

plot_grid(before_beta_plot,after_beta_plot,before_gamma_plot,
after_gamma_plot,nrow=2,ncol=2)
```



Indeed, as we decrease the value of ϵ, the ABC-estimated posterior distribution of β and γ diverges further and further from the uniform priors $U[0,3]$, $U[0,1]$, and converges towards a distribution centered around the MAP (maximum a priori) estimate.

We can see this in the plot below, which begins with a tolerance value of 0.4, producing a wider plot, and with each decrease in ϵ, more and more mass is shifted towards the 'true' value. Whilst not included here, the same is true for γ also.

```
epsilons <- c(0.5,0.4,0.3,0.2)
datas <- list(c(),c(),c(),c())
for (i in 1:4){
  sims <- ABC(1e6, epsilons[i], 2, x, 4, 1,NA)
  datas[[i]] <- sims[!rowSums(sims==0),][,1]
}

ggplot(aes(x=datas[[1]]),data=as.data.frame(datas[[1]]))+
  geom_density(fill='blue',alpha=.2)+
  geom_density(data=as.data.frame(datas[[2]]),
  aes(x=datas[[2]]),fill='blue',alpha=.2)+
  geom_density(data=as.data.frame(datas[[3]]),
  aes(x=datas[[3]]),fill='blue',alpha=.2)+
  geom_density(data=as.data.frame(datas[[4]]),
  aes(x=datas[[4]]),fill='blue',alpha=.2)+
  xlim(c(0,3))+
  labs(x="Beta",y="Density",
  title=expression(paste("Distribution of ", beta,"  as  ",
  epsilon, " approaches 0.1")))+
  theme_light()
```



Parameter Estimates

We will now simulate the SIR model using the optimal values of β and γ. To best gain these optimal point estimates from our posterior distributions produced by ABC, we use a method of kernel density estimation.

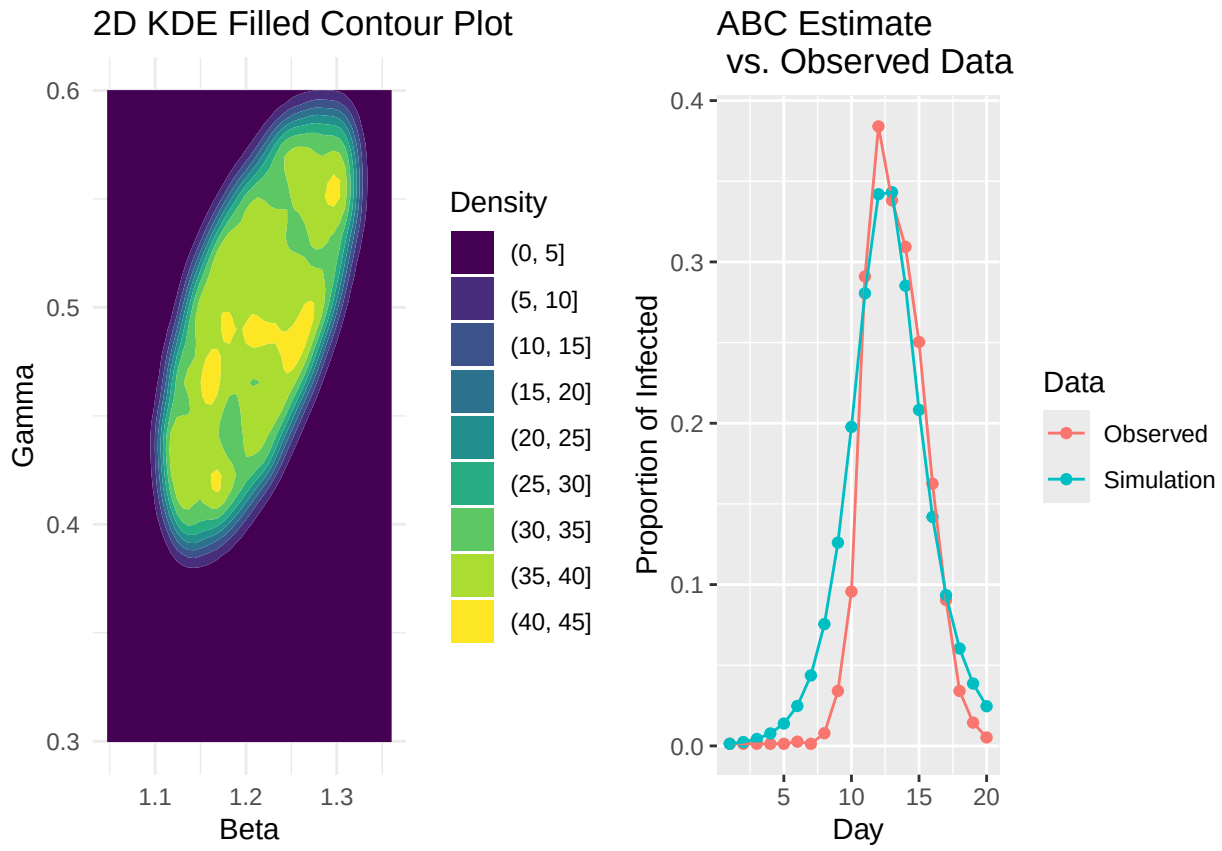
Below is a function which calculates these estimates for $\hat{\beta}$ and $\hat{\gamma}$, then is able to plot the associated heatmap used to find these posterior maximisers, as well as the fit to the observed data produced when inputting these estimates into the SIR simulator.

```
library(MASS)
library(ks)

Optimal_Plot<-function(sims,plot.heat){

  colnames(sims) <- c("beta", "gamma")

  kde_result <- kde2d(sims[,1], sims[,2], n=50)

  kde_df <- data.frame(
  x = rep(kde_result$x, each = length(kde_result$y)),
  y = rep(kde_result$y, times = length(kde_result$x)),
  z = as.vector(kde_result$z)
  )

  heat_plot <- ggplot(kde_df, aes(x = x, y = y, z = z)) +
  geom_contour_filled() +
  labs(title = "2D KDE Filled Contour Plot",
       x = "Beta",
       y = "Gamma",
       fill = "Density") +
  theme_minimal()

  # Extract KDE results
  x <- kde_result$x
  y <- kde_result$y
  z <- kde_result$z

  # Find the indices of the maximum density value
  max_density_index <- which(z == max(z), arr.ind = TRUE)

  # Get the corresponding x and y values for the maximum density
  max_x <- x[max_density_index[1]]
  max_y <- y[max_density_index[2]]

  I <- SIR(numeric(20), numeric(20), numeric(20),
  1 - 1/763, 1/763, 0, max_x, max_y)

  obs_data <- c(rep(1/763,5), 2/763, bsflu$B/763)

  fit_plot <- ggplot(data=data.frame(z=c(seq(1,20),seq(1,20)),
  y=c(obs_data,I),Data=c(rep("Observed",20  ),rep("Simulation",20))),
  aes(x=z,y=y,col=Data)) +
    geom_point() + geom_line() + xlab("Day") +
    ylab("Proportion of Infected") +
    ggtitle("ABC Estimate \n vs. Observed Data")

  if(plot.heat){

    plot_grid(heat_plot,fit_plot)
```

```
  }
  else{

    return(fit_plot)

  }
}
```

As an example therefore, we can use the mean squared error distance metric, along with a very small tolerance value of 0.01 to produce this fit to the data.

```
accepted_samples <- ABC(1e6, .1, 2, x, 4, 1,NA)
accepted_samples <- accepted_samples[!rowSums(accepted_samples==0),]


Optimal_Plot(accepted_samples,TRUE)
```



By observing the contour plot, we can see that the optimal parameters using this method are approximately $\hat{\gamma} = 0.45$, and $1.2 < \hat{\beta} < 1.3$. Then by plugging these into the SIR simulator, we get a simulation that matches the observed data well. However it is worth noting that to achieve such a good fit required an incredibly low acceptance rate: only 0.86%. This means that a high number of parameter samples are rejected, which is computationally costly.

Extension: Summary Statistics

One other distance metric we could have used was that of looking at the difference between a handful of *summary statistics* of the data, rather than the raw data itself. This reduces the dimensionality of the problem and can reduce the amount of simulations needed for a good fit.

The example we use here looks at the empirical mean and variance of the distribution, as well as the 'peak' of the distribution. I.e. for observed data points (x_i, y_i) where x_i is normalised to be a probability,

$$\mu = \sum_{i=1}^{N} x_i y_i$$

$$\sigma^2 = \sum_{i=1}^{N} y_i (\mu - x_i)^2$$

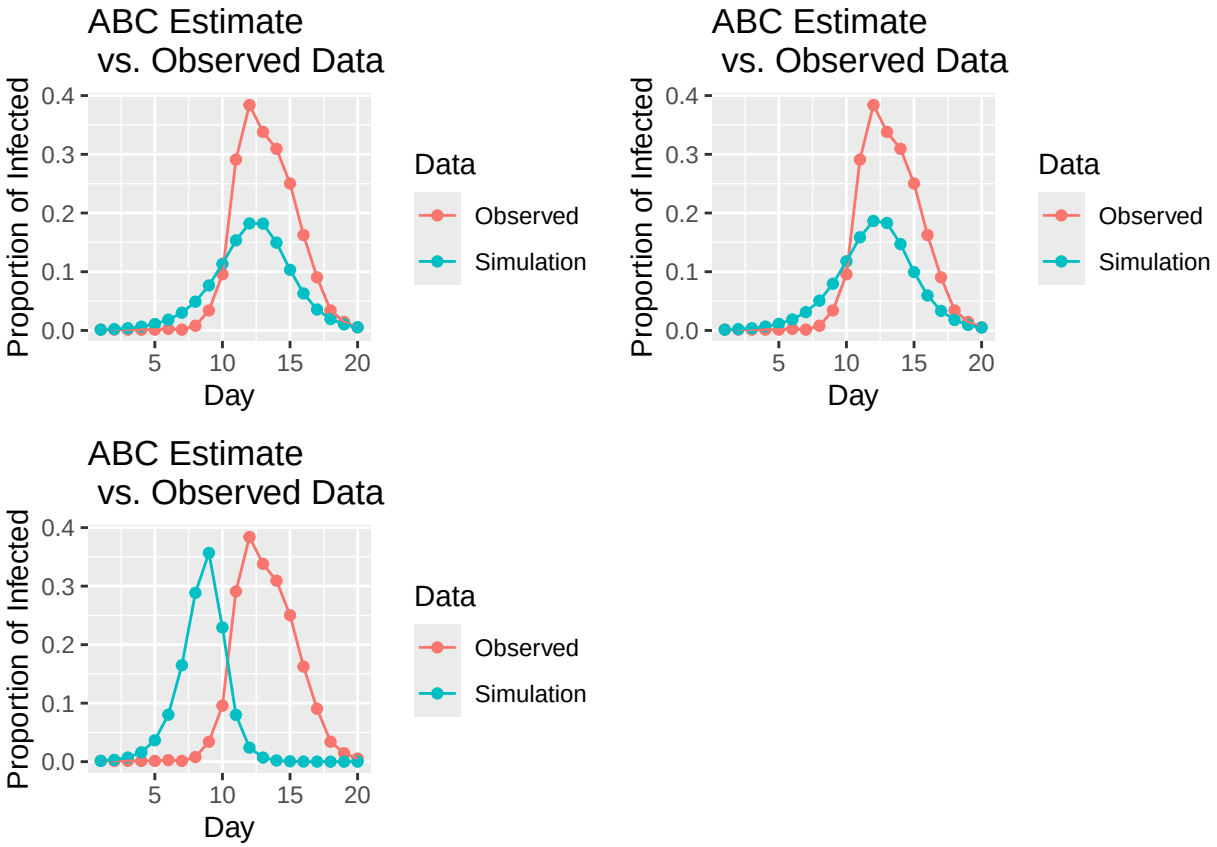
Then by observing $d_{obs} = (x_{obs}, y_{obs})$ and simulating $d_{sim} = (x_{sim}, y_{sim})$, we define our distance metric to be:

$$D(d_{obs}, d_{sim}) = w_1 \frac{(\mu_{obs} - \mu_{sim})^2}{\mu_{obs}^2} + w_2 \frac{(\sigma_{obs} - \sigma_{sim})^2}{\sigma_{obs}^2} + w_3 \frac{(max(d_{obs}) - max(d_{sim}))^2}{max(d_{obs})^2}$$

Where w_1, w_2, w_3 are the weights assigned to each distance measure.

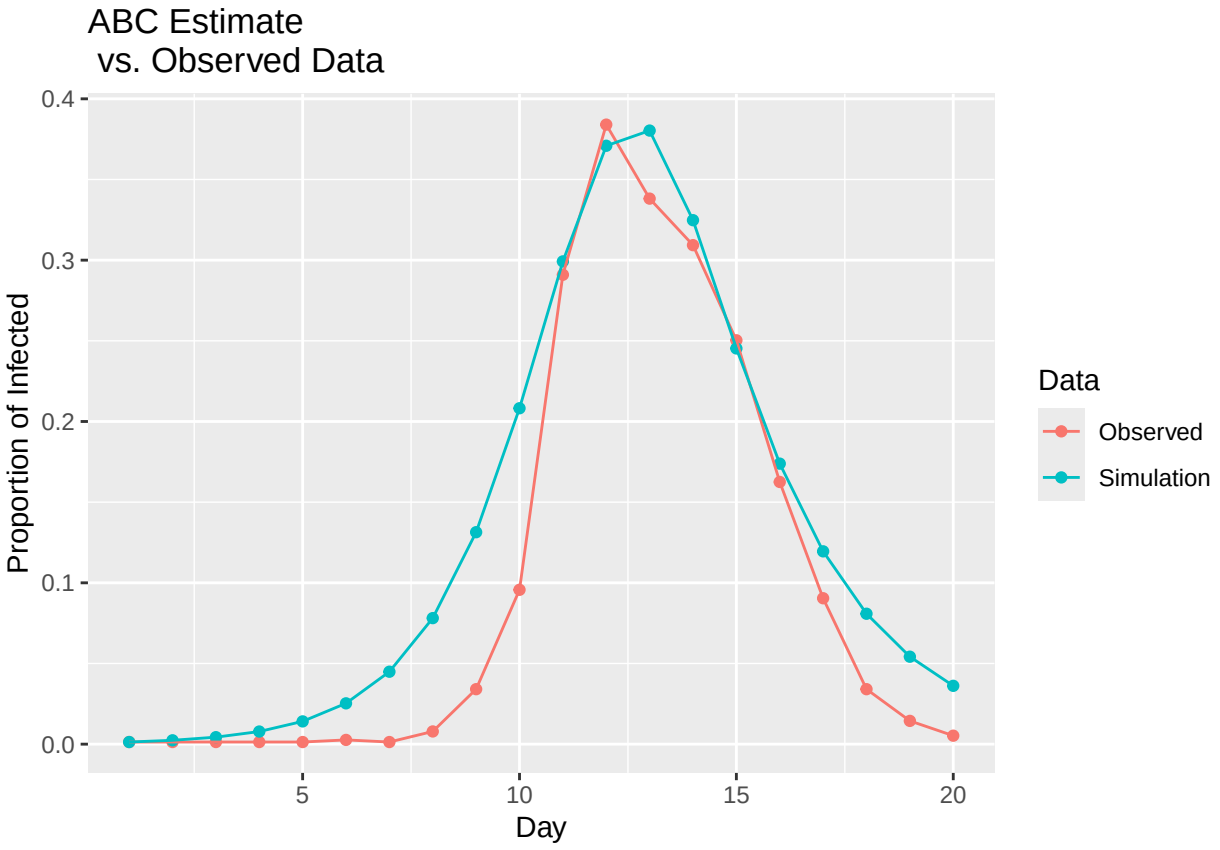
Then by using a different set of weights, we can assign more importance to the matching of mean, variance, and whether the peaks of the distributions coincide. For example, by prioritizing each we obtain the plots below, where the mean weighted one matches the mean perfectly, but at the expense of standard deviation. The estimate weighted by peak however matches the peak and shape of distribution correctly, but with the wrong mean.

```
weights <- list(c(2,.2,.2),c(.2,1,.2),c(.2,.2,2))
plots <- list(before_gamma_plot,before_beta_plot,before_beta_plot)

for (i in 1:3){

  accepted_samples <- ABC(1e6, .4, 2, x, 4, 3, weights[[i]])
  accepted_samples <- accepted_samples[!rowSums(accepted_samples==0),]
  plots[[i]] <- Optimal_Plot(accepted_samples,FALSE)

}

plot_grid(plots[[1]],plots[[2]],plots[[3]])
```



Then by choosing an appropriate set of weights and a low value of ϵ, we can achieve a fit to the observed data of similar quality to that which we got using the mean squared error of raw data. The difference here is the acceptance probability, while low, is much higher than that case: 4% as opposed to 0.8%. Hence requiring less computational power due to fewer simulations required.

```
accepted_samples <- ABC(1e6, .05, 2, x, 4, 3, c(1,.1,.5))
accepted_samples <- accepted_samples[!rowSums(accepted_samples==0),]
Optimal_Plot(accepted_samples,FALSE)
```

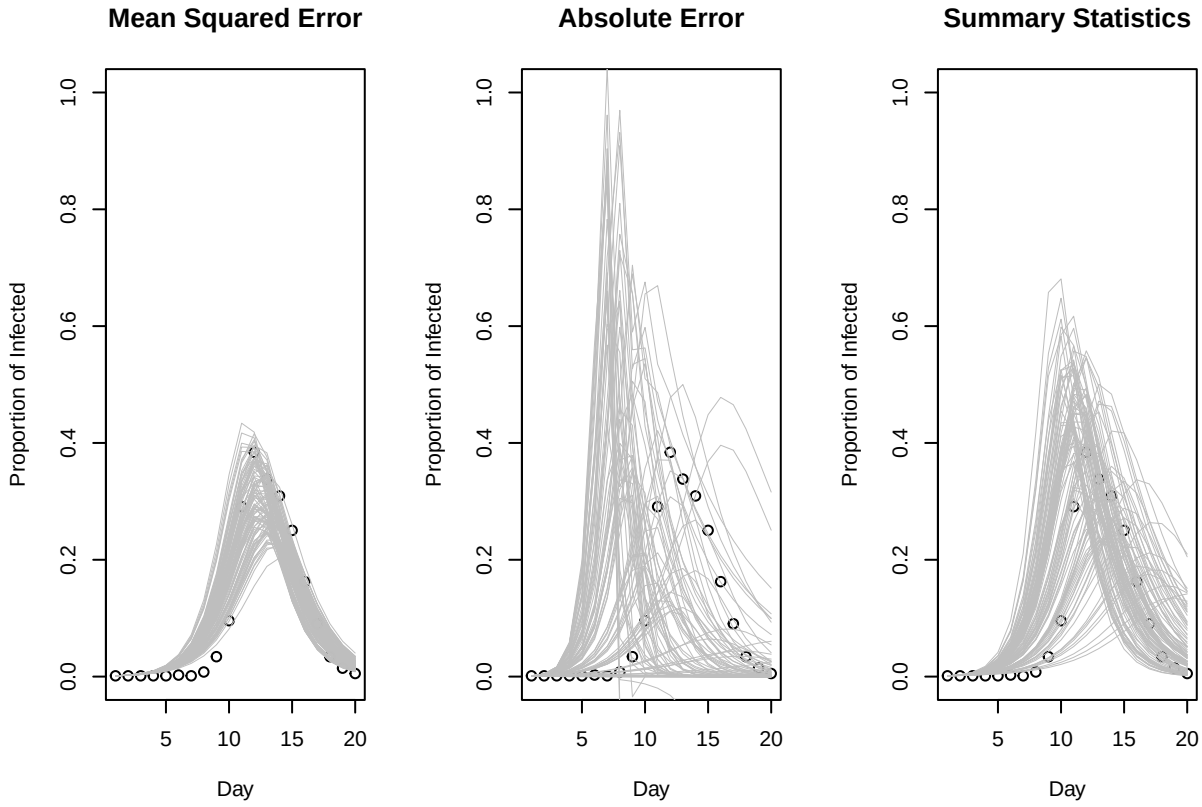


Final Results

Finally, we can sample randomly from the posterior distribution of β and γ to produce a range of simulations to fit the infection curve for each metric.

```
# set up 3 horizontally adjacent plots
par(mfrow=c(1,3))

n_samples <- 100

# mean squared metric
plot(x, xlim=c(1,20), ylim=c(0,1), xlab="Day",
ylab="Proportion of Infected", main="Mean Squared Error")
accepted_samples <- ABC(1e6, .1, 2, x, 4, 1,NA)
accepted_samples <- accepted_samples[!rowSums(accepted_samples==0),]
for (i in 1:n_samples) {
  beta <- sample(accepted_samples[,1], 1)
  gamma <- sample(accepted_samples[,2], 1)
  I <- SIR(numeric(20), numeric(20), numeric(20),
  1 - 1/763, 1/763, 0, beta, gamma)
  lines(I, type = "l", col = "grey", xlab = "Day",
  ylab = "Proportion of Infected", main = "Mean Squared Error", lwd=0.5)
}

# absolute error metric
```

```r
plot(x, xlim=c(1,20), ylim=c(0,1), xlab="Day", ylab="Proportion of Infected", main="Absolute Error")
accepted_samples <- ABC(1e6, .005, 2, x, 4, 2,NA)
accepted_samples <- accepted_samples[!rowSums(accepted_samples==0),]
for (i in 1:n_samples) {
  beta <- sample(accepted_samples[,1], 1)
  gamma <- sample(accepted_samples[,2], 1)
  I <- SIR(numeric(20), numeric(20), numeric(20),
  1 - 1/763, 1/763, 0, beta, gamma)
  lines(I, type = "l", col = "grey", xlab = "Day",
  ylab = "Proportion of Infected", main = "Mean Squared Error", lwd=0.5)
}

# summary statistics metric
plot(x, xlim=c(1,20), ylim=c(0,1), xlab="Day",
ylab="Proportion of Infected", main="Summary Statistics")
accepted_samples <- ABC(1e6, .05, 2, x, 4, 3, c(1,.1,.5))
accepted_samples <- accepted_samples[!rowSums(accepted_samples==0),]
for (i in 1:n_samples) {
  beta <- sample(accepted_samples[,1], 1)
  gamma <- sample(accepted_samples[,2], 1)
  I <- SIR(numeric(20), numeric(20), numeric(20),
  1 - 1/763, 1/763, 0, beta, gamma)
  lines(I, type = "l", col = "grey", xlab = "Day",
  ylab = "Proportion of Infected", main = "Mean Squared Error", lwd=0.5)
}
```



In these plots we see that the mean squared error metric produces a range of simulations that fit the observed data well, but the absolute error metric produces a range of simulations that do not fit the observed data well. The summary statistics metric produces a range of simulations that fit the observed data well, but not as well as the mean squared error metric. However, the mean squared error metric is unable to capture the behaviour of the model at the start of the outbreak, where the second infection occurs 6 days after the first.

Conclusion

In conclusion, we have implemented the ABC algorithm to estimate the parameters of the SIR model for an influenza outbreak in a boarding school. We have shown that the choice of distance metric and tolerance value can have a significant impact on the acceptance rate and the quality of the fit to the observed data. We have also shown that using summary statistics can reduce the dimensionality of the problem and improve the acceptance rate. We found that the mean squared error distance metric produced the most consistently accurate fits to the data, but at the cost of not properly modelling the start of the infection. Perhaps this suggets that the infection rate parameter should be allowed to vary in time in order to reflect some aspects of the data that the SIR model cannot. An example of this might be that the first infected boy did not come into contact with many people until a few days into the model, so the infection rate is low to begin with. Finally, we have demonstrated how to use kernel density estimation to estimate the optimal parameters of the SIR model and produce a range of simulations to fit the infection curve. In future work, we could test different priors for the parameters, and different simulation-based inference techniques such as MCMC-ABC and SMC-ABC; these techniques are known to help to mitigate the problem of having a high rejection rate in the standard ABC algorithm, which we encountered in this project. Given more data, we could also consider using a more complex model, such as the SEIR model, which includes an exposed compartment to account for the incubation period of the disease, or spatially explicit models to account for the spatial spread of the disease.

References

[1] J.-M. Marin, P. Pudlo, C. P. Robert, and R. J. Ryder, "Approximate bayesian computational methods" Stat Comput, vol. 22, pp. 1167–1180, Nov. 2012.

[2] A. Minter, and R. Retkute, "Approximate Bayesian Computation for infectious disease modelling". Epidemics, vol. 29, p.100368.