Stochastic SIR Forecasting Showdown Part 1: Parameter Fitting

Dexter Barrows

21:33 14 December 2015

Setup

Now that we have established which methods we wish to evaluate the efficacy of for epidemic forecasting, it is prudent to see how they perform when fitting parameters for a known epidemic model. We have already seen how they perform when fitting parameters for a model with a deterministic evolution process and observation noise, but a more realistic model will have both process and observation noise.

To form such a model, we will take a deterministic SIR ODE model given by

$$\begin{aligned}\frac{dS}{dt} &= -\beta SI \\ \frac{dI}{dt} &= \beta SI - \gamma \\ \frac{dR}{dt} &= \gamma I,\end{aligned}$$

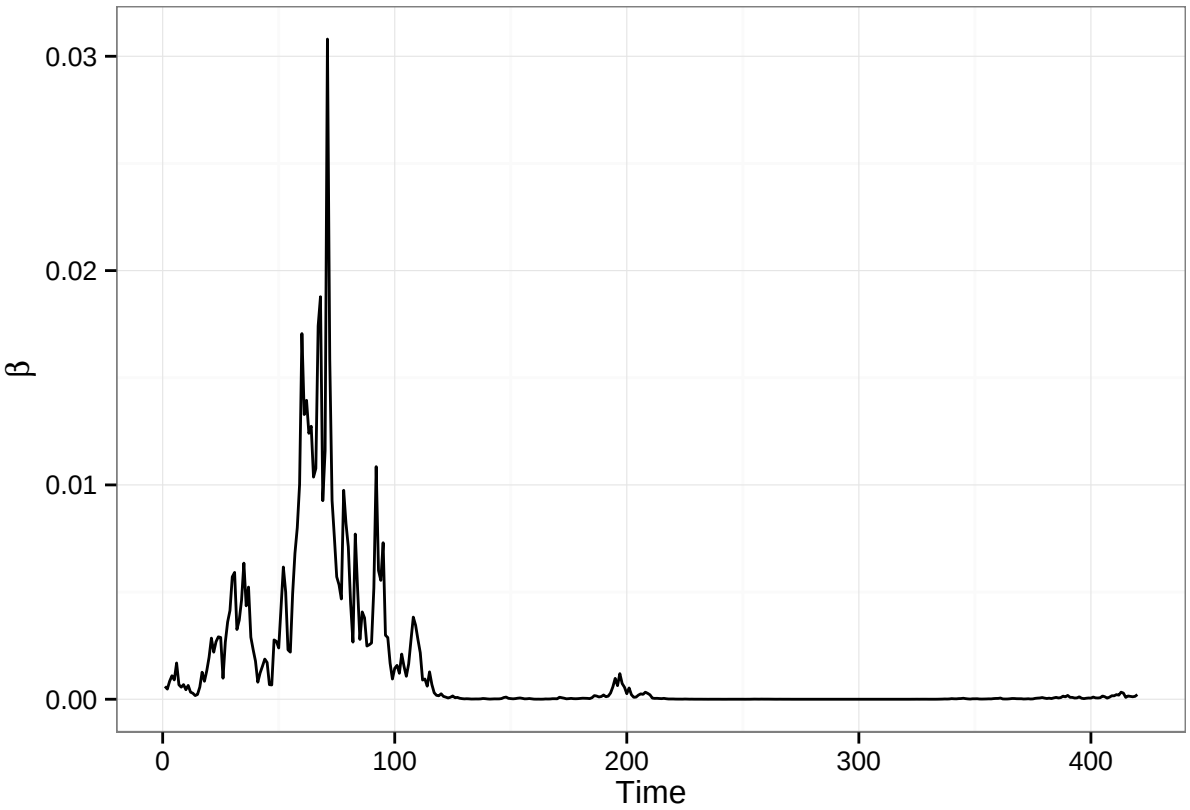
and add process noise by allowing β to embark on a geometric random walk given by

$$\beta_{t+1} = \exp\left(\log(\beta_t) + \eta(\log(\bar{\beta}) - \log(\beta_t)) + \epsilon_t\right).$$

We will take ϵ_t to be normally distributed with variance ρ such that $\epsilon_t \sim \mathcal{N}(0, \rho)$. The geometric attraction term constrains the random walk, the force of which is $\eta \in [0, 1]$. If we take $\eta = 0$ then the walk will be unconstrained; if we let $\eta = 1$ then all values of β_t will be independent from the previous value (and consequently all other values in the sequence).

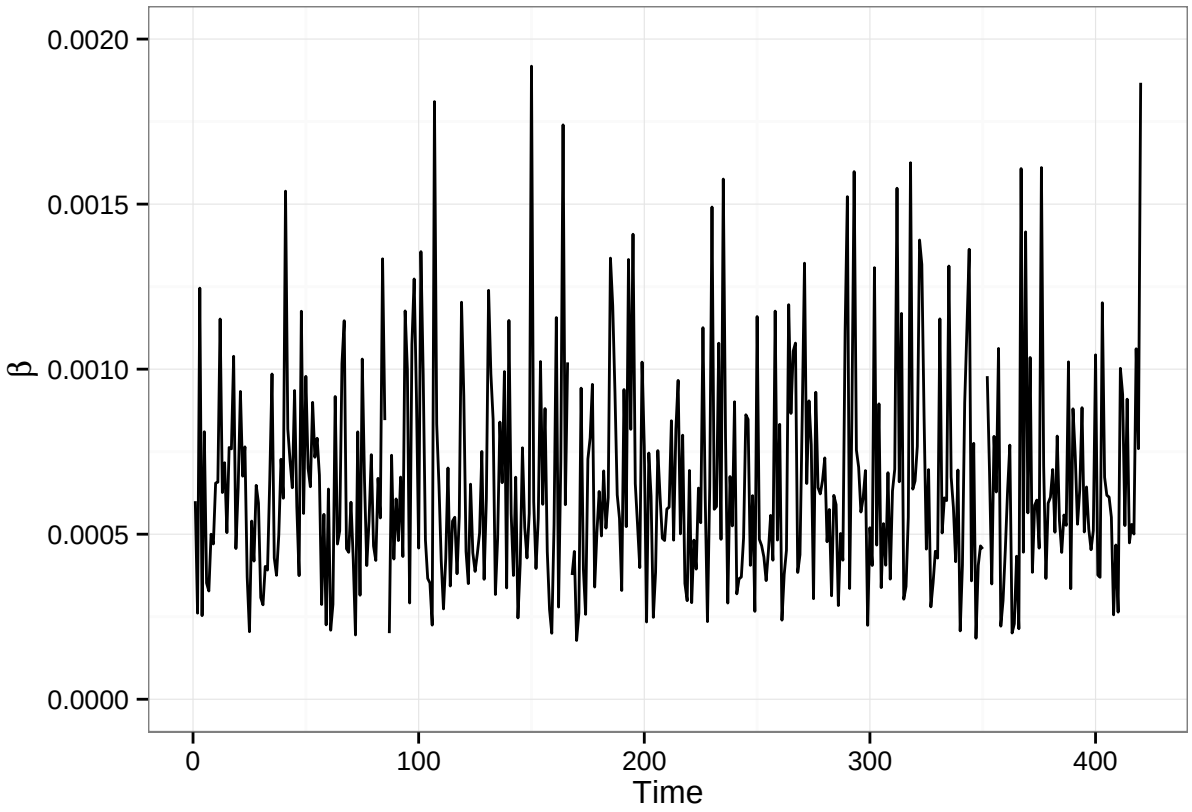
We can observe the effect of η visually by setting η to each extreme value and then to an intermediate value and plotting the results. The anchoring value β_0 was set to $\beta_0 = R_0 \cdot r/N$ where $R_0 = 3.0$, $r = 0.1$ and $N = 500$, giving $B_0 = 6 \times 10^{-4}$. The noise parameter ρ was set to 0.5.

For $\eta = 0$ we have

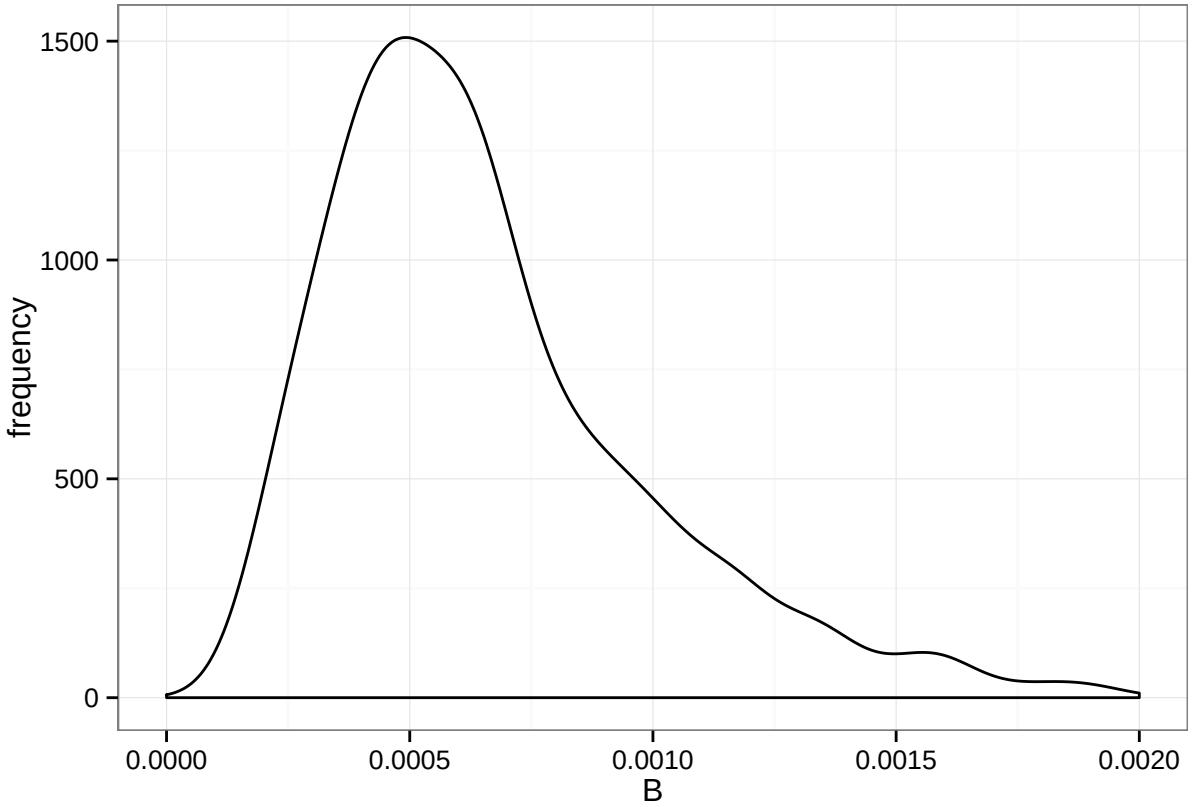


Here we can see how the unconstrained walk allows the sequence to climb several orders of magnitude higher than the starting value, which is undesirable for the model formulation and unrealistic from a biological perspective.

Now if we set $\eta = 1$ we have



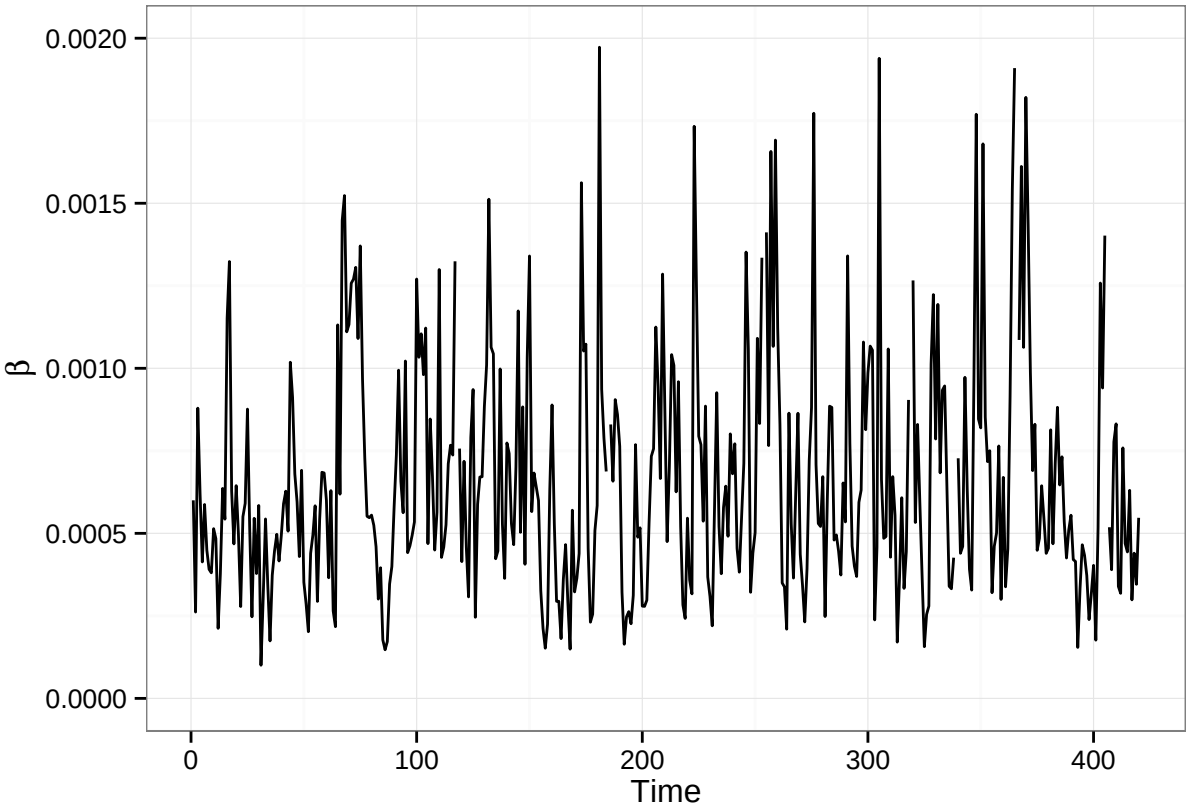
and corresponding density plot



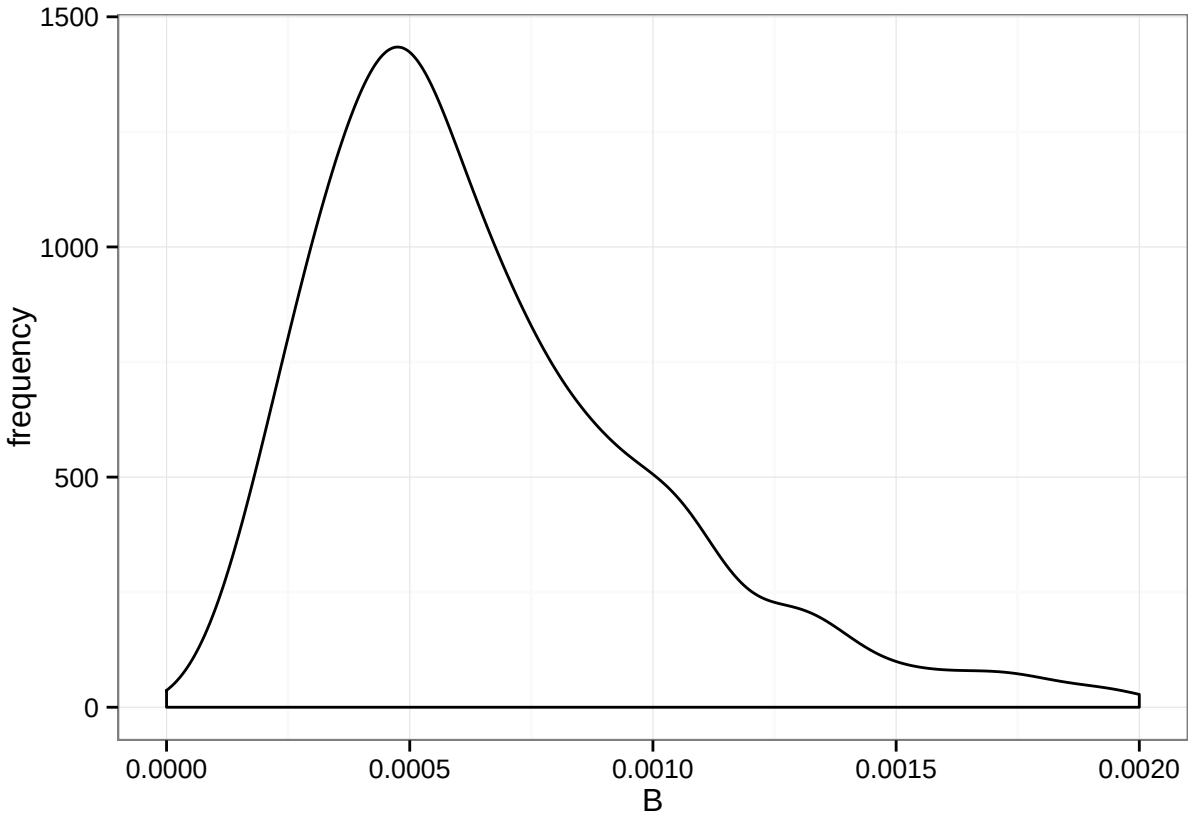
Notice that we have frequent oscillations between smaller and larger values within only a few

time steps, but the range of values is desirable.

Finally choosing an intermediate value of $\eta = 0.5$ gives us

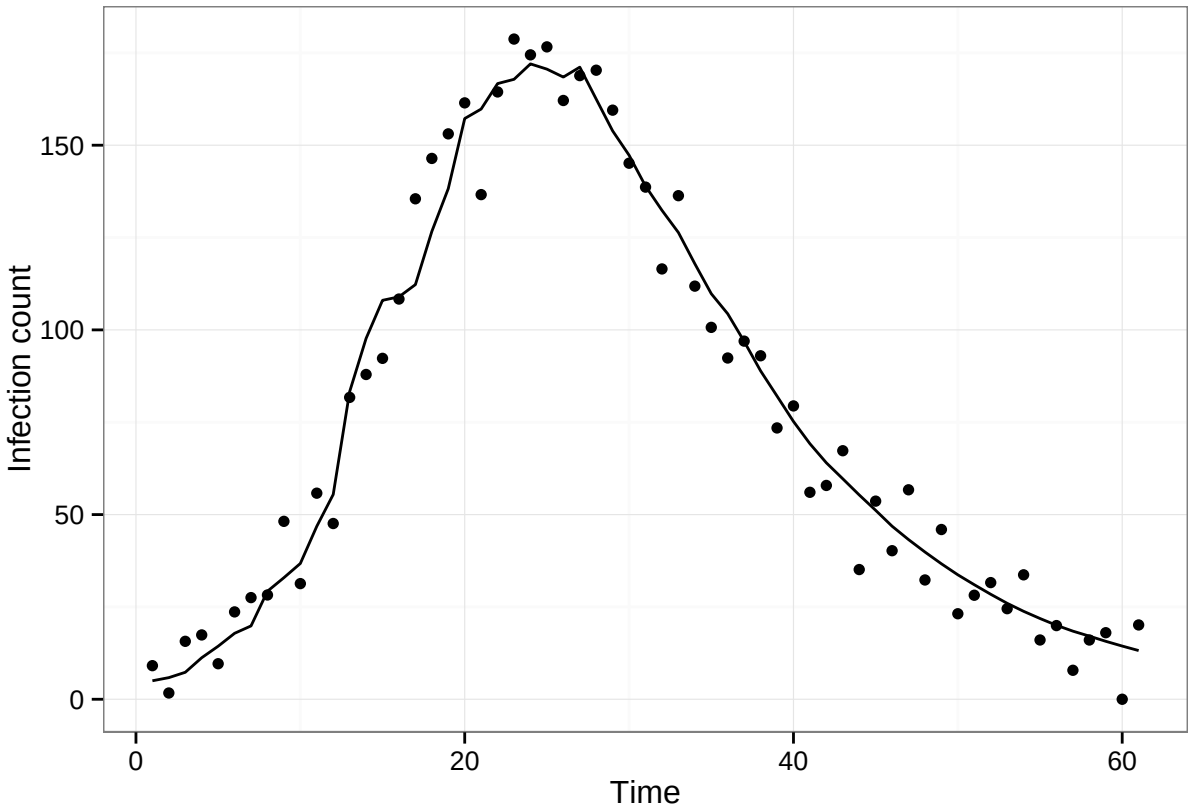


and corresponding density plot

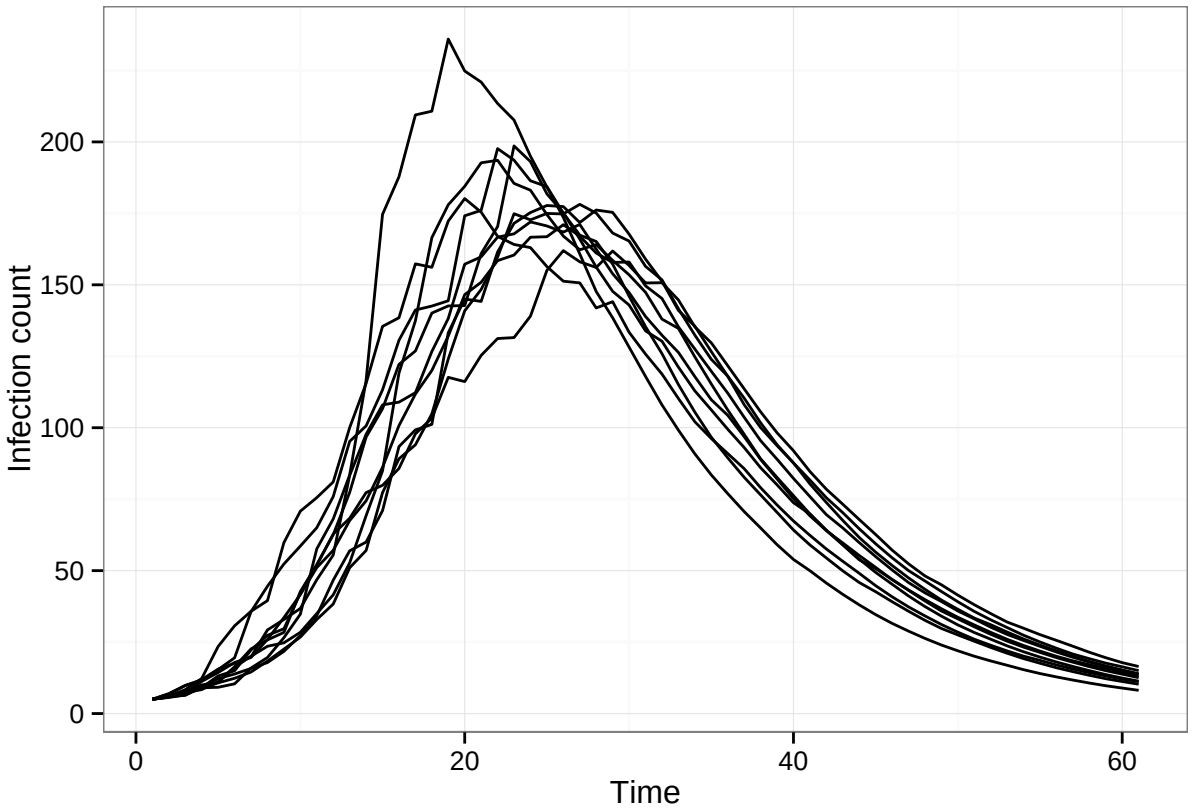


Now we see a density plot similar in shape to the desired density, but now the geometric random walk displays dependence on previous values.

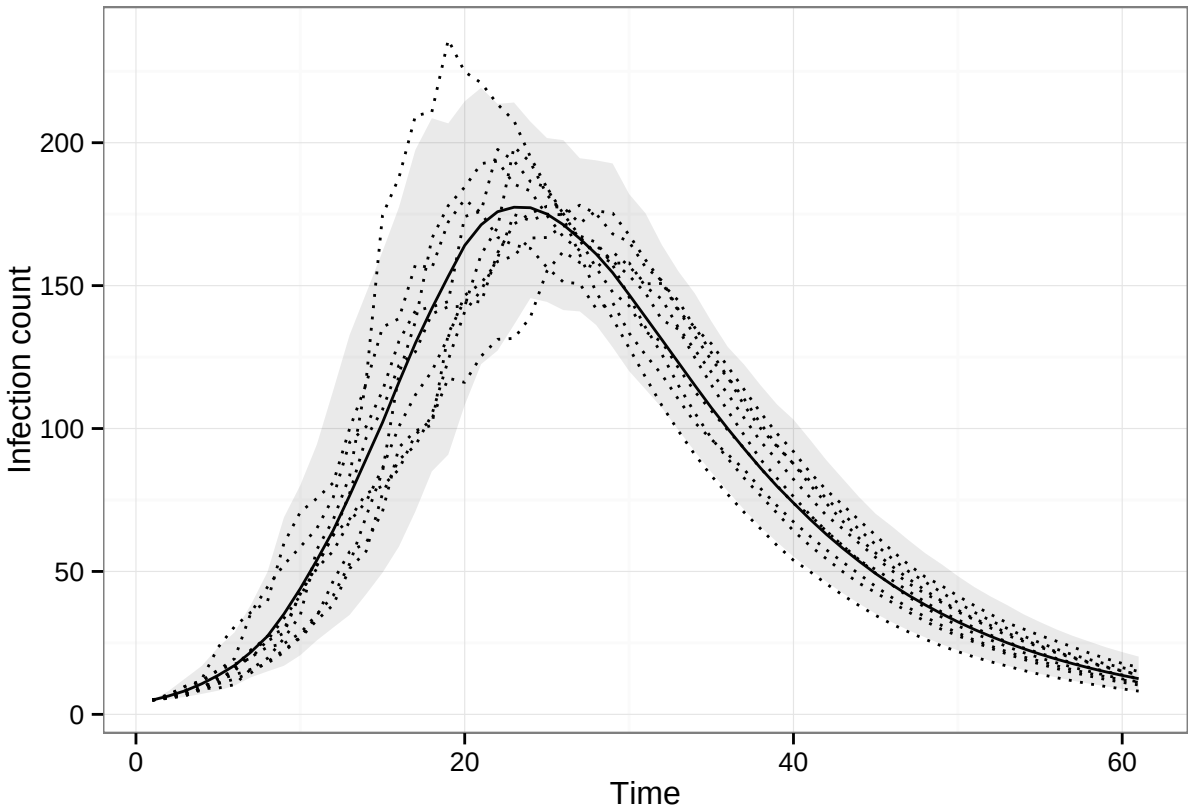
If we take the full stochastic SIR system and evolve it using an Euler stepping scheme with a step size of $h = 1/7$, for 1 step per day, we obtain the following plot



where the solid line is the true trajectory and the dots show the data points obtained by adding in observation error defined as $\epsilon_{obvs} = \mathcal{N}(0, 10)$. Note that this is only one possible realization of the system, and that different random numbers drawn from the system's random number generator will produce varying results. The plot below shows 9 additional example trajectories in addition to the one shown in the previous plot



We can see the average behaviour of the system by taking the average over 100 trajectories. This is shown below with the 10 sample trajectories from the previous plot overlayed.



Here the solid line is the mean behaviour, the dotted lines are the sample trajectories from the previous plot, and the grey ribbon is centre 95th quantile.

IF2 fitting

Now we will use an implementation of the IF2 algorithm to attempt to fit the stochastic SIR model to the data from Figure [?]. The goal here is just parameter inference, but since IF2 works by applying a series on particle filters we essentially get the average system state estimates for a very small additional computational cost. Hence, we will will also look at that estimated behaviour in addition the the parameter estimates.

The code used here is a mix of R and C++ implemented using RCpp. The fitting was undertaken using 10^4 particles with 10 IF2 passes and a cooling schedule given by a reduction in particle spread determined by 0.8^p, where p is the pass number starting with 0.

The total runtime for the fitting was determined using the R `system.time()` function, and yielded

```
##    user  system elapsed
##  32.608   0.346  33.304
```

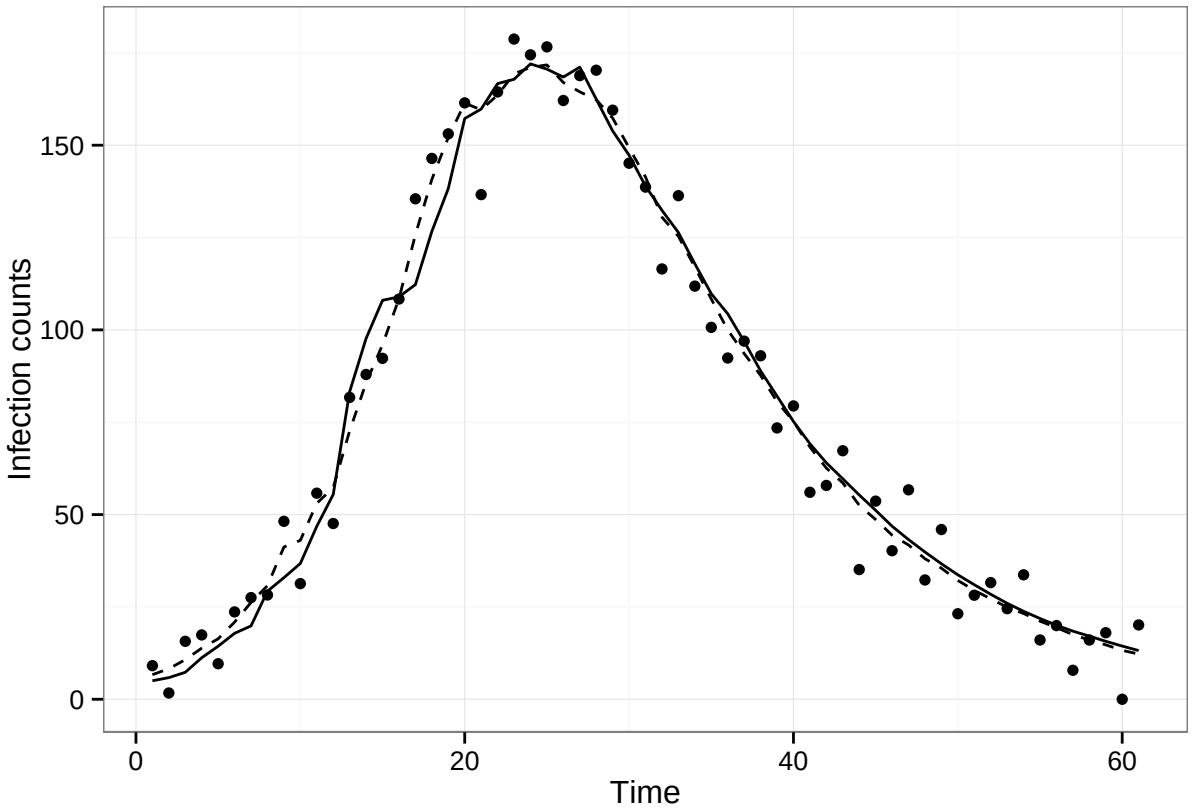
The MLE parameter estimates, taken to be the mean of the particle swarm values after the final pass was

```
##        R0         r        I0     sigma       eta      berr
## 3.3250544 0.1026381 6.7121591 8.8374208 0.7592928 0.1359986
```

giving a relative error of

```
##          R0           r          I0       sigma         eta        berr
##  0.10835146  0.02638106  0.34243183 -0.11625792  0.51858566 -0.72800271
```

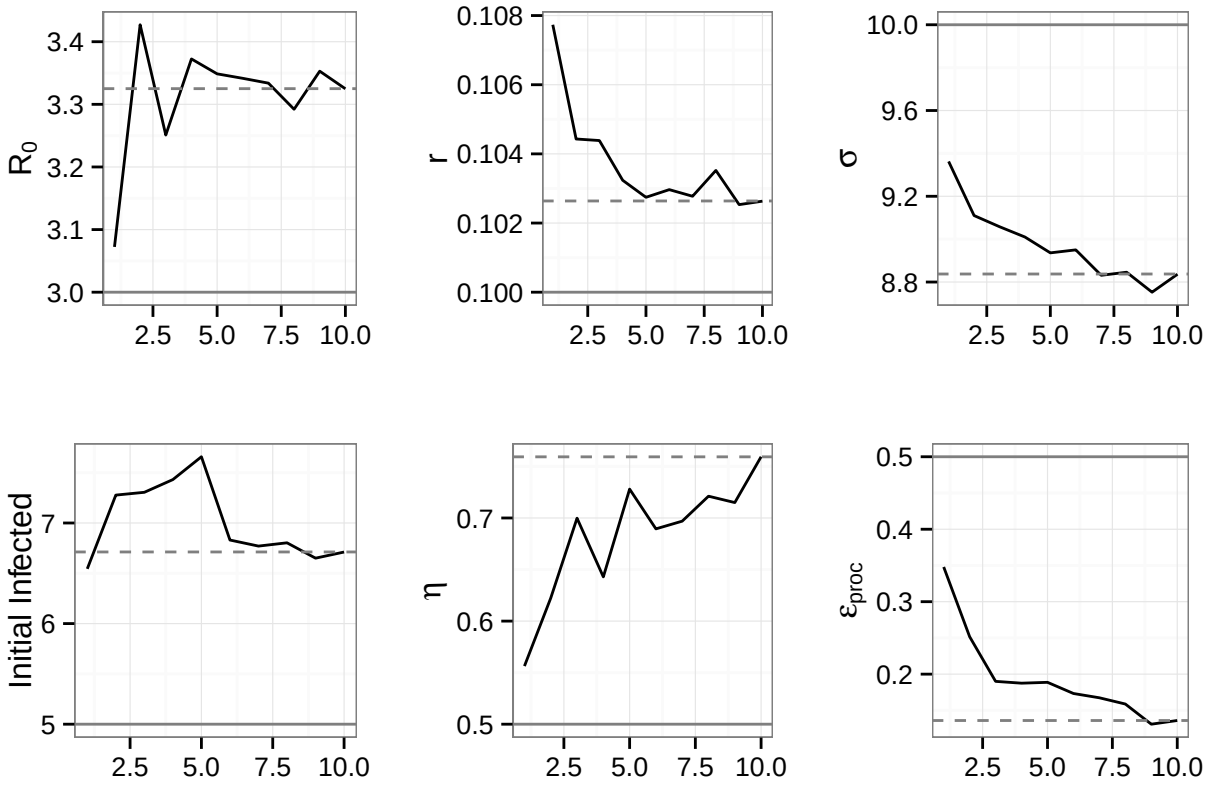
From last IF2 particle filtering iteration, the mean state values from the particle swarm at each time step are shown with the true underlying state and data in the plot below



where the solid line shows the true states, the points are the data, and the dashed line shows the state estimates from the last IF2 filtering pass.

IF2 convergence plots

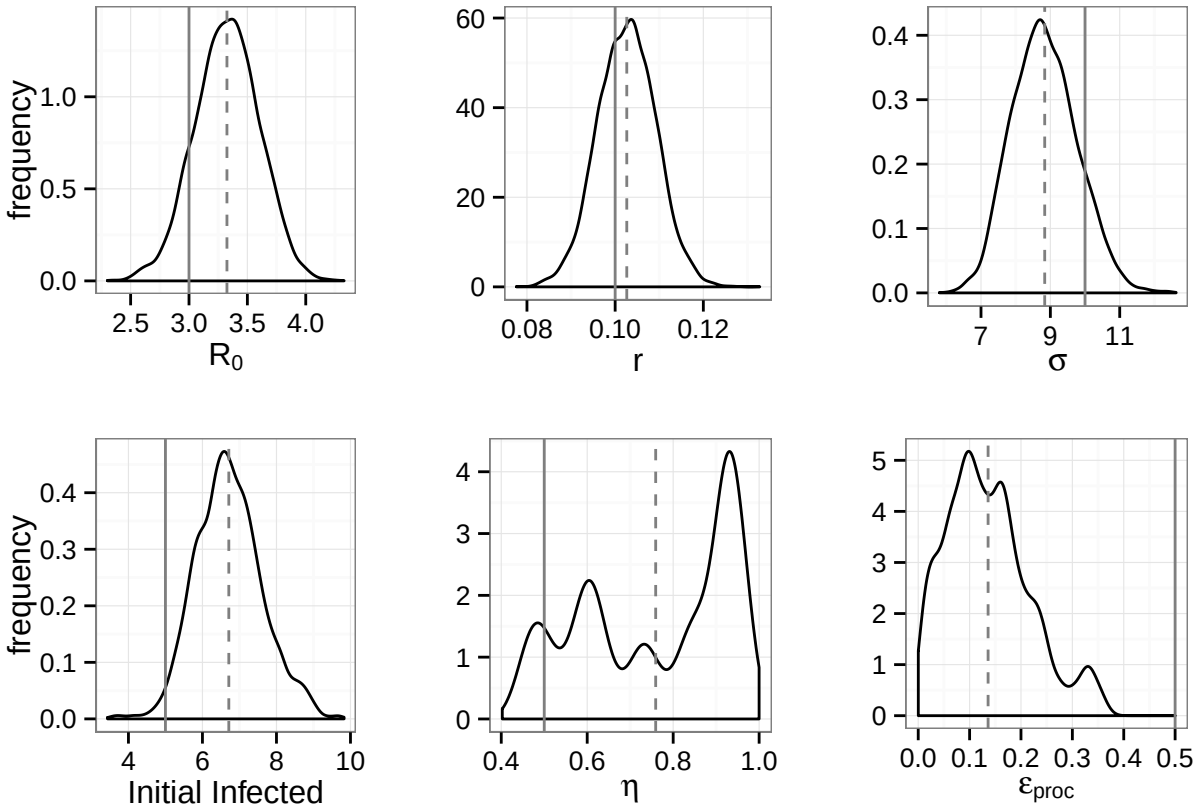
Since IF2 is an iterative algorithm where each pass through he data is expected to push the parameter estimates towards the MLE, we can see the evolution of these estimates as a function of the pass number. Plots showing this evolution are shown below for the six most critical parameters



The horizontal axis shows the IF2 pass number. The solid black lines show the evolution of the ML estimates, the solid grey lines show the true value, and the dashed grey lines show the mean parameter estimates from the particle swarm after the final pass.

IF2 density plots

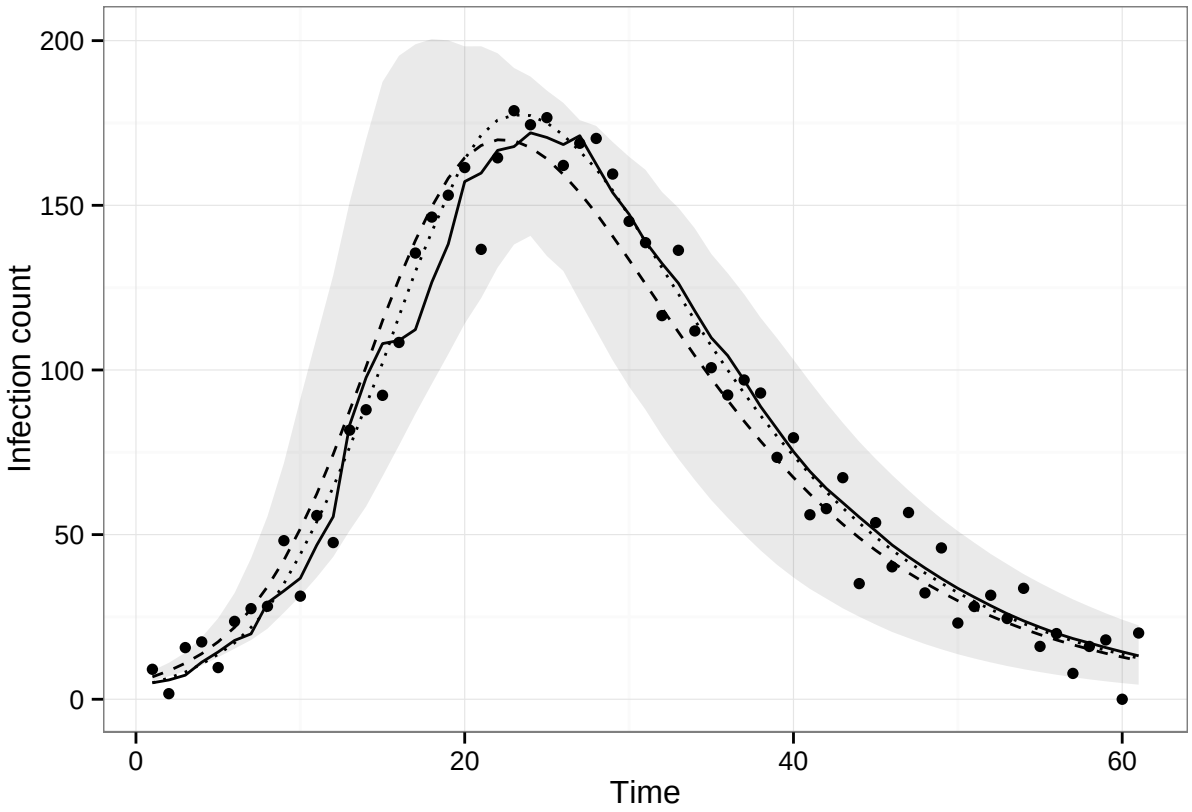
Of particular importance are the densities of the parameter estimates given by the final parameter swarm. These are shown below.



As before, the solid grey lines show the true parameter values and the dashed grey lines show the density means. It is worth noting that the IF2 parameters chosen were in part chosen so as to not artificially narrow these densities; a more aggressive cooling schedule and/or an increased number of passes would have resulted in much narrower densities, and indeed have the potential to collapse them to point estimates.

IF2 bootstrapping

We can now use the parameter estimate densities to perform bootstrapping. Here we have drawn 100 parameter estimates randomly from the final particle swarm, run a single trajectory from each of them, and plotted the results below:



Here the solid line shows the true system states at each time, the dots show the data, the dotted line shows the mean system behaviour from Figure [?], the dashed line is the bootstrap mean, and the grey ribbon shows the centre 95th quantile of the bootstrap results.

Note that the bootstrap mean is less critical now as we are working with "complete" data over the infection cycle and thus the particle filter state estimates provide more easily obtained and arguably better information, but will become more important in the context of forecasting, to be shown in the next section.

POMP fitting

Given that the previous IF2 fitting was carried out using a (slightly) customised IF2 implementation, it is prudent to compare these results to that produced by the 'official' IF2 implementation in the POMP package. All IF2 parameters were chosen so as to match the ones chosen previously as closely as possible.

The POMP fitting runtime, again determined using R's `system.time()` function was

```
##    user  system elapsed
##  14.664   0.345  15.194
```

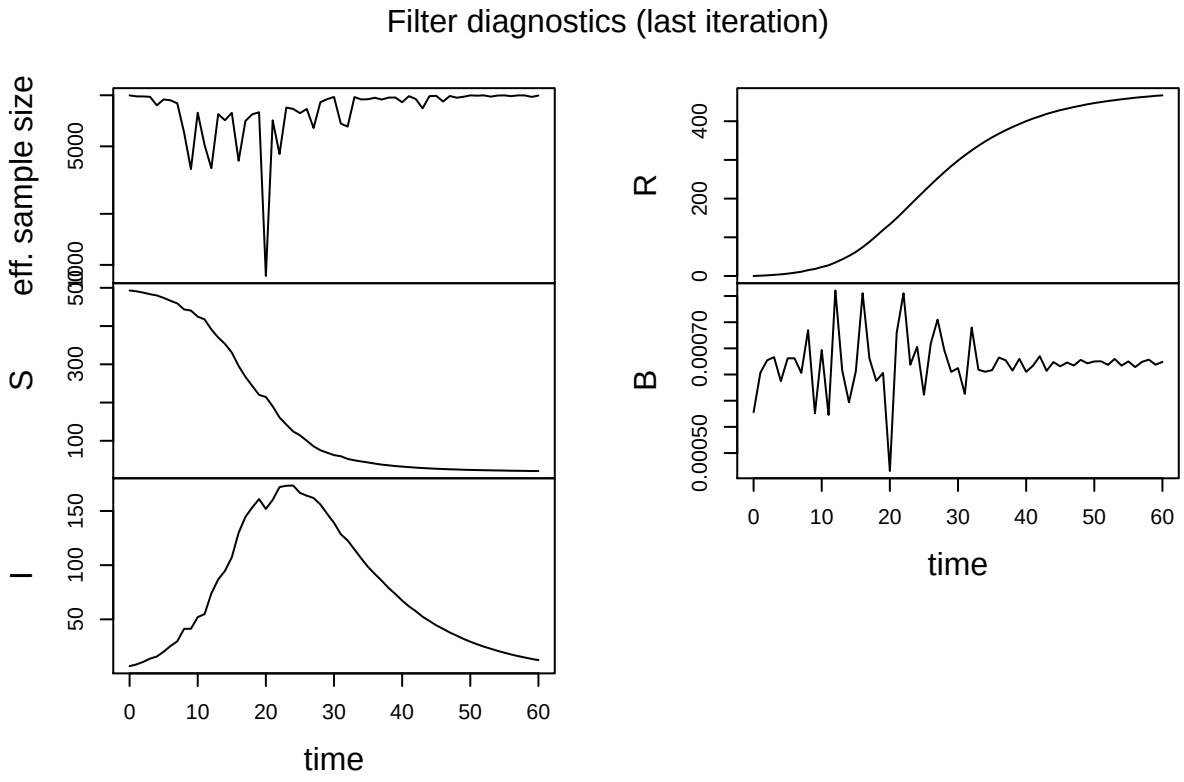
The MLE for the parameters after the final pass was

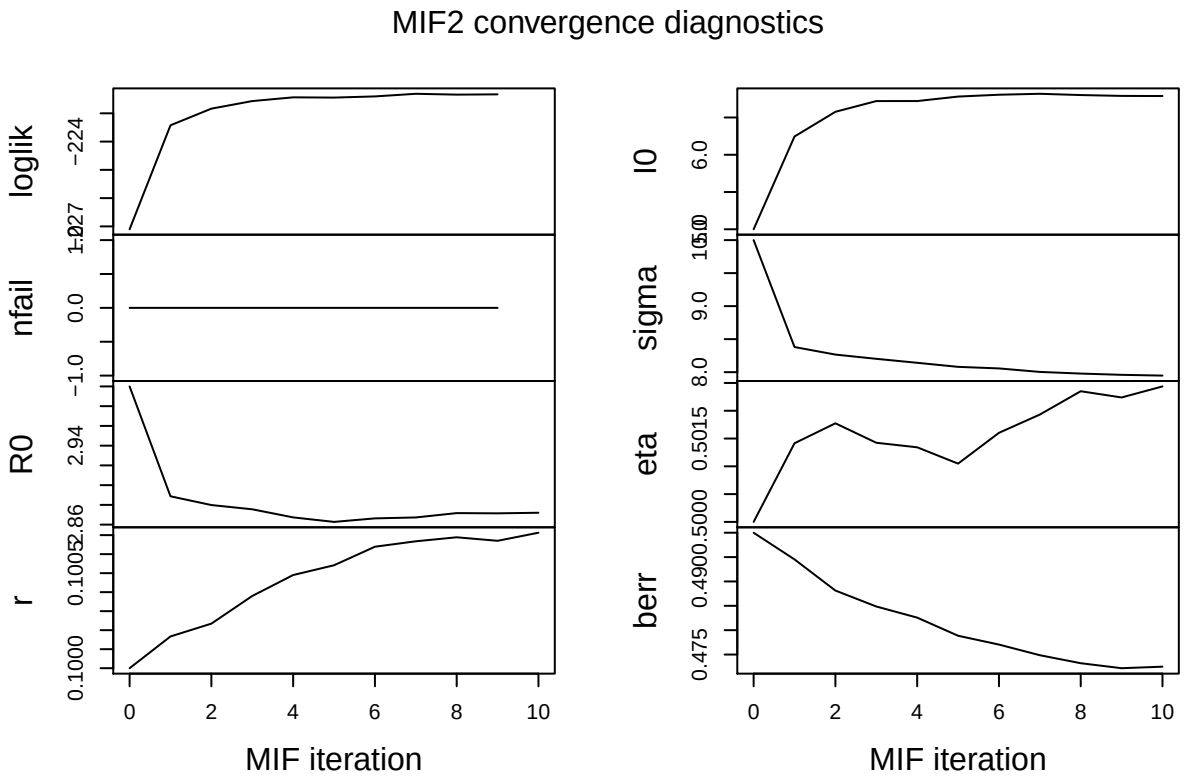
```
##        R0         r        I0     sigma       eta      berr
## 2.8720348 0.1007125 6.7878701 7.9480034 0.5024374 0.4725095
```

giving a relative error of

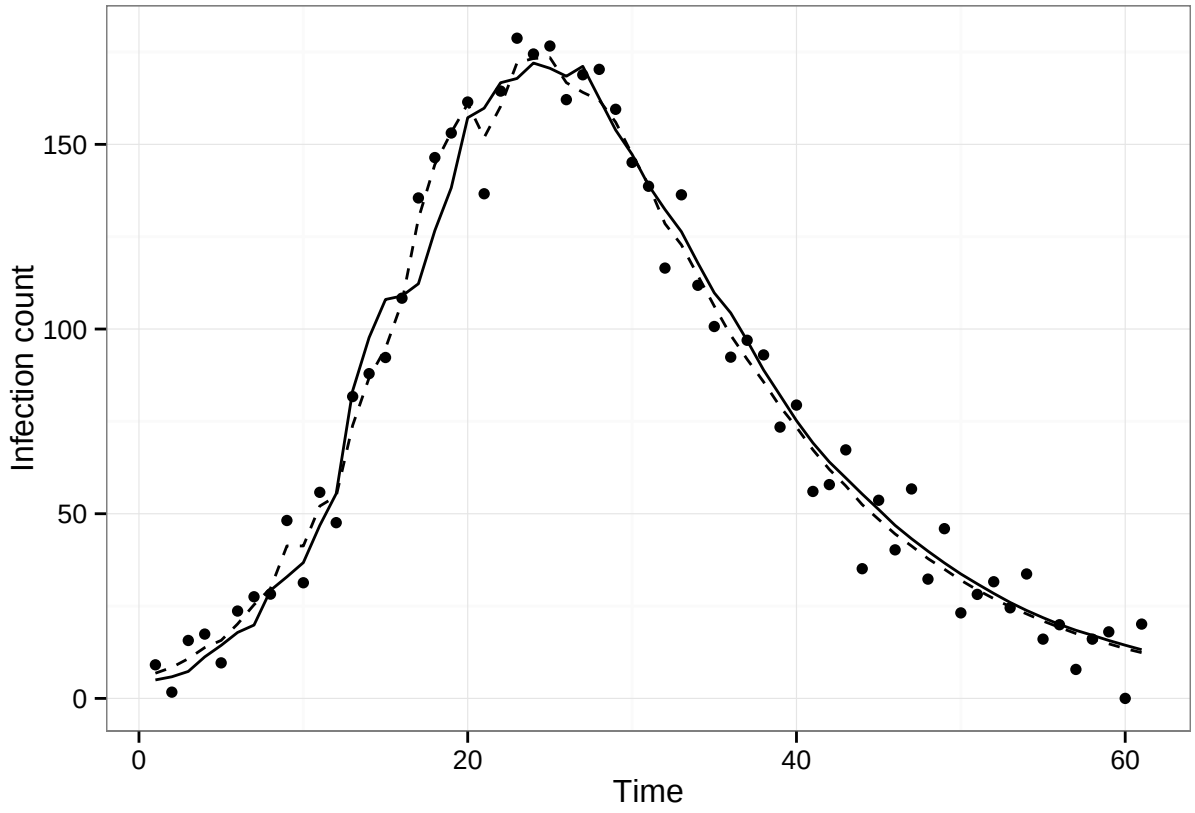
```
##           R0            r           I0        sigma          eta
## -0.042655069  0.007125066  0.357574024 -0.205199659  0.004874786
##         berr
## -0.054980932
```

POMP's built-in diagnostic plots show the following





As before, we can extract the system state estimates at each time step form the last IF2 pass and plot it as below:



Here again the solid line shows the true states, the points show the data, and the dashed line shows the filter mean states.

HMC fitting

We can use the Hamiltonian Monte Carlo algorithm implemented in the `Rstan` package to fit the stochastic SIR model as above. This was done with a single HMC chain of 5000 iterations with 100 warm-up iterations and a tinning value of 10.

The runtime retrieved again using R's `system.time()` shows

```
##    user  system elapsed
## 163.298   3.920 170.134
```

which is significantly slower than either the custom IF2 algorithm or the POMP implementation.

The MLE parameter estimates, taken to be the means of the samples in the chain were

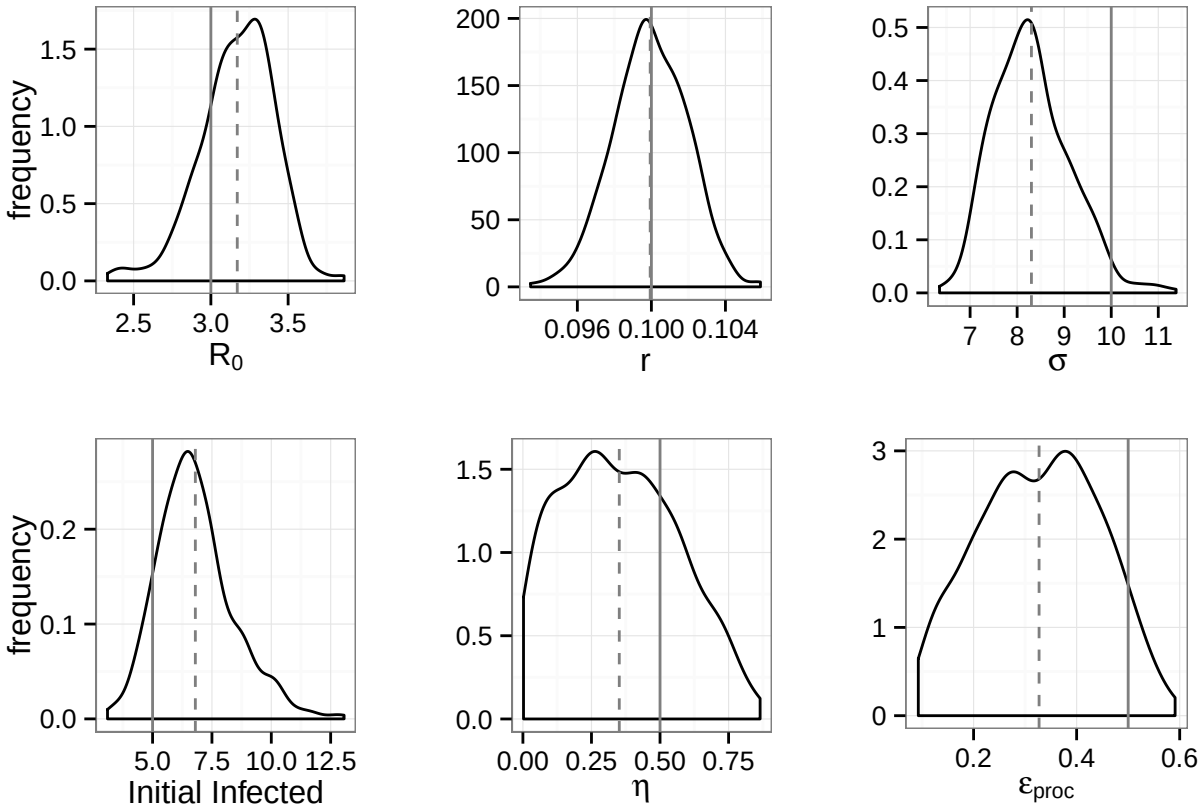
```
##        R0         r        I0     sigma       eta      berr
## 3.1708714 0.0999343 6.8012592 8.3038000 0.3513886 0.3271267
```

giving a relative error of

```
##            R0             r            I0         sigma           eta
##  0.0569571434 -0.0006570413  0.3602518490 -0.1696199997 -0.2972228343
##          berr
## -0.3457466887
```

HMC kernels

The densities produced by Stan's HMC implementation are shown below:

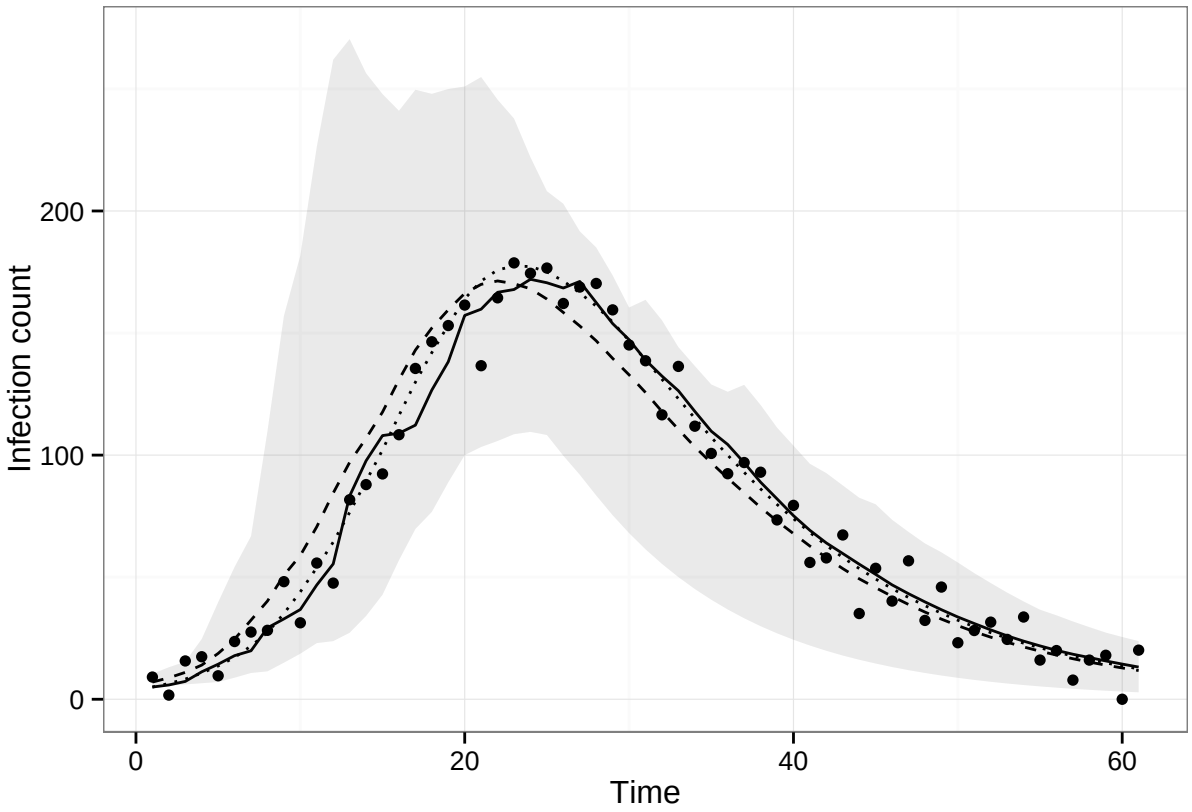


As before the solid grey lines show the true values and the dashed grey lines show the density means.

It is worth noting that the densities shown here represent a "true" MLE density estimate in that they represent HMC's attempt to directly sample from the parameter space according to the likelihood surface, unlike IF2 which is in theory only trying to get a ML point estimate. Further, these densities are much smoother and potentially more robust than those produced by the custom IF2 implementation.

HMC Bootstrapping

Unlike particle particle-filtering-based approaches, HMC does not produce state estimates as a by-product of parameter fitting, and so we must rely on a bootstrap alone to get state estimates. The results of 100 bootstrap trajectories is shown below:



As before the solid line shows the true states, the dots show the data, the dotted line shows the average system behaviour, the dashed line shows the bootstrap mean, and the grey ribbon shows the centre 95th quantile of the bootstrap trajectories.