

Forecasting Frameworks

Dexter Barrows

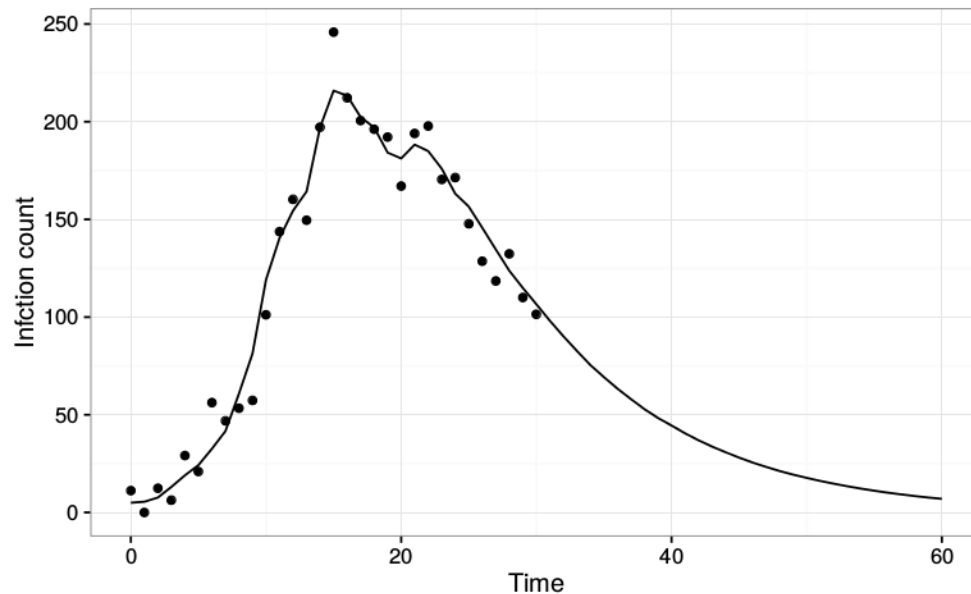
March 11, 2016

1 Data Setup

This section will focus on taking the stochastic SIR model from the previous section, truncating the synthetic data output from realizations of that model, and seeing how well IF2 and HMCMC can reconstruct out-of-sample forecasts.

I often prefer active voice, e.g. "Figure 1 shows an example of a simulated system with truncated data"

An example of a simulated system with truncated data can be seen in Figure 1 below.



Q: the true underlying system states are smooth even though there's some process error (autoregressive variation in beta) ... ?

(or is there no process error in this example?)

Figure 1: Infection count data truncated at $T = 30$. The solid line shows the true underlying system states, and the dots show those states with added observation noise. Parameters used were $R_0 = 3.0$, $r = 0.1$, $\eta = .05$, $\sigma_{proc} = 0.5$, and additive observation noise was drawn from $\mathcal{N}(0, 10)$.

In essence we want to be able to give either IF2 or HMCMC only the data points and have it reconstruct the entirety of the true system states.

2 IF2

For IF2, we will take advantage of the fact that the particle filter will produce state estimates for every datum in the time series given to it, as well as producing ^{MLE point estimates for the parameters} parameter maximum likelihood point estimates. Both of these sources of information will be used to produce forecasts by parametric bootstrapping using the final parameter estimates from the particle swarm after the last IF2 pass, then using the newly generated parameter sets along with the system state point estimates from the first fitting to simulate the systems forward into the future.

We will truncate the data at half the original time series length (to $T = 30$), and fit the model as previously described.

^{Figure 2 shows...} First, we can see the state estimates for each time point produced by the last IF2 pass in Figure [2](#) below.

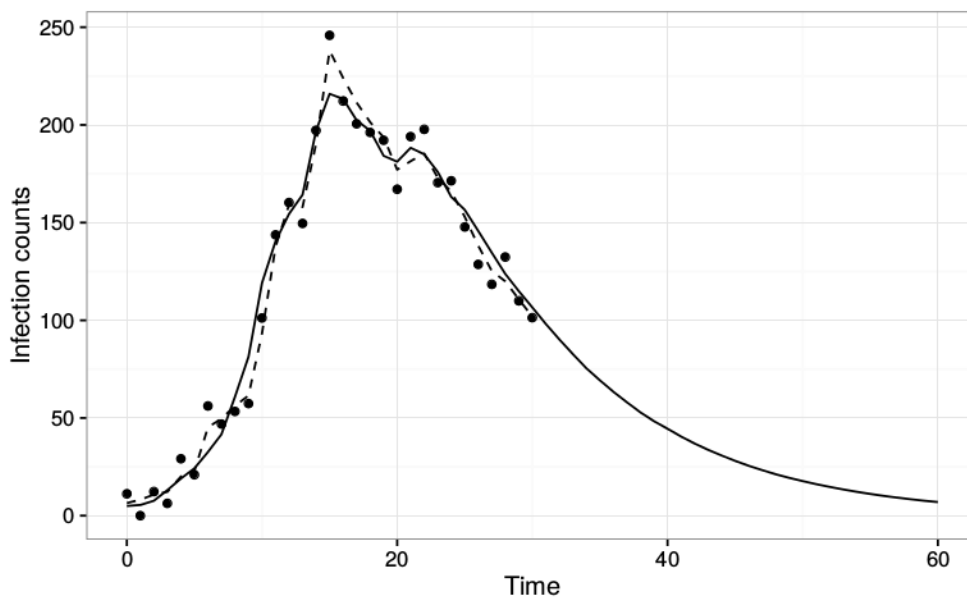


Figure 2: Infection count data truncated at $T = 30$ from Figure [1](#) above. The dashed line shows IF2's attempt to reconstruct the true underlying state from the observed data points.

Recall that IF2 is not trying to generate ^{be more precise? ("posterior probability densities" or "interval estimates"?)} parameter estimation densities, but rather produce a point estimate. Since we wish to determine the approximate distribution of each of the parameters in addition to the point estimate, we must turn to another method, parametric bootstrapping. ^{add a layer atop the IF2 machinery ... ?}

2.1 Parametric Bootstrapping

... such that the further samples represent samples from the `_sampling distribution_` of theta?

The goal of the parametric bootstrap is use an initial density sample θ^* to generate further samples $\theta_1, \theta_2, \dots, \theta_M$. It works by using θ to generate artificial data sets D_1, D_2, \dots, D_M to which we can refit our model of interest and generate new parameter sets.

[I'm still trying to dig up a good paper that talks about applicability to dynamical systems, there will be a paragraph here about it.]

OK

An algorithm for parametric bootstrapping using IF2 and our stochastic SIR model is shown in Algorithm [1](#).

Algorithm 1: Parametric Bootstrap

Input : Forward simulator $S(\theta)$, data set D

```
/* Initial fit */
1  $\theta^* \leftarrow IF2(D)$ 
/* Generate artificial data sets */
2 for  $i = 1 : M$  do
3    $D_i \leftarrow S(\theta^*)$ 
/* Fit to new data sets */
4 for  $i = 1 : M$  do
5    $\theta_i \leftarrow IF2(D_i)$ 
```

Output: Distribution samples $\theta_1, \theta_2, \dots, \theta_M$

as I may have said when we met, I'm not sure a "normal bootstrap" means anything in particular, nor that this procedure constitutes a standard (nonparametric) bootstrap. By convention, "bootstrapping" without modifiers suggests refitting the model to data points resampled with replacement

2.2 IF2 Forecasts

Using the parameter sets $\theta_1, \theta_2, \dots, \theta_M$ and the point estimate of the state provided by the initial IF2 fit, we can use a normal bootstrap to produce estimates of the future state. A plot showing a projection of the data from the previous plots can be seen in Figure [3](#).

see previous notes "Figure 3 shows" vs "... can be seen"

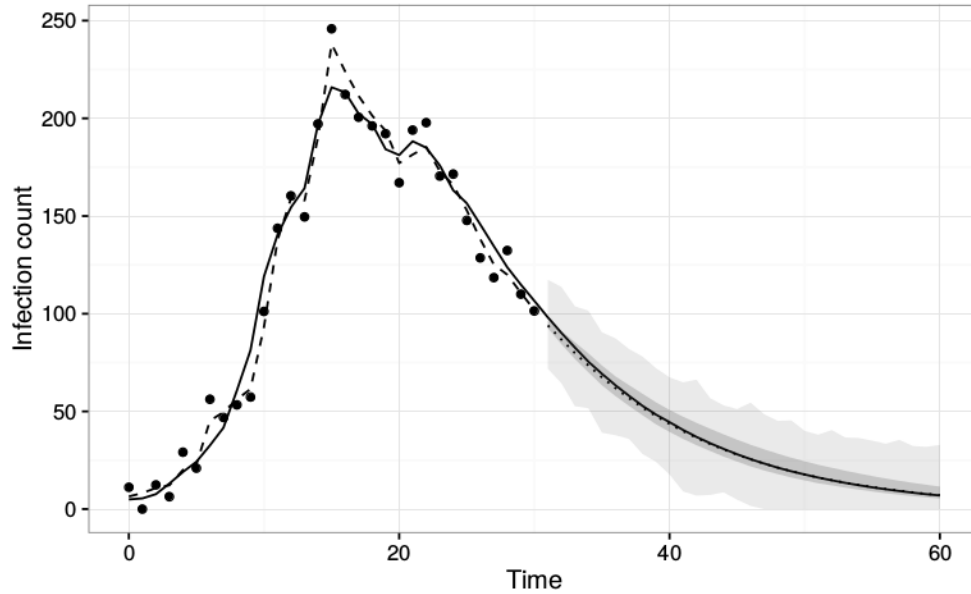


Figure 3: Forecast produced by the IF2 / parametric bootstrapping framework. The dotted line shows the mean estimate of the forecasts, the dark grey ribbon shows the centre 95th quantile of the true state estimates, and the lighter grey ribbon shows the centre 95th quantile of the true state estimates with added observation noise drawn from $\mathcal{N}(0, \sigma)$.

"centre 95th quantile" doesn't actually mean anything (I think) could say 95% confidence interval, based on the 0.025 and 0.975 quantiles (should probably define CI in this way once)

We can define a metric to gauge forecast effectiveness by calculating the SSE and dividing that value by the number of values predicted to get the average squared error per point. For the data in Figure 3 the value was $\overline{SSE} = 1.67$.

I think coverage is not feasible because we would have to further multiply the computational cost of everything by ~100-fold to get it ... right? Possibly worth mentioning.

3 HMCMC

again, not sure about "bootstrapping" -- "approach to CIs/interval estimates" ?

For HMCMC we can use a simpler bootstrapping approach. We do not get state estimates directly from the RStan fitting due to the way we implemented the model, but we can construct them using the process noise latent variables. Once we've done this we can forward simulate the system from the state estimate into the future.

As before we fit the stochastic SIR model to the partial data, but now perform bootstrapping as described above, and obtain the plot in Figure 4.

??

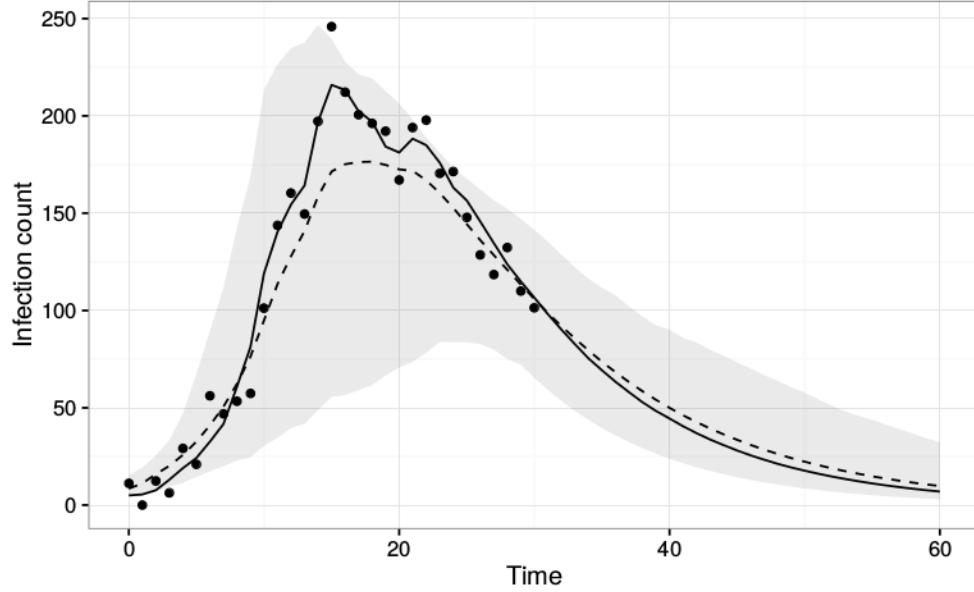


Figure 4: Forecast produced by the HMCMC / bootstrapping framework with $M = 200$ trajectories. The dotted line shows the mean estimate of the forecasts, and the grey ribbon shows the centre 95th quantile.

OK, centre 95th quantile is **much** wider. Is (1) HMCMC giving much less accurate results, with correct CIs, or (2) IF2/PB is underestimating the CIs and HMCMC is correct in this regard ... ?

And as before we can evaluate the averaged SSE of the forecast for the data shown, giving $\overline{SSE} = 20.27$.

4 Truncation vs. Error

Of course the above mini-comparison only shows one truncation value for one trajectory. Really, we need to know how each method performs on average given different trajectories and truncation amounts. In effect we wish to “starve” each method of data and see how poor the estimates become with each successive data point loss.

Using each method, we can fit the stochastic SIR model to successively smaller time series to see the effect of truncation on forecast averaged SSE. This was performed with 10 new trajectories drawn for each of the desired lengths. The results are shown in Figure [5](#).

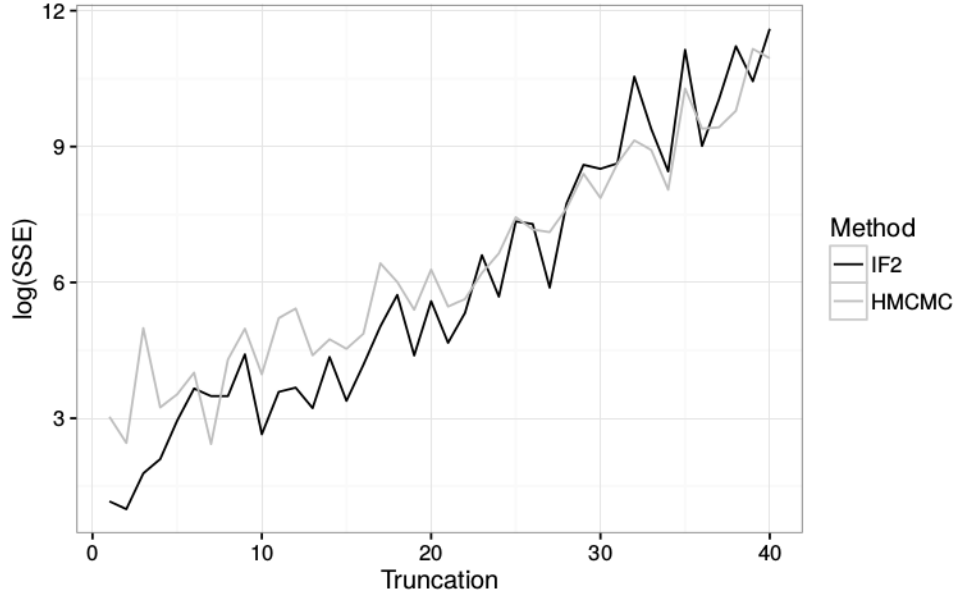


Figure 5: Error growth as a function of data truncation amount. Both methods used 200 bootstrap trajectories. Note that the y-axis shows the natural log of the averaged SSE, not the total SSE.

Maybe use log10? a little easier to interpret. Maybe use RELATIVE sse? (one step easier still ...)

IF2 and HMC MC perform very closely, with IF2 maintaining a small advantage up to a truncation of about 25-30 data points.

Since the parametric bootstrapping approach used by IF2 requires a significant number of additional fits, its computational cost is significantly higher than the simpler bootstrapping approach used by the HMC MC framework, about 35.5x as expensive. However the now much longer running time can somewhat alleviated by parallelizing the parametric bootstrapping process; as each of the parametric bootstrap fittings in entirely independent, this can be done without a great deal of additional effort. The code used here has this capability, but it was not utilised in the comparison so as to accurately represent total computational cost, not potential running time.

rather than

Appendices

A IF2 Parametric Bootstrapping Function

The parametric bootstrapping machinery used to produce forecasts.

```
1 # Dexter Barrows
2 #
3 # IF2 parametric bootstrapping function
4
5 library(foreach)
6 library(parallel)
7 library(doParallel)
8 library(Rcpp)
9
10 if2_paraboot ← function(if2data_parent, T, Tlim, steps, N, nTrials, if2file
11   , if2_s_file, stoc_sir_file, NP, nPasses, coolrate) {
12
13   source(stoc_sir_file)
14
15   if (nTrials < 2)
16     ntrials ← 2
17
18   # unpack if2 first fit data
19   # ...parameters
20   paramdata_parent ← data.frame( if2data_parent$paramdata )
21   names(paramdata_parent) ← c("R0", "r", "sigma", "eta", "berr", "Sinit", "
22     Iinit", "Rinit")
23   parmeans_parent ← colMeans(paramdata_parent)
24   names(parmmeans_parent) ← c("R0", "r", "sigma", "eta", "berr", "Sinit", "
25     Iinit", "Rinit")
26   # ...states
27   statedata_parent ← data.frame( if2data_parent$statedata )
28   names(statedata_parent) ← c("S", "I", "R", "B")
29   statemeans_parent ← colMeans(statedata_parent)
30   names(statemeans_parent) ← c("S", "I", "R", "B")
31
32   ## use parametric bootstrapping to generate forecasts
33   ##
34   trajectories ← foreach( i = 1:nTrials, .combine = rbind, .packages = "
35     Rcpp" ) %dopar% {
36
37     source(stoc_sir_file)
38
39     ## draw new data
40     ##
41
42     pars ← with( as.list(parmmeans_parent),
43       c(R0 = R0,
```

```

41         r = r,
42         N = N,
43         eta = eta,
44         berr = berr) )
45
46 init_cond ← with( as.list(parmmeans_parent),
47                   c(S = Sinit,
48                     I = Iinit,
49                     R = Rinit) )
50
51 # generate trajectory
52 sdeout ← StocSIR(init_cond, pars, Tlim + 1, steps)
53 colnames(sdeout) ← c('S', 'I', 'R', 'B')
54
55 # add noise
56 counts_raw ← sdeout[, 'I'] + rnorm(dim(sdeout)[1], 0, parmeans_parent[['sigma']])
57 counts      ← ifelse(counts_raw < 0, 0, counts_raw)
58
59 ## refit using new data
60 ##
61
62 rm(if2) # because stupid things get done in packages
63 sourceCpp(if2file)
64 if2time ← system.time( if2data ← if2(counts, Tlim+1, N, NP, nPasses,
65                                     coolrate) )
66
67 paramdata ← data.frame( if2data$paramdata )
68 names(paramdata) ← c("R0", "r", "sigma", "eta", "berr", "Sinit", "Iinit",
69                     "Rinit")
70 parmeans ← colMeans(paramdata)
71 names(parmeans) ← c("R0", "r", "sigma", "eta", "berr", "Sinit", "Iinit",
72                    "Rinit")
73
74 ## generate the rest of the trajectory
75 ##
76
77 # pack new parameter estimates
78 pars ← with( as.list(parmeans),
79              c(R0 = R0,
80                r = r,
81                N = N,
82                eta = eta,
83                berr = berr) )
84
85 init_cond ← c(S = statemeans_parent[['S']],
86               I = statemeans_parent[['I']],
87               R = statemeans_parent[['R']])
88
89 # generate remaining trajectory part
90 sdeout_future ← StocSIR(init_cond, pars, T-Tlim, steps)
91 colnames(sdeout_future) ← c('S', 'I', 'R', 'B')
92
93 return ( c( counts = unname(sdeout_future[, 'I']),
94            parmeans,

```



```

91         time = if2time[['user.self']] )
92
93
94     }
95
96     return(trajectories)
97
98 }

```

B RStan Forward Simulator

The code used to reconstruct the state estimates, then project the trajectory forward past data.

```

1 StocSIRstan <- function(y, pars, T, steps, berrvec, bveclim) {
2
3     out <- matrix(NA, nrow = (T+1), ncol = 4)
4
5     R0 <- pars[['R0']]
6     r <- pars[['r']]
7     N <- pars[['N']]
8     eta <- pars[['eta']]
9     berr <- pars[['berr']]
10
11     S <- y[['S']]
12     I <- y[['I']]
13     R <- y[['R']]
14
15     B0 <- R0 * r / N
16     B <- B0
17
18     out[1,] <- c(S,I,R,B)
19
20     h <- 1 / steps
21
22     for ( i in 1:(T*steps) ) {
23
24         if (i <= bveclim) {
25             B <- exp( log(B) + eta*(log(B0) - log(B)) + berrvec[i])
26         } else {
27             B <- exp( log(B) + eta*(log(B0) - log(B)) + rnorm(1, 0, berr))
28         }
29
30         BSI <- B*S*I
31         rI <- r*I
32
33         dS <- -BSI
34         dI <- BSI - rI
35         dR <- rI
36

```

```
37     S ← S + h*dS   #newInf
38     I ← I + h*dI   #newInf - h*dR
39     R ← R + h*dR   #h*dR
40
41     if (i %% steps == 0)
42         out[i/steps+1,] ← c(S,I,R,B)
43
44 }
45
46 return(out)
47
48 }
```