EEID WORKSHOP 2010 - PARTICLE FILTERS

MJF

Contents

1. A General Framework for Model Fitting

(1) Look at the data (plotting, etc)
(2) Quantify patterns in the data (summary measures)
(3) Posit a model (or models) to explain the biological and observational processes that generated the data
(4) Evaluate whether or not the candidate model(s) are capable of producing patterns consistent with those in the data.
(5) Find the set of model parameters that best reproduce the patterns seen in the data
(6) Compare the ability of candidate models to reproduce the observed patterns using the "best" parameters for each

2. The Generic Problem

For many biological or ecological problems it is easier to simulate random realizations of a model than it is to analytically evaluate the probability of observing a specific outcome. Thus, working directly with the likelihood for a dynamic, time series model is often difficult or limiting (i.e. we are restricted to simple models to make the analysis tractable). However, if we can simulate realizations of the model, for a given set of parameters, we can generate a distribution of outcomes that we can compare against the observed data.

Date: June 7, 2010.

3. An example: The flu outbreak

Consider the following model for a closed epidemic. First we will write the following function to simulate stochastic realizations of the SIR model.

```
> sir.birth.death.onestep <- function (x, params, timestep) {
+
+  S <- x[,2]
+  I <- x[,3]
+  R <- x[,4]
+  N <- S+I+R
+  Np<-dim(x)[1]
+  with(
+  as.list(params),
+  {
+
+  dSI <- pmin(S,rpois(Np,beta*S*I/N * timestep))
+  dIR <- pmin(I,rpois(Np,gamma*I * timestep))
+
+  new.sir<-cbind(S - dSI , I + dSI - dIR, R + dIR)
+  cbind(timestep,new.sir)
+
+  }
+  )
+  }
>  sir.birth.death.model <- function (x, params, time.max, timestep=1) {
+  Np<-dim(x)[1]
+  X <- array(dim=c(time.max/timestep+1,Np,4))
+  X[1,,] <- x
+  for (k in 1:(time.max/timestep)) {
+  X[k,,]<- x <- sir.birth.death.onestep(x,params, timestep)
+
+  }
+  X[,,1]<-cumsum(X[,,1])
+  X
+  }
```

Note that this has the same structure as the model shown in the earlier example for the flu outbreak in a British boarding school.

Question: Based on the methods you've already seen for model fitting, how could you use these stochastic simulations to fit this model to data?

Answer: First consider how you would do it for a single stochastic realization . . .

```
> #generate stochastic simulations, and plot trajectories#
>  set.seed(38499583)
>  nsims <- 100
>  pop.size <- 762
>  I0 <- 2
>  S0 <- round(0.98*pop.size)
>  params <- c(beta=2.5,gamma=1, p.obs=.9)
>  xstart <- matrix(c(time=0,S=S0,I=I0,R=pop.size-I0-S0),nr=nsims,nc=4,byrow=T)
>  tmp<- sir.birth.death.model(xstart,params,time.max=14, timestep=.2)
```

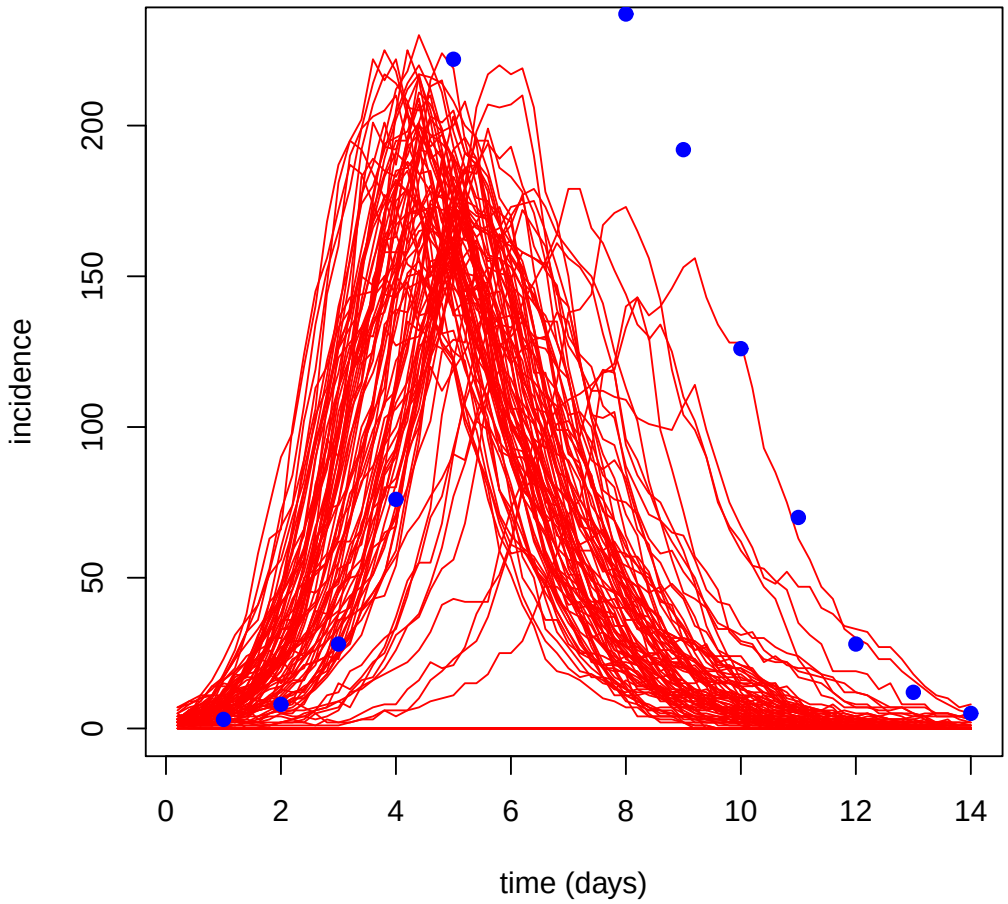

FIGURE 1. 100 realizations of the SIR closed epidemic.

```
> I<-tmp[,,3]
> time<-matrix(tmp[,1,1],nr=dim(I)[1],nc=dim(I)[2],byrow=F)
> save(time,I,file=binary.file)
```

Exercise 1. Simulate many realizations of the stochastic model. How many iterations of the simulation would you need to characterize the average behavior well?

4. State Space Models

For many of the systems in which we work, we cannot directly observed the biological quantities of interest directly or without error. For example, surveillance data during an epidemic may come from cases that are admitted to health clinics. Those individuals that seek care (and are correctly diagnosed – i.e. the recent H1N1 flu epidemic) are likely to be only a subset of the true number of incident cases in the population. However, the models that we write to explore the dynamics of epidemics depend on the true incidence, I_t, not the number of cases reported, C_t. Naively substituting the reported cases for the

true incidence is likely to result in biases (the magnitude of which will depend on the particular system). We refer to the true, but unobservable elements, of the biological system (e.g. the true numbers infected and susceptible individuals through time) as **states**. And the model that describes the time progression of these **states** is called the **process model**. Often it is reasonable to posit a model that connects the unobserved **states** to the observed data. For example if some fraction, p, of the incident cases are severe enough to present to clinics for care, then the observed cases, C_t could be modelled as a random process with mean pC_t. We refer to the model that connects the unobserved **states** to the observed data as the **observation model**. Note, again, that the biological process that we want to understand is described in terms of the **states**, which progress in time according to a model of biological interest, but our observation of the system is in terms of observations that are connected to the **states** only through an **observation model** that is likely to introduce both biases and additional variation.

The general set of statistical methods that deal with systems like this, a linked **process model** and **observation model** are referred to as **state space models**.

Exercise 2. Consider a simple observation model that says that the observed cases are Poisson distributed with mean equal to an observation parameter, p.obs, times the true number of incident cases. Modify the code above to generate realizations of the observed data, assuming that p.obs = 0.9. Consider how the

5. A generic algorithm using the observation likelihood

Often, we are willing to accept a relatively simple, phenomenological model for the observation processs. Thus it is common that it might be difficult to write an explicit likelihood for the biological **process model**, but not too hard to do so for the observations. In that event, we can use simulation to generate a distribution of outcomes for the unobserved **states** and use the likelihood for the **observation model** to evaluate how likely we are to have observed the data, given that distribution of **states**.

(1) Start with a specific set of parameter values.
(2) Simulate many (Np) stochastic realizations of the process model to get timeseries of the unobserved states.
(3) For each simulated time series, evaluate the log-likelihood of each observation under the observation model (i.e. Np x T calls to the likelihood function)
(4) For each of the Np timeseries, sum the log-likelihoods to get the likelihood for that realization of the stochastic model.
(5) Take the average over the Np log-likelihoods as an estimate of hte likelihood for the parameter set in step 1. This gives a simulated approximation to the likelihood.

Question: Then what? How would you use this to fit parameters?

Answer:

Question: How big does Np have to be?

Answer: Depends on how well the simulations reflect the average. But, in general, pretty big.

Now that we've simulated many realizations of the epidemic, we can evaluate likelihood of each simulated realization according to the observation model, conditional on the observed data.

```
> lik.calc.pois<-function(dat,X,params){
+         dpois(dat,X*params["p.obs"],log=T)
+ }
```

Now load the data . . .

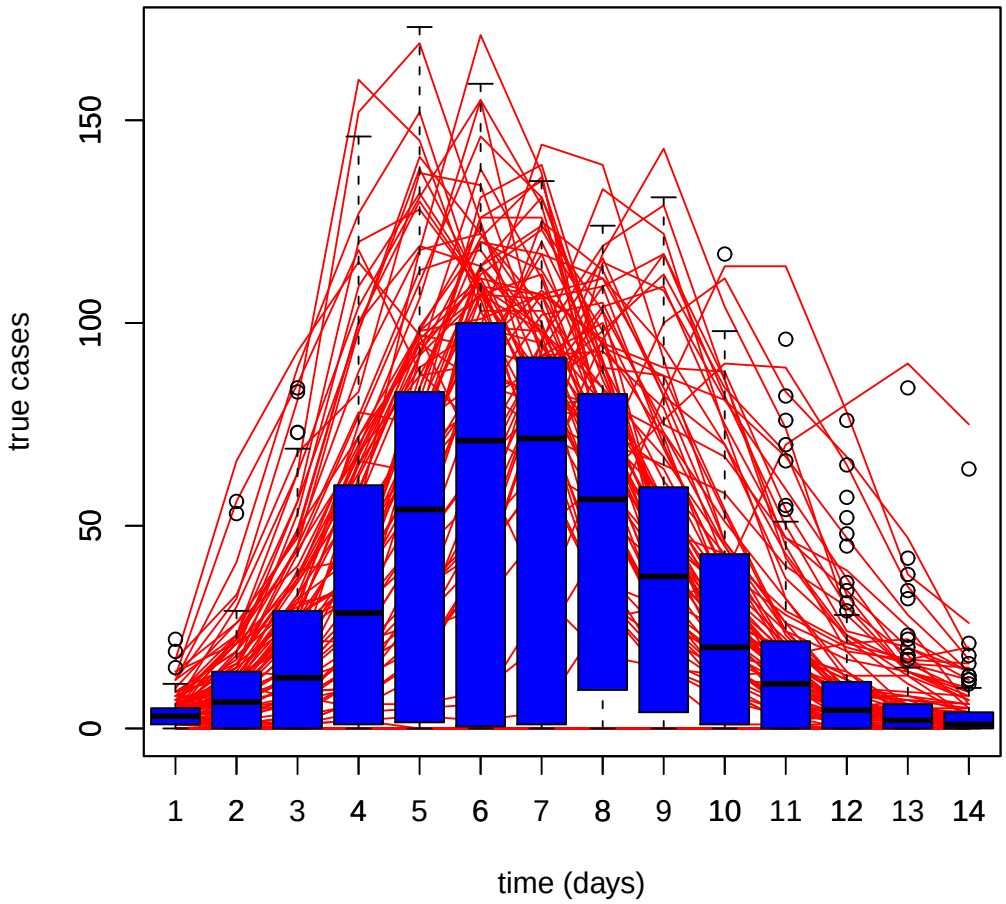

FIGURE 2. Stochastic realizations of the epidemic model (red lines). The data that would be observed under the observation model are shown in the blue box plots.

```
> load("/Users/matthewferrari/Library/Mail Downloads/flu.RData")
> names(flu)<-c("time","data")
```

Now, for a particular set of parameters, simulate realizations of the stochastic model, and evaluate the likelihood of each realization . . .

```
>  set.seed(38499583)
>  nsims <- 10000                          # number of simulations
>  pop.size <- 762                          # population size
>  I0 <- 2                                          # initial infections
>  S0 <- round(0.98*pop.size)          # initial susceptibles
>  params <- c(beta=2,gamma=1, p.obs=.9)          # parameters
>  xstart <- matrix(c(time=0,S=S0,I=I0,R=pop.size-I0-S0),nr=nsims,nc=4,byrow=T)
>  tmp<- sir.birth.death.model(xstart,params,time.max=14, timestep=.2)        # run simulations
```

```
>  I<-tmp[1:70,,3]                                          # extract the incidence
>  time<-matrix(tmp[1:70,1,1],nr=dim(I)[1],nc=dim(I)[2],byrow=F)        # extract the times
> save(time,I,file=binary.file)
> # pull out just the incidence at the timestep of the observations (days 1-14)
> # recall that the simulation is at .2 day increments
> observation.times<-which(!is.na(match(time[,1],1:14)))
> incidence<-I[observation.times,]
> ########step 3 in the algorithm###############
> loglik.mat<-lik.calc.pois(matrix(flu$data,nr=14,nc=nsims,byrow=F),incidence,params)
> ########step 4 in the algorithm###############
> loglik<-apply(loglik.mat,2,sum)
> ########step 5 in the algorithm###############
> lik<-sum(exp(loglik))
> save(time,I,loglik,file=binary.file)
```

Question: Are the "failed" epidemics informative?

Answer: No, because conditional on the data, we know that an epidemic took place. Look at the historgram of likelihood values. In fact, once an epidemic trajectory diverges far from the data, it is not worth continuing to simulate, because the associated particles carry such little weight.

Question: Is there a better way to do this?

Answer: Yes . . .

- Part of the problem is that once a simulation gets away from the data, it is highly unlikely to come back
- So we waste a lot of effort simulating trajectories that we know will have vanishingly small likelihoods
- So what if, as we went along with the simulation, we eliminated those trajectories that were the least likely?
- This innovation is at the heart of particle filtering
  - each trajectory is called a particle
  - the elimination of unlikely trajectories is called "filtering"

6. The Particle Filtering Algorithm

As we did before:

(1) Start with a specific set of parameters, θ.
(2) Simulate N realizations (particles) of the model forward by one observation step.
(3) Evaluate the log-likelihood of each simulated particle, p^i_{t+1}, conditional on the data at that observation point.
(4) Calculate the relative weight of each particle

$$q^i_{t+1} = \frac{p^i_{t+1}}{\sum p^i_{t+1}} \tag{1}$$

(5) Resample indices j from $1:N$ with replacement with probability equal to q^i_{t+1}.
(6) Store the states, X^i_{t+1}, and the weights, p^i_{t+1}.
(7) Return to 2 starting from X^i_{t+1}
(8) Iterate until the end of the time series.

Then, as before, the mean of the p's is an estimate of the likelihood.

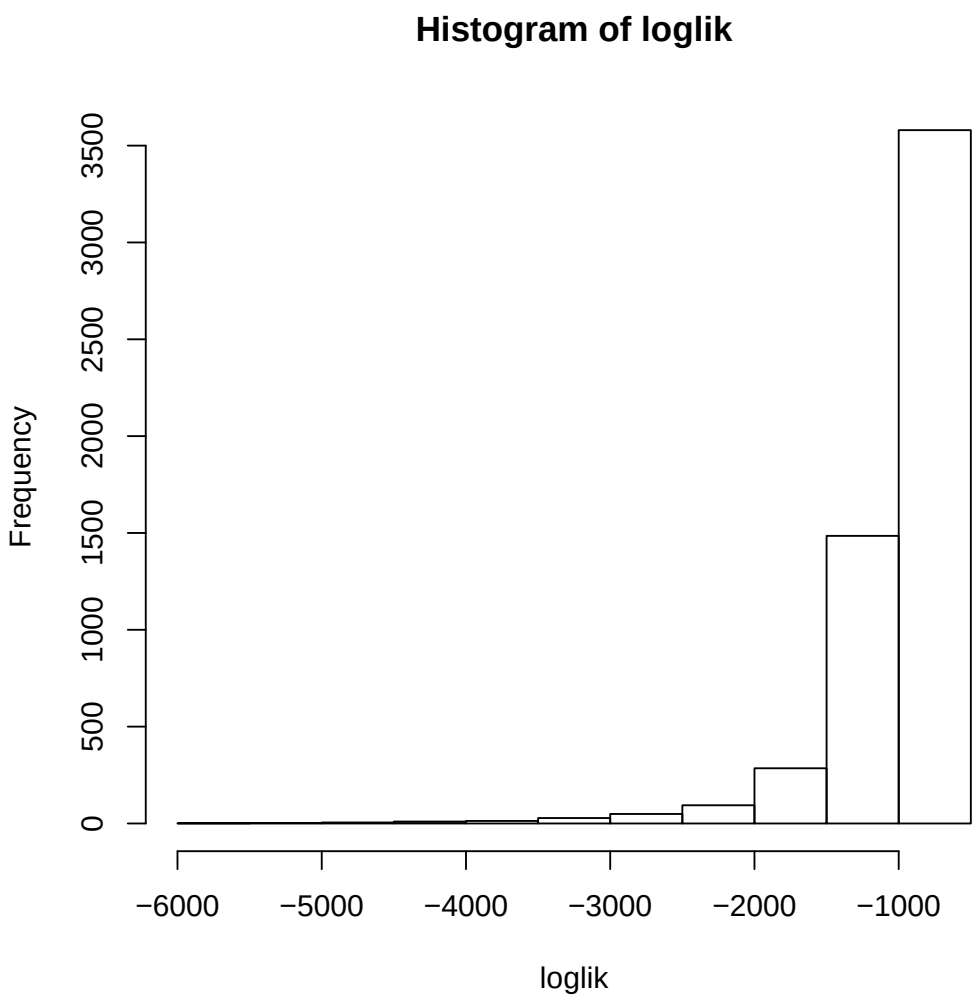

FIGURE 3. Histogram of likelihood values for trajectories (particles). Note that many very, very small.

7. An example: The flu outbreak

First, because we need to evaluate the likelihood of each particle, at each time step (before filtering), I need to introduce an additional function that increments the process model by one observation time step, rather than the small fractional step we used earlier.

```
> sir.birth.death.obsstep<-function (x, obstime, params, timestep) {
+
+  Np<-dim(x)[1]
+  X <- array(dim=c((obstime-x[1])/timestep+10,Np,4))
+  X[1,,] <- x
+  time.end <- obstime
+  k <- 1
+  while(sum(X[1:k,1,1]) <= time.end){
```

```
+  k<-k+1
+  X[k,,] <- sir.birth.death.onestep(x,params, timestep)
+  }
+  X[k-1,,1] <- sum(X[1:(k-1),1,1])  #THIS IS WRONG
+  X[k-1,,]
+  }
```

Now, we need to write code that will increment the process model one step forward (for many particles), calculate the weights, and do the filtering.

```
> pfilt_lik<-function(par, vecs){
+ Np<-vecs$Np                                        # number of particles
+ data<-vecs$data                                  # the data
+ timestep<-vecs$timestep                 # the timestep for tau leap algorithm
+ params <- c(beta=par[1],gamma=par[2], p.obs=par[3])
+ T<- dim(data)[1]                          # the maximum time step
+ states<- array(NA,dim=c(T+1,Np,4))        # set up an array for the states
+ for(i in 1:Np){states[1,i,]<- c(time=0,S=S0,I=I0,R=763-I0-S0) }        # set the initial conditions
+ weights<- matrix(NA,nr=T,nc=Np)                # set up an array for the weights
+ for(t in 2:(T+1)){
+         ##########project states -- step 2##############
+         states[t,,] <- sir.birth.death.obsstep(states[t-1,,],data[t-1,"time"],
+         params, timestep)
+         ##########evaluate weights -- step 3###########
+         weights[(t-1),] <- lik.calc.pois(data[t-1,"data"],states[t,,3],params)
+         ##########resample indices -- step 4&5#########
+         new.ind<-sample(1:Np,Np,replace=T,prob=exp(weights[(t-1),]))
+         ##########store states -- step 6################
+         states[t,,] <- states[t,new.ind,]
+         ##########store weights -- step 6##############
+         weights[(t-1),] <- weights[(t-1),new.ind]
+ }
+ ######### sum the weights -- step 8############
+ loglik<-sum(weights)
+ if(!is.finite(loglik)){loglik<-sign(loglik)*1e8}
+ ifelse(vecs$optim==1,return(-loglik),return(list(loglik=loglik,states=states,weights=weights)))
+ }
```

So, lets see how this looks in comparison

```
> init <- c(5,1,.9)
> fit<-optim(init,pfilt_lik,method="L-BFGS-B",lower=c(1,.99,.5),upper=c(6,1.01,.95),
+            vecs=list(Np=1000,data=flu,optim=1,timestep=.33))
> tmp2<-pfilt_lik(fit$par,vecs=list(Np=10000,data=flu,optim=0,timestep=.33))
> save(tmp2,file=binary.file)
```

Question: How would you use this to fit parameters?

Answer:

```
> init <- c(5,1,.9)        #starting values for optim()
> S0<-762                          #population initial conditions
> I0<-1                          #population initial conditions
> #####run optim()#########
```

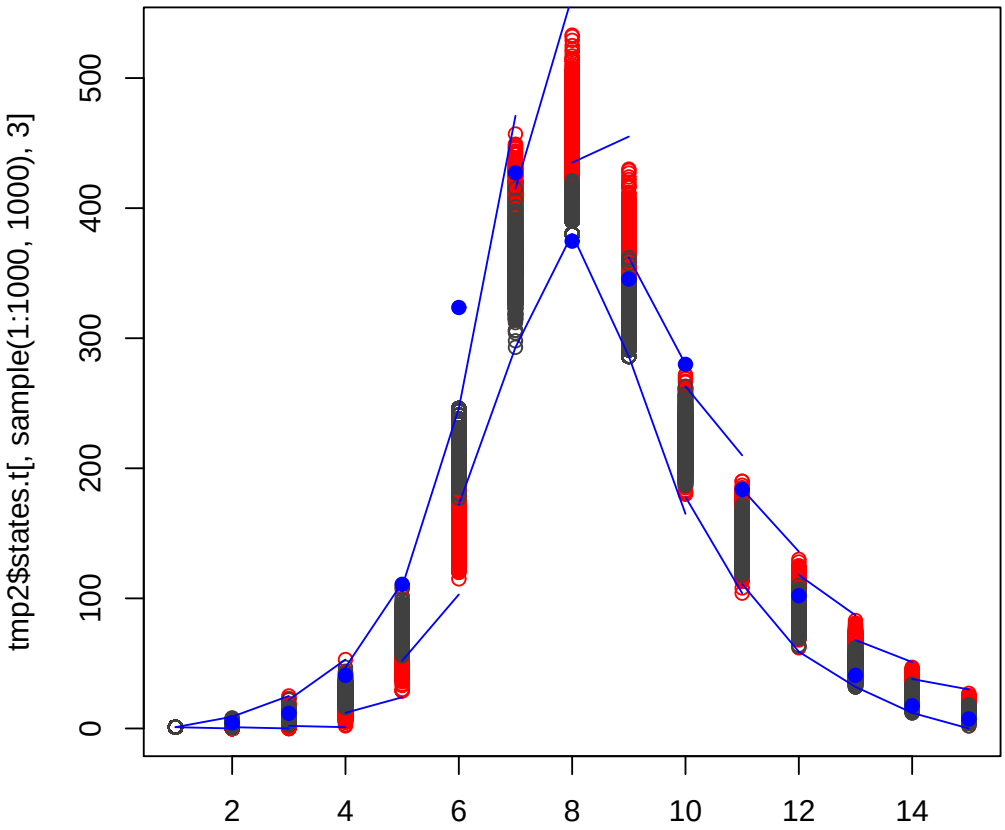

FIGURE 4. Projected and filtered states. In red are the one-step ahead projections of the states (step 2). In black are the resampled states after filtering according to the data likelihood (step 4). The data (corrected for under-reporting) are shown in blue.

```
> fit<-optim(init,pfilt_lik,method="L-BFGS-B",lower=c(1,.99,.5),upper=c(6,1.01,.95),
+            vecs=list(Np=1000,data=flu,optim=1,timestep=.33))
> #####run particle filter at fit values#########
> tmp2<-pfilt_lik(fit$par,vecs=list(Np=10000,data=flu,optim=0,timestep=.33))
> #####sample states at estimated observation rate######
> sim.data<-matrix(rpois(14000,fit$par[3]*tmp2$states[2:15,sample(1:10000,1000),3]),nr=14,nc=1000,byrow=
> save(fit,tmp2,sim.data,file="binary.file")
```

8. PRACTICAL ISSUES – PARTICLE DEPLETION

The most serious practical issue with particle filtering, other than the computational time, is that when the likelihood weights are highly skewed, the tendency will be to sample only a few particles. Because

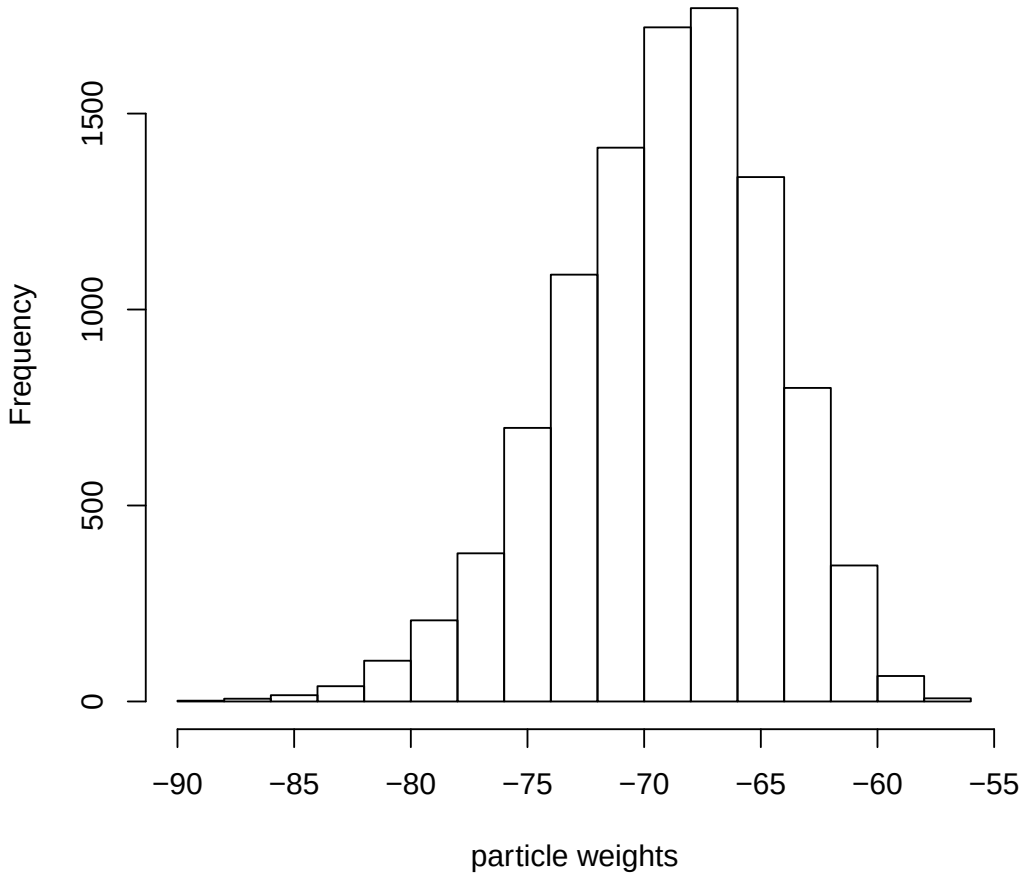

FIGURE 5. Histogram of likelihood weights for filtered particles.

we sample with replacement, that means that many of the particles will be the same. The resulting "particle depletion" means that the effective sample size t hat one is using to approxmiate the likelihood is much smaller than the actual number suggests. This is an algorithmic issue that is beyond the scope of this lesson, but is an issue worth raising. There are a variety of ways around particle depletion. The simplest way is just to use more particles to insure that there are enough to get through these bottlenecks. Alternative methods involve jittering, or adding noise to the particles at each time step; though because this noise is not due to the process model, corrections must be made in order for the variance in the resulting particles to correctly reflect the process model likelihood.

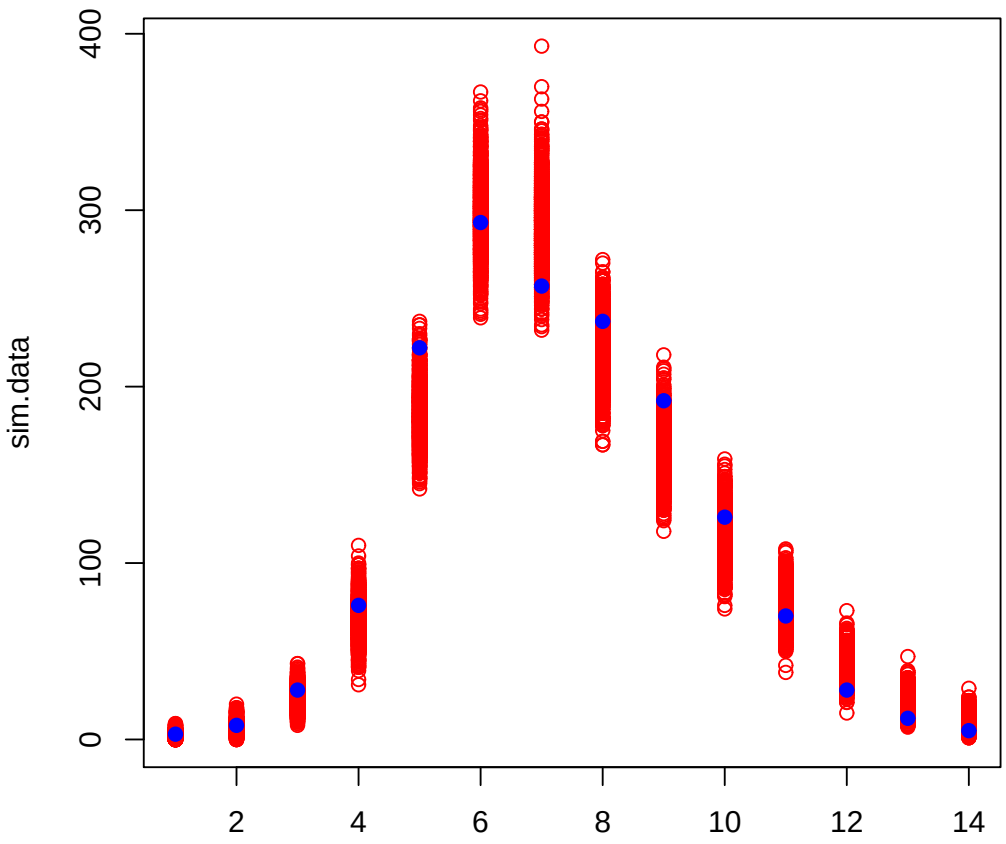

FIGURE 6. Filtered states and data. Red points show the filtered states assuming Poisson under reporting, the blue points show the observed data.