12 Practical likelihood-based inference for POMP models

Edward Ionides

4/1/2018

Contents

Produced with R version 3.4.3 and **pomp** version 1.16.

Objectives

1. To explain the simplest **particle filter**, allowing Monte Carlo solution of the POMP filtering and prediction recursions and therefore providing computation of the likelihood.
2. To provide examples of visualizing and exploring likelihood surfaces using the particle filter for computation of the likelihood.
3. To consider how to apply the fundamental tools of likelihood-based inference in situations where the likelihood cannot be written down explicitly but can be evaluated and maximized via Monte Carlo methods.
4. To gain some experience at carrying out likelihood-based inferences for dynamic models using simulation-based statistical methodology in the R package **pomp**.
5. Understand how iterated filtering algorithms carry out repeated particle filtering operations, with randomly perturbed parameter values, in order to maximize the likelihood.
6. Gain experience carrying out statistical investigations using iterated filtering in a relatively simple situation (fitting an SIR model to a boarding school flu outbreak).

Theory of the particle filter

Indirect specification of the statistical model via a simulation procedure

- For simple statistical models, we may describe the model by explicitly writing the density function $f_{Y_{1:N}}(y_{1:N}\,;\,\theta)$. One may then ask how to simulate a random variable $Y_{1:N} \sim f_{Y_{1:N}}(y_{1:N}\,;\,\theta)$.

- For many dynamic models it is convenient to define the model via a procedure to simulate the random variable $Y_{1:N}$. This implicitly defines the corresponding density $f_{Y_{1:N}}(y_{1:N}\,;\,\theta)$. For a complicated simulation procedure, it may be difficult or impossible to write down $f_{Y_{1:N}}(y_{1:N}\,;\,\theta)$ exactly.
- It is important for us to bear in mind that the likelihood function exists even when we don't know what it is! We can still talk about the likelihood function, and develop numerical methods that take advantage of its statistical properties.

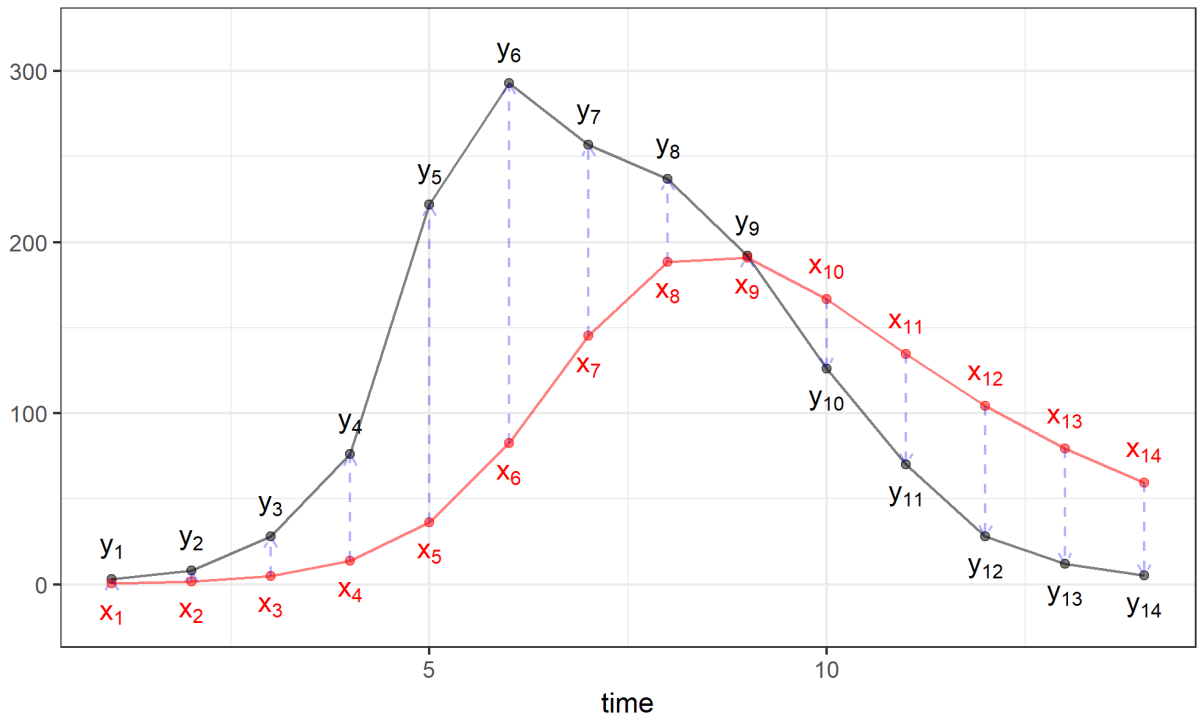
Special case: a deterministic unobserved state process

- Lets' begin with a special case where the unobserved state process is deterministic. That is, $X_n = x_n(\theta)$ is a known function of θ for each n. What is the likelihood?
- Since the distribution of the observable random variable, Y_n, depends only on X_n and θ, and since, in particular Y_m and Y_n are independent given X_m and X_n, we have

$$\mathcal{L}(\theta) = \prod_n f_{Y_n|X_n}(y_n^* \,|\, x_n(\theta)\,;\,\theta)$$

or

$$\ell(\theta) = \log \mathcal{L}(\theta) = \sum_n \log f_{Y_n|X_n}(y_n^* \,|\, x_n(\theta)\,;\,\theta).$$



Likelihood for stochastic models by direct simulation

- We can introduce the particle filter by first proposing a simpler method that usually doesn't work on anything but very short time series.
- **Warning: this section is an example of what NOT to do!**
- We can imagine using Monte Carlo integration for computing the likelihood of a state space model,

$$\begin{aligned}\mathcal{L}(\theta) &= f_{Y_{1:N}}(y^*_{1:N}\,;\,\theta) &(1)\\ &= \int_{x_{0:N}} f_{X_0}(x_0\,;\,\theta)\prod_{n=1}^{N} f_{Y_n|X_n}(y^*_n\,|\,x_n\,;\,\theta)\,f_{X_n|X_{n-1}}(x_n|x_{n-1}\,;\,\theta)\,dx_{0:N}. &(2)\end{aligned}$$

- Specifically, if we have some probabilistic means of proposing trajectories for the unobserved state process, then we could just generate a large number of these and approximate $\mathcal{L}(\theta)$ by its Monte Carlo estimate.
- Suppose we can generate J trajectories of length N,

$$\big\{X_{n,j},\ j=1\ \ldots,J,\ n=1,\ldots,N\big\},$$

  from a joint density $g_{X_{0:N}}(x_{0:N})$.
- Let G_j denote the probability density for trajectory j under the model used to generate the trajectory, i.e.,

$$G_j = g_{X_0}(X_{0,j})\prod_{n=1}^{N} g_{X_n|X_{n-1}}(X_{n,j}\,|\,X_{n-1,j}).$$

- Let F_j denote the probability density for trajectory j under our dynamic model, i.e.,

$$F_j = f_{X_0}(X_{0,j}\,;\,\theta)\prod_{n=1}^{N} f_{X_n|X_{n-1}}(X_{n,j}\,|\,X_{n-1,j}\,;\,\theta).$$

- For each trajectory, we can compute the measurement likelihood of that trajectory, i.e., the measurement density

$$\begin{aligned}&f_{Y_{1:N}|X_{1:N}}(y^*_{1:N}\,|\,X_{1:N,j}\,;\,\theta) &(3)\\ &\quad= \prod_{n=1}^{N} f_{Y_n|X_n}(y^*_n\,|\,X_{n,j}\,;\,\theta). &(4)\end{aligned}$$

  Then the Monte Carlo importance sampling theorem gives us

$$\mathcal{L}(\theta) \approx \frac{1}{J}\sum_{j=1}^{J} f_{Y_{1:N}|X_{1:N}}(y^*_{1:N}\,|\,X_{1:N,j}\,;\,\theta)\frac{F_j}{G_j}.$$

- How shall we choose our trajectories? What is a good choice of the importance sampling proposal density, $g_{X_{0:N}}(x_{0:N})$?
- A convenient choice is

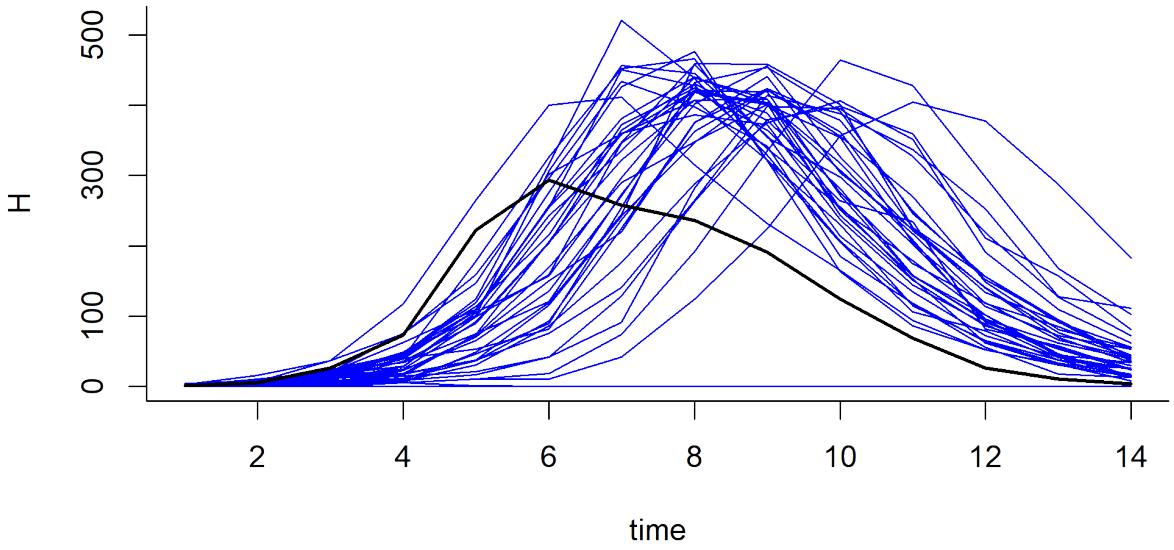
$$g_{X_{0:N}}(x_{0:N}) = f_{X_{0:N}}(x_{0:N}\,;\,\theta).$$

- This choice of $g_{X_{0:N}}(x_{0:N})$ means the numerator and denominator cancel in the importance sampling weights.
- As a consequence, for this choice of $g_{X_{0:N}}(x_{0:N})$ we never have to compute these weights.
- This means, as long as we can simulate from the dynamic model, $f_{X_{0:N}}(x_{0:N}\,;\,\theta)$, we never have to evaluate this density.
- We get the **plug-and-play** property that our algorithm depends on `rprocess` but does not require `dprocess`.
- We see that, if we generate trajectories by simulation, all we have to do to get a Monte Carlo estimate of the likelihood is evaluate the measurement density of the data at each trajectory and average.
- Let's go back to the boarding school influenza outbreak to see what this would look like.
- Let's reconstruct the toy SIR model we were working with.

```
bsflu <- read.table("bsflu_data.txt")

sir_step <- Csnippet("
  double dN_SI = rbinom(S,1-exp(-Beta*I/N*dt));
  double dN_IR = rbinom(I,1-exp(-gamma*dt));
  S -= dN_SI;
  I += dN_SI - dN_IR;
  R += dN_IR;
  H += dN_IR;
")

sir_init <- Csnippet("
  S = nearbyint(N)-1;
  I = 1;
  R = 0;
  H = 0;
")

dmeas <- Csnippet("lik = dbinom(B,H,rho,give_log);")
rmeas <- Csnippet("B = rbinom(H,rho);")

pomp(bsflu,times="day",t0=0,
     rprocess=euler.sim(sir_step,delta.t=1/5),
     initializer=sir_init,rmeasure=rmeas,dmeasure=dmeas,
     zeronames="H",statenames=c("H","S","I","R"),
     paramnames=c("Beta","gamma","rho","N"),
     cdir = getwd()) -> sir
```

- Let's generate a bunch of simulated trajectories at some particular point in parameter space.

```
simulate(sir,params=c(Beta=2,gamma=1,rho=0.5,N=2600),
         nsim=10000,states=TRUE) -> x
matplot(time(sir),t(x["H",1:50,]),type='l',lty=1,
        xlab="time",ylab="H",bty='l',col='blue')
lines(time(sir),obs(sir,"B"),lwd=2,col='black')
```



- We can use the function `dmeasure` to evaluate the log likelihood of the data given the states, the model, and the parameters:

```
ell <- dmeasure(sir,y=obs(sir),x=x,times=time(sir),log=TRUE,
                params=c(Beta=2,gamma=1,rho=0.5,N=2600))
dim(ell)
```

```
## [1] 10000    14
```

- According to the equation above, we should sum up the log likelihoods across time:

```
ell <- apply(ell,1,sum); summary(exp(ell)); logmeanexp(ell,se=TRUE)
```

```
##      Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
##  0.00e+00  0.00e+00  0.00e+00 5.37e-106  0.00e+00 5.37e-102
```

```
##                    se
## -242.39328   16.09139
```

- The likelihood appears to be very low, but the error in the estimate is very high and therefore the estimated likelihood is very imprecise.
- We are going to need very many simulations to get an estimate of the likelihood sufficiently precise to be of any use in parameter estimation or model selection.
- What's the problem? Essentially, far too many of the trajectories don't pass near the data. Moreover, once a trajectory diverges from the data, it almost never comes back.
- This is a consequence of the fact that we are proposing trajectories in a way that is completely unconditional on the data.
- The problem will only get worse with longer data sets!

The particle filter

- Fortunately, we can compute the likelihood for a POMP model by a much more efficient algorithm than direct Monte Carlo integration.
- We proceed by factorizing the likelihood in a different way:

$$\begin{aligned}\mathcal{L}(\theta) &= f_{Y_{1:N}}(y^*_{1:N}\,;\,\theta) && (5)\\ &= \prod_n f_{Y_N|Y_{1:N-1}}(y^*_n|y^*_{1:n-1}\,;\,\theta) && (6)\\ &= \prod_n \int f_{Y_n|X_n}(y^*_n|x_n\,;\,\theta)\, f_{X_n|Y_{1:n-1}}(x_n|y^*_{1:n-1}\,;\,\theta)\, dx_n. && (7)\end{aligned}$$

- The Markov property leads to the prediction formula:

$$\begin{aligned}&f_{X_n|Y_{1:n-1}}(x_n|y^*_{1:n-1}\,;\,\theta) && (8)\\ &= \int_{x_{n-1}} f_{X_n|X_{n-1}}(x_n|x_{n-1}\,;\,\theta)\, f_{X_{n-1}|Y_{1:n-1}}(x_{n-1}\,|\,y^*_{1:n-1}\,;\,\theta)\, dx_{n-1}. && (9)\end{aligned}$$

- Bayes' theorem gives the filtering formula:

$$\begin{aligned}&f_{X_n|Y_{1:n}}(x_n|y^*_{1:n}\,;\,\theta) && (10)\\ &= f_{X_n|Y_n,Y_{1:n-1}}(x_n|y^*_n, y^*_{1:n-1}\,;\,\theta) && (11)\\ &= \frac{f_{Y_n|X_n}(y^*_n|x_n\,;\,\theta)\, f_{X_n|Y_{1:n-1}}(x_n|y^*_{1:n-1}\,;\,\theta)}{\int f_{Y_n|X_n}(y^*_n|u_n\,;\,\theta)\, f_{X_n|Y_{1:n-1}}(u_n|y^*_{1:n-1}\,;\,\theta)\, du_n}. && (12)\end{aligned}$$

- This suggests that we keep track of two key distributions. We'll refer to the distribution of $X_n|Y_{1:n-1}{=}y^*_{1:n-1}$ as the **prediction distribution** at time t_n and the distribution of $X_n|Y_{1:n}{=}y^*_{1:n}$ as the **filtering distribution** at time t_n.
- Let's use Monte Carlo techniques to estimate the integrals in the prediction and filtering recursion equations.
- Consider the following Monte Carlo scheme:

1. Suppose $X^F_{n-1,j}$, $j = 1,\dots,J$ is a set of J points drawn from the filtering distribution at time $n-1$.
2. We obtain a sample $X^P_{n,j}$ of points drawn from the prediction distribution at time t by simply simulating the process model:
$$X^P_{n,j} \sim \text{process}(X^F_{n-1,j},\theta), \qquad j = 1,\dots,J.$$
3. Having obtained $x^P_{n,j}$, we obtain a sample of points from the filtering distribution at time t_n by *resampling* from $\left\{X^P_{n,j}, j \in 1:J\right\}$ with weights
$$w_{n,j} = f_{Y_n|X_n}(y^*_n|X^P_{n,j}\,;\,\theta).$$
4. The Monte Carlo principle tells us that the conditional likelihood

$$\begin{aligned}\mathcal{L}_n(\theta) &= f_{Y_n|Y_{1:n-1}}(y^*_n|y^*_{1:n-1}\,;\,\theta) && (13)\\ &= \int f_{Y_n|X_n}(y^*_n|x_n\,;\,\theta)\, f_{X_n|Y_{1:n-1}}(x_n|y^*_{1:n-1}\,;\,\theta)\, dx_n && (14)\end{aligned}$$

is approximated by

$$\hat{\mathcal{L}}_n(\theta) \approx \frac{1}{N} \sum_j f_{Y_n|X_n}(y_n^*|X_{n,j}^P\,;\,\theta).$$

5. We can iterate this procedure through the data, one step at a time, alternately simulating and resampling, until we reach $n = N$.

6. The full log likelihood then has approximation

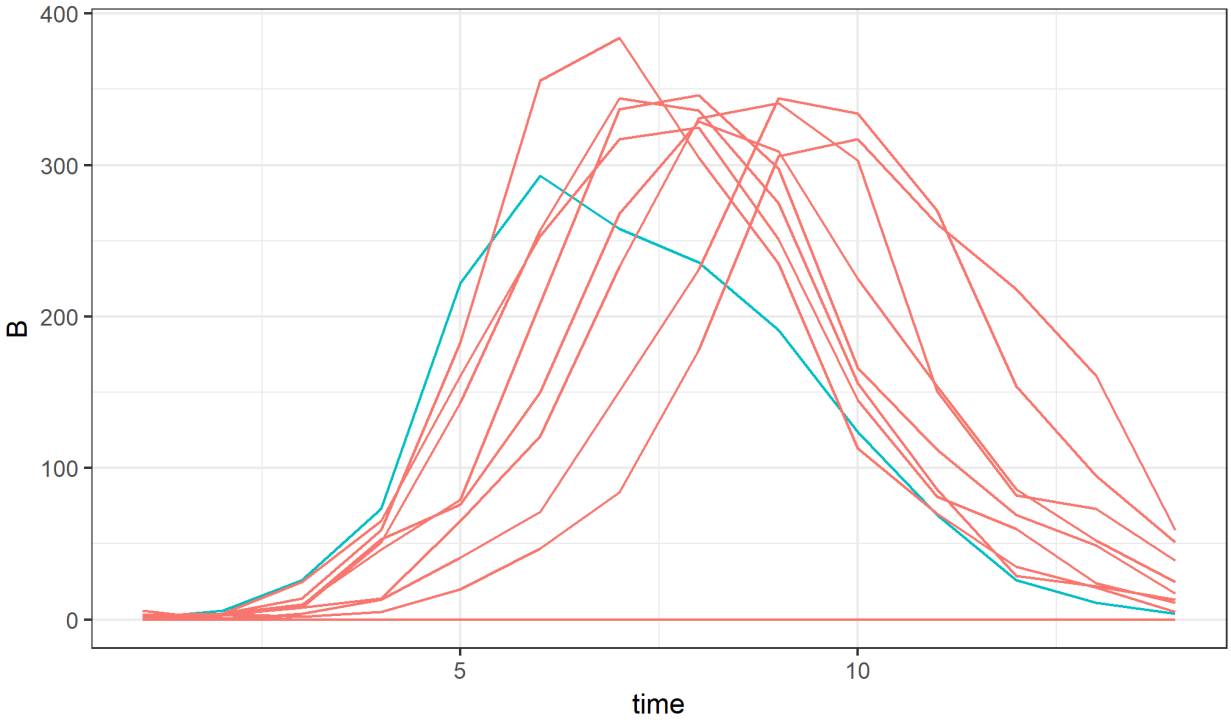
$$\begin{aligned} \ell(\theta) &= \log \mathcal{L}(\theta) && (15)\\ &= \sum_n \log \mathcal{L}_n(\theta) && (16)\\ &\approx \sum_n \log \hat{\mathcal{L}}_n(\theta). && (17) \end{aligned}$$

- The above procedure is known as the **sequential Monte Carlo** (SMC) algorithm or the **particle filter**.
- Good references for further reading include Kitagawa (1987), Arulampalam et al. (2002) and the book edited by Doucet et al. (2001).

Sequential Monte Carlo in pomp

- Here, we'll get some practical experience with the particle filter, and the likelihood function, in the context of our influenza-outbreak case study.

```
sims <- simulate(sir,params=c(Beta=2,gamma=1,rho=0.8,N=2600),nsim=20,
                 as.data.frame=TRUE,include.data=TRUE)

ggplot(sims,mapping=aes(x=time,y=B,group=sim,color=sim=="data"))+
  geom_line()+guides(color=FALSE)
```



- In **pomp**, the basic particle filter is implemented in the command `pfilter`. We must choose the number of particles to use by setting the `Np` argument.

```
pf <- pfilter(sir,Np=5000,params=c(Beta=2,gamma=1,rho=0.8,N=2600))
logLik(pf)
```

```
## [1] -69.73054
```

- We can run a few particle filters to get an estimate of the Monte Carlo variability:

```
pf <- replicate(10,pfilter(sir,Np=5000,params=c(Beta=2,gamma=1,rho=0.8,N=2600)))
ll <- sapply(pf,logLik); ll
```

```
##  [1] -76.60686 -82.19224 -80.33033 -88.14655 -71.78196 -71.64952 -84.32846
##  [8] -82.05719 -76.90641 -77.36237
```

```
logmeanexp(ll,se=TRUE)
```

```
##                    se
## -73.3146129   0.8277326
```

- A theoretical property of the particle filter is that it gives us an unbiased Monte Carlo estimate of the likelihood.
- This theoretical property, combined with Jensen's inequality and the observation that $\log(x)$ is a concave function, ensures that the average of the log likelihoods from many particle filter replications will have negative bias as a Monte Carlo estimator of the log likelihood.
- We've been careful to avoid this bias in the code above, by using `logmeanexp` to average the likelihood estimates on the natural scale not the logarithmic scale.

The graph of the likelihood function: The likelihood surface

- Intuitively, it can be helpful to think of the geometric surface defined by the likelihood function.
- If Θ is two-dimensional, then the surface $\ell(\theta)$ has features like a landscape.
  - Local maxima of $\ell(\theta)$ are peaks
  - local minima are valleys
  - peaks may be separated by a valley or may be joined by a ridge. If you go along the ridge, you may be able to go from one peak to the other without losing much elevation. Narrow ridges can be easy to fall off, and hard to get back on to.
- In higher dimensions, one can still think of peaks and valleys and ridges. However, as the dimension increases it quickly becomes hard to imagine the surface.
- To get an idea of what the likelihood surface looks like in the neighborhood of the default parameter set supplied by `sir`, we can construct a **likelihood slice**.
- We'll make slices in the β and γ directions. Both slices will pass through the default parameter set.

```
sliceDesign(
  c(Beta=2,gamma=1,rho=0.8,N=2600),
  Beta=rep(seq(from=0.5,to=4,length=40),each=3),
  gamma=rep(seq(from=0.5,to=2,length=40),each=3)) -> p

require(foreach)
require(doMC)

registerDoMC(cores=5)
## number of cores
## usually the number of cores on your machine, or slightly smaller

set.seed(998468235L,kind="L'Ecuyer")
mcopts <- list(preschedule=FALSE,set.seed=TRUE)

foreach (theta=iter(p,"row"),.combine=rbind,
         .inorder=FALSE,.options.multicore=mcopts) %dopar%
 {
   pfilter(sir,params=unlist(theta),Np=5000) -> pf
   theta$loglik <- logLik(pf)
   theta
 } -> p
```

- You can see from the code what is meant by a likelihood slice.

Exercise: Write down the definition of a likelihood slice in mathematical notation.

Exercise: Note that a likelihood slice is different from a likelihood profile

- What is the difference between a likelihood slice and a profile from a computational perspective?
- What is the difference between the statistical interpretation of a likelihood slice and a profile. For example, compare their roles in constructing confidence intervals and hypothesis tests.

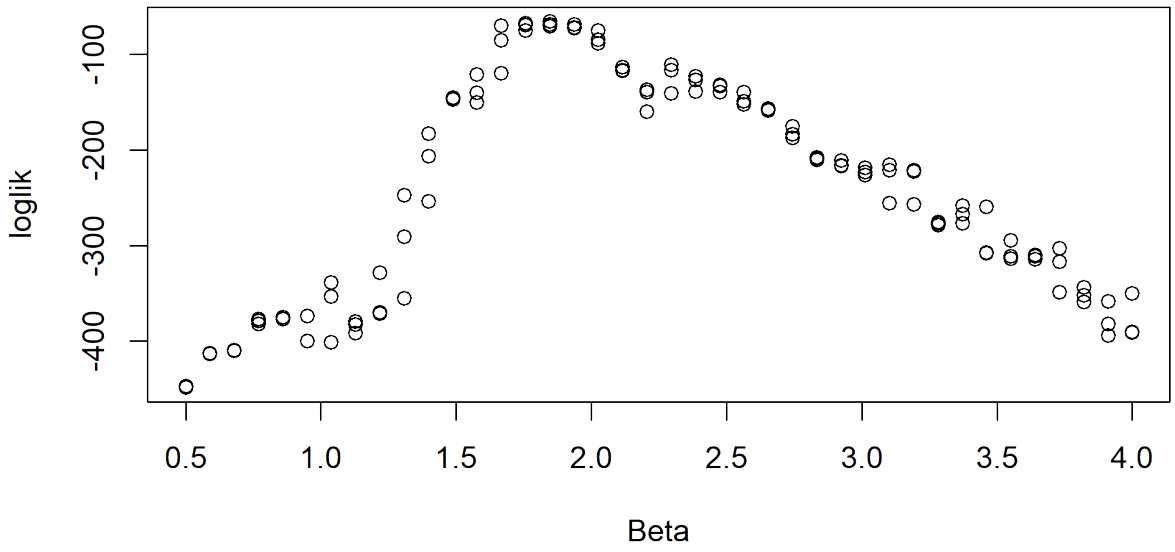
- What is the purpose of computing a likelihood slice? What is the purpose of computing a likelihood profile?

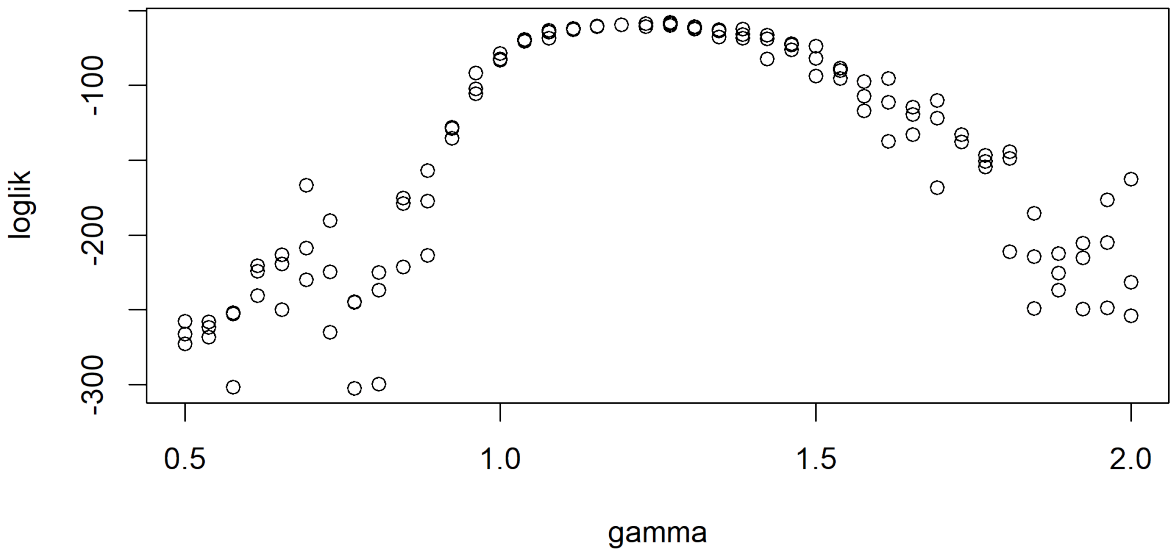
_____

_____

- Note that the slice computation used the **foreach** package with the multicore backend (**doMC**) to parallelize these computations.
- To ensure that we have high-quality random numbers in each parallel *R* session, we use a parallel random number generator. Notice the specification `kind="L'Ecuyer"` when we set the seed for the random number generator, and the specification `.options.multicore=mcopts` for `foreach`, the parallel for loop
- Now, we can plot the likelihood slices:

```
foreach (v=c("Beta","gamma")) %do%
{
  x <- subset(p,slice==v)
  plot(x[[v]],x$loglik,xlab=v,ylab="loglik")
}
```





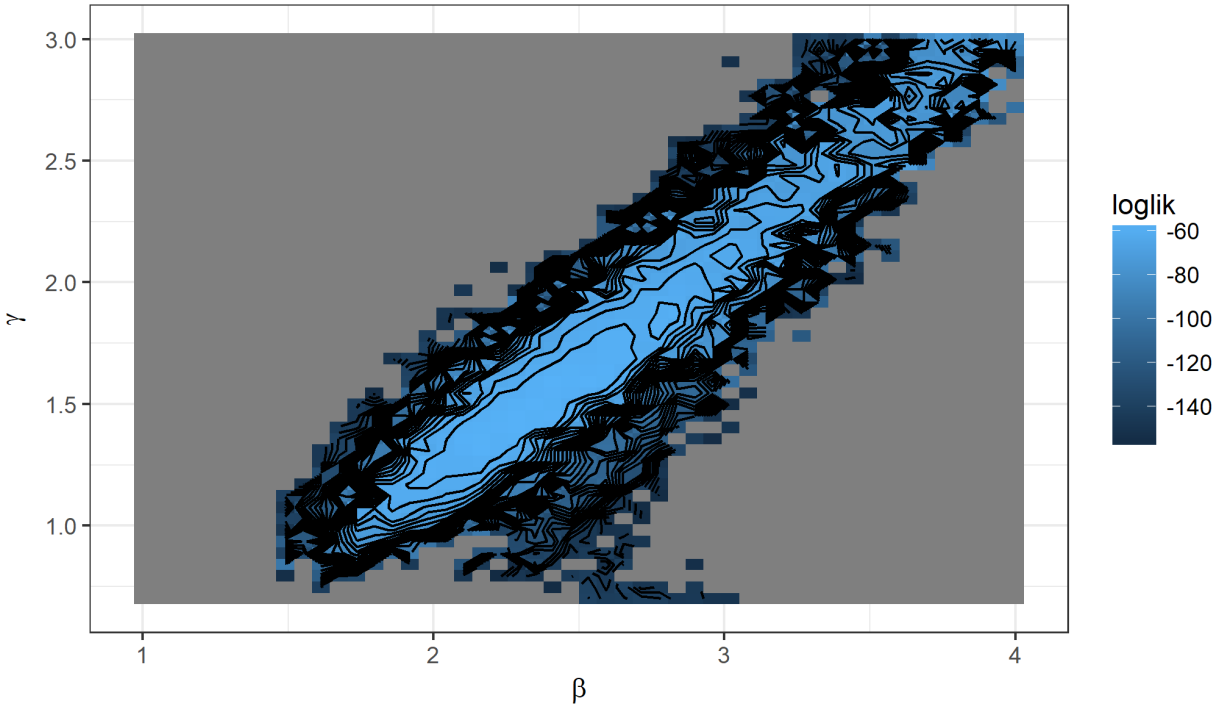
Exercise: Likelihood slices

Add likelihood slices along the ρ direction.

Slices offer a very limited perspective on the geometry of the likelihood surface. With just two parameters, we can evaluate the likelihood at a grid of points and visualize the surface directly.

```
expand.grid(Beta=seq(from=1,to=4,length=50),
            gamma=seq(from=0.7,to=3,length=50),
            rho=0.8,
            N=2600) -> p

foreach (theta=iter(p,"row"),.combine=rbind,
         .inorder=FALSE,.options.multicore=mcopts) %dopar%
 {
   pfilter(sir,params=unlist(theta),Np=5000) -> pf
   theta$loglik <- logLik(pf)
   theta
 } -> p
```

```
pp <- mutate(p,loglik=ifelse(loglik>max(loglik)-100,loglik,NA))
ggplot(data=pp,mapping=aes(x=Beta,y=gamma,z=loglik,fill=loglik))+
  geom_tile(color=NA)+
  geom_contour(color='black',binwidth=3)+
  scale_fill_gradient()+
  labs(x=expression(beta),y=expression(gamma))
```



Exercise: 2D likelihood slice

Compute a slice of the likelihood in the β-N plane.

Maximizing the likelihood using the particle filter

- First, a few cautionary words about parameter estimation for dynamic models

Causal, mechanistic interpretation of parameter estimates

- When we write down a mechanistic model for a system, we have some idea of what we intend parameters to mean. In epidemiology, for example, we interpret parameters as a reporting rate, a contact rate between individuals, an immigration rate, a duration of immunity, etc.
- The data and the parameter estimation procedure do not know about our intended interpretation of the model. It can and does happen that some parameter estimates statistically consistent with the data may be scientifically absurd according to the scientific reasoning that went into building the model.
- This can arise as a consequence of weak identifiability.
- It can also be a warning that the data do not agree that our model represents reality in the way we had hoped. Perhaps more work is needed on model development.
- Scientifically unreasonable parameter estimates can sometimes be avoided by fixing some parameters at known, reasonable values. However, this risks suppressing the warning that the data were trying to give about weaknesses in the model, or in the biological interpretation of it.

- This issue will be discussed further when it arises in case studies.

Likelihood maximization for the boarding school flu example

- We saw above that the default parameter set for the 'bsflu' pomp object is not particularly close to the MLE.
- One way to find the MLE is to try optimizing the estimated likelihood, computed by the particle filter, directly.
- There are many optimization algorithms to choose from, and many implemented in R.
- Three issues arise immediately.

1. The particle filter gives us a stochastic estimate of the likelihood. We can reduce this variability by making the number of particles, `Np`, larger. However, we cannot make it go away. If we use a deterministic optimizer (i.e., one that assumes the objective function is evaluated deterministically), then we must control this variability somehow. For example, we can fix the seed of the pseudo-random number generator. A side effect will be that the objective function becomes jagged, marked by many small local knolls and pits. Alternatively, we can use a stochastic optimization algorithm, with which we will be only be able to obtain estimates of our MLE. This is the trade-off between a noisy and a rough objective function.
2. Because the particle filter gives us just an estimate of the likelihood and no information about the derivative, we must choose an algorithm that is "derivative-free". There are many such, but we can expect less efficiency than would be possible with derivative information. Note that finite differencing is not an especially promising way of constructing derivatives. The price would be a n-fold increase in cpu time, where n is the dimension of the parameter space. Also, since the likelihood is only estimated, we would expect the derivative estimates to be noisy.
3. Finally, the parameters are constrained to be positive, and $\rho < 1$. We must therefore select an optimizer that can solve this *constrained maximization problem*, or figure out some of way of turning it into an unconstrained maximization problem. For the latter purpose, one can transform the parameters onto a scale on which there are no constraints.

An iterated filtering algorithm (IF2)

- We use the IF2 algorithm of Ionides et al. (2015).
- A particle filter is carried out with the parameter vector for each particle doing a random walk.
- At the end of the time series, the collection of parameter vectors is recycled as starting parameters for a new particle filter with a smaller random walk variance.
- Theoretically, this procedure converges toward the region of parameter space maximizing the maximum likelihood.
- Empirically, we can test this claim on examples.

IF2 algorithm pseudocode

model input: Simulators for $f_{X_0}(x_0;\theta)$ and $f_{X_n|X_{n-1}}(x_n|x_{n-1};\theta)$; evaluator for $f_{Y_n|X_n}(y_n|x_n;\theta)$; data, $y^*_{1:N}$

algorithmic parameters: Number of iterations, M; number of particles, J; initial parameter swarm, $\{\Theta^0_j, j=1,\dots,J\}$; perturbation density, $h_n(\theta|\varphi;\sigma)$; perturbation scale, $\sigma_{1:M}$

output: Final parameter swarm, $\{\Theta^M_j, j=1,\dots,J\}$

1. For m in 1:M
2. $\quad\Theta^{F,m}_{0,j} \sim h_0(\theta|\Theta^{m-1}_j;\sigma_m)$ for j in 1:J
3. $\quad X^{F,m}_{0,j} \sim f_{X_0}(x_0;\Theta^{F,m}_{0,j})$ for j in 1:J
4. $\quad$For n in 1:N
5. $\quad\quad\Theta^{P,m}_{n,j} \sim h_n(\theta|\Theta^{F,m}_{n-1,j},\sigma_m)$ for j in 1:J
6. $\quad\quad X^{P,m}_{n,j} \sim f_{X_n|X_{n-1}}(x_n|X^{F,m}_{n-1,j};\Theta^{P,m}_j)$ for j in 1:J
7. $\quad\quad w^m_{n,j} = f_{Y_n|X_n}(y^*_n|X^{P,m}_{n,j};\Theta^{P,m}_{n,j})$ for j in 1:J
8. $\quad\quad$Draw $k_{1:J}$ with $P[k_j=i] = w^m_{n,i}\Big/\sum_{u=1}^J w^m_{n,u}$
9. $\quad\quad\Theta^{F,m}_{n,j} = \Theta^{P,m}_{n,k_j}$ and $X^{F,m}_{n,j} = X^{P,m}_{n,k_j}$ for j in 1:J
10. $\quad$End For
11. $\quad$Set $\Theta^m_j = \Theta^{F,m}_{N,j}$ for j in 1:J
12. End For

comments:

- The N loop (lines 4 through 10) is a basic particle filter applied to a model with stochastic perturbations to the parameters.
- The M loop repeats this particle filter with decreasing perturbations.
- The superscript F in $\Theta^{F,m}_{n,j}$ and $X^{F,m}_{n,j}$ denote solutions to the *filtering* problem, with the particles $j=1,\dots,J$ providing a Monte Carlo representation of the conditional distribution at time n given data $y^*_{1:n}$ for filtering iteration m.
- The superscript P in $\Theta^{P,m}_{n,j}$ and $X^{P,m}_{n,j}$ denote solutions to the *prediction* problem, with the particles $j=1,\dots,J$ providing a Monte Carlo representation of the conditional distribution at time n given data $y^*_{1:n-1}$ for filtering iteration m.
- The *weight* $w^m_{n,j}$ gives the likelihood of the data at time n for particle j in filtering iteration m.

Choosing the algorithmic settings for IF2

- The initial parameter swarm, $\{\Theta^0_j, j=1,\dots,J\}$, usually consists of J identical replications of some starting parameter vector.
- J is set to be sufficient for particle filtering. By the time of the last iteration ($m=M$) one should not have effective sample size close to 1.
- Perturbations are usually chosen to be Gaussian, with σ_m being a scale factor for iteration m:

$$h_n(\theta|\varphi;\sigma) \sim N[\varphi, \sigma^2_m V_n].$$

- V_n is usually taken to be diagonal,

$$V_n = \begin{pmatrix} v^2_{1,n} & 0 & 0 & \to & 0 \\ 0 & v^2_{2,n} & 0 & \to & 0 \\ 0 & 0 & v^2_{3,n} & \to & 0 \\ \downarrow & & & \searrow & \downarrow \\ 0 & 0 & 0 & \to & v^2_{p,n} \end{pmatrix}.$$

- If θ_i is a parameter that affects the dynamics or observations throughout the timeseries, it is called a **regular parameter**, and it is often appropriate to specify

$$v_{i,n} = v_i,$$

- If θ_j is a parameter that affects only the initial conditions of the dynamic model, it is called an **initial value parameter** (IVP) and it is appropriate to specify

$$v_{j,n} = \begin{cases} v_j & \text{if } n = 0 \\ 0 & \text{if } n > 0 \end{cases}$$

- If θ_k is a break-point parameter that models how the system changes at time t_q then θ_k is like an IVP at time t_q and it is appropriate to specify

$$v_{j,n} = \begin{cases} v_j & \text{if } n = q \\ 0 & \text{if } n \neq q \end{cases}$$

- $\sigma_{1:M}$ is called a **cooling schedule**, following a thermodynamic analogy popularized by simulated annealing. As σ_m becomes small, the system cools toward a "freezing point". If the algorithm is working sucessfully, the freezing point should be close to the lowest-energy state of the system, i.e., the MLE.
- It is generally helpful for optimization to provide transformations of the parameters so that (on the estimation scale) they are real-valued and have uncertainty on the order of 1 unit. For example, one typically takes a logarithmic transformation of positive parameters and a logistic transformation of $[0, 1]$ valued parameters.
- On this scale, it is surprisingly often effective to take

$$v_i = 0.02$$

  for regular parameters (RPs) and

$$v_j = 0.1$$

  for initial value parameters (IVPs).
- We suppose that $\sigma_1 = 1$, since the scale of the parameters is addressed by the matrix V_n . Early on in an investigation, one might take $M = 100$ and $\sigma_M = 0.1$. Later on, consideration of diagnostic plots may suggest refinements.
- It is surprising that useful general advice exists for these quantities that could in principle be highly model-specific. Here is one possible explanation: the precision of interest is often the second significant figure and there are often order 100 observations (10 monthly obsevations would be too few to fit a mechanistic model; 1000 would be unusual for an epidemiological system).

Applying IF2 to a boarding school influenza outbreak

- We're going to reintroduce the boarding school flu example. This provides a reminder, but also lets us develop the model and the corresponding pomp object in a way that generalizes to other situations. It will be a template for the case studies that follow.
- For a relatively simple epidemiological example of IF2, we consider fitting a stochastic SIR model to an influenza outbreak in a British boarding school (Anonymous 1978). Reports consist of the number of children confined to bed for each of the 14 days of the outbreak. The total number of children at the school was 763, and a total of 512 children spent time away from class. Only one adult developed influenza-like illness, so adults are omitted from the data and model. First, we read in the boarding school flu (bsflu) data:

```
bsflu_data <- read.table("bsflu_data.txt")
```

- Our model is a variation on a basic SIR Markov chain, with state $X(t) = (S(t), I(t), R_1(t), R_2(t), R_3(t))$ giving the number of individuals in the susceptible and infectious categories, and three stages of recovery.
- The recovery stages, R_1, R_2 and R_3, are all modeled to be non-contagious.
- R_1 consists of individuals who are bed-confined if they show symptoms.
- R_2 consists of individuals who are convalescent if they showed symptoms.
- R_3 consists of recovered individuals who have returned to schoolwork if they were symtomatic.
- The observation on day n of the observed epidemic (with $n = 1$ being 22 January) consists of the numbers of children who are bed-confined and convalescent.
- These measurements are modeled as $Y_n = (B_n, C_n)$ with $B_n \sim \text{Poisson}(\rho R_1(t_n))$ and $C_n \sim \text{Poisson}(\rho R_2(t_n))$.
- Here, ρ is a reporting rate corresponding to the chance of being symptomatic.
- The index case for the epidemic was proposed to be a boy returning to Britain from Hong Kong, who was reported to have a transient febrile illness from 15 to 18 January. It would therefore be reasonable to initialize the epidemic with $I(t_0) = 1$ at $t_0 = -6$. This is a little tricky to reconcile with the rest of the data; for now, we avoid this issue by instead initializing with $I(t_0) = 1$ at $t_0 = 0$. All other individuals are modeled to be initially susceptible.
- Our Markov transmission model is that each individual in S transitions to I at rate $\beta I(t)$; each individual in I transitions at rate μ_I to R_1. Subsequently, the individual moves from R_1 to R_2 at rate μ_{R_1}, and finally from R_2 to R_3 at rate μ_{R_2}.
- Therefore, $1/\mu_I$ is the mean infectious time prior to bed-confinement; $1/R_1$ is the mean duration of bed-confinement for symptomatic cases; $1/R_2$ is the mean duration of convalescence for symptomatic cases. All rates have units day^{-1}.
- This model has limitations and weaknesses. Writing down and fitting a model is a starting point for data analysis, not an end point. In particular, one should try model variations. For example, one could include a latency period for infections, or one could modify the model to give a better description of the bed-confinement and convalescence processes. Ten individuals received antibiotics for secondary infections, and they had longer bed-confinement and convalescence times. Partly for this reason, we will initially fit only the bed-confinement data, using $Y_n = B_n$ for our `dmeasure`.
- For the code, we represent the states (S, I, R_1, R_2) and the parameters (β, μ_I, ρ, μ_{R_1}, μ_{R_2}) as follows:

```
bsflu_statenames <- c("S","I","R1","R2")
bsflu_paramnames <- c("Beta","mu_I","rho","mu_R1","mu_R2")
```

The observation names (B, C) are the names of the data variables:

```
(bsflu_obsnames <- colnames(bsflu_data)[1:2])
```

```
## [1] "B" "C"
```

We do not need a representation of R_3 since the total population size is fixed at $P = 763$ and hence $R_3(t) = P - S(t) - I(t) - R_1(t) - R_2(t)$. Now, we write the model code:

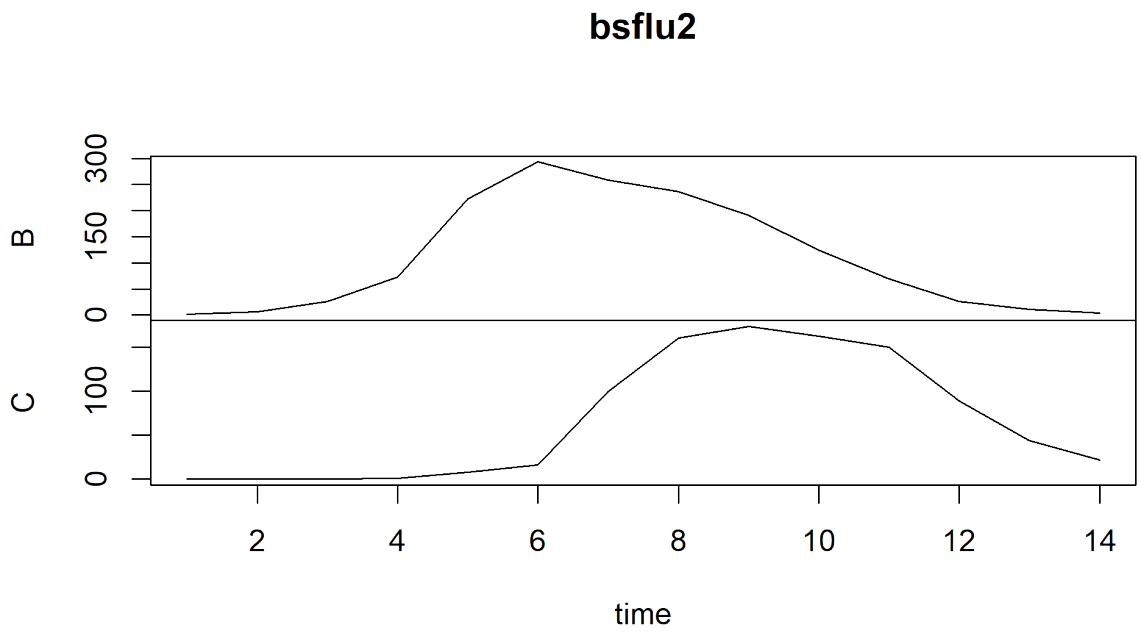
```
bsflu_dmeasure <- "
  lik = dpois(B,rho*R1+1e-6,give_log);
"

bsflu_rmeasure <- "
  B = rpois(rho*R1+1e-6);
```

```
  C = rpois(rho*R2);
"

bsflu_rprocess <- "
  double t1 = rbinom(S,1-exp(-Beta*I*dt));
  double t2 = rbinom(I,1-exp(-dt*mu_I));
  double t3 = rbinom(R1,1-exp(-dt*mu_R1));
  double t4 = rbinom(R2,1-exp(-dt*mu_R2));
  S -= t1;
  I += t1 - t2;
  R1 += t2 - t3;
  R2 += t3 - t4;
"

bsflu_fromEstimationScale <- "
 TBeta = exp(Beta);
 Tmu_I = exp(mu_I);
 Trho = expit(rho);
"

bsflu_toEstimationScale <- "
 TBeta = log(Beta);
 Tmu_I = log(mu_I);
 Trho = logit(rho);
"

bsflu_initializer <- "
 S=762;
 I=1;
 R1=0;
 R2=0;
"
```

We can now build the new `bsflu` pomp object.

```
require(pomp)
stopifnot(packageVersion("pomp")>="0.75-1")
bsflu2 <- pomp(
  data=bsflu_data,
  times="day",
  t0=0,
  rprocess=euler.sim(
    step.fun=Csnippet(bsflu_rprocess),
    delta.t=1/12
  ),
  rmeasure=Csnippet(bsflu_rmeasure),
  dmeasure=Csnippet(bsflu_dmeasure),
  fromEstimationScale=Csnippet(bsflu_fromEstimationScale),
  toEstimationScale=Csnippet(bsflu_toEstimationScale),
  obsnames = bsflu_obsnames,
  statenames=bsflu_statenames,
  paramnames=bsflu_paramnames,
  initializer=Csnippet(bsflu_initializer),
  cdir=getwd()
```

```
)
plot(bsflu2)
```



- To develop and debug code, it is nice to have a version that runs extra quickly, for which we set `run_level=1`. Here, `Np` is the number of particles (i.e., sequential Monte Carlo sample size), and `Nmif` is the number of iterations of the optimization procedure carried out below. Empirically, `Np=5000` and `Nmif=200` are around the minimum required to get stable results with an error in the likelihood of order 1 log unit for this example; this is implemented by setting `run_level=2`. One can then ramp up to larger values for more refined computations, implemented here by `run_level=3`.

```
run_level <- 3
switch(run_level,
       {bsflu_Np=100; bsflu_Nmif=10; bsflu_Neval=10; bsflu_Nglobal=10; bsflu_Nlocal=10},
       {bsflu_Np=20000; bsflu_Nmif=100; bsflu_Neval=10; bsflu_Nglobal=10; bsflu_Nlocal=10},
       {bsflu_Np=60000; bsflu_Nmif=300; bsflu_Neval=10; bsflu_Nglobal=100; bsflu_Nlocal=20}
)
```

Running a particle filter

Before engaging in iterated filtering, it is often a good idea to check that the basic particle filter is working since iterated filtering builds on this technique. Here, carrying out slightly circular reasoning, we are going to test `pfilter` on a previously computed point estimate read in from bsflu_params.csv:

```
bsflu_params <- data.matrix(read.table("mif_bsflu_params.csv",row.names=NULL,header=TRUE))
bsflu_mle <- bsflu_params[which.max(bsflu_params[,"logLik"]),][bsflu_paramnames]
```

We are going to treat μ_{R_1} and μ_{R_2} as known, fixed at the empirical mean of the bed-confinement and convalescence times for symptomatic cases:

```
bsflu_fixed_params <- c(mu_R1=1/(sum(bsflu_data$B)/512),mu_R2=1/(sum(bsflu_data$C)/512))
```

- It is convenient to do some parallelization to speed up the computations. Most machines are multi-core nowadays, and using this computational capacity involves only
  1. the following lines of code to let R know you plan to use multiple processors;
  2. using the parallel for loop provided by 'foreach'.

```
require(doParallel)
cores <- 20  # The number of cores on this machine
registerDoParallel(cores)
mcopts <- list(set.seed=TRUE)

set.seed(396658101,kind="L'Ecuyer")
```

- Note that this code uses the `doParallel` library, whereas before we used `doMC`, as below.

```
require(doMC)
registerDoMC(cores=20)
```

- `doParallel` and `doMC` are functionally equivalent for our current purposes, but give you two options if you hit technical problems accessing the multiple cores on your machine.
- We proceed to carry out replicated particle filters at this tentative MLE:

```
stew(file=sprintf("pf-%d.rda",run_level),{
  t_pf <- system.time(
    pf <- foreach(i=1:20,.packages='pomp',
                  .options.multicore=mcopts) %dopar% try(
                    pfilter(bsflu2,params=bsflu_mle,Np=bsflu_Np)
                  )
  )

},seed=1320290398,kind="L'Ecuyer")

(L_pf <- logmeanexp(sapply(pf,logLik),se=TRUE))
```

```
##                      se
## -73.3241678   0.3197182
```

- In 9.4 seconds, we obtain an unbiased likelihood estimate of -73.32 with a Monte standard error of 0.32.

Caching computations in Rmarkdown

- It is not unusual for computations in a POMP analysis to take hours to run on many cores.
- The computations for a final version of a manuscript may take days.
- Usually, we use some mechanism like the different values of `run_level` so that preliminary versions of the manuscript take less time to run.
- However, when editing the text or working on a different part of the manuscript, we don't want to re-run long pieces of code.
- Saving results so that the code is only re-run when necessary is called **caching**.
- You may already be familiar with Rmarkdown's own version of caching.

- In the notes, we tell Rmarkdown to cache. For example, in (notes13.Rmd) the first R chunk, called `knitr-opts`, contains the following:

```
opts_chunk$set(
  cache=TRUE,
  )
```

- Rmarkdown uses a library called `knitr` to process the Rmd file, so options for Rmarkdown are formally options for knitr.
- Having set the option `cache=TRUE`, Rmarkdown caches every chunk, meaning that a chunk will only be re-run if code in that chunk is edited.
- You can force Rmarkdown to recompute all the chunks by deleting the `cache` subdirectory.
- What if changes elsewhere in the document affect the proper evaluation of your chunk, but you didn't edit any of the code in the chunk itself?
  - Rmarkdown will get this wrong. **It will not recompute the chunk**.
- A perfect caching system doesn't exist. **Always delete the entire cache and rebuild a fresh cache before finishing a manuscript.**
- Rmarkdown caching is good for relatively small computations, such as producing figures or things that may take a minute or two and are annoying if you have to recompute them every time you make any edits to the text.
- For longer computations, it is good to have full manual control. In **pomp**, this is provided by two related functions, `stew` and `bake`.

`stew` and `bake`

- Notice the function `stew` in the replicated particle filter code above.
- Here, `stew` looks for a file called `pf-[run_level].rda`.
- If it finds this file, it simply loads the contents of this file.
- If the file doesn't exist, it carries out the specified computation and saves it in a file of this name.
- `bake` is similar to `stew`. The difference is that `bake` uses `readRDS` and `saveRDS`, whereas `stew` uses `load` and `save`.
- either way, the computation will not be re-run unless you manually delete `pf-[run_level].rda`.
- `stew` and `bake` reset the seed appropriately whether or not the computation is recomputed. Othewise, caching risks adverse consequences for reproducibility.

A local search of the likelihood surface

- Let's carry out a local search using `mif2` around this previously identified MLE. For that, we need to set the `rw.sd` and `cooling.fraction.50` algorithmic parameters:

```
bsflu_rw.sd <- 0.02
bsflu_cooling.fraction.50 <- 0.5

stew(file=sprintf("local_search-%d.rda",run_level),{

  t_local <- system.time({
```

```
    mifs_local <- foreach(i=1:bsflu_Nlocal,.packages='pomp', .combine=c, .options.multicore=mcopts) %do
      mif2(
        bsflu2,
        start=bsflu_mle,
        Np=bsflu_Np,
        Nmif=bsflu_Nmif,
        cooling.type="geometric",
        cooling.fraction.50=bsflu_cooling.fraction.50,
        transform=TRUE,
        rw.sd=rw.sd(
          Beta=bsflu_rw.sd,
          mu_I=bsflu_rw.sd,
          rho=bsflu_rw.sd
        )
      )

    }
  })

},seed=900242057,kind="L'Ecuyer")
```

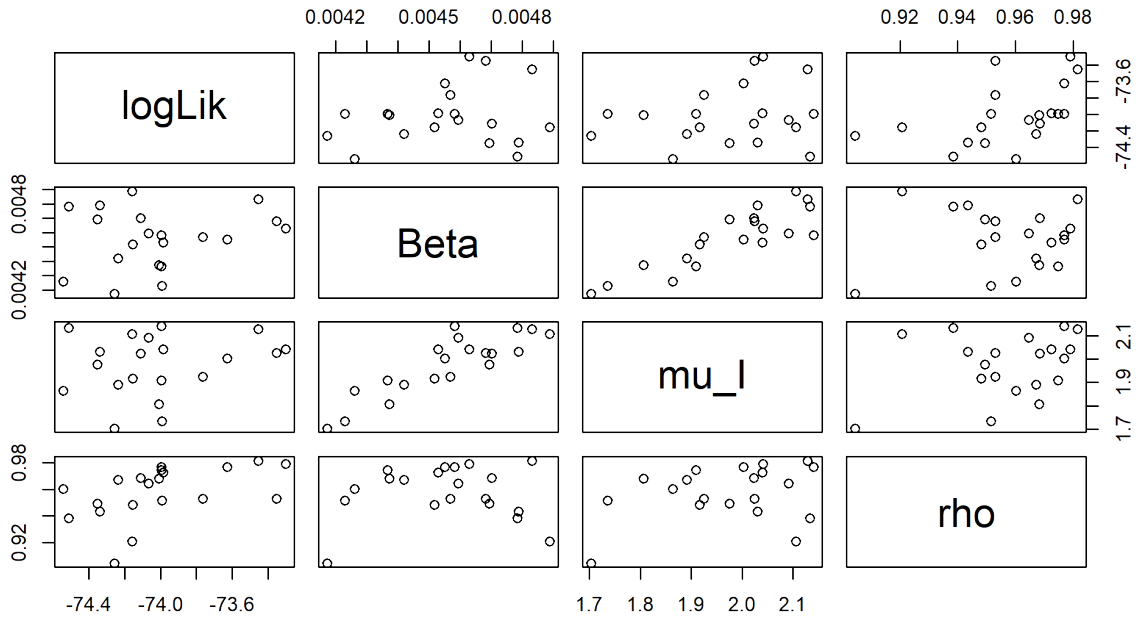
- Although the filtering carried out by `mif2` in the final filtering iteration generates an approximation to the likelihood at the resulting point estimate, this is not usually good enough for reliable inference. Partly, this is because some parameter perturbations remain in the last filtering iteration. Partly, this is because `mif2` is usually carried out with a smaller number of particles than is necessary for a good likelihood evaluation—the errors in `mif2` average out over many iterations of the filtering.
- Therefore, we evaluate the likelihood, together with a standard error, using replicated particle filters at each point estimate:

```
stew(file=sprintf("lik_local-%d.rda",run_level),{
    t_local_eval <- system.time({
    liks_local <- foreach(i=1:bsflu_Nlocal,.packages='pomp',.combine=rbind) %dopar% {
      evals <- replicate(bsflu_Neval, logLik(pfilter(bsflu2,params=coef(mifs_local[[i]]),Np=bsflu_Np)))
      logmeanexp(evals, se=TRUE)
    }
  })
},seed=900242057,kind="L'Ecuyer")

results_local <- data.frame(logLik=liks_local[,1],logLik_se=liks_local[,2],t(sapply(mifs_local,coef)))
summary(results_local$logLik,digits=5)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  -74.54  -74.24  -74.04  -74.01  -73.93  -73.30
```

- This investigation took 48.1 minutes for the maximization and 1.5 minutes for the likelihood evaluation. These repeated stochastic maximizations can also show us the geometry of the likelihood surface in a neighborhood of this point estimate:

```
pairs(~logLik+Beta+mu_I+rho,data=subset(results_local,logLik>max(logLik)-50))
```



A global search of the likelihood surface using randomized starting values

- When carrying out parameter estimation for dynamic systems, we need to specify beginning values for both the dynamic system (in the state space) and the parameters (in the parameter space). By convention, we use *initial values* for the initialization of the dynamic system and *starting values* for initialization of the parameter search.
- Practical parameter estimation involves trying many starting values for the parameters. One can specify a large box in parameter space that contains all parameter vectors which seem remotely sensible. If an estimation method gives stable conclusions with starting values drawn randomly from this box, this gives some confidence that an adequate global search has been carried out.
- For our flu model, a box containing reasonable parameter values might be

```
bsflu_box <- rbind(
  Beta=c(0.001,0.01),
  mu_I=c(0.5,2),
  rho = c(0.5,1)
)
```

- We are now ready to carry out likelihood maximizations from diverse starting points. To simplify the code, we can reset only the starting parameters from `mifs_global[[1]]` since the rest of the call to `mif2` can be read in from `mifs_global[[1]]`:

```
stew(file=sprintf("box_eval-%d.rda",run_level),{

  t_global <- system.time({
    mifs_global <- foreach(i=1:bsflu_Nglobal,.packages='pomp', .combine=c, .options.multicore=mcopts) %
```

```
      mifs_local[[1]],
      start=c(apply(bsflu_box,1,function(x)runif(1,x[1],x[2])),bsflu_fixed_params)
    )
  })
},seed=1270401374,kind="L'Ecuyer")
```

- As noted above, the approximate likelihood evaluation generated by `mif2` in the final filtering iteration is not usually good enough for reliable inference. Therefore, we evaluate the likelihood, together with a standard error, using replicated particle filters at each point estimate:

```
stew(file=sprintf("lik_global_eval-%d.rda",run_level),{
  t_global_eval <- system.time({
    liks_global <- foreach(i=1:bsflu_Nglobal,.packages='pomp',.combine=rbind, .options.multicore=mcopts)
      evals <- replicate(bsflu_Neval, logLik(pfilter(bsflu2,params=coef(mifs_global[[i]]),Np=bsflu_Np)))
      logmeanexp(evals, se=TRUE)
    }
  })
},seed=442141592,kind="L'Ecuyer")

results_global <- data.frame(logLik=liks_global[,1],logLik_se=liks_global[,2],t(sapply(mifs_global,coef)
summary(results_global$logLik,digits=5)
```

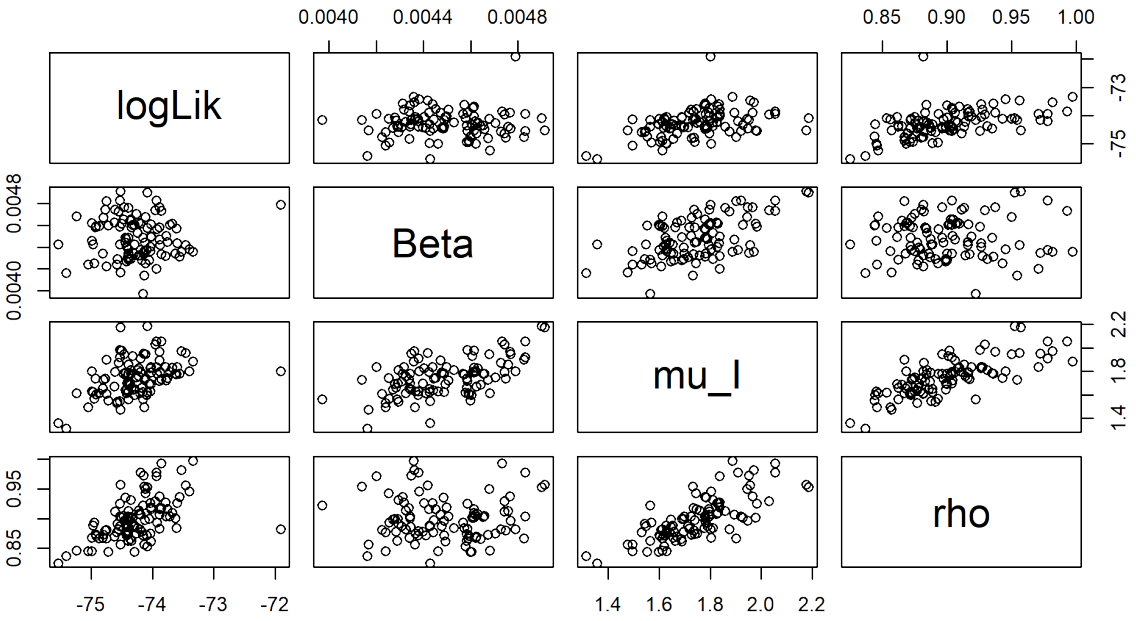
```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  -75.54  -74.53  -74.30  -74.27  -74.03  -71.91
```

- It is good practice to build up a file of successful optimization results for subsequent investigation:

```
if (run_level>2)
  write.table(rbind(results_local,results_global),
              file="mif_bsflu_params.csv",append=TRUE,col.names=FALSE,row.names=FALSE)
```

- Evaluation of the best result of this search gives a likelihood of -71.9 with a standard error of 2.8. This took in 240.9 minutes for the maximization and 7.6 minutes for the evaluation. Plotting these diverse parameter estimates can help to give a feel for the global geometry of the likelihood surface

```
pairs(~logLik+Beta+mu_I+rho,data=subset(results_global,logLik>max(logLik)-250))
```



- We see that optimization attempts from diverse remote starting points end up with comparable likelihoods, even when the parameter values are quite distinct. This gives us some confidence in our maximization procedure.

Diagnosing success or failure of the maximization procedure

- The `plot` method for an object of class `mif2d.pomp` presents some graphical convergence and filtering diagnostics for the maximization procedure.
- It is often useful to look at superimposed convergence diagnostic plots for multiple Monte Carlo replications of the maximization procedure, perhaps with different starting values.
- Concatenating objects of class `mif2d.pomp` gives a list of class `mif2List`. The `plot` method for a mif2List object gives us the superimposed convergence diagnostic plots.
- Above, we built a list of mif2d.pomp objects for the global maximum likelihood search, fitting a model to the boarding school flu data. Let's first check the classes of the resulting objects.

```
class(mifs_global)
```

```
## [1] "mif2List"
## attr(,"package")
## [1] "pomp"
```

```
class(mifs_global[[1]])
```

```
## [1] "mif2d.pomp"
## attr(,"package")
## [1] "pomp"
```

```
class(c(mifs_global[[1]],mifs_global[[2]]))
```

```
## [1] "mif2List"
## attr(,"package")
## [1] "pomp"
```

- Now, we can look at the diagnostics.

Question: Interpret the diagnostics

1. Do these plots suggest we have successfully maximized the likelihood, or not? Why?
2. What would the convergence plots look like if we cooled too quickly? Or too slowly? Can you find evidence for either of these here? (The algorithmic parameter `cooling.fraction.50` is the fraction by which we decrease the random walk standard deviation in 50 filtering iterations.)
3. Here, we've done 300 mif iterations. Should we have done more? Could we have saved ourselves computational effort by doing less, without compromising our analysis?
4. Some parameter estimates show strong agreement between the different mif runs from different starting values. Others less so. How do you interpret this? Diversity in parameter estimates could be a signal of poor numerical maximization. It could signal a multi-modal likelihood surface. Or, it could simply correspond to a flat likelihood surface where the maximum is not precisely identifiable. Can we tell from the diagnostic plots which of these is going on here?
5. Maximization via particle filtering requires that the particle filter is working effectively. One way to monitor this is to pay attention to the effective sample size on the last filtering iteration. The effective sample size (ESS) is computed here as

$$\mathrm{ESS}_n = \frac{\left(\sum_{j=1}^{J} w_{n,j}\right)^2}{\sum_{j=1}^{J} w_{n,j}^2},$$

where $\{w_{n,j}\}$ are the weights defined in step 3 of the particle filter. The ESS approximates the number of independent, equally weighted, samples from the filtering distribution that would be equally informative to the one weighted sample that we have obtained by the particle filter. Do you have any concerns about the number of particles?

Exercises

Exercise: Construct a profile likelihood

- How strong is the evidence about the contact rate, β, given this model and data? Use `mif2` to construct a profile likelihood. Due to time constraints, you may be able to compute only a preliminary version.
- It is also possible to profile over the basic reproduction number, $R_0 = \beta P/\mu_I$. Is this more or less well determined that β for this model and data?

Exercise: Check the model source code

- Check the source code for the `bsflu pomp` object. Does the code implement the model described?
- For various reasons, it can be surprisingly hard to make sure that the written equations and the code are perfectly matched. Here are some things to think about:

1. Papers should be written to be readable to as broad a community as possible. Code must be written to run successfully. People do not want to clutter papers with numerical details which they hope and belief are scientifically irrelevant. What problems can arise due to this, and what solutions are available?
2. Suppose that there is an error in the coding of `rprocess`. Suppose that plug-and-play statistical methodology is used to infer parameters. A conscientious researcher carries out a simulation study, using `simulate` to generate some realizations from the fitted model and checking that the inference methodology can successfully recover the known parameters for this model, up to some statistical error. Will this procedure help to identify the error in `rprocess`? If not, how does one debug `rprocess`? What research practices help minimize the risk of errors in simulation code?

Exercise: Assessing and improving algorithmic parameters

Develop your own heuristics to try to improve the performance of `mif2` in the previous example. Specifically, for a global optimization procedure carried out using random starting values in the specified box, let $\hat{\Theta}_{\max}$ be a random Monte Carlo estimate of the resulting MLE, and let $\hat{\theta}$ be the true (unknown) MLE. We can define the maximization error in the log likelihood to be

$$e = \ell(\hat{\theta}) - E[\ell(\hat{\Theta}_{\max})].$$

We cannot directly evaluate e, since there is also Monte Carlo error in our evaluation of $\ell(\theta)$, but we can compute it up to a known precision. Plan some code to estimates e for a search procedure using a computational effort of $JM = 2 \times 10^7$, comparable to that used for each mif computation in the global search. Discuss the strengths and weaknesses of this quantification of optimization success. See if you can choose J and M subject to this constraint, together with choices of `rw.sd` and the cooling rate, `cooling.fraction.50`, to arrive at a quantifiably better procedure. Computationally, you may not be readily able to run your full procedure, but you could run a quicker version of it.

Exercise: Finding sharp peaks in the likelihood surface

- Even in this small, 3 parameter, example, it takes a considerable amount of computation to find the global maximum (with values of β around 0.004) starting from uniform draws in the specified box. The problem is that, on the scale on which "uniform" is defined, the peak around $\beta \approx 0.004$ is very narrow. Propose and test a more favorable way to draw starting parameters for the global search, with better scale invariance properties.

Exercise: Adding a latent class

- Modify the model to include a latent period between becoming exposed and becoming infectious. See what effect this has on the maximized likelihood.

Acknowledgment

These notes draw on material developed for a short course on Simulation-based Inference for Epidemiological Dynamics by Aaron King and Edward Ionides, taught at the University of Washington Summer Institute in Statistics and Modeling in Infectious Diseases, 2015.

References

Anonymous. 1978. Influenza in a boarding school. British Medical Journal 1:587.

Arulampalam, M. S., S. Maskell, N. Gordon, and T. Clapp. 2002. A tutorial on particle filters for online nonlinear, non-Gaussian Bayesian tracking. IEEE Trans. Sig. Proc. 50:174–188.

Doucet, A., N. de Freitas, and N. Gordon, editors. 2001. Sequential Monte Carlo Methods in Practice. Springer-Verlag, New York.

Ionides, E. L., D. Nguyen, Y. Atchadé, S. Stoev, and A. A. King. 2015. Inference for dynamic and latent variable models via iterated, perturbed Bayes maps. Proceedings of the National Academy of Sciences of USA 112:719–724.

Kitagawa, G. 1987. Non-Gaussian state-space modeling of nonstationary time series. Journal of the American Statistical Association 82:1032–1041.