Chapter 12. Practical likelihood-based inference for POMP models

Objectives

1. Understanding the simplest **particle filter** and how it enables Monte Carlo solution of the POMP filtering and prediction recursions and computation of a Monte Carlo evaluation of the likelihood.
2. Using the particle filter to visualize and exploring likelihood surfaces
3. Understanding how iterated filtering algorithms carry out repeated particle filtering operations, with randomly perturbed parameter values, in order to maximize the likelihood.
4. Carrying out likelihood-based inferences for dynamic models using simulation-based statistical methodology in the R package `pomp`, demonstrated by fitting an SIR model to a boarding school flu outbreak.

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

- Suppose $X_n = x_n(\theta)$ is a known function of θ for each n. Equivalently, consider fitting the deterministic skeleton for a POMP. 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

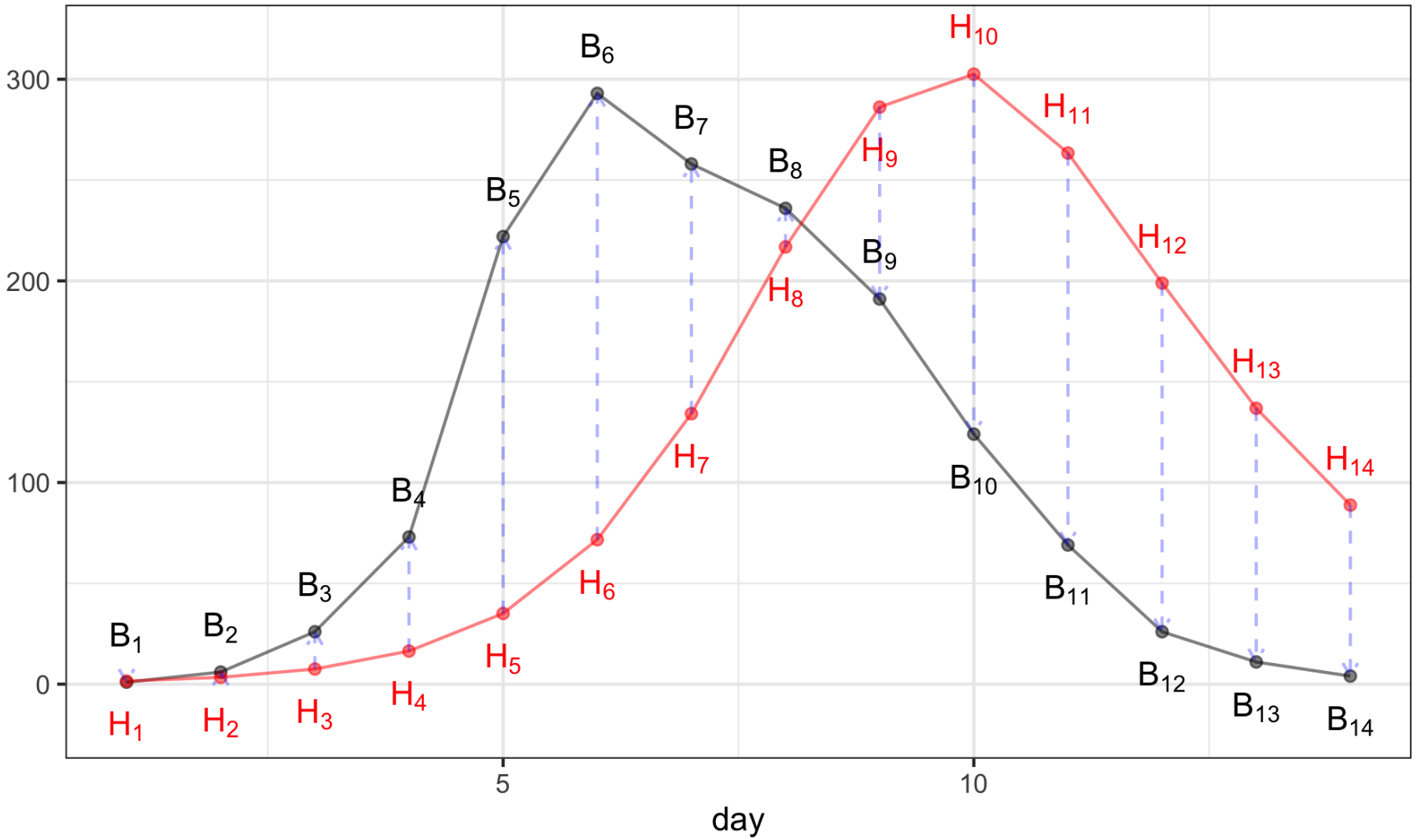
$$\mathcal{L}(\theta) = \prod_n f_{Y_n|X_n}(y_n^* \,|\, x_n(\theta)\,;\theta)$$

or

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

- This situation includes linear or nonlinear regression: with a Gaussian measurement model, maximum likelihood corresponds to least squares.

The skeleton of an SIR compared to the flu dataset



- Minimizing a sum of squared errors for the skeleton is a simple comparison. We want to do better, and fit the full POMP model.

An ineffective method: Likelihood by direct simulation

To motivate the particle filter, we first introduce a simpler method which usually does not work on anything but very short time series. We calculate:

$$\begin{aligned}
\mathcal{L}(\theta) &= f_{Y_{1:N}}(y_{1:N}\,;\theta) \\
&= \int_{x_{0:N}} f_{X_0}(x_0\,;\theta) \prod_{n=1}^{N} f_{Y_n|X_n}(y_n \mid x_n\,;\theta)\, f_{X_n|X_{n-1}}(x_n|x_{n-1}\,;\theta)\, dx_{0:N} \\
&= \mathbb{E}\left[\prod_{n=1}^{N} f_{Y_n|X_n}(y_n \mid X_n\,;\theta)\right] \\
&\approx \frac{1}{J}\sum_{j=1}^{J}\prod_{n=1}^{N} f_{Y_n|X_n}(y_n \mid X_{n,j}\,;\theta)
\end{aligned}$$

where we have J independent simulated trajectories $\{X_{nj}, n = 1, \ldots, N\}$, and $\approx$ is justified by the laws of large numbers.

Question 12.1. Why is this approach ineffective unless time series is very short?

The particle filter

- Fortunately, we can compute the likelihood for a POMP model much more efficiently by using Monte Carlo representations of the prediction and filtering recursions.
- This gives the **particle filter** algorithm, also known as sequential Monte Carlo (SMC):

1. Suppose $X^F_{n-1,j}$, $j = 1, \ldots, J$ 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:

$$X^P_{n,j} \sim \text{process}(X^F_{n-1,j}, \theta), \qquad j = 1, \ldots, J.$$

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

$$w_{n,j} = f_{Y_n|X_n}(y^*_n|X^P_{n,j}\,;\theta).$$

The likelihood via the particle filter

- The Monte Carlo principle tells us that the conditional likelihood

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

  is approximated by

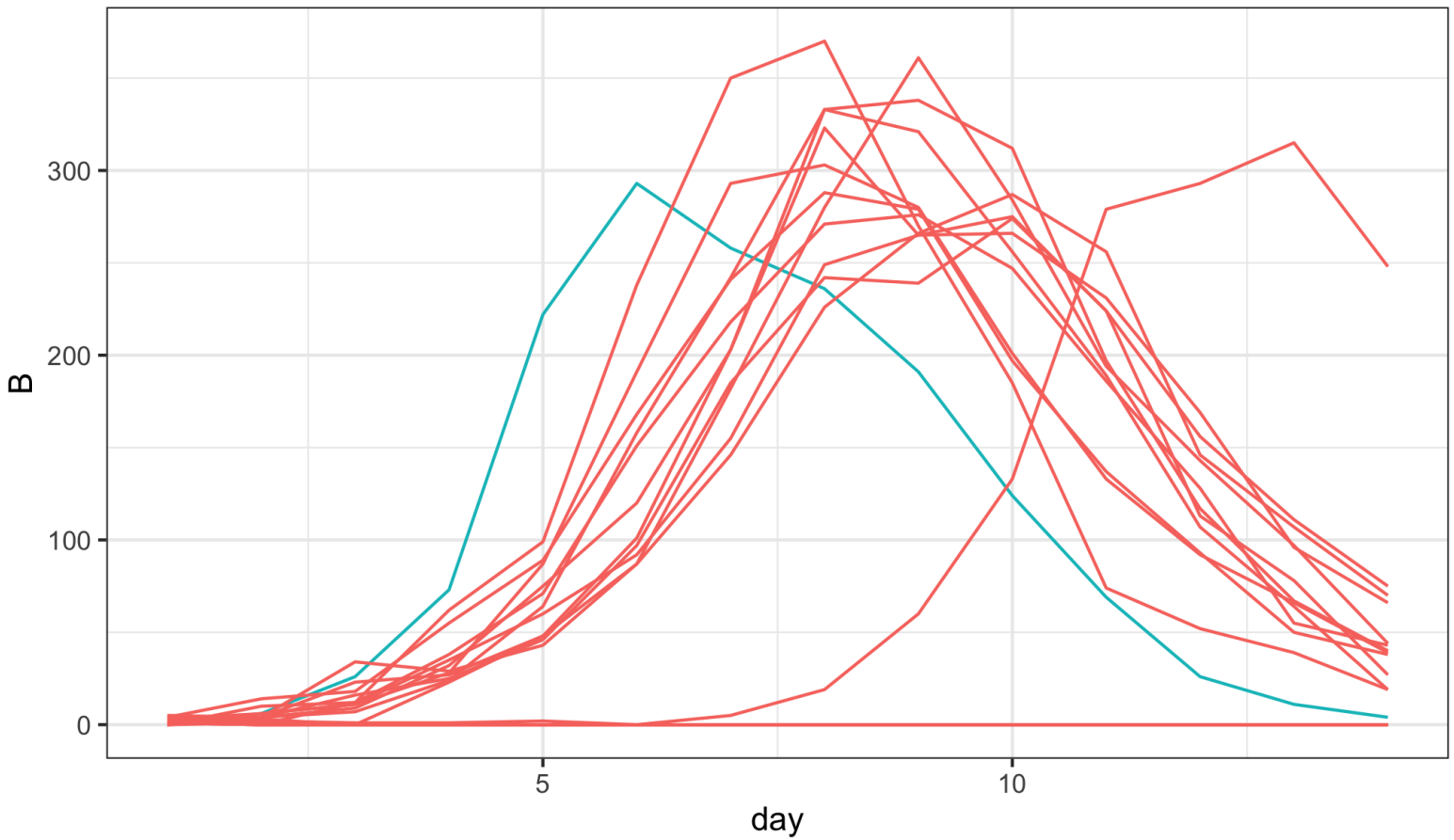
$$\hat{\mathcal{L}}_n(\theta) \approx \frac{1}{N}\sum_j f_{Y_n|X_n}(y_n^*|X_{n,j}^P\,;\theta).$$

- The full log likelihood then has approximation

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

Recall the SIR model from Chapter 11

```
sims <- simulate(sir,params=c(Beta=1.8,mu_IR=1,rho=0.9,N=2600),
  nsim=20,format="data.frame",include=TRUE)
ggplot(sims,mapping=aes(x=day,y=B,group=.id,color=.id=="data"))+
  geom_line()+guides(color=FALSE)
```



Sequential Monte Carlo in pomp

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,mu_IR=1,rho=0.8,N=2600))
logLik(pf)

## [1] -75.08432
```

Running a few particle filters gives an estimate of Monte Carlo variability:

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

##  [1] -73.60527 -77.57437 -76.89373 -81.53621 -75.17748
##  [6] -74.89806 -78.67495 -71.34814 -75.82188 -79.11634
```

Unbiasedness of the particle filter likelihood estimate

- 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.
- For other quantities, the particle filter has bias which decreases to zero as the number of particles increases. It is a special property of the likelihood that the bias in this case is zero.

Averaging log likelihood estimates

- The unbiased property of the particle filter for the **likelihood** suggests we average log likelihood estimates on the natural scale.
- After returning to the log scale, a standard error is available from the delta method.
- `logmeanexp()` does these computations.

```
logmeanexp(ll,se=TRUE)

##                    se
## -73.490733   1.707026
```

Graphing the likelihood function: The likelihood surface

- We can 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.
- Moving along a 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**. A slice varies one parameter at a time, fixing the others.

Parallel statistical computing in R

Parallelization is helpful for computational statistics. You can tell R to access multiple processors on your machine.

```
library(doParallel)
registerDoParallel()
library(doRNG)
registerDoRNG(3899882)
```

- `registerDoRNG` sets up a parallel random number generator.
- Most statistical computing is **embarrassingly parallel**.
- This means we simply have to learn to use a parallel for loop via `foreach()`

Slicing in the β and μ_{IR} directions

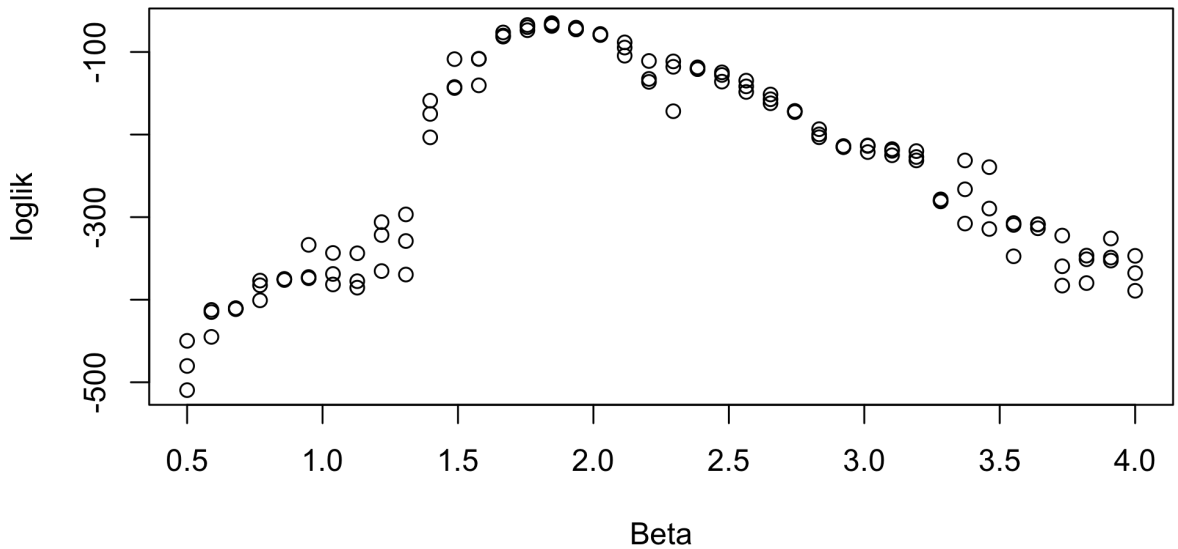
```
p <- sliceDesign(
  c(Beta=2,mu_IR=1,rho=0.8,N=2600),
  Beta=rep(seq(from=0.5,to=4,length=40),each=3),
  mu_IR=rep(seq(from=0.5,to=2,length=40),each=3))

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

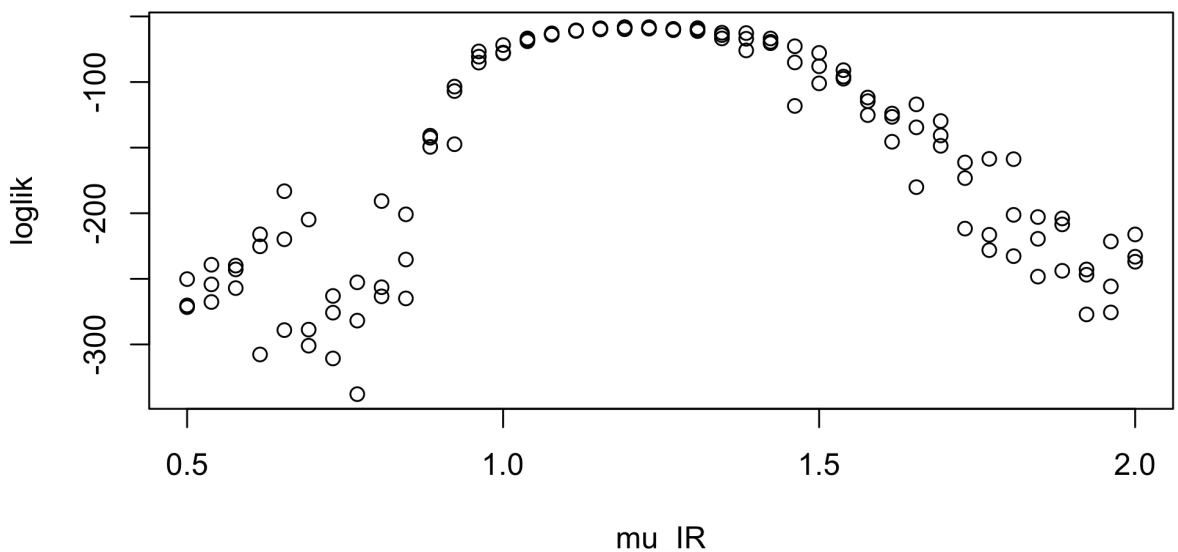
- `sliceDesign()` builds a dataframe where each row is a parameter vector.
- `foreach()` carries out particle filters for each row, distributed across processors

Question 12.2. Write down the definition of a likelihood slice in mathematical notation.

Question 12.3. Explain the difference between a likelihood slice and a likelihood profile,
(a) from a computational perspective.

(b) from the perspective of constructing confidence intervals and hypothesis tests.


loglik
-100
-300
-500
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
Beta

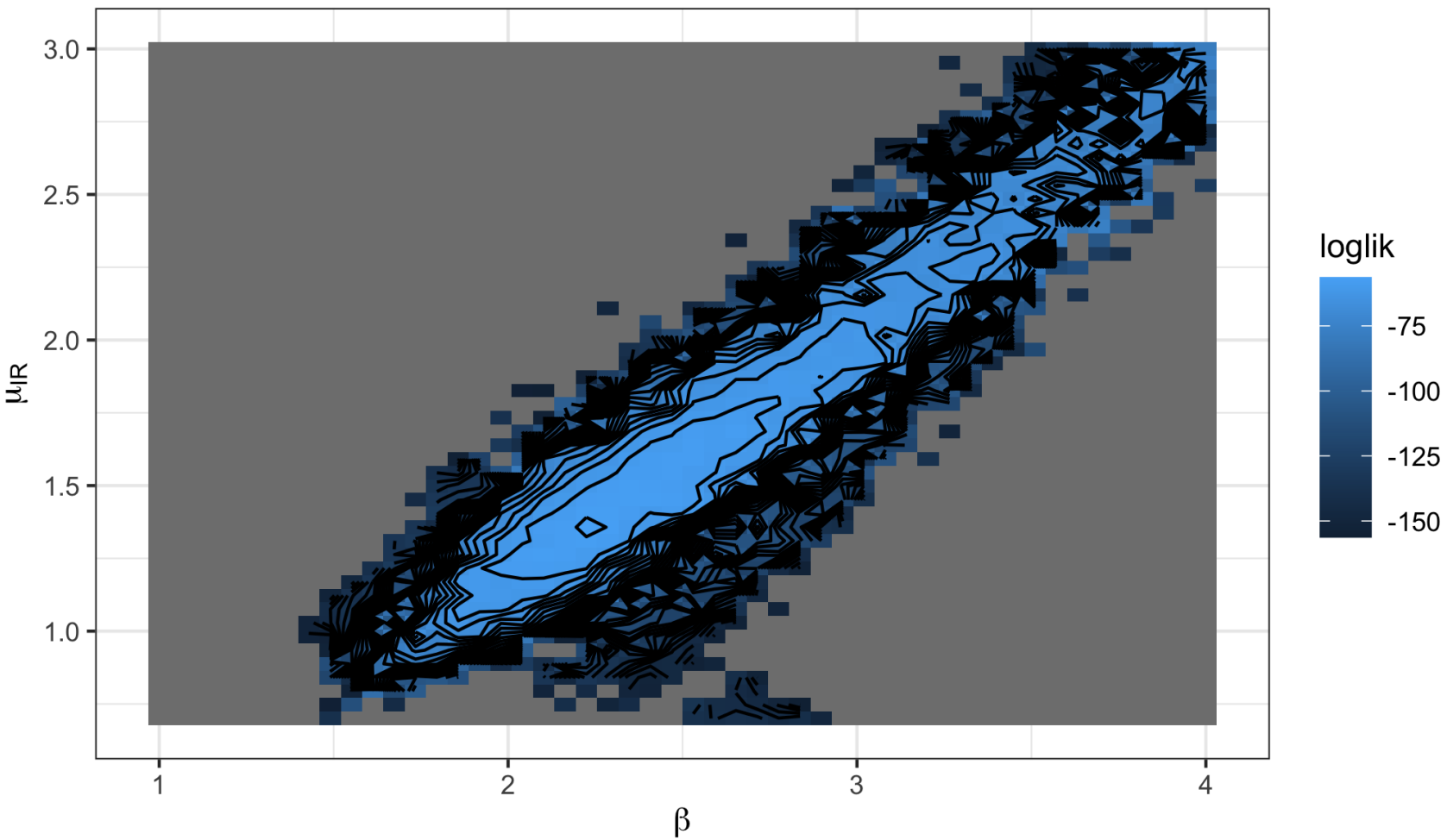

loglik
-100
-200
-300
0.5
1.0
1.5
2.0
mu_IR

A two-dimensionsional likelihood cross-section

- Slices offer a limited perspective on the geometry of the likelihood surface. With two parameters, we can evaluate the likelihood at a grid of points and visualize the surface.
- We compute a likelihood cross-section in the β and μ_{IR} directions.

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

foreach (theta=iter(p,"row"),.combine=rbind,
  .inorder=FALSE) %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=mu_IR,z=loglik,fill=loglik))+
  geom_tile(color=NA)+
  scale_fill_gradient()+
  geom_contour(color='black',binwidth=3)+
  labs(x=expression(beta),y=expression(mu[IR]))
```



Likelihood maximization

- We saw above that the default parameter set for the 'bsflu' pomp object is not particularly close to the MLE.
- One approach to find the MLE is to apply an optimizer to the particle filter estimate of the likelihood.
- There are many optimization algorithms to choose from, and many implemented in R.
- Three issues arise immediately (discussed more on following slides):

1. The particle filter gives us a stochastic estimate of the likelihood.
2. Lack of derivatives.
3. Constrained parameters.

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.
- We can use deterministic optimization, by fixing the seed of the pseudo-random number generator, a side effect is that the objective function can become jagged, with many small local maxima and minima.
- If we use stochastic optimization, the underlying surface may be smoother but we see it only with Monte Carlo noise.
- This is the trade-off between a noisy and a rough objective function.

2. Lack of derivatives

- 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 (i.e., a direct numerical estimate of the derivative) is not an especially promising way of constructing derivatives in the presence of Monte Carlo noise.

3. Constrained parameters

- For the boarding school flu example, the parameters are constrained to be positive, and $\rho < 1$.
- Such constraints are common, especially for rate parameters.
- We must 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, we transform the parameters onto a scale on which there are no constraints.

Cautions about parameter estimation for dynamic models

- When we propose a mechanistic model for a system, we have some idea of what we intend parameters to mean. In our epidemiology 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. Parameter estimates statistically consistent with the data may be absurd according to the scientific reasoning used to build the model.
- This can arise as a consequence of weak identifiability, or it can be a warning that the data show our model does not represent reality in the way we had hoped.
- Fixing some parameters at known, scienficially reasonable values is tempting. However, this can suppress warnings that the data were giving about weaknesses in the model, or in our biological interpretation of it.

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 input and output

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; starting 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, $\{\Theta^M_j, j = 1,\dots,J\}$

- This algorithm requires `rprocess` but not `dprocess`. It is **simulation-based**, also known as **plug-and-play**.

IF2 algorithm pseudocode

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

Comments on the IF2 algorithm

- 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 $\Theta_{n,j}^{F,m}$ and $X_{n,j}^{F,m}$ denote solutions to the filtering problem, with the particles $j = 1, \ldots, J$ 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 $\Theta_{n,j}^{P,m}$ and $X_{n,j}^{P,m}$ denote solutions to the prediction problem, with the particles $j = 1, \ldots, J$ providing a Monte Carlo representation of the conditional distribution at time n given data $y_{1:n-1}^*$ for filtering iteration m.
- The weight $w_{n,j}^m$ gives the likelihood of the data at time n for particle j in filtering iteration m.

Choosing the algorithmic settings for IF2

- The starting parameter swarm, $\{\Theta_j^0, j = 1, \ldots, J\}$, usually consists of J identical replications of some starting parameter vector.
- J should be sufficient for particle filtering. By 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:

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

- V_n is usually taken to be diagonal,

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

- If θ_i is a parameter that affects the dynamics or observations throughout the timeseries, it is called a **regular parameter** (RP) and we can set $v_{i,n} = v_i$.

Initial value parameters (IVPs)

- 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

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

Choosing the cooling schedule

- $\sigma_{1:M}$ 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**.
- The freezing point should be close to the lowest-energy state of the system, i.e., the MLE.
- We aim to transform parameters so that (on the estimation scale) they are unconstrained and have uncertainty on the order of 1 unit.
- Usually, we do 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

$$\begin{aligned} v_i &= 0.02 \quad \text{for regular parameters (RPs)} \\ v_j &= 0.1 \quad \text{for initial value parameters (IVPs)} \end{aligned}$$

- 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.

Applying IF2 to a boarding school influenza outbreak

- We redevelop a model for the boarding school flu data, as template for the cases studies to follow.
- We use an $SIR_1R_2R_3$ model 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 counts individuals who are bed-confined if they show symptoms; R_2 counts individuals who are convalescent if they showed symptoms; R_3 counts recovered individuals who have returned to schoolwork if they were symtomatic.
- We use abbreviations $\mu_{IR} = \mu_{IR_1}$, $\mu_{R_1} = \mu_{R_1R_2}$, $\mu_{R_2} = \mu_{R_2R_3}$.

```
bsflu_rprocess <- "
  double dN_SI = rbinom(S,1-exp(-Beta*I*dt));
  double dN_IR1 = rbinom(I,1-exp(-dt*mu_IR));
  double dN_R1R2 = rbinom(R1,1-exp(-dt*mu_R1));
  double dN_R2R3 = rbinom(R2,1-exp(-dt*mu_R2));
  S -= dN_SI;
  I += dN_SI - dN_IR1;
  R1 += dN_IR1 - dN_R1R2;
  R2 += dN_R1R2 - dN_R2R3;
"
```

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

The measurement model

- The observation on day n of the observed epidemic (with $n = 1$ being 22 January) has two measurements: the numbers of children who are bed-confined and convalescent.
- To start simply, we will just take $Y_n = B_n$ with $B_n \sim \text{Poisson}(\rho R_1(t_n))$.
- Here, ρ is a reporting rate corresponding to the chance of being symptomatic.
- Multivariate measurement models can be coded in `pomp`.

```
bsflu_dmeasure <- "
  lik = dpois(B,rho*R1+1e-10,give_log);
"

bsflu_rmeasure <- "
  B = rpois(rho*R1+1e-10);
"
```

- The `1e-10` **tolerance** value stops the code crashing when all particles have `R1=0`.

Initial conditions

- 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 tricky to reconcile with the rest of the data; we simply initialize with $I(t_0) = 1$ at $t_0 = 0$.

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

Limitations and weaknesses

- All models have 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.
- One could include a latency period for infections.
- 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. A model including the convalescence data might have to address this.

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

- The names are needed only for compiling the Csnippets, but writing them down also helps clarify the map from the mathematical representation of the model to the computational representation.

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

bsflu2 <- pomp(
  data=subset(bsflu_data,select=c(day,B)),
  times="day",
  t0=0,
  rprocess=euler(
    step.fun=Csnippet(bsflu_rprocess),
    delta.t=1/12),
  rmeasure=Csnippet(bsflu_rmeasure),
  dmeasure=Csnippet(bsflu_dmeasure),
  partrans=parameter_trans(
    log=c("Beta","mu_IR","mu_R1","mu_R2"),
    logit="rho"),
  statenames=bsflu_statenames,
  paramnames=bsflu_paramnames,
  rinit=Csnippet(bsflu_rinit)
)
```

Controlling the run time

- To develop and debug code, we want a version that runs extra quickly. Setting `run_level=1` gives a low number of particles, `Np`, etc.
- For this model and data, `Np=5000` and `Nmif=200` are empirically around the minimum to get stable results with an error in the likelihood of order 1 log unit for this example. This is done by `run_level=2`.
- For more precise time-consuming computations, `run_level=3`.

```
run_level <- 2
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 running iterated filtering, we check that the basic particle filter is working.
- We 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))
which_mle <- which.max(bsflu_params[,"logLik"])
bsflu_mle <- bsflu_params[which_mle,][bsflu_paramnames]
```

- We 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) )
```

- 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') %dopar% try(
                    pfilter(bsflu2,params=bsflu_mle,Np=bsflu_Np)
                  )
  )

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

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

##                    se
## -73.154557   1.274774
```

Caching computations in Rmarkdown

- In 15 seconds, we obtain an unbiased likelihood estimate of -73.15 with a Monte Carlo standard error of 1.27.
- 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.

Practical advice for caching

- 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

```
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) %dopar%  {
      mif2(bsflu2,
        params=bsflu_mle,
        Np=bsflu_Np,
        Nmif=bsflu_Nmif,
        cooling.fraction.50=bsflu_cooling.fraction.50,
        rw.sd=rw.sd(
          Beta=bsflu_rw.sd,
          mu_IR=bsflu_rw.sd,
          rho=bsflu_rw.sd)
      )
    }
  })
},seed=900242057,kind="L'Ecuyer")
```

- The final filtering iteration carried out by `mif2` generates an approximation to the likelihood at the resulting point estimate.
- This approximation is not usually good enough for reliable inference. Partly, because some parameter perturbations remain in the last filtering iteration. Partly, because `mif2` may be carried out with fewer particles than necessary for a good likelihood evaluation.
- Therefore, we evaluate the likelihood, together with a standard error, using replicated particle filters at each point estimate:

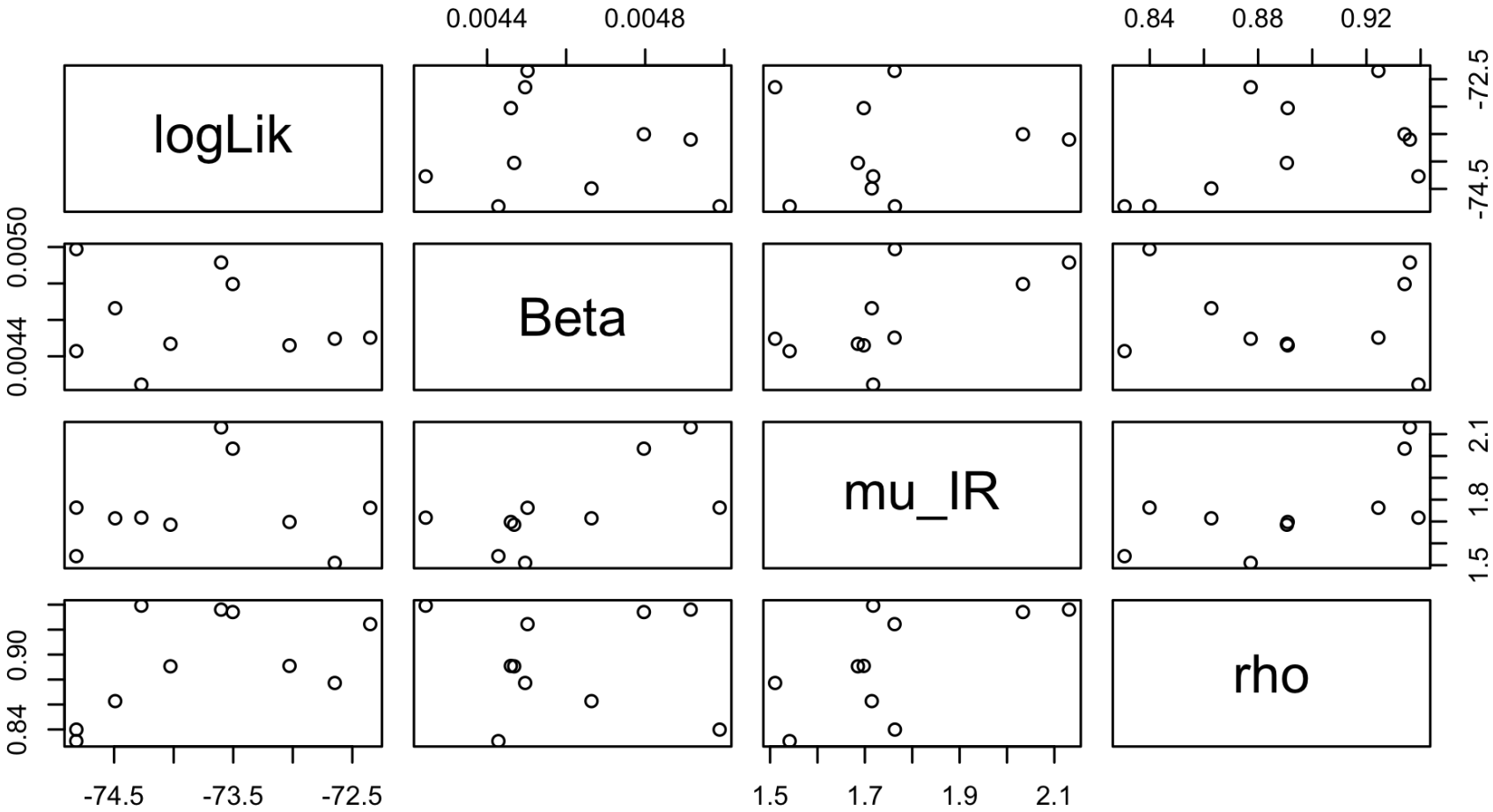
```
stew(file=sprintf("lik_local-%d.rda",run_level),{
  t_local_eval <- system.time({
  liks_local <- foreach(i=1:bsflu_Nlocal,.combine=rbind)%dopar% {
    evals <- replicate(bsflu_Neval, logLik(
      pfilter(bsflu2,params=coef(mifs_local[[i]]),Np=bsflu_Np)))
    logmeanexp(evals, se=TRUE)
    }
  })
},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.
##  -74.82  -74.44  -73.81  -73.75  -73.15  -72.35
```

- This investigation took 14.6 minutes for the maximization and 1.3 minutes for the likelihood evaluation.
- These repeated stochastic maximizations can show us the geometry of the likelihood surface in a neighborhood of this point estimate.
- A pairs plot is helpful to interpret these results.

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



Question 12.4. What do you conclude from this pairs plot?

A global likelihood search using random 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. 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, we have 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_IR=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,.combine=c) %dopar% {
      mif2(
        mifs_local[[1]],
        params=c(
          apply(bsflu_box,1,function(x)runif(1,x[1],x[2])),
          bsflu_fixed_params)
      )}
  })
},seed=1270401374,kind="L'Ecuyer")
```

Repeated likelihood evaluations 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,
      .combine=rbind) %dopar% {
        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.
##  -75.56  -75.43  -75.01  -75.01  -74.73  -74.36
```

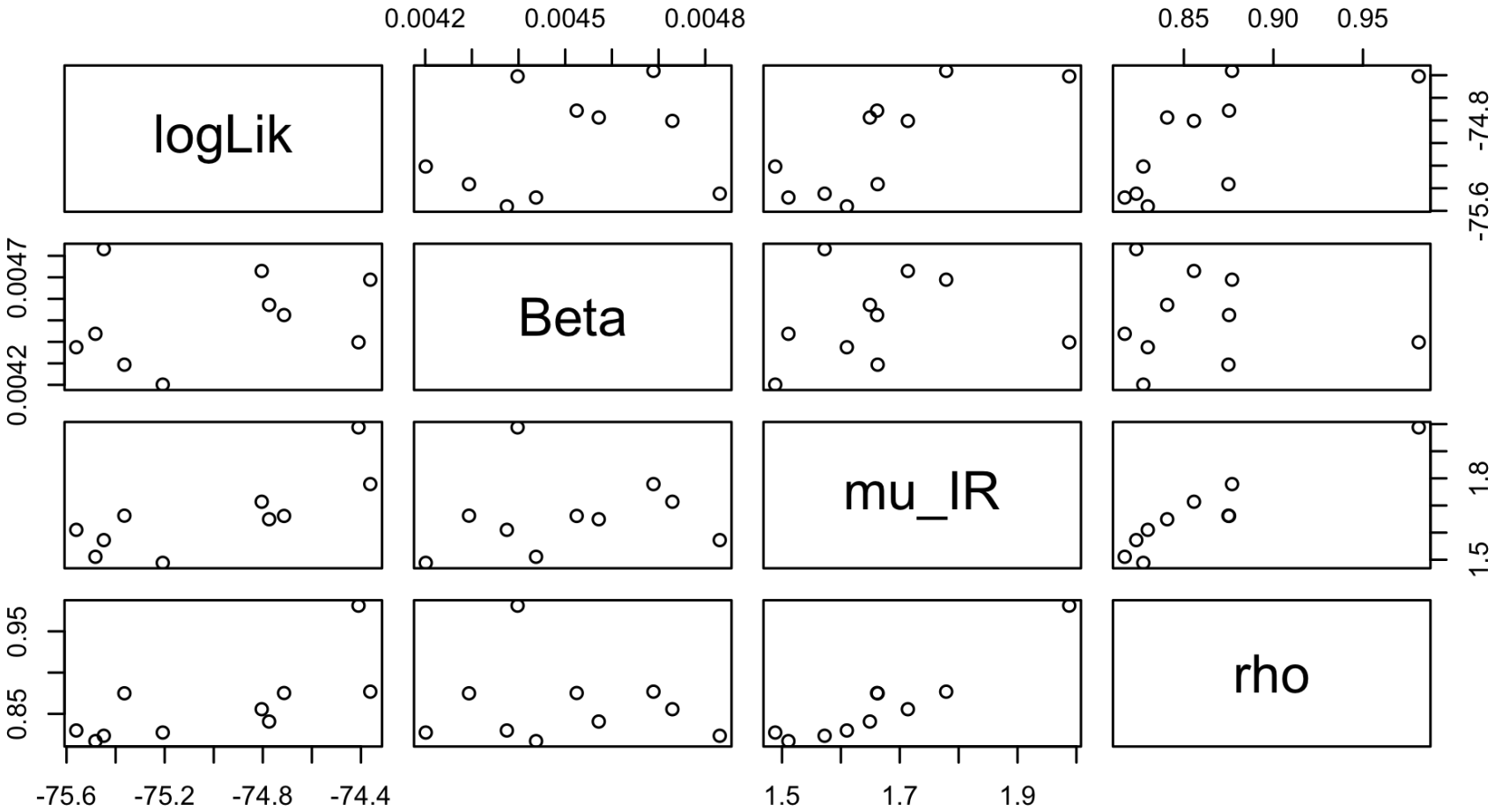
Building up evidence about the high likelihood region

- It is good practice to collect 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 `rround(max(results_global$logLik),1)` with a standard error of 0.3. This took in 14.7 minutes for the maximization and 1.4 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_IR+rho,
  data=subset(results_global,logLik>max(logLik)-250))
```

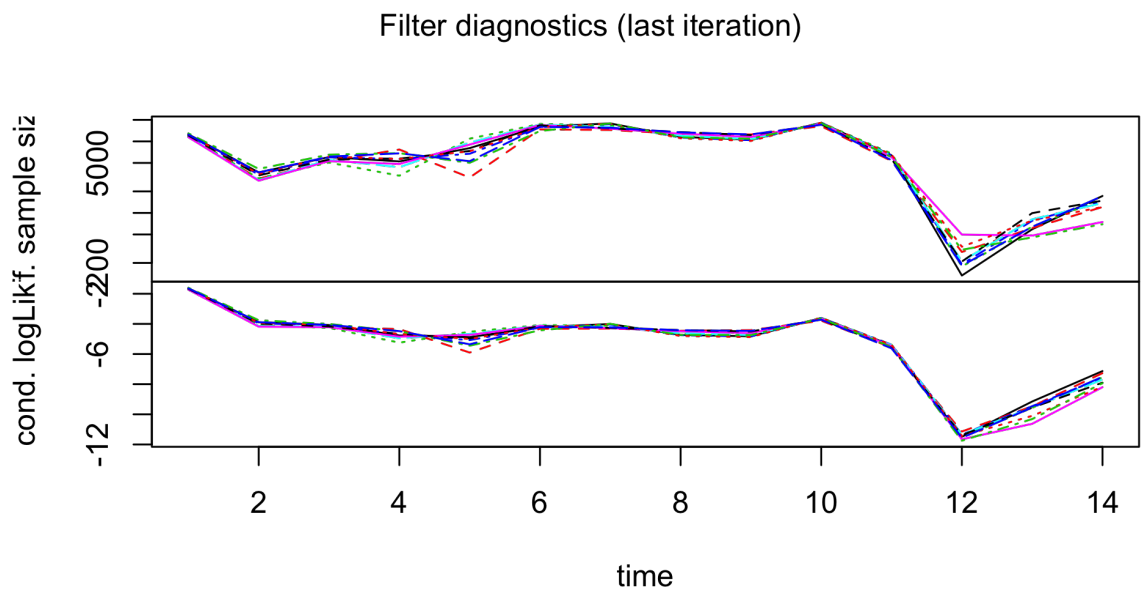


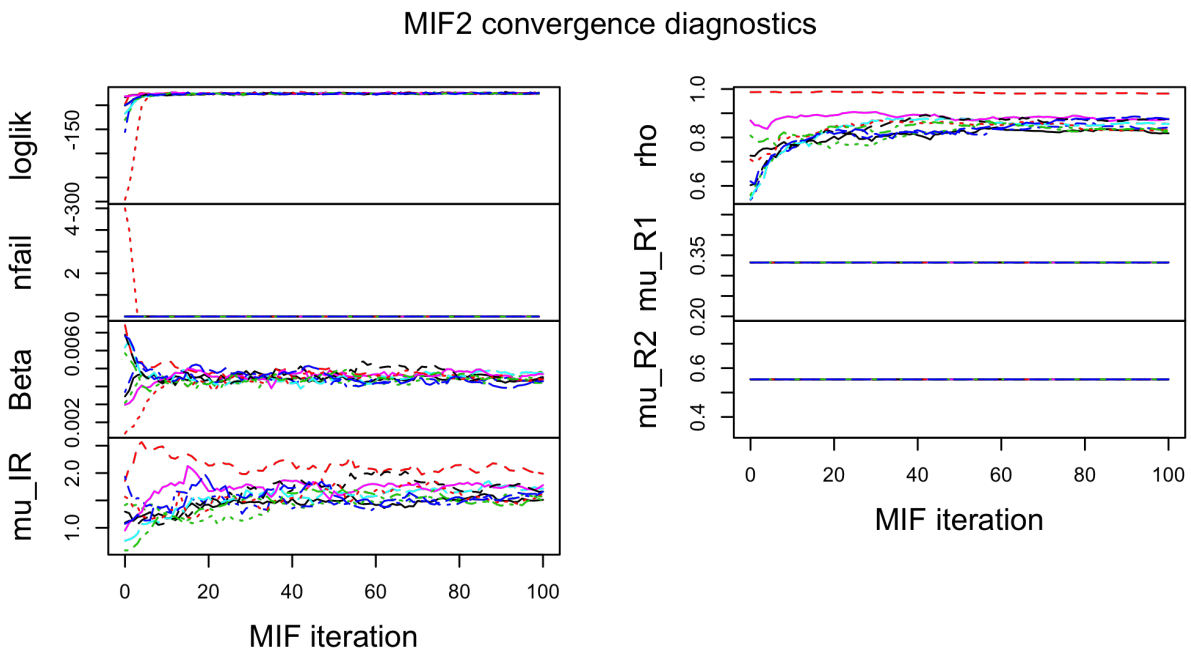
- We see that optimization attempts from diverse remote starting points end up with comparable likelihoods, even when the parameter values are quite distinct. This gives us some confidence in our maximization procedure.

Diagnostic plots for the maximization procedure

- The `plot` method for an object of class `mif2d.pomp` gives graphical convergence and filtering diagnostics for the maximization procedure.
- Concatenating objects of class `mif2d.pomp` gives a list of class `mif2List`.
- The `plot` method for a mif2List object gives us superimposed convergence diagnostic plots from different starting values, a useful tool.

```
class(mifs_global)

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

class(mifs_global[[1]])

## [1] "mif2d_pomp"
## attr(,"package")
## [1] "pomp"
```


Filter diagnostics (last iteration)
cond. logLik
f. sample siz
5000
-200
-6
-12
2
4
6
8
10
12
14
time


MIF2 convergence diagnostics
loglik
-150
nfail
-300
4
2
Beta
0.006
0
0.002
mu_IR
2.0
1.0
0
20
40
60
80
100
MIF iteration
rho
1.0
0.8
0.6
mu_R2
mu_R1
0.35
0.20
0.6
0.4
0
20
40
60
80
100
MIF iteration

Interpreting the diagnostics

1. What would the convergence plots look like if we cooled too quickly? Or too slowly? Can you find evidence for either of these above? (The algorithmic parameter `cooling.fraction.50` is the fraction by which we decrease the random walk standard deviation in 50 filtering iterations.)
2. Here, we did 100 `mif` iterations. Should we have done more? Could we have saved ourselves computational effort by doing less, without compromising our analysis?
3. 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?

Effective sample size

- 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 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 pseudo code.
- 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.
- For our example, do you have any concerns about the number of particles?

Question 12.5. **Constructing 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.

- How would you profile over the basic reproduction number, $R_0 = \beta P/\mu_{IR}$. Is this more or less well determined that β for this model and data?

Checking model source code

It is surprisingly hard to ensure that written equations and 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?

Question 12.6. **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.

Question 12.7. **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.

Acknowledgments and License

- Produced with R version 3.6.2 and `pomp` version 2.7.
- These notes build on previous versions at `ionides.github.io/531w16` and `ionides.github.io/531w18`.
- Those notes draw on material developed for a short course on Simulation-based Inference for Epidemiological Dynamics (`http://kingaa.github.io/sbied/`) by Aaron King and Edward Ionides, taught at the University of Washington Summer Institute in Statistics and Modeling in Infectious Diseases, from 2015 through 2019.

References I

Ionides, E. L., Nguyen, D., Atchadé, Y., Stoev, S. and King, A. A. (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.