Lesson 9:
Modeling stochasticity: Overdispersion in the Consett measles data.
IN DEVELOPMENT

Aaron A. King, Edward L. Ionides, Kidus Asfaw

July 12, 2020

Contents

1 Introduction

Objectives

- We investigates the consequences of modeling overdispersion in the on the Consett measles outbreak dataset.
- More broadly, we consider the question of whether a model has sufficient stochasticity to explain the data.
- We extend the model used to demonstrate iterated filtering in Lesson 4.

Introduction

- This investigation provides a relatively simple example to better understand overdispersed Markov chain models that are essential for large populations such as the [city-level measles case study](../measles/measles.html).
- We investigate whether fitting models with dynamic overdispersion (i.e., environmental stochasticity, also known as extra-demographic stochasticity) is helpful even for this relatively small population, for which demographic stochasticity is relatively large.
- We consider the respective roles of overdispersion in the measurement model and the dynamic model.
- We think about scientific interpretations of overdispersion if it is statistically evident.
- We demonstrate a way to implement overdispersion for POMP models using **pomp**.

background on mean-variance relationships and overdispersion

- A good statistical model should describe both the mean (center) and variance (spread) of the data.
- In the language of forecasting, we want point predictions and uncertainty estimates.
- Appropriate modeling of uncertainty in the data is closely linked to appropriate assessment of uncertainty in parameter estimates (confidence intervals and hypothesis tests).
- Some basic models (especially for count data) constrain the variance as a function of the mean.
- A famous example arises in Poisson regression. The Poisson distribution has variance equal to its mean. What if the data have higher variance than its mean? This **overdispersion** is common. Checking and correcting for it are standard practice in generalized linear model regression analysis.
- Overdispersion is also common in dynamic models. It can be natural to write down models building on Poisson or binomial increments, which therefore include specific mean-variance assumptions.
- These dynamic models may contain nonlinearities that complicate the mean-variance relationship, but the underlying issue remains.

Adding overdispersion to the basic SIR model

- Recall the case report data on the 1948 measles outbreak in Consett, UK.

```r
library(tidyverse)

courseurl <- "https://kingaa.github.io/sbied/"
datafile <- "mif/Measles_Consett_1948.csv"

read_csv(paste0(courseurl,datafile)) %>%
  select(week,reports=cases) %>%
  filter(week<=42) -> consett_data
```

Specification of a basic SIR model

- Our model is a variation on a basic SIR Markov chain
- State: $X(t) = (S(t), I(t), R(t))$; numbers of hosts in susceptible, infectious, and recovered classes.
- Assume: a single infection in week 0, i.e., that $I(0) = 1$.
- Each individual in S transitions to I at rate $\mu_{SI} = \beta\, I(t)/N$.
- Each individual in I transitions at rate μ_{IR} to R.
- $1/\mu_{IR}$ is the mean infectious period.

Adding overdispersion to the latent stochastic process

- We are going to extend the basic model to include stochastic variation by incorporating multiplicative noise,
$$\mu_{SI} = \beta \frac{I(t)}{N} d\Gamma/dt.$$
- Here, $\Gamma(t)$ is a gamma process with $\mathbb{E}\left[\Gamma(t)\right] = t$ and $\mathrm{Var}\left[\Gamma(t)\right] = \sigma^2 t$.

- Thus, σ^2 is the **infinitesimal variance parameter** of the noise process, which we will call the **extrademographic process noise** parameter.
- We do not include overdispersion in the $I \to R$ transition, on the assumption this is a purely demographic process. This assumption could be checked.

Exercise 9.1. Why do we use multiplicative noise?
What difficulties arise if we use additive noise such as

$$\mu'_{SI} = \beta \frac{I(t)}{N} + dW/dt$$

Where dW/dt is some white noise process?

Gamma noise

- The gamma process has the property of being non-negative, which is the main reason it may be preferable to Gaussian noise.
- The gamma process is a pure jump process, constant between jumps. There are an infinite number of jumps in any finite time interval, but almost all of them are negligibly small. Thus, the derivative of the gamma process doesn't exist in the usual sense, but one can still give formal meaning to $d\Gamma/dt$. All this is similar to Gaussian noise, which can also be considered as the formal derivative of a non-differentiable process (Brownian motion).

Adding overdispersion to the measurement model

- Overdispersion in the measurement model may be appropriate for similar reasons. Measurement overdispersion might be considered together with, or in place of, dynamic overdispersion.
- Adding gamma noise to a Poisson measurement model leads to a negative binomial measurement model. The overdispersion parameter is ψ, with the variance for mean μ being $\mu + \mu^2/\psi$ and the Poisson model being recovered in the limit as $\psi \to \infty$.

Latent variables, observed variables and parameters

- We code the state variables (S, I and the accumulator variable H) as follows, noting that we do not need a representation of $R = N - S - I$.

```
consett_statenames <- c("S","I","H")
```

- Similarly, the parameters (β, η, μ_{IR}, ρ, N, σ, ψ) are:

```
consett_paramnames <- c("Beta","mu_IR","eta","rho","N","sigma","psi")
```

- The data variable name is taken from `consett_data`:

```
colnames(consett_data)

[1] "week"    "reports"
```

The model code is available for inspection:

```
consett_dmeas <- Csnippet("
  lik = dnbinom_mu(reports, psi, rho*H, give_log);
")

consett_rmeas <- Csnippet("
  reports = rnbinom_mu(psi, rho*H);
")
```

```
consett_step <- Csnippet("
  double dGamma = rgammawn(sigma,dt);
  double dN_SI = rbinom(S,1-exp(-Beta*I/N*dGamma));
  double dN_IR = rbinom(I,1-exp(-mu_IR*dt));
  S -= dN_SI;
  I += dN_SI - dN_IR;
  H += dN_IR;
  ")
```

```
  consett_rinit <- Csnippet("
  S = nearbyint(eta*N);
  I = 1;
  H = 0;
  ")
```

```
consett_data %>%
  pomp(
    times="week",t0=0,
    rprocess=euler(consett_step,delta.t=1/7),
    rinit=consett_rinit,
    rmeasure=consett_rmeas,
    dmeasure=consett_dmeas,
    accumvars="H",
    statenames=consett_statenames,
    paramnames=consett_paramnames
  ) -> consett
```

Testing the codes

To develop and debug code, it is useful to have testing codes that run quickly and fail if the codes are not working correctly.

As such a test, here we run some simulations and a particle filter.

We'll use the following parameters, derived from our earlier explorations, but adding in some small amount of overdispersion.

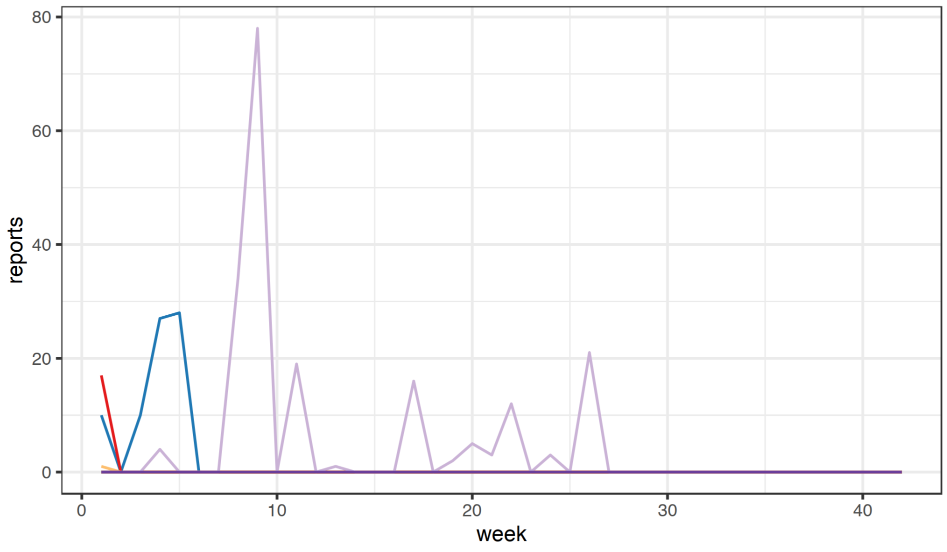
```
params <- c(Beta=20,mu_IR=2,rho=0.5,eta=0.1,N=38000,
  sigma=0.1,psi=0.1)
```

Testing the codes: simulation

Now to run and plot some simulations:

```
consett %>%
  simulate(params=params,nsim=10,format="data.frame") -> y
```

```
y %>%
  ggplot(aes(x=week,y=reports,group=.id,color=factor(.id)))+
  geom_line()+
  scale_color_brewer(type="qual",palette=3)+
  guides(color=FALSE)
```

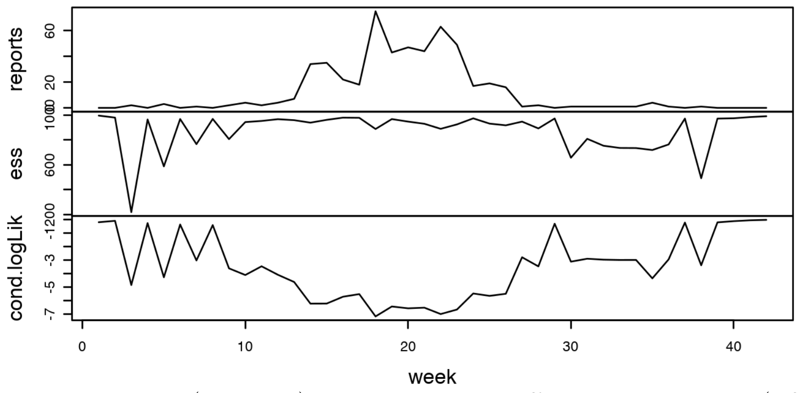


Testing the codes: filtering

Before engaging in iterated filtering, it is a good idea to check that the basic particle filter is working since we can't iterate something unless we can run it once! The simulations above check the `rprocess` and `rmeasure` codes; the particle filter depends on the `rprocess` and `dmeasure` codes and so is a check of the latter.

```
consett %>%
  pfilter(Np=if(DEBUG) 100 else 1000,params=params) -> pf
```

```
plot(pf)
```



The above plot shows the data (`reports`), along with the *effective sample size* (ESS) of the particle filter (`ess`) and the log likelihood of each observation conditional on the preceding ones (`cond.logLik`). The ESS is the equivalent number of independent particles. In this case, the ESS is very small at several points.

Setting up the estimation problem

We follow the assumptions from the iterated filtering lesson, while adding overdispersion. We treat the population size, N, as known from the census. We also fix the infectious period in our model to 3.5 da, i.e., $\mu_{IR} = 2\ \mathrm{wk}^{-1}$.

```
fixed_params <- c(N=38000, mu_IR=2)
```

We proceed to estimate β, η, ρ, σ and ψ.

Setup for parallel computing

```
library(doParallel)
registerDoParallel(detectCores())
library(doRNG)
registerDoRNG(582998)
```

Running a particle filter

We proceed to carry out replicated particle filters at an initial guess of $\beta = 20$, $\eta = 0.1$, and $\rho = 0.5$.

```
foreach(i=1:10,.combine=c) %dopar% {
  library(pomp)
  consett %>% pfilter(params=params,Np=10000)
} -> pf

pf %>% logLik() %>% logmeanexp(se=TRUE) -> L_pf
L_pf
```

```
                              se
-147.58396382    0.01533597
```

In 3.28 seconds, using 8 cores, we obtain an unbiased likelihood estimate of -147.6 with a Monte Carlo standard error of 0.015.

A local search of the likelihood surface

A local search of the likelihood surface

Let's carry out a local search using `mif2` around this point in parameter space.

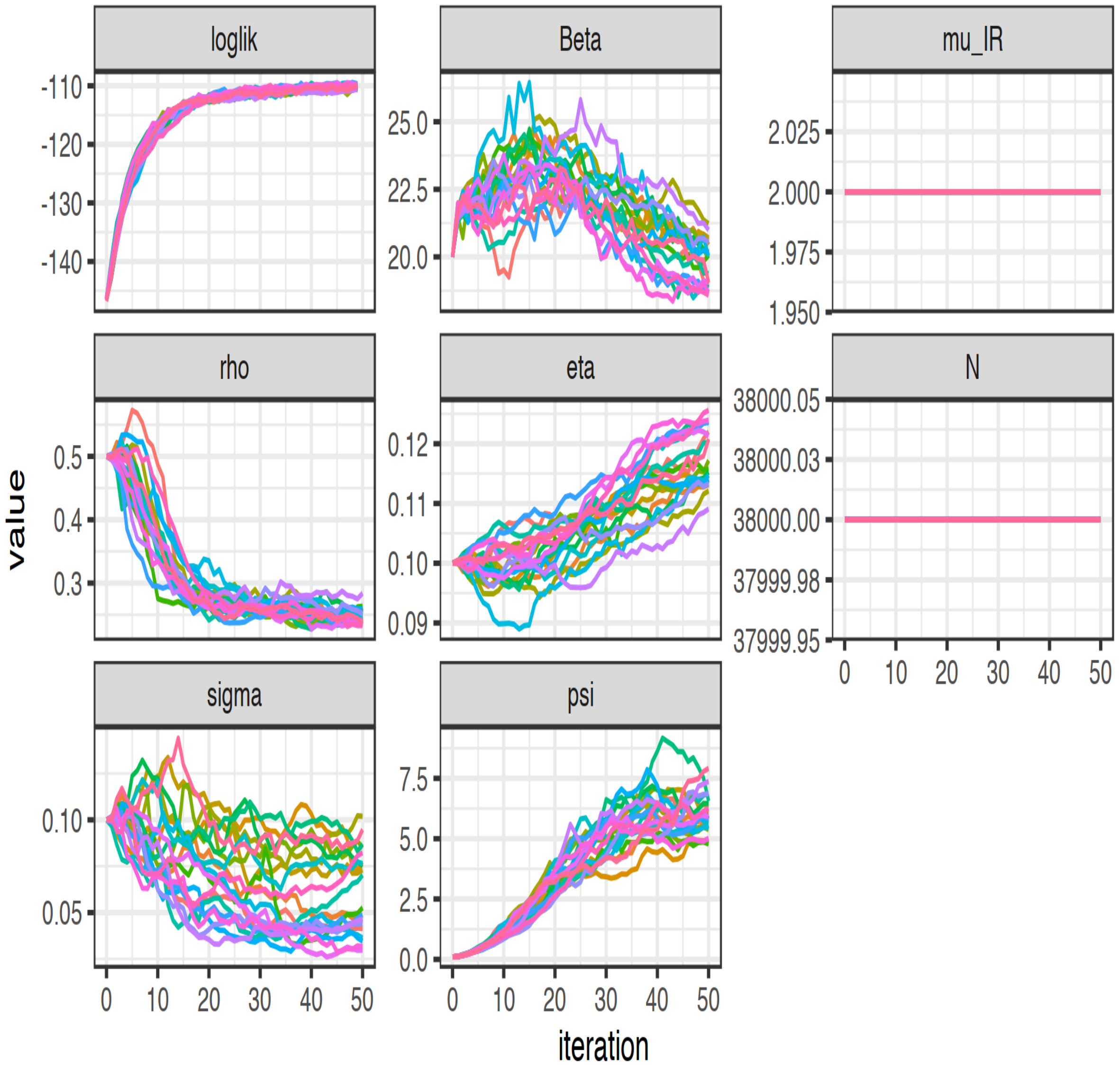
- We need to choose the `rw.sd` and `cooling.fraction.50` algorithmic parameters.
- Since β, σ and ψ will be estimated on the log scale, and we expect that multiplicative perturbations of these parameters will have roughly similar effects on the likelihood, we'll use a perturbation size of 0.02, which we imagine will have a small but non-negligible effect.
- For simplicity, we'll use the same perturbation size on ρ and η.
- We fix `cooling.fraction.50=0.5`, so that after 50 `mif2` iterations, the perturbations are reduced to half their original magnitudes.

```
foreach(i=1:20,.combine=c) %dopar% {
  library(pomp)
  library(tidyverse)
  consett %>%
    mif2(
      params=params,
      Np=if(DEBUG) 50 else 2000, Nmif=if(DEBUG) 2 else 50,
      partrans=parameter_trans(
        log=c("Beta","sigma","psi"),
        logit=c("rho","eta")
      ),
      paramnames=consett_paramnames,
      cooling.fraction.50=0.5,
      rw.sd=rw.sd(Beta=0.02, rho=0.02, sigma=0.02, psi=0.02, eta=ivp(0.02))
    )
} -> mifs_local
```

Iterated filtering diagnostics

We obtain some diagnostic plots with the `plot` command applied to `mifs_local`. Here is a way to get a prettier version:

```
mifs_local %>%
  traces() %>%
  melt() %>%
  ggplot(aes(x=iteration,y=value,group=L1,color=factor(L1)))+
  geom_line()+
  guides(color=FALSE)+
  facet_wrap(~variable,scales="free_y")
```



- We see that the likelihood eventually increases as the iterations proceed, though there is considerable variability due to

  (a) the poorness of our starting guess and

  (b) the stochastic nature of this Monte Carlo algorithm.

- We see movement in the parameters, though considerable variability remains.

Estimating the likelihood
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 good enough for reliable inference.

- Partly, this is because parameter perturbations are applied in the last filtering iteration, so that the likelihood reported by `mif2` is not identical to that of the model of interest.

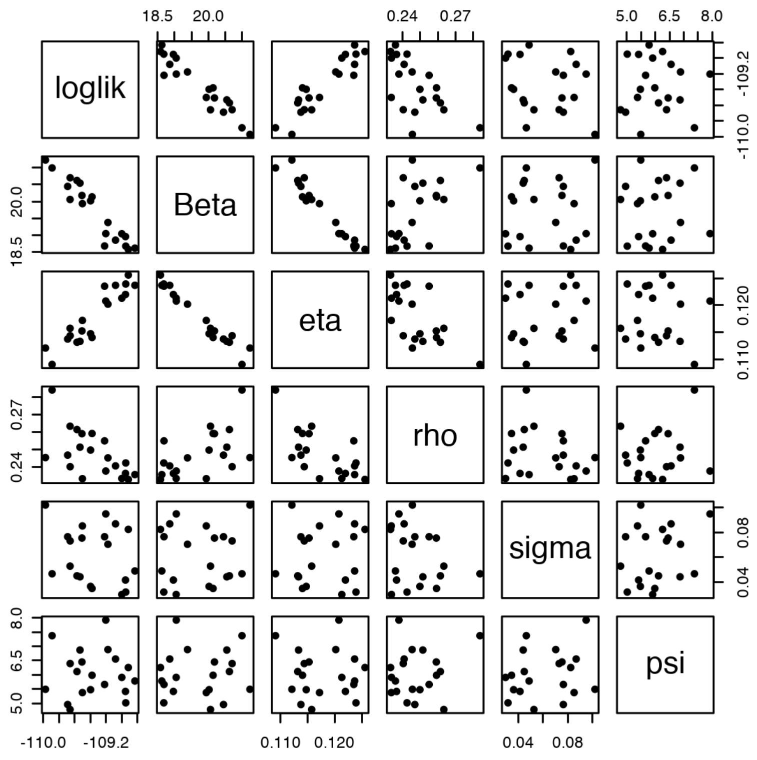
- Partly, this is because `mif2` is usually carried out with fewer particles than are needed for a good likelihood evaluation.

Therefore, we evaluate the likelihood, together with a standard error, using replicated particle filters at each point estimate.

```r
foreach(mf=mifs_local,.combine=rbind) %dopar% {
  library(pomp)
  library(tidyverse)
  evals <- replicate(if(DEBUG) 2 else 10,
    logLik(pfilter(mf,Np=if(DEBUG) 50 else 20000)))
  ll <- logmeanexp(evals,se=TRUE)
  mf %>% coef() %>% bind_rows() %>%
    bind_cols(loglik=ll[1],loglik.se=ll[2])
} -> results
```

On 8 processors, this local investigation took 50 sec for the maximization and 71 sec 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:

```r
pairs(~loglik+Beta+eta+rho+sigma+psi,data=results,pch=16)
```



Building up a picture of the likelihood surface
This plot shows a hint of a ridge in the likelihood surface (cf. the β-η panel). However, the sampling is as yet too sparse to give a clear picture.
We add these newly explored points to our database,

```
read_csv("measles_params.csv") %>%
  bind_rows(results) %>%
  arrange(-loglik) %>%
  write_csv("measles_params.csv")
```

and move on to a more thorough exploration of the likelihood surface.

2 Searching for the MLE

A global search

A global search of the likelihood surface

- 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).
- To avoid confusion, we use the term "initial values" to refer to the state of the system at t_0 and "starting values" to refer to the point in parameter space at which a search is initialized.
- Practical parameter estimation involves trying many starting values for the parameters.
- One way to approach this is to choose a large box in parameter space that contains all remotely sensible parameter vectors.
- 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 measles model, a box containing reasonable parameter values might be $\beta \in (5, 80)$, $\rho \in (0.2, 0.9)$, $\eta \in (0, 0.4)$.
We are now ready to carry out likelihood maximizations from diverse starting points.

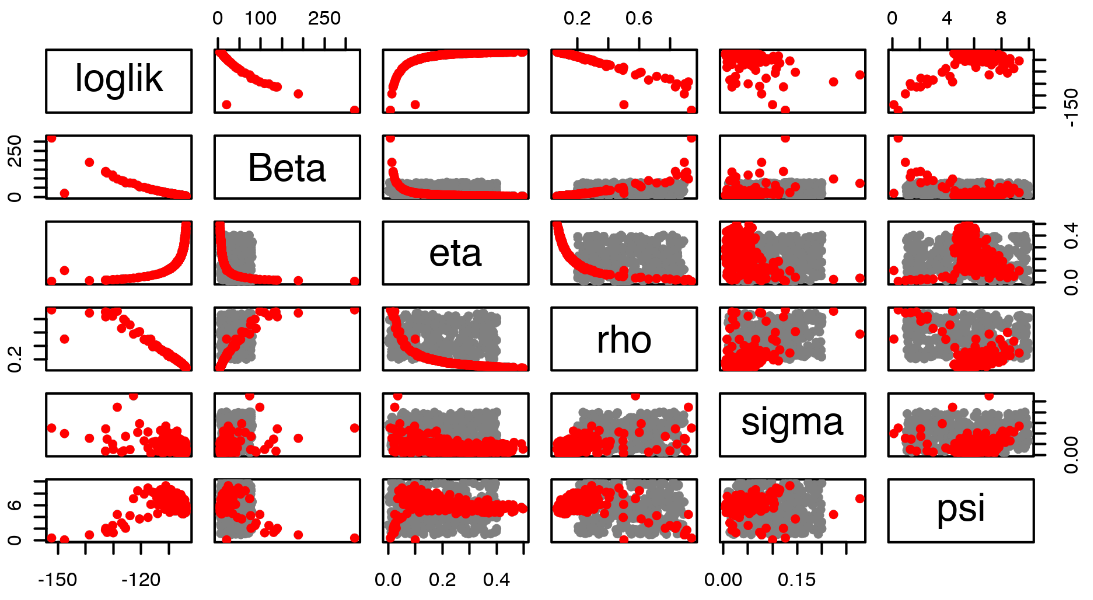
```
set.seed(2062379496)

runifDesign(
  lower=c(Beta=5,rho=0.2,eta=0,sigma=0.01,psi=1),
  upper=c(Beta=80,rho=0.9,eta=0.4,sigma=0.2,psi=10),
  nseq=300
) -> guesses

mf1 <- mifs_local[[1]]
```

```
registerDoRNG(1270401374)
foreach(guess=iter(guesses,"row"), .combine=rbind) %dopar% {
  library(pomp)
  library(tidyverse)
  mf1 %>%
    mif2(params=c(unlist(guess),fixed_params)) %>%
    mif2(Nmif=if(DEBUG) 2 else 100) -> mf
  replicate(
    if(DEBUG) 2 else 10,
    mf %>% pfilter(Np= if(DEBUG) 100 else 100000) %>% logLik()
  ) %>%
    logmeanexp(se=TRUE) -> ll
  mf %>% coef() %>% bind_rows() %>%
    bind_cols(loglik=ll[1],loglik.se=ll[2])
} -> results
```

- The above codes run one search from each of 300 starting values.
- Each search consists of an initial run of 50 IF2 iterations, followed by another 100 iterations.
- These codes exhibit a general **pomp** behavior:
  - Re-running a command on an object (i.e., `mif2` on `mf1`) created by the same command preserves the algorithmic arguments.
  - In particular, running `mif2` on the result of a `mif2` computation re-runs IF2 from the endpoint of the first run.
  - In the second computation, by default, all algorithmic parameters are preserved; here we overrode the default choice of `Nmif`.
- Following the `mif2` computations, the particle filter is used to evaluate the likelihood, as before.
- In contrast to the local-search codes above, here we return only the endpoint of the search, together with the likelihood estimate and its standard error in a named vector.
- The best result of this search had a likelihood of -103.7 with a standard error of 0.01.
- This took 107.1 minutes altogether using 8 processors.

Again, we attempt to visualize the global geometry of the likelihood surface using a scatterplot matrix. In particular, here we plot both the starting values (grey) and the IF2 estimates (red).

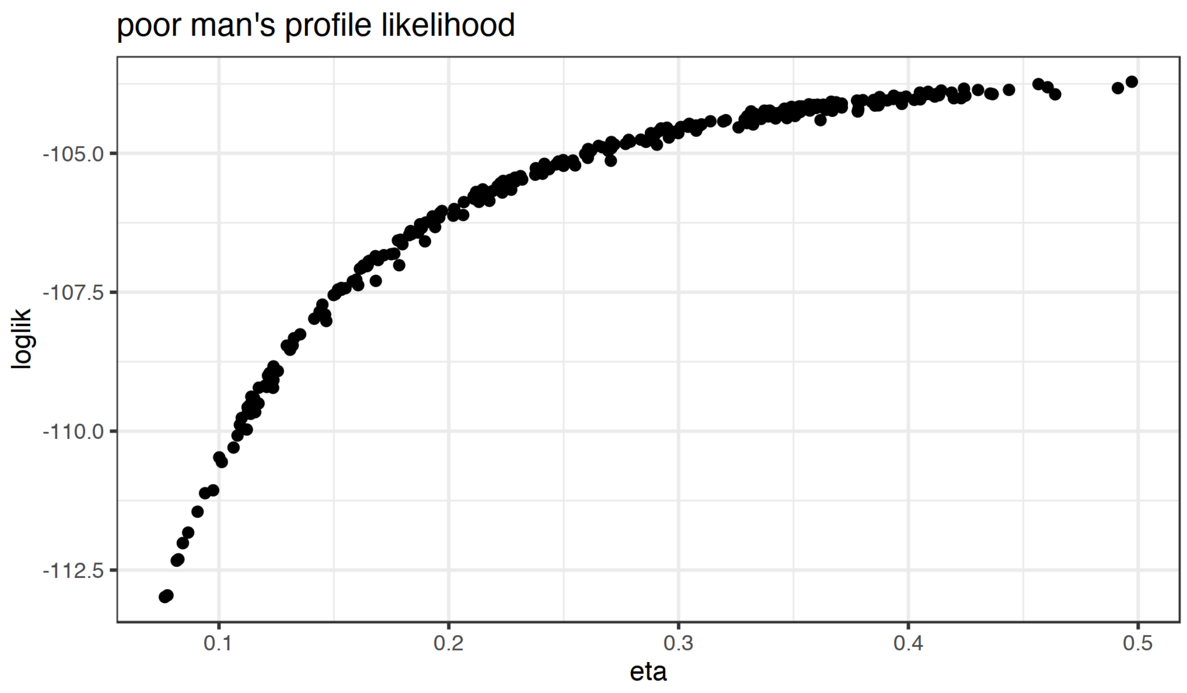
```
read_csv("measles_params.csv") %>%
  filter(loglik>max(loglik)-50) %>%
  bind_rows(guesses) %>%
  mutate(type=if_else(is.na(loglik),"guess","result")) %>%
  arrange(type) -> all

pairs(~loglik+Beta+eta+rho+sigma+psi, data=all,
      col=ifelse(all$type=="guess",grey(0.5),"red"),pch=16)
```



- We see that optimization attempts from diverse remote starting points converge on a particular region in parameter space.
- The estimates have comparable likelihoods, despite their considerable variability.
- This gives us some confidence in our maximization procedure.

The projections of the estimates give us "poor man's profiles":

```r
all %>%
  filter(type=="result") %>%
  filter(loglik>max(loglik)-10) %>%
  ggplot(aes(x=eta,y=loglik))+
  geom_point()+
  labs(
    x=expression("eta"),
    title="poor man's profile likelihood"
  )
```



Exercise 9.2. A profile over σ

- Is there evidence for the inclusion of overdispersion on the latent process?
- The profile for ρ shows very little pattern for σ, which could be because the amount of information about σ is small in the context of this model (i.e., the profile has low curvature – it is close to flat).
- Construct a profile likelihood for σ to investigate whether this is the case.
- You'll have to work out a suitable range of σ for the profile. The above figure suggests that the interval used for σ in the profile for ρ may be too narrow.

References

License, acknowledgments, and links

- This lesson is prepared for the Simulation-based Inference for Epidemiological Dynamics module at the 2020 Summer Institute in Statistics and Modeling in Infectious Diseases, SISMID 2020.

- The materials build on previous versions of this course and related courses.

- Produced with R version 4.0.2 and **pomp** version 3.1.0.2.

- Compiled on July 12, 2020.

Back to course homepage
R codes for this lesson