Basic SI Model for pompitis

Katie Montovan, Aaron King, Eli Goldwyn & Julie Blackwood

2025-7-23

Introduction

This document was created to support the book chapter *Computational methods for efficient fitting of ODE models to time series data using pomp*. It contains an example of a disease model setup to fit the model to data and prepare for possible model comparison. There are four models shared in the github repository that show different aspects.

This first one walks through the method step by step. We recommend working through this file and the chapter together to understand each step of the process. Then model 2 is setup more in the way you might structure your own model later. Work through it - the steps are all the same but they are organized slightly differently and some optional steps are skipped. After that, we recommend making a copy of the model 2 file and modifying the model to make your own model of the disease. The model 2 file is more streamlined for modifying to implement your own model once you understand what all of these pieces do. This method hopefully will help you learn the method and learn how to troubleshoot problems should you encounter them.

As explained in the chapter, we are going to model the spread of pompitis through a population. The disease and data are fictional but the method applies equally well to real problems. This allows us to more clearly explain the model fitting process which is complex even in the most basic of applications.

Disclaimer This code was written with pomp version 5.11. If you are running into errors with this code a good place to start might be by looking at the updates made to pomp and how the code needs to be modified accordingly. We will try to keep this up to date and you are welcome to email kmontovan@bennington.edu if you think this code needs updating to the current version of these packages.

Some of these chunks of code take a very long time to run. This can make knitting this file take a very long time. You may just want to play with the individual chunks and not knit the file each time. If you would like to be able to see all the pieces together, there is a pdf available on github that might be useful to hand on hand.

Load in needed libraries and get setup for parallel processing

```
options(
  tidyverse.quiet=TRUE,
  pomp_archive_dir="results",
  stringsAsFactors=FALSE
)
library(tidyverse)
library(pomp)
stopifnot(packageVersion("pomp")>="6.0.4")
library(doParallel)
library(doRNG)
library(ggplot2)
library(dplyr)
```

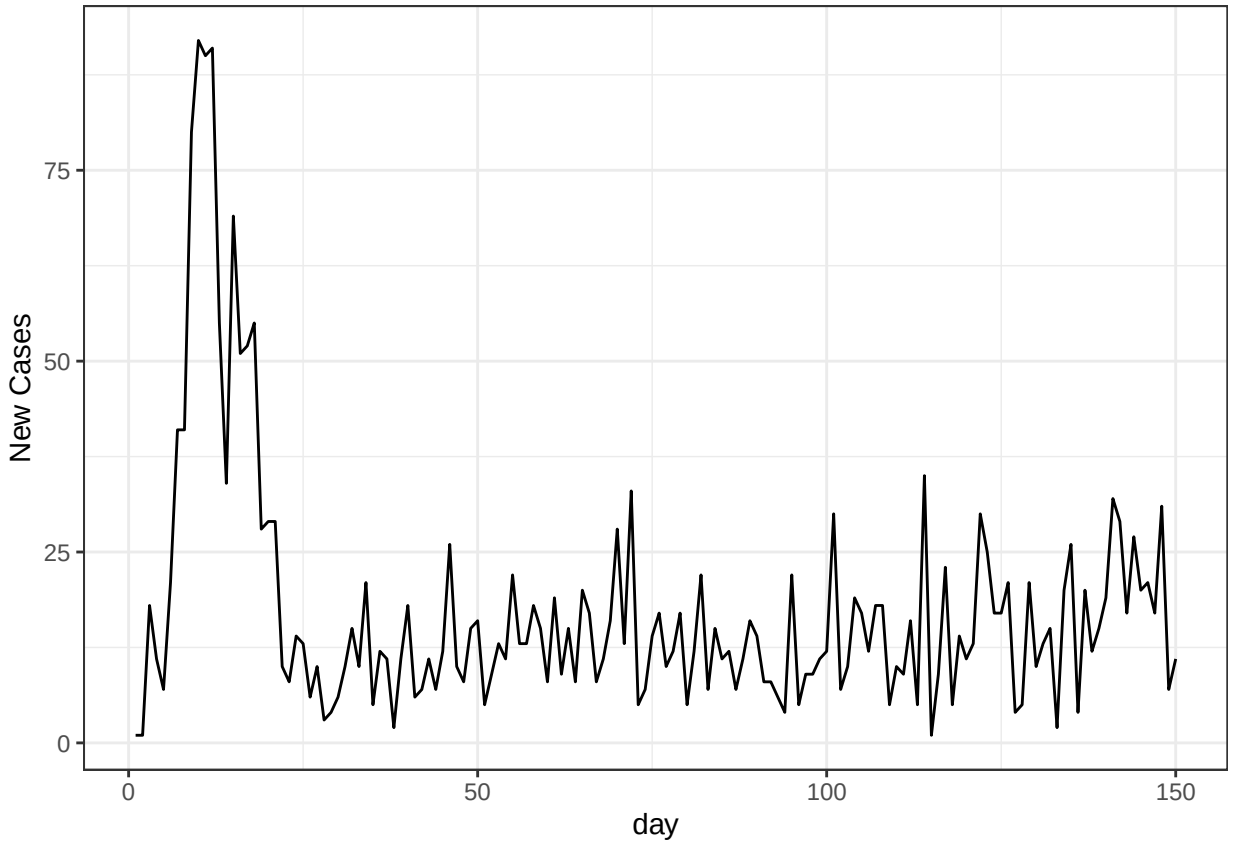
First, lets set it up so that files get saved in the same place that this ".rmd'' file is stored.

```
library(rstudioapi)
# Getting the path of your current open file
current_path = rstudioapi::getActiveDocumentContext()$path
#Set the working directory to that location
setwd(dirname(current_path ))
```

Load in data from github and plot

This dataset was artificially made. The goal is to figure out what model and parameters led to these observations. The data show the number of new cases of an illness each day.

```
library(readr) #this library is used to help load the csv file.
simuldata <-read_csv("https://raw.githubusercontent.com/kmontovan/pomp/refs/heads/main/SimulatedPompitis


ggplot(
  data=simuldata,
  mapping=aes(x=day,y=NewCases)
)+
  geom_line()+
  labs(y="New Cases")+
  theme_bw()
```



Model Info

Setup a function to model the populations in S, I, andn $NewI$ based on the parameters, the timestep, and the random variables accounting for the process noise.

```
#Setup Model Step function
si_step <- function(S, I, NewI, Beta, gamma, delta.t, ...){
  infections <- rbinom(n=1,size=S, prob=1-exp(-Beta*I*delta.t))
  recover <- rbinom(n=1,size=I, prob=1-exp(-gamma*delta.t))
  S <- S - infections + recover
  I <- I+ infections - recover
  NewI <- NewI + infections
  c(S=S, I=I, NewI=NewI)
}
```

Setup the initial conditions. These are defined in a way that allows us to change the parameter defining the fraction of the population that is infected (ϕ) if we want to. The initial conditions need to be integers so the `nearbyint()` function helps ensure that the are integers.

```
si_rinit <- function(N, phi, ...){
  c(S = round((1-phi)*N),
    I= round(phi*N), NewI=0)
}
```

We also need to setup the dmeasure and rmeasure functions to use the negative binomial distribution to assess the probability of seeing the observed number of cases given the model population sizes and parameters (dmeasure) and performing a random draw from the distribution to simulate our own sampling and reported number of cases (rmeasure). Note: we only need to setup rmeasure to be able to simulate results but will need dmeasure to assess the likelihood. Since the two are pretty closely related it is usually easier to set them up together.

```
si_rmeas <- function(NewI, rho,k,...){
  c(NewCases=rnbinom(n=1,size=k,
                     mu=rho*NewI))
}

si_dmeas <- function(log,NewCases, NewI, rho, k, ...){
  dnbinom(x=NewCases, size=k, mu=rho*NewI, log=log)
}
```

Create POMP structure

This creates a POMP structure that holds key information about our model. It is an optional step but can be helpful so that we don't have to enter these pieces eact time we run things. These pieces can (and will) be modified and added to later, but for now, let's focus on the pieces we need right now.

```
SI_Model <- pomp(
  data = simuldata,
    times="day",
    t0=0,
    rinit=si_rinit,
    rprocess=euler(si_step,delta.t= 1/4),
    rmeasure=si_rmeas,
    dmeasure=si_dmeas,
    statenames=c("S","I","NewI"),
```

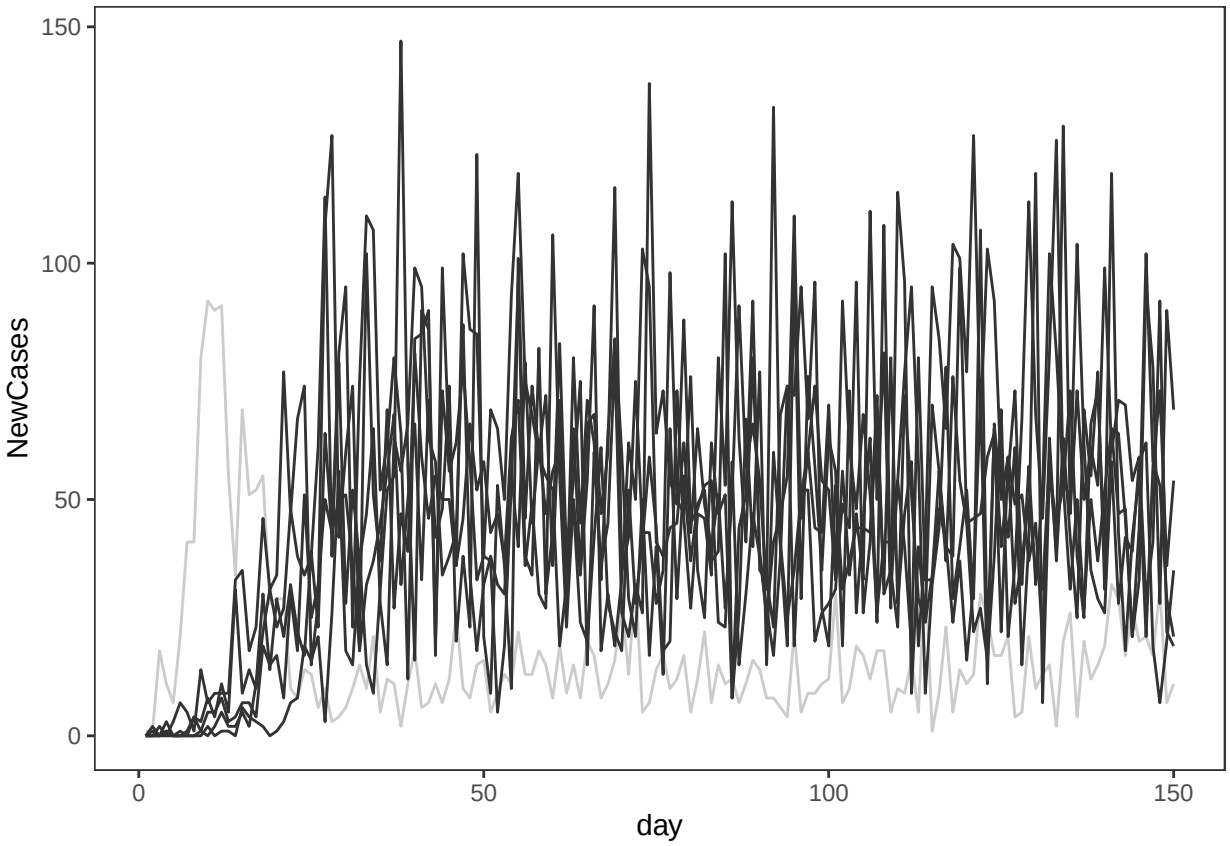
```
    accumvars="NewI",
    paramnames=c("Beta","gamma","rho","k","N","phi")
  )
```

You can skip this step and just feed the pieces into the model each time but if there are parts of your model that will not be changing it can make it easier to have them all together.

Optional: Run simulations of a single parameter set

This is a helpful first step to make sure this everything so far is working before moving on. Note: The data is shown in teal and the simulation results are shows in red.

```
sims <- SI_Model |>
  simulate(
    params= c(N=1000, #population size
              phi = 0.004, #fraction of population infected at time 0
              Beta=.0003, # rate of spread of illness - per day
              gamma=.1, #recovery rate
              rho=.8, #fraction of new infections that are recorded
              k=5 ), #changes the variability of reporting accuracy
    nsim=5,
    format="data.frame",
    include.data=TRUE
  )

sims |>
  ggplot(aes(x=day,y=NewCases,group=.id,color=.id=="data"))+
  geom_line()+
  guides(color="none")+
  theme_bw()+ scale_color_grey()+
  theme(panel.grid.major = element_blank(), panel.grid.minor = element_blank(),
panel.background = element_blank())
```



As you are getting warmed up with the model, it can also be helpful to play with how changing the parameter values changes the results. This is not only helpful to getting your mind around the model, but can also sometimes help catch errors in your code if something is responding in a really unexpected way.But keep in mind that in complex systems it isn't always clear how something should respond to a specific parameter. So unexpected results will often occur.

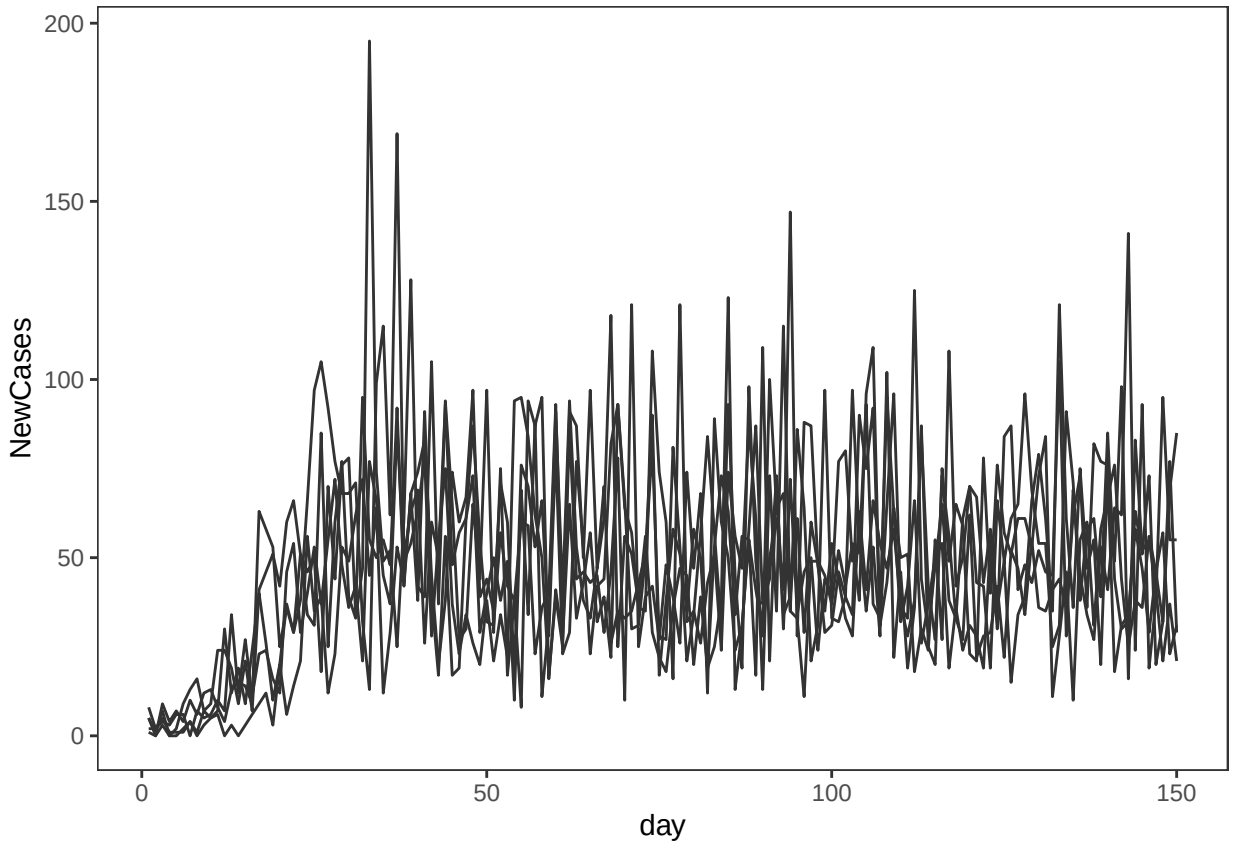
You might see that sometimes the number of infected people drops to zero and stays at zero. Anytime the number of people in I becomes zero, our model is set so that there will be no new infections, so I will stay at zero forever. Sometimes models have a term for infections coming in from outside of the population. This is more realistic and you could add this to the model if you wanted to but we will keep our model relatively simple and not add it.

Switching to Csnippets

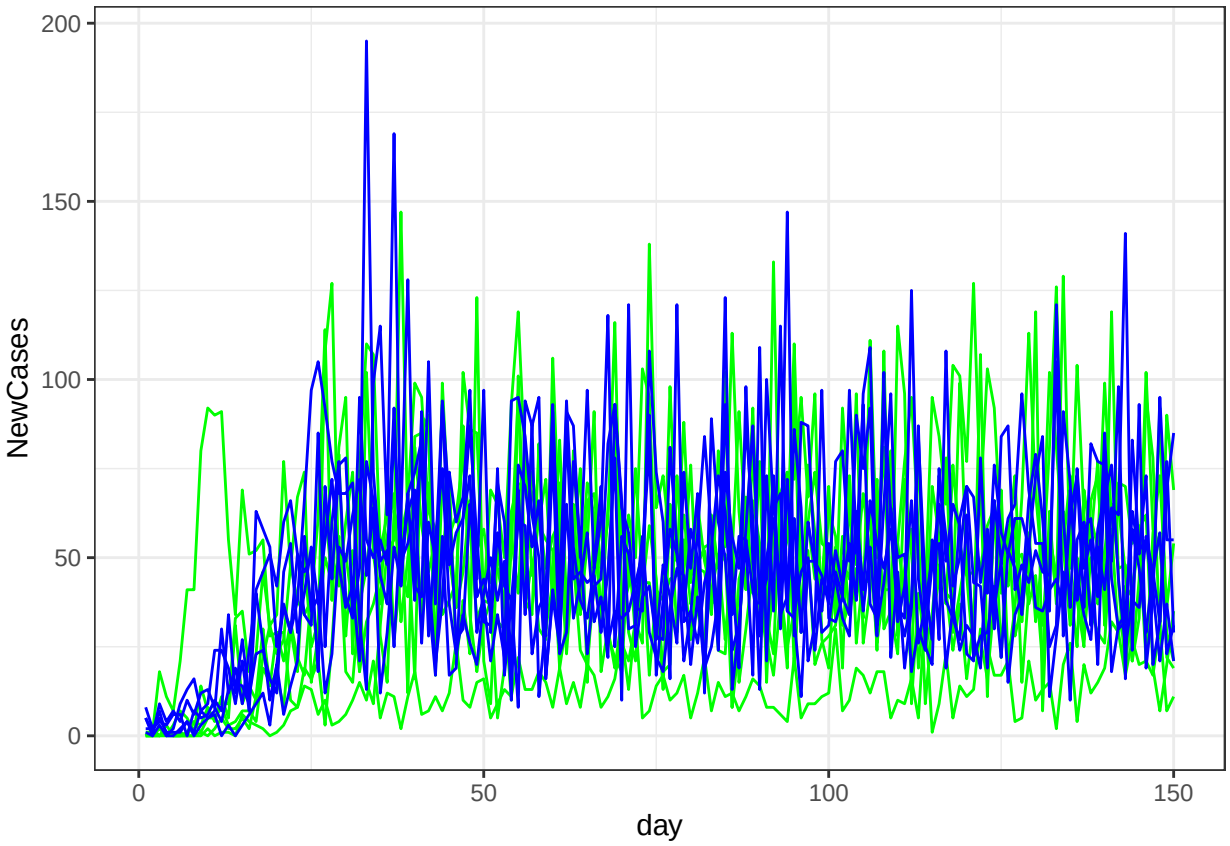
We delayed on this piece because it can be tricky to troubleshoot problems that arise with the Csnippets and getting your computer to run Csnippets correctly. You can go on without the Csnippets but your codes will take longer to run.

```
si_stepC <- Csnippet("
  double infections = rbinom(S,1-exp(-Beta*I*dt));
  double recoveries = rbinom(I,1-exp(-gamma*dt));
  S += -infections + recoveries;
  I += infections - recoveries;
  NewI += infections;
")

si_rinitC <- Csnippet("
```

```
  S = nearbyint((1-phi)*N);
  I = nearbyint(phi*N);
  NewI = 0;
")

#Something about the rmeasure function is leading to differences
si_rmeasC <- Csnippet("
  NewCases = rnbinom_mu(k,rho*NewI);
  ")

si_dmeasC <- Csnippet("
    lik = dnbinom_mu(NewCases,k,rho*NewI, give_log);
  ")

SI_ModelC <- pomp(
  data = simuldata,
    times="day",
    t0=0,
    rinit=si_rinitC,
    rprocess=euler(si_stepC,delta.t= 1/4),
    rmeasure=si_rmeasC,
    dmeasure=si_dmeasC,
    statenames=c("S","I","NewI"),
    accumvars="NewI",
    paramnames=c("Beta","gamma","rho","k","N","phi")
  )

sims2 <- SI_ModelC |>
  simulate(
    params= c(N=1000, #population size
              phi = 0.004, #fraction of population infected at time 0
              Beta=0.0003, # rate of spread of illness - per day
              gamma=.1, #recovery rate
              rho=.8, #fraction of new infections that are recorded
              k=5 ), #changes the variability of reporting accuracy
    nsim=5,
    format="data.frame",
    include.data=FALSE
  )

sims2 |>
  ggplot(aes(x=day,y=NewCases,group=.id,color=.id=="data"))+
  geom_line()+
  guides(color="none")+
  theme_bw()+ scale_color_grey()+
  theme(panel.grid.major = element_blank(), panel.grid.minor = element_blank(),
panel.background = element_blank())
```




```
#This approach using Csnippets should (roughly)
#match the approach from before.
ggplot()+
  geom_line(data=sims, aes(x=day,y=NewCases,group=.id),color="green")+
  geom_line(data=sims2, aes(x=day,y=NewCases,group=.id),color="blue")+
  guides(color="none")+
  theme_bw()
```



Once we are sure it is working, we will set SI_Model to be the version with Csnippets.

```
SI_Model <- pomp(
  data = simuldata,
    times="day",
    t0=0,
    rinit=si_rinitC,
    rprocess=euler(si_stepC,delta.t= 1/4),
    rmeasure=si_rmeasC,
    dmeasure=si_dmeasC,
    statenames=c("S","I","NewI"),
    accumvars="NewI",
    paramnames=c("Beta","gamma","rho","k","N","phi")
  )
```

When you adjust the Csnippets for a new model, don't forget to move slow and test that things work in the following steps frequently.

If you can't get the Csnippets to work you can comment out the previous section and proceed with the R functions instead. But you will have to switch to them again when you open and work on Models 2-4.

Estimate the log likelihood

As a reminder, the likelihood function is a way to quantify how well the model explains the observed data across different values of a parameter. For simple models, the likelihood function can sometimes be computed exactly based on the model and on the probability distributions of the random variables. But for most models, this will not be possible so we instead rely on a particle filter to estimate the likelihood for each specific set of

parameters. Then we will combine the results of many runs to create an estimate of the likelihood function.

To run the particle filter, we use the command `pfilter()` in pomp. The number of particles, Np, affects the accuracy of the estimates. The following code returns the log-likelihood of the data for the model with the present parameter values.

Note: you can use `Np=100` when developing things to make them run faster. But then switch to `Np=1000` for the final runs to get more accurate estimates of the log-likelihood. We will set things up with `Np=100` so that things run nfaster, but know that the estimates will ot be as accurate.

```
pf <- SI_Model |>
  pfilter(params= c(N=1000, phi = 0.004,Beta=.0024, gamma=.1, rho=.8, k=5 ),
          Np=100)
logLik(pf)
```

```
## [1] -1051.151
```

The random variables involved in the model make it so that each time we run the particle filter we get a slightly different result. Let's estimate the loglikelihood multiple times to see how much the estimates vary. Because this gets computationally expensive, we set up parrallel processing to increase efficiency. This requires the libraries "doParallel" and "doRNG"

```
foreach (i=1:10, .combine=c) %dopar% {
  library(pomp)
  SI_Model |>
    pfilter(
      params= c(N=1000, phi = 0.004,Beta=.0024, gamma=.1, rho=.8, k=5 ),
      Np=500 #Quicker: Np=100, More accurate Np=500-1000
      )
} -> pf
logLik(pf) -> ll
ll
```

```
##  [1] -1050.335 -1049.673 -1052.479 -1050.462 -1052.443 -1047.376 -1047.559
##  [8] -1049.399 -1050.631 -1051.286
```

These are each independent estimates of the logliklihood. To understand the variability (and error) in these estimates, we can use the `logmeannexp()` function to find the mean and calculate the standard error across the estimates.

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

```
##           est            se
## -1048.8744795     0.6166039
```

Increasing the number of particles tends to decrease the variability of the estimates but only to a certain point. Beyond that the variability is coming more from the random variables in the model than from the errors in the estimation process. We will come back to this later to explore how to know if your number of particles is enough. Usually 1000 is enough, and we will use this number of particles for the rest of this example, but in your own work this might be a good place to explore what the right number of particles might be for your model. It also can be helpful to reduce the number of particles while developing the code to make it run quicker, and then increase it before performing the final analysis.

Performing a local search for optimal parameter sets

Soon we are going to talk about how to search a large multidimensional parameter space. But before we do that, we are going to perform a local search. To do this, we start with one set of parameter values and then

let the parameters wander a bit and move them in the direction that increases the log likelihood. This can be a good way to find local maxima and to improve our ability to find the optimal parameter set.

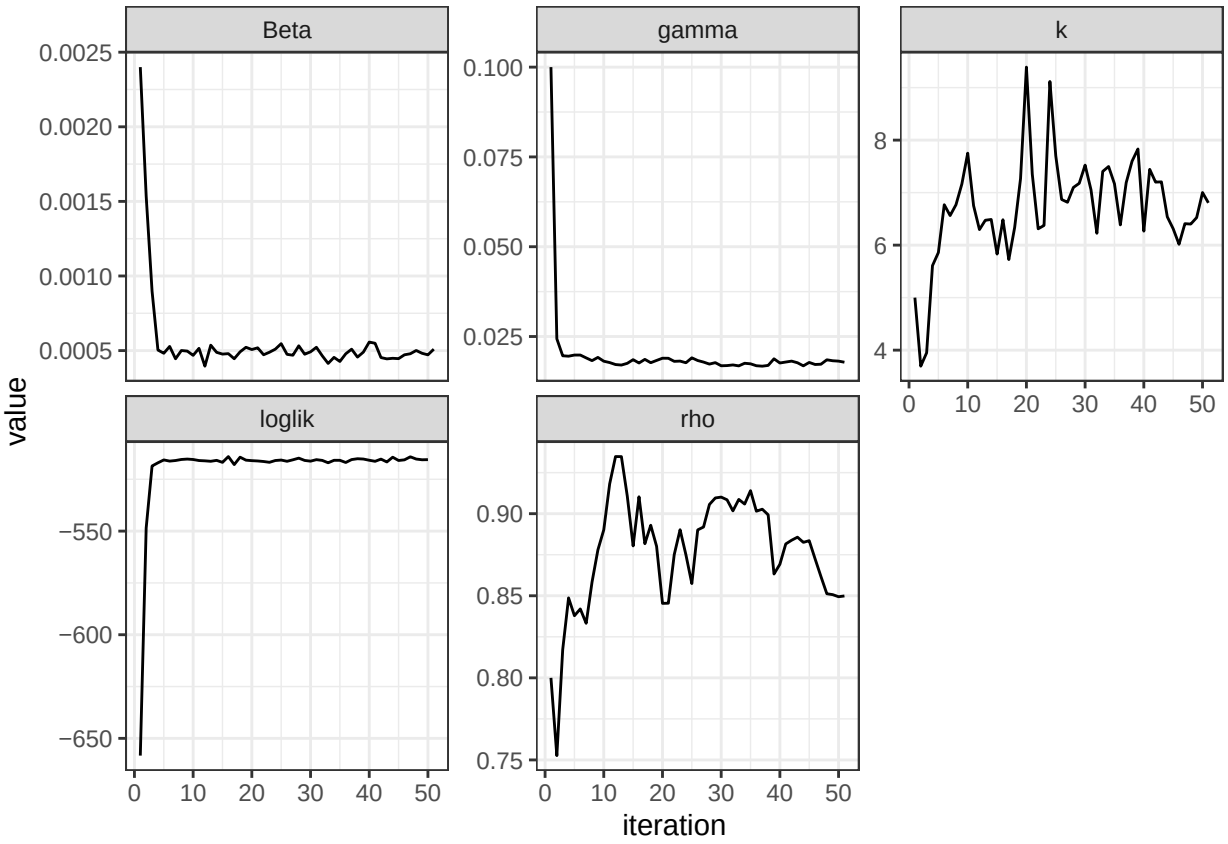
At this point it can be helpful to think about which of the parameters are going to be fixed and which will be adjusted to better fit the data. In this example we were told that we know the population is 1000 people and that 4 people started with the infection on day 0. So our population size (N) and our fraction infected (fI) are fixed.

Now we will see what happens when we start at the parameter values we specified before (or whichever ones you choose) and let them change a bit to maximize the log-likelihood.

```
mifs_local <-
  SI_Model |>
  mif2(
    params= c(N=1000, phi = 0.004,Beta=.0024, gamma=.1, rho=.8, k=5 ),
    Np=500, #Set small for speed, later increase to Np=1000
    Nmif=50, #number of iterations to perform in the local search
    cooling.fraction.50=0.5,

    #names of all parameters to fit
    paramnames=c("Beta","gamma","k","rho"),

    #Setup parameter transformations
    partrans=parameter_trans(
      log=c("Beta","gamma","k"), #always positive
      logit=c("rho") #always between 0 and 1
    ),

    #this defines how much to change the parameter values each iteration
    rw.sd=rw_sd(
      Beta=0.02,
      gamma=0.02,
      rho=0.02,
      k=0.02
    )
  )
```

Graph the results:

```
mifs_local |>
  traces(c("Beta","gamma","rho","k","loglik")) |>
  melt() |>
  ggplot(aes(x=iteration,y=value))+
  geom_line()+
  facet_wrap(~name,scales="free_y")+
  theme_bw()
```



We see the likelihood rapidly rise in the early iterations and then levels off. This tells us that we have fairly quickly found a nearby parameter set that greatly increases the log-likelihood. While the parameters are still drifting, the number of iterations (`Nmif`) is probably large enough. These plots also show how each parameter changes in this local search.

Setting up to save results

Decide on a name for your model results. We will want a different name for the saved results from different models. Our two models for this example will be the SI model and later the SIR model so `SI_Model_results.csv'' will work for our model name and we can add the`R'' when we get to the second model.

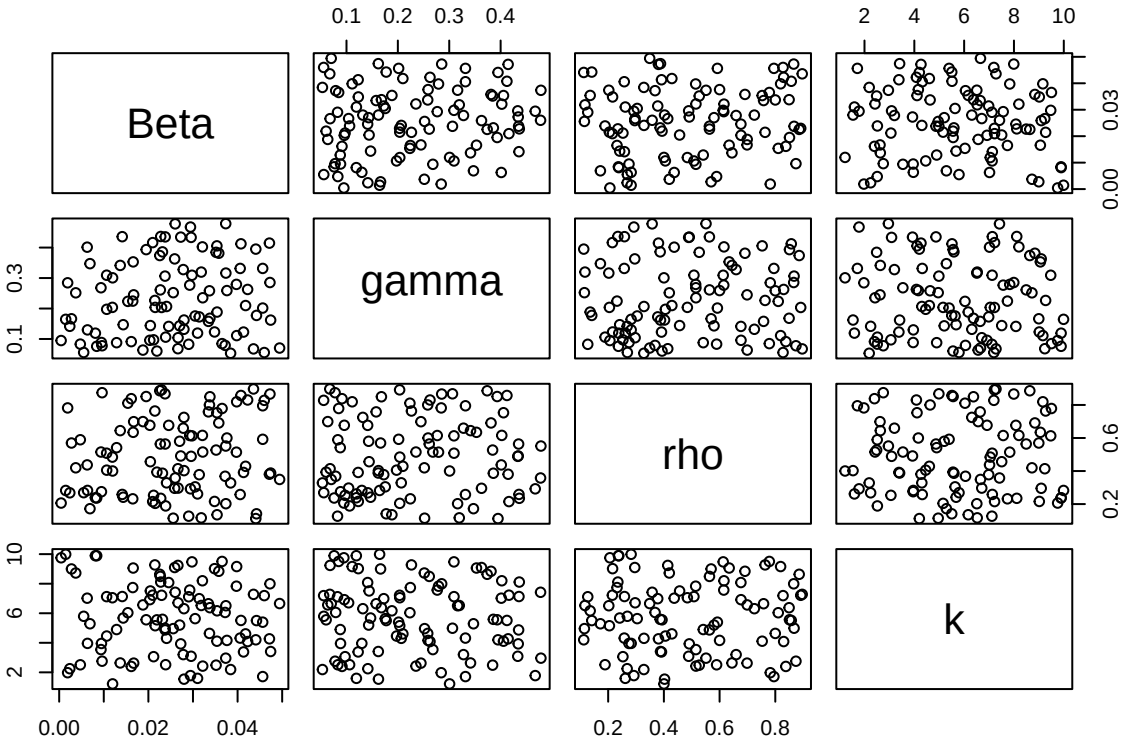
```
savefile <- "SI_Model_results.csv"
```

Global Search

The `mif` process we learned to use above starts at a combination of parameter values and then walks around and goes inn whatever direction leads to better (larger) loglikelihood values. This usually find a better fitting parameter set **nearby** but will get stuck on little local maxima. To find a best parameter set across all the possible values of each of the parameters we combine this local **mif** search with sets of parameters that span reasonable parameter space. We start at a lot of different places in parameter space and search for nearby parameter sets that minimize the log likelihood.

The `runif_design` creates a sequence of each parameter spanning the range from the lower to the upper bound and giving `nseq` results.

```
#Note: for fixed parameters, the lower and upper bounds are equal.
guesses <-
  runif_design(
    lower=c(Beta=0.0001,gamma=0.05,rho=0.1,k=1,N=1000,phi=0.004),
    upper=c(Beta=0.05,gamma=0.5,rho=0.9,k=10,N=1000,phi=0.004),
    nseq=100 #Use 20-50 when getting thing working
             #Change to nseq>=200 later
  )

#Show the starting parameter sets by looking at all possible pairs of parameters
pairs(~Beta+gamma+rho+k,data=guesses)
```



This shows that the parameters are spread across parameter space and there are no clear relationships between pairs of parameters in our starting parameter sets.

Now we will run the local search starting at each of these parameter sets.

```
par_trans=parameter_trans(
      log=c("Beta","gamma","k"), #always positive
      logit=c("rho") #always between 0 and 1
    )

foreach(
    #Start with a guess from the set of guesses created before
    guess=iter(guesses,"row"),
    .combine=bind_rows
    ) %dopar% {
      library(pomp)

      #Perform the local search -
      #to find better parameter sets near the randomly selected ones
      mf <- mif2(
```

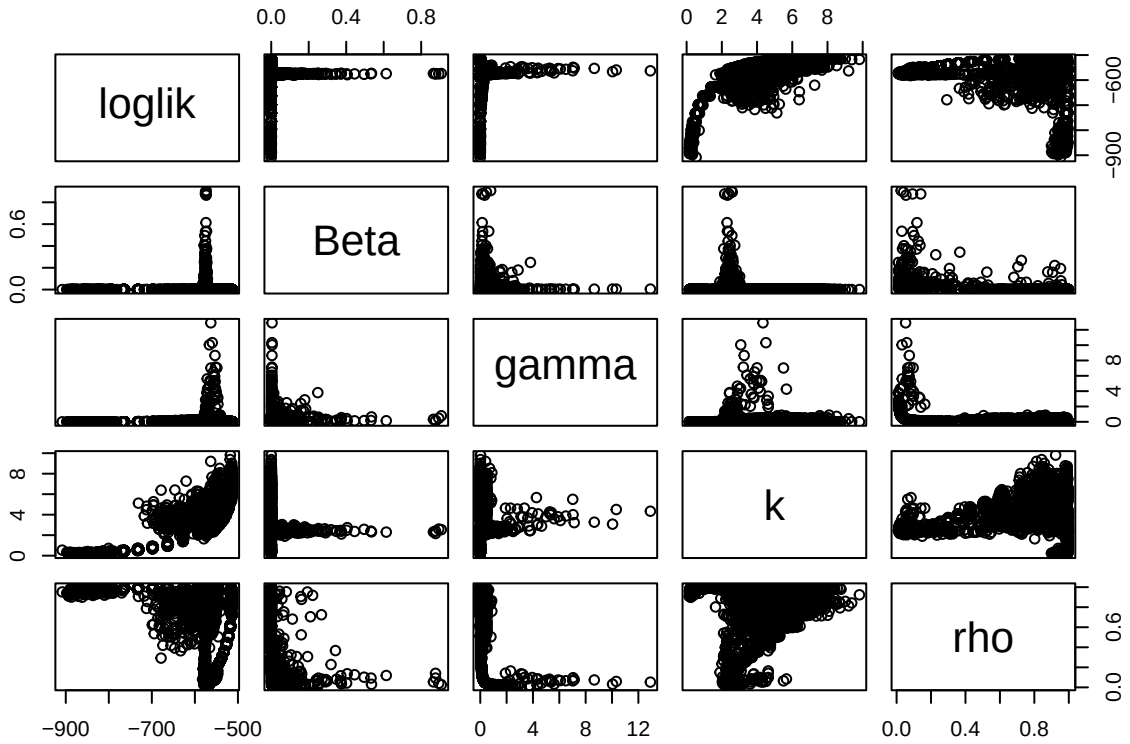
```
        SI_Model,
        params=guess,
        paramnames=c("Beta","gamma","k","rho"),
        rw.sd=rw_sd(Beta=0.02,gamma=0.02,rho=0.02,k=0.02),
        Np=500, #Quick: Np=50-100, More precise: to Np=500-1000
        Nmif=50, #Quick: Nmif=10-20, More precise: Nmif>=50
        cooling.fraction.50=0.5,
        partrans=par_trans)

      #estimate the likelihood n=10 times for each resulting parameter set
      ll <- mf |>
        pfilter(Np=500) |> #Quick: Np=50-100, More precise: to Np=500-1000
        logLik() |>
        replicate(n=10) |>
        logmeanexp(se=TRUE,ess=FALSE)

      #
      mf |>
        coef() |>
        c(loglik=ll)
    } -> results

#related to efficient parallel processing
attr(results,"ncpu") <- getDoParWorkers()

#Manage data a bit
results |>
  rename(loglik=loglik.est) |>
  filter(is.finite(loglik)) -> results
```

The `append_data()` function is in the `pomp` library and adds the results to a saved CSV file and pulls the cumulative data from that same saved csv file. We then save this whole dataset as `all'' so we can use it here in R. You can overwrite saved results using the option`overwrite=TRUE"

```
all <- append_data(data=results,file=savefile)
```

The `pairs()` function plots all of the pairs of variables in the data. This helps us see relationships between parameters and between parameters and the loglikelihood.

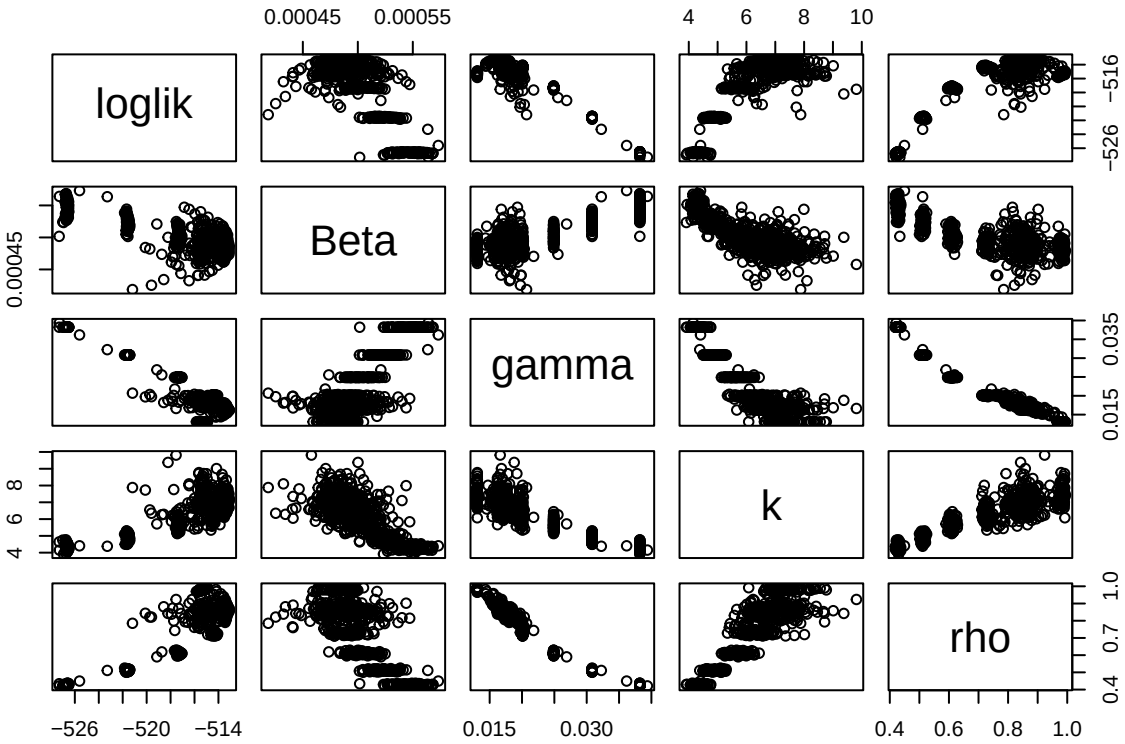
```
pairs(~loglik + Beta + gamma + k + rho ,data=all)
```



We can see that although the parameter sets started out evenly distributed across parameter space, the local search to maximize the likelihood has resulted in structured relationship between some pairs of variables. For example, it looks like the better combinations for `rho` and `k` have them linearly related to each other (with larger values of k being associated with larger values of ρ). We also see that the best log-liklihood values tend to be happening for small values of β.

What other things do you notice?

```
plot_data <- all |>
  filter(loglik>max(loglik)-15)
pairs(~loglik + Beta + gamma + k + rho, data= plot_data)
```



For the paramter sets which achieved within 20 of the maximum log-likelihood, speed up the cooling to refine the estimates.

```
guesses <-
  results |>
  filter(loglik>max(loglik)-20) |>
  select(!starts_with("loglik"))

## Search for best Likelihood
registerDoParallel()
registerDoRNG(1929342697)
foreach(
    guess=iter(guesses,"row"),
    .combine=bind_rows
) %dopar% {
    library(pomp)
    mifs_local |>
      mif2(
        params=guess,
        cooling.fraction.50=0.1, # cf with previous mif2 runs
        Nmif=50, #Set Nmif=50 for final run
        rw.sd=rw_sd(Beta=0.02,gamma=0.02,rho=0.02,k=0.02)
        ) -> mf
      mf |>
        pfilter(Np=1000) |>
        logLik() |>
        replicate(n=10) |>
        logmeanexp(se=TRUE,ess=TRUE) -> ll
      mf |>
        coef() |>
        c(loglik=ll)
```

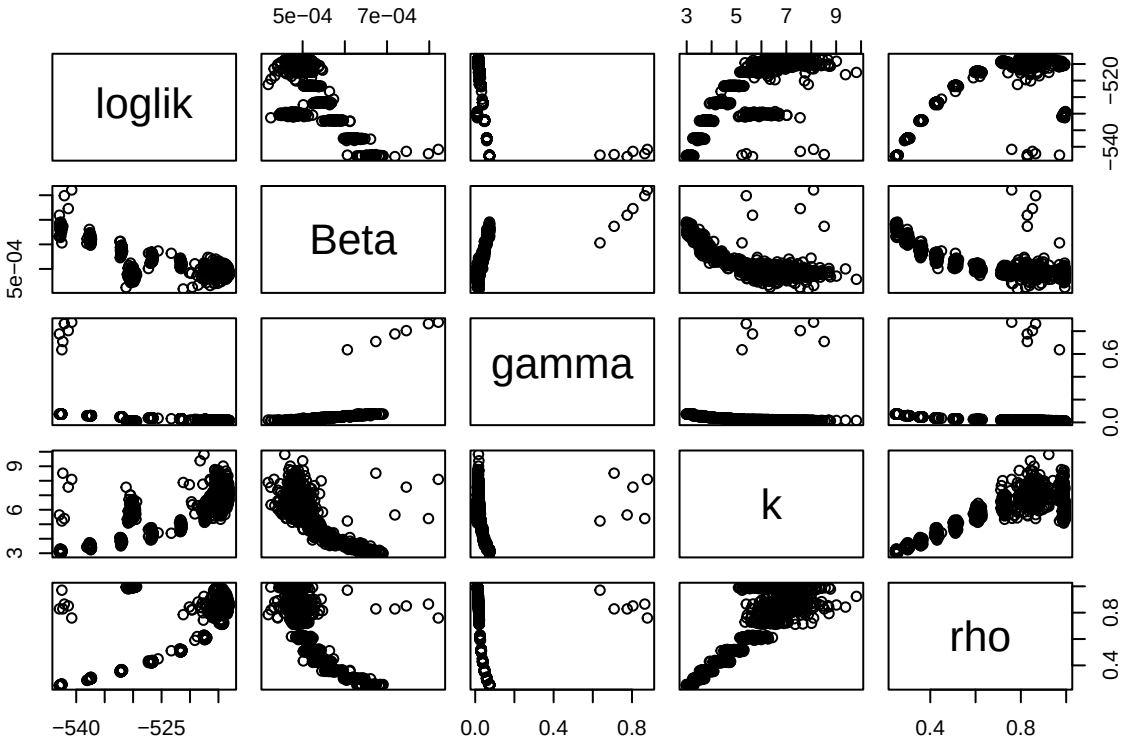
```
    } -> results
    attr(results,"ncpu") <- getDoParWorkers()

results |>
  rename(loglik=loglik.est) |>
  filter(is.finite(loglik)) -> results
```

Add these new results to the saved file

```
append_data(results,file=savefile) -> all
```

Visualization of global parameter space with a focus on areas with higher likelihood values.

```
plot_data <- all |>
  filter(loglik>max(loglik)-30)
pairs(~loglik + Beta + gamma + k + rho, data= plot_data)
```



Profile likelihood

When we make the profile likelihood, we look across the full range of one parameter of interest and hold that parameter constant while exploring the rest of the parameter space for the best parameter set. Here we hold gamma constant.

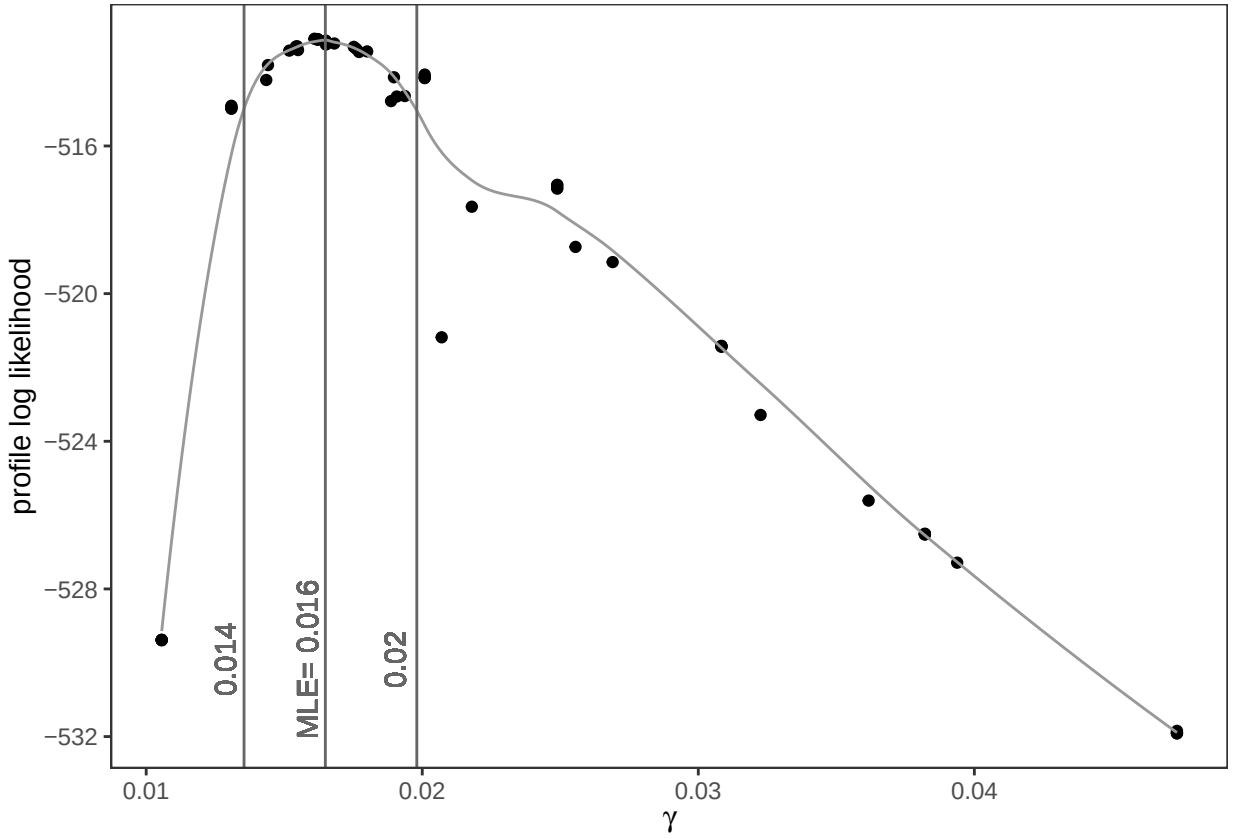
```
#Pull bounds for parameters from those that resulted in pretty good loglik values
box <-
    all |>
    filter(loglik>max(loglik)-20) |>
    sapply(range)

## Create parameter ranges for each fit parameter and the parameter of interest
## Note: we distribute gamma on a log scale so that we get heavier sampling for smaller values of gamma
```

```
guesses <-
    profile_design(
      gamma=exp(seq(log(0.001),log(0.5),length=30)),
      lower=c(box[1,c("Beta","rho","k")],N=1000,phi=0.004),
      upper=c(box[2,c("Beta","rho","k")],N=1000,phi=0.004),
      nprof=20,
      type="runif"
    )

registerDoParallel()
registerDoRNG(1268275250)
foreach(
    guess=iter(guesses,"row"),
    .combine=bind_rows
  ) %dopar% {
    library(pomp)
    mifs_local |>
      mif2(
        params=guess,
        cooling.fraction.50=0.1,
        Nmif=50,
        rw.sd=rw_sd(
          Beta=0.02,
          rho=0.02,
          k=0.02
        )
      ) -> mf
    mf |>
      pfilter(Np=1000) |> #set to Np=1000 for final data runs.
      logLik() |>
      replicate(n=10) |>
      logmeanexp(se=TRUE,ess=TRUE) -> ll
    mf |>
        coef() |>
        c(loglik=ll)
    } -> results
    attr(results,"ncpu") <- getDoParWorkers()

results |>
  rename(loglik=loglik.est) |>
  filter(is.finite(loglik)) -> results

append_data(results,file=savefile) -> all
```

`mcap` is a function in the pomp library that is designed to create the likelihood profile and confidence interval from the estimates of the log-likleihood. It adjusts the profile likelihood for Monte Carlo variability.

```
prof <- all |>
  filter(loglik>(max(loglik)-20) )|>
  group_by(round(gamma,3)) |>
  filter(rank(-loglik)<=4) |>
  ungroup()

prof |>
```

```
  with(
    mcap(loglik,gamma,span=0.5)
  ) -> mc

prof |>
  ggplot(aes(x=gamma,y=loglik))+
  geom_point()+
  geom_line(data=mc$fit,aes(x=parameter,y=smoothed),color="gray60")+
  geom_vline(xintercept=c(mc$mle,mc$ci),color="gray40")+
  labs(x=expression(gamma),y="profile log likelihood")+
  theme_bw()+
  theme(panel.grid.major = element_blank(), panel.grid.minor = element_blank(),
panel.background = element_blank())+
  geom_text(x=mc$ci[1]*.95, y=min(prof$loglik)+2, label=paste(round(mc$ci[1],3)),color="gray40",angle =
  geom_text(x=mc$ci[2]*.96, y=min(prof$loglik)+2, label=paste(round(mc$ci[2],3)),color="gray40",angle =
  geom_text(x=mc$mle*.96, y=min(prof$loglik)+2, label=paste("MLE=",round(mc$mle,3)),color="gray40",angl
```



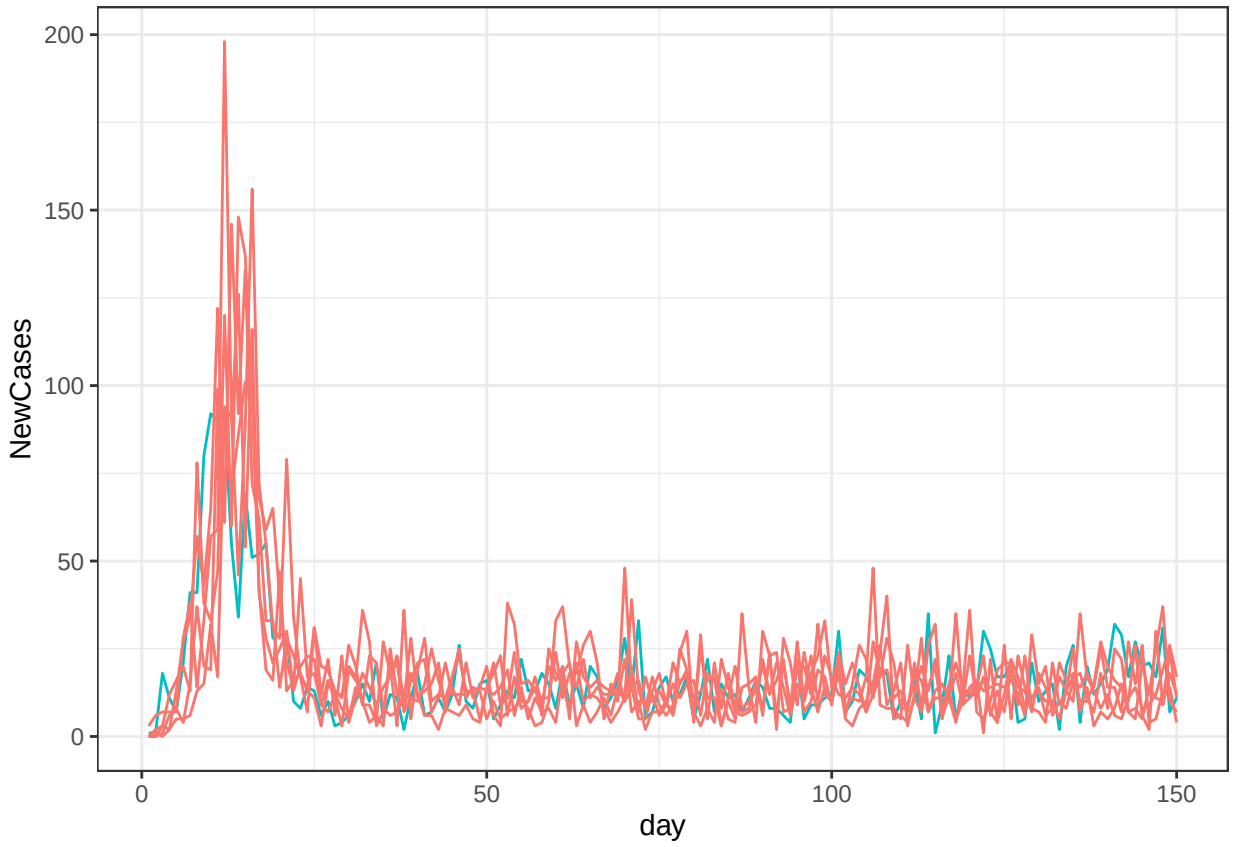
The 95% confidence interval for γ is show in gray If you don't limits on both sides for this interval, that is a sign that either we haven't searched sufficiently broadly to estimate the CI or the parameter is not well defined.

Lets simulate the results for one of the best parameter sets and see what it does.

```
best_param <- all |>
  filter(loglik==max(loglik))|>
  select(N, phi, Beta,gamma,rho,k)
best_param
```

```
##         N   phi          Beta       gamma       rho       k
##     <int> <num>         <num>       <num>     <num>   <num>
## 1:  1000 0.004 0.0004865289 0.01609389 0.8686807 7.21301
```

```
sims2 <- SI_ModelC |>
  simulate(
    params= best_param,
    nsim=5,
    format="data.frame",
    include.data=TRUE
  )

#Plot the simulated reported new cases and the data to see how well the model looks like it is doing.
sims2 |>
  ggplot(aes(x=day,y=NewCases,group=.id,color=.id=="data"))+
  geom_line()+
  guides(color="none")+
  theme_bw()
```

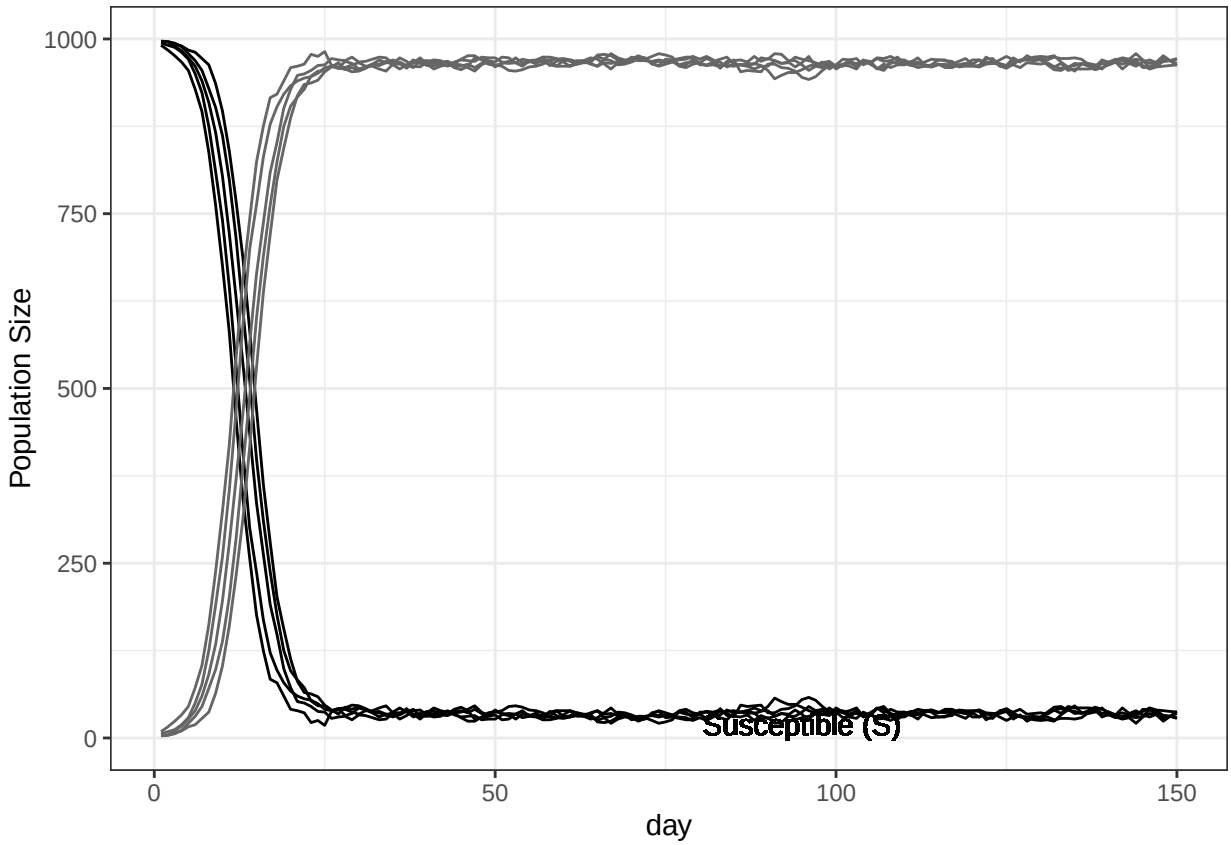


It looks like it is able to fit the data reasonably well, but sometimes models end up with less than reasonable parameter values. Let's think about these values to see if they make sense. The length of time someone is infected and infectious is, on average, $1/\gamma$ or about 62 days. This seems really long for an infection.

Now let's plot the populations of S, I, and R to see what the populationn looks like through time.

```
sims2 |>
  ggplot()+
  geom_line(aes(x=day,y=S,group=.id),color="black")+
  geom_line(aes(x=day,y=I,group=.id),color="gray40")+
```

```
  guides(color="none")+
  theme_bw()+
  ylab("Population Size")+
  geom_text(x=95, y=sims2$S[500]*.75, label="Susceptible (S)",color="black")+
  geom_text(x=95, y=sims2$I[500]*1.15, label="Infected (I)",color="gray40")
```



The number of people infected is shown in gray and the number of susceptible prople are the black lines. We see the results from many simulations but they are all pretty similar. The model is choosing parameters that keep people in I for a long time, this is how we get the reduction in new infections after the initial spike; there aresn't many people left in S to become infected so the number of new infections drops. This is not reasonable for infections (and people with *pomitis* recover in at most 2 weeks according to the experts) so might inicate that we need to constrain the values γ can take on and find a better model structure. In Model 2 we will explore a model that adds in a recovered class. Gaining immunity from infection is common ffor many infections and early in the spread of the disease can lead to a decrease in new infections after the initial spike because of a growing number of recovered (and immune) individuals. See the model 2 rmd file to explore this new model.

Best log likelihood value

```
all |>
  filter(loglik==max(loglik))
```

```
##            Beta      gamma       rho       k     N    fI    loglik loglik.se
##           <num>      <num>     <num>   <num> <int> <num>     <num>     <num>
## 1: 0.0004865289 0.01609389 0.8686807 7.21301  1000    NA -513.0943 0.1046361
```

```
##    loglik.ess   phi
##         <num> <num>
## 1:   9.105625 0.004
```

Compute the AIC value. To do this we need to know how many parameters were fit. For this analysis we fit 4 parameters: Beta, gamma, rho, and k.

```
all |>
  filter(loglik==max(loglik)) |>
  summarize(
    loglik=loglik,
    K=4, #number of parameters fit when maximizing the likelihood
    AIC=-2*loglik+2*K
  )
```

```
##      loglik K      AIC
## 1 -513.0943 4 1034.189
```

We will use these estimates of the best log-likelihood value and the AIC soon when we compare models.