deBInfer logistic ODE example

Philipp H Boersch-Supan

2016-05-09

Preliminaries

First we install the deBInfer package. For this you need to install and load the devtools package first. You can do this from CRAN.

```
install.packages("devtools")
```

```
#Load the devtools package.
library(devtools)
```

```
## Warning: package 'devtools' was built under R version 3.2.3
```

Then you install deBInfer from github

```
install_github("pboesu/debinfer")
```

```
## Downloading GitHub repo pboesu/debinfer@master
## from URL https://api.github.com/repos/pboesu/debinfer/zipball/master
```

```
## Installing deBInfer
```

```
## '/Library/Frameworks/R.framework/Resources/bin/R' --no-site-file  \
##   --no-environ --no-save --no-restore CMD INSTALL  \
##   '/private/var/folders/4c/46l2mj4n4s7fzz1cyjr_wq9h0000gn/T/RtmpjecSZ1/devtools127ba733a945/pboesu-d
##   --library='/Library/Frameworks/R.framework/Versions/3.2/Resources/library'  \
##   --install-tests
```

```
##
```

```
## Reloading installed deBInfer
```

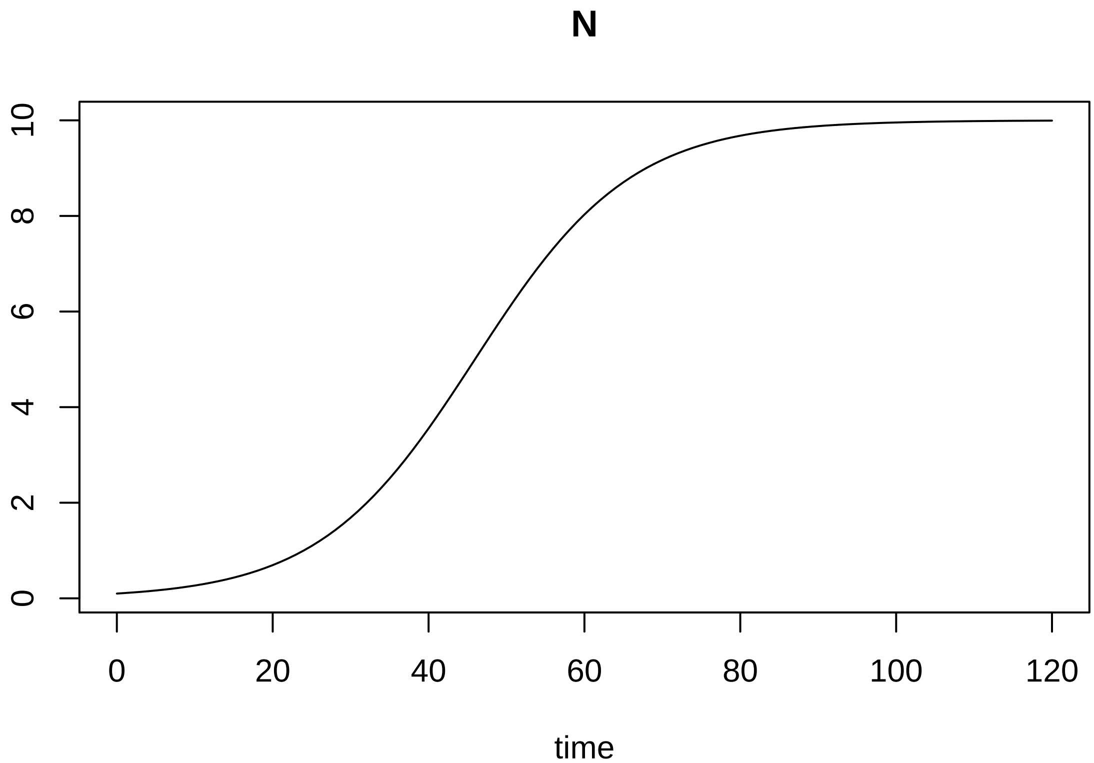
Defining the DE model

We define a logistic growth ODE for deSolve.

```
library(deSolve)
logistic_model <- function (time, y, parms) {
with(as.list(c(y, parms)), {
dN <- r * N * (1 - N / K)
list(dN)
})
```

```
}
y <- c(N = 0.1)
parms <- c(r = 0.1, K = 10)
times <- seq(0, 120, 1)
out <- ode(y, times, logistic_model, parms, method='lsoda')
```

Which gives us the numerical solution



Simulationg observations

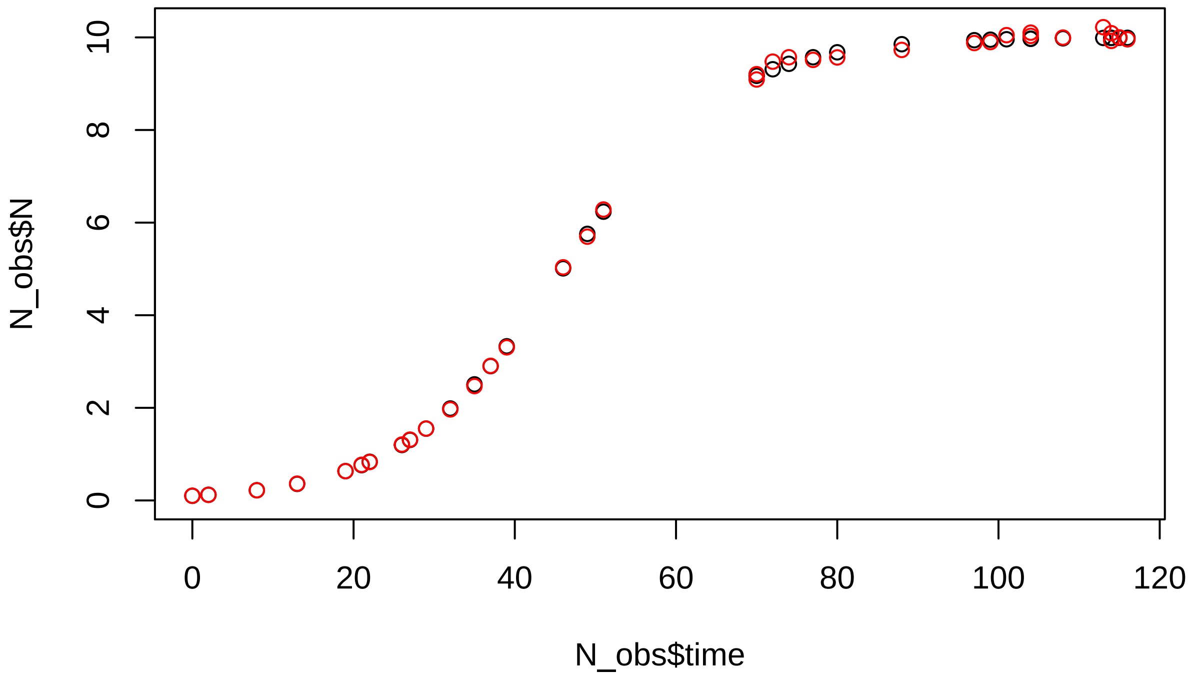
Now we simulate a noisy dataset from this equation. We sample a random subset from the integration output

```
set.seed(143)
N_obs <- as.data.frame(out[c(1,runif(35, 0, nrow(out))),]) #force include the first time-point (t=0)
```

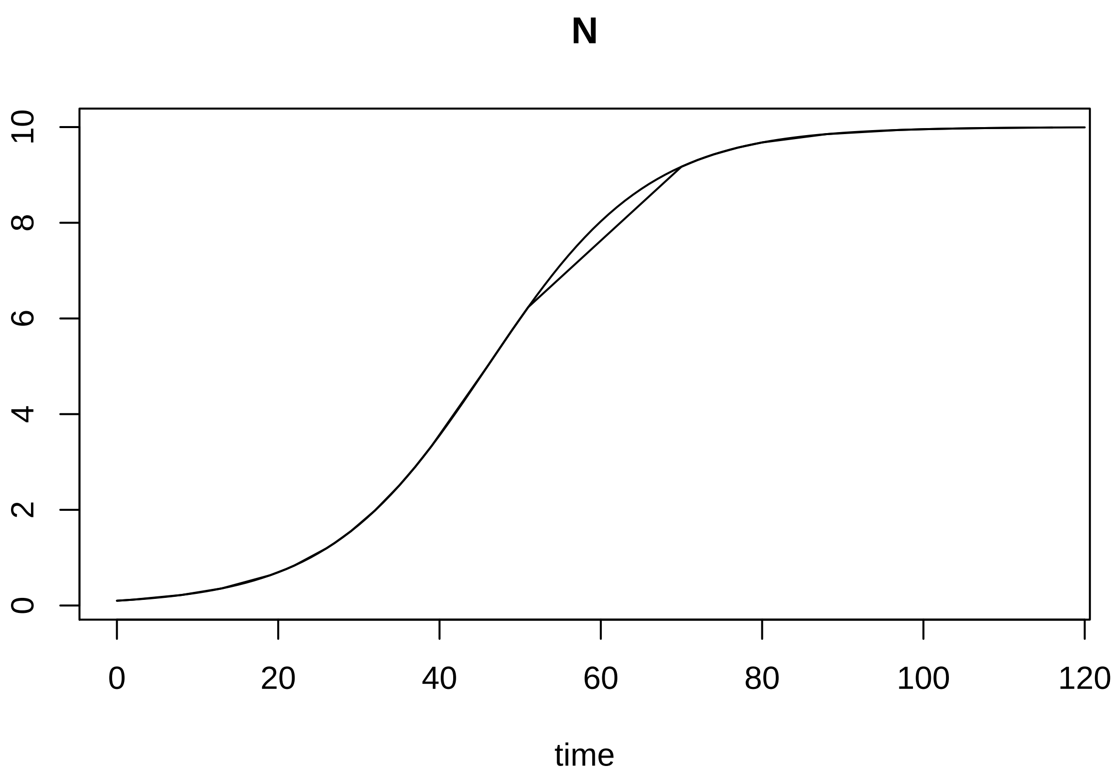
and we "add" lognormal noise

```
# add lognormal noise
  parms['loglogsd.N'] <- -4.6
  N_obs$N_noisy <- rlnorm(nrow(N_obs), log(N_obs$N),exp(parms['loglogsd.N']))
#observations must be ordered for solver to work
N_obs <- N_obs[order(N_obs$time),]

plot(N_obs$time, N_obs$N, ylim=c(0, max(N_obs$N,N_obs$N_noisy)))
points(N_obs$time, N_obs$N_noisy, col="red")
```




```r
out_obs <- ode(y, c(0,N_obs$time), logistic_model, parms, method='lsoda')
plot(out_obs)
lines(out)
```



Defining an observation model and parameters for inference

We define an observation model. Note that we are sampling the log of the observation standard deviation, to ensure sampled values are strictly positive. We also use an epsilon correction for the meanlog, as the DE model can return values of 0 (or even less due to numerical precision).

```
  # the observation model
  logistic_obs_model<-function(data, sim.data, sds, samp){

    llik.N<-sum(dlnorm(data$N_noisy, meanlog=log(sim.data$N + 1e-6),
                       sdlog=exp(samp[['loglogsd.N']]), log=TRUE)
               )

    llik<-llik.N

    return(llik)
  }
```

We declare the parameters for inference:

```
library(deBInfer)
r <- debinfer_par(name = "r", var.type = "de", fixed = FALSE,
                value = 0.5, prior="norm", hypers=list(mean = 0, sd = 1),
                prop.var=0.005, samp.type="rw")

K <- debinfer_par(name = "K", var.type = "de", fixed = FALSE,
                value = 5, prior="lnorm", hypers=list(meanlog = 1, sdlog = 1),
                prop.var=0.1, samp.type="rw")

loglogsd.N <- debinfer_par(name = "loglogsd.N", var.type = "obs", fixed = FALSE,
                value = -2, prior="norm", hypers=list(mean = 0, sd = 1),
                prop.var=0.5, samp.type="rw")
```

and we also need to provide an initial condition for the differential equation:

```
N <- debinfer_par(name = "N", var.type = "init", fixed = TRUE, value = 0.1)
```

All declared parameters are collated using the setup_debinfer function

```
mcmc.pars <- setup_debinfer(r, K, loglogsd.N, N)
```

Conduct inference

Finally we use deBInfer to estimate the parameters of the original model.

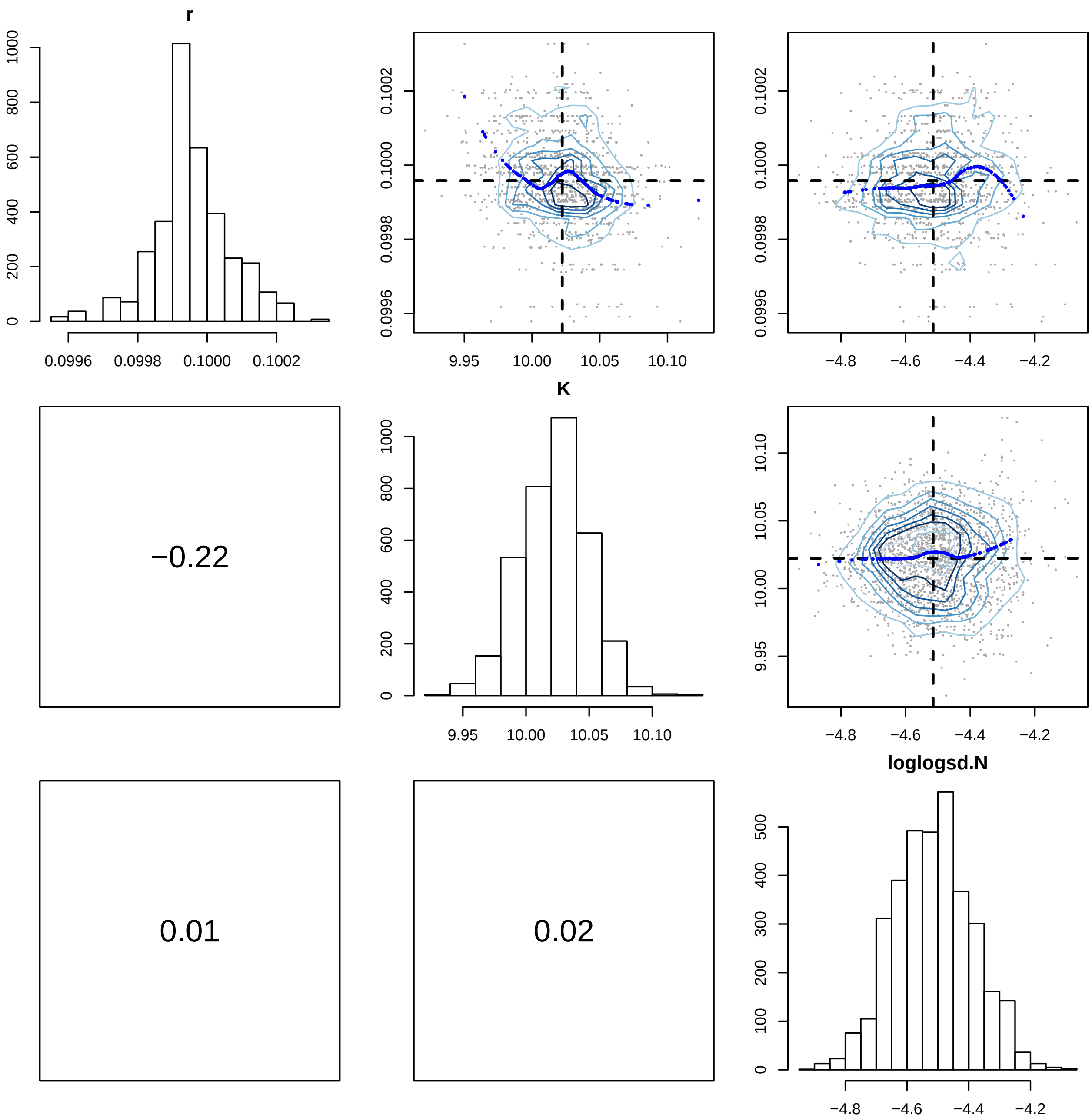
```
# do inference with deBInfer
  # MCMC iterations
  iter = 5000
  # inference call
  mcmc_samples <- de_mcmc(N = iter, data=N_obs, de.model=logistic_model,
```

```
                         obs.model=logistic_obs_model, all.params=mcmc.pars,
                         Tmax = max(N_obs$time), data.times=N_obs$time, cnt=iter,
                         burnin=0.1, plot=FALSE, sizestep=0.1, which=1)
```

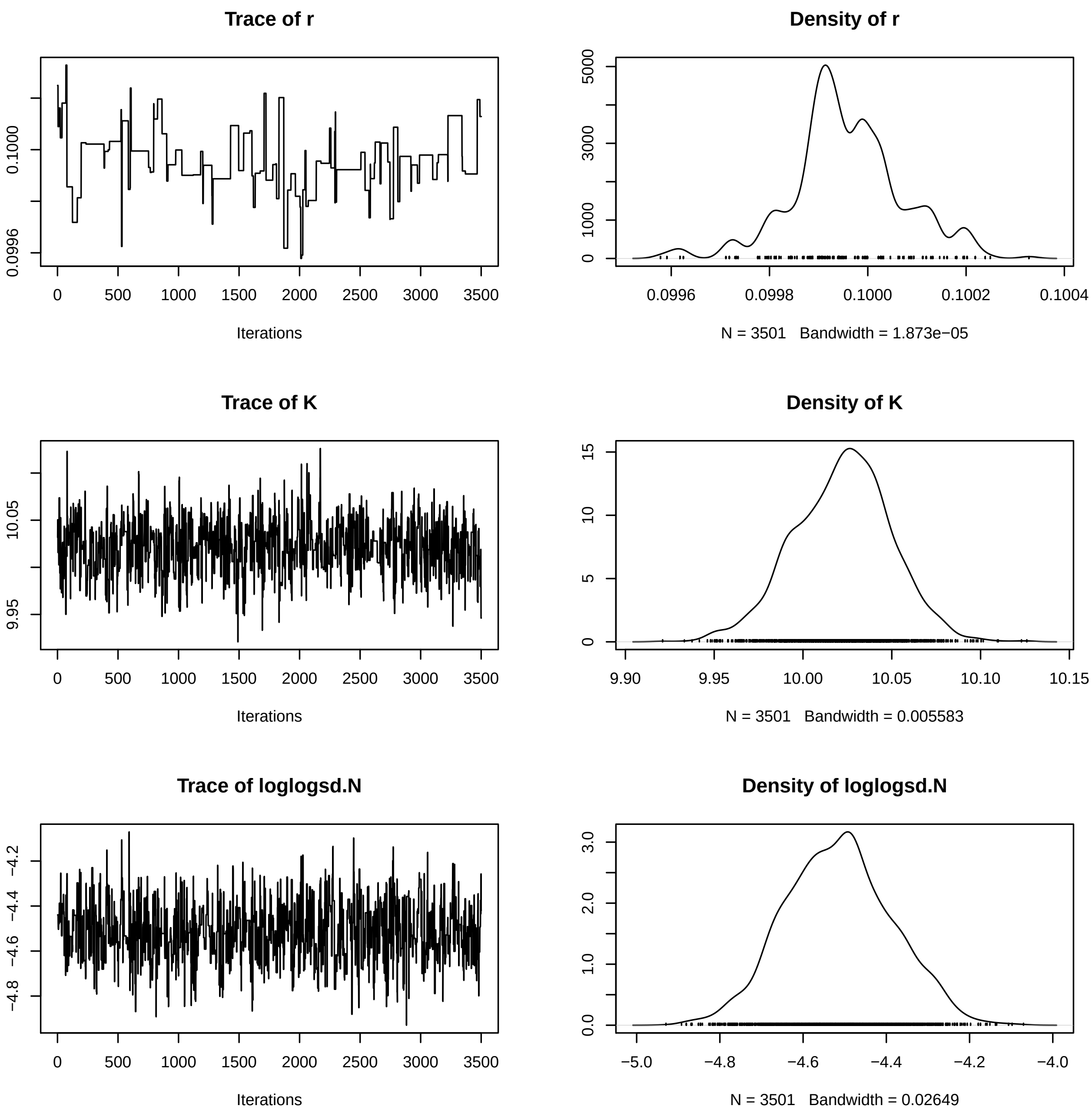
We plot the results

```
burnin = 1500
pretty_pairs(mcmc_samples$samps[burnin:iter,], scatter=TRUE, trend=TRUE)

library(coda)
```



```r
coda.samples <- as.mcmc(mcmc_samples$samps[burnin:iter,])
plot(coda.samples)
```



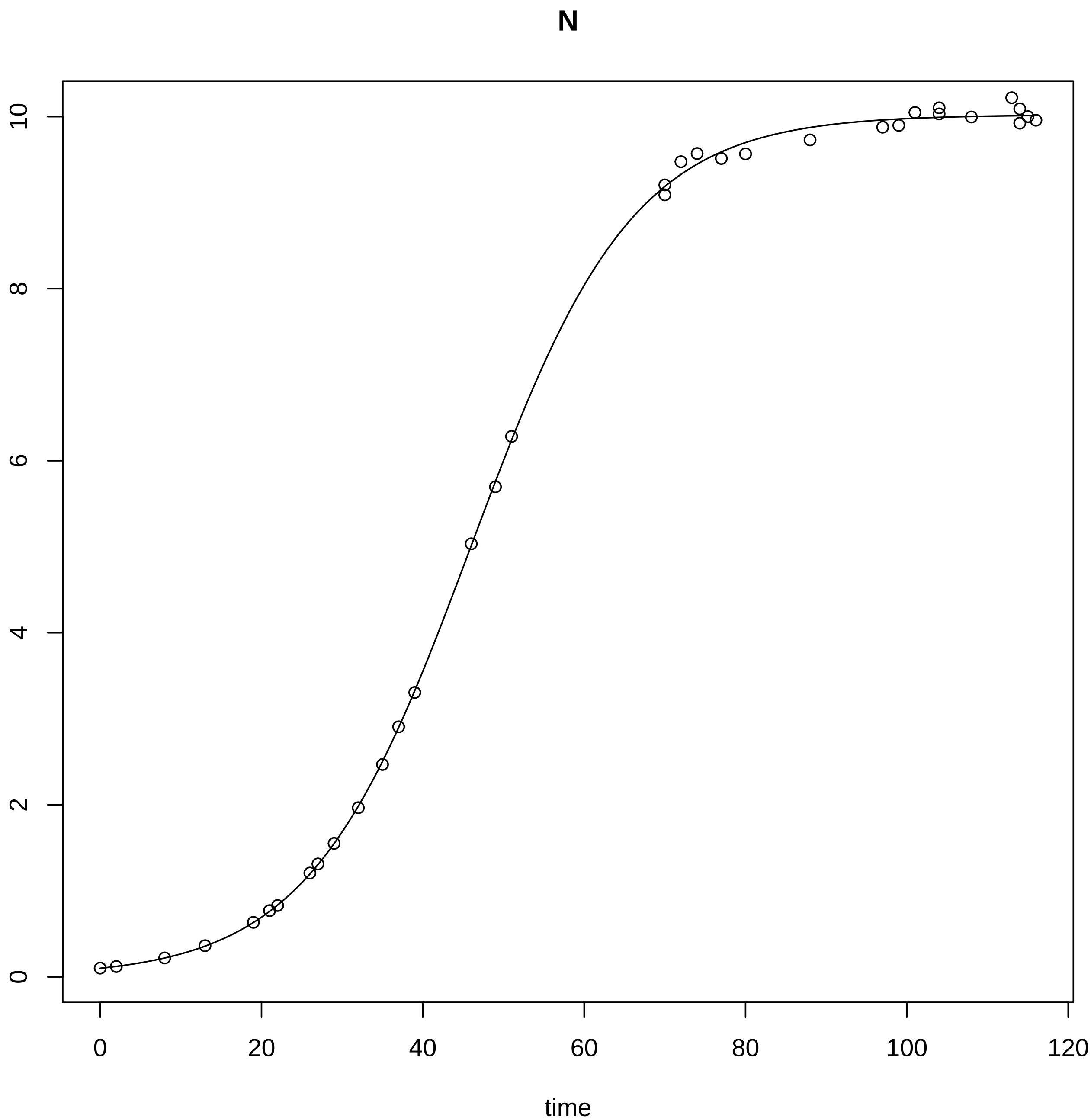

```r
summary(coda.samples)
```

```
##
## Iterations = 1:3501
## Thinning interval = 1
## Number of chains = 1
## Sample size per chain = 3501
##
## 1. Empirical mean and standard deviation for each variable,
```

```
##    plus standard error of the mean:
##
##                 Mean        SD  Naive SE Time-series SE
## r           0.09996 0.0001131 1.912e-06      1.439e-05
## K          10.02225 0.0271891 4.595e-04      1.172e-03
## loglogsd.N -4.51501 0.1278483 2.161e-03      5.522e-03
##
## 2. Quantiles for each variable:
##
##                 2.5%     25%      50%   75%   97.5%
## r           0.09972  0.0999  0.09994  0.10  0.1002
## K           9.96640 10.0040 10.02337 10.04 10.0737
## loglogsd.N -4.76560 -4.6044 -4.51439 -4.43 -4.2686
```

```r
#plot(N_obs$time,N_obs$N_noisy)

posterior.mean.sim <- solve_de(logistic_model, params=c(r=mean(mcmc_samples$samps$r[burnin:iter]),K=mea
plot(posterior.mean.sim)
points(N_obs$time,N_obs$N_noisy)
```



Posterior credible interval (equal-tailed)

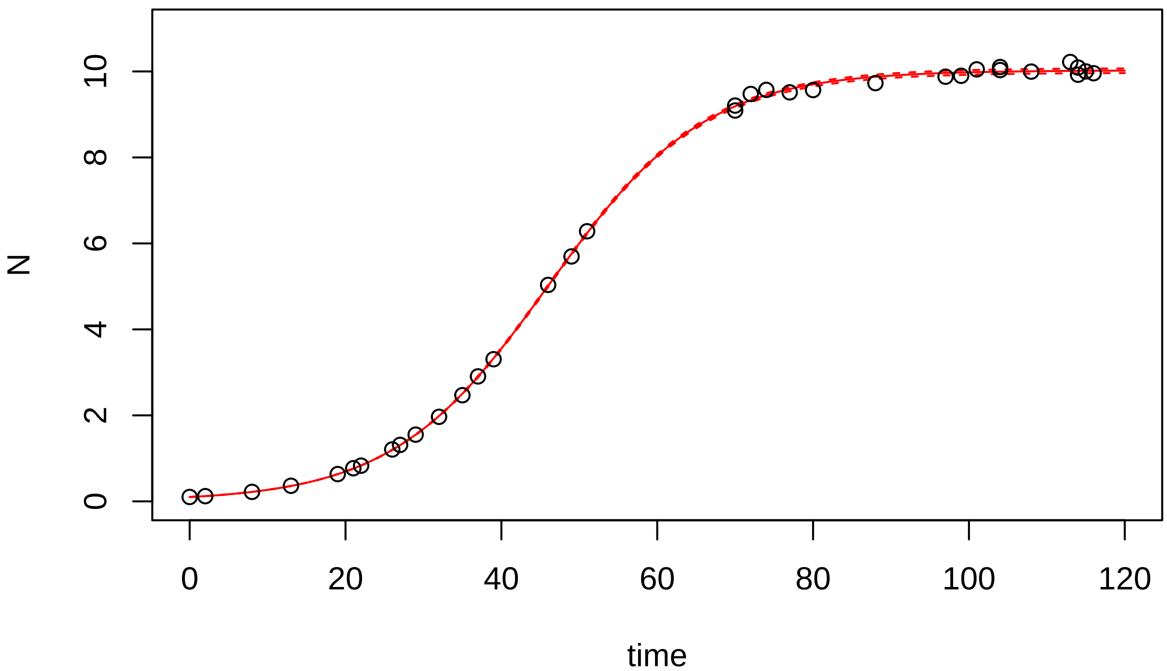
```
library(plyr)
simlist <- llply(sample(nrow(mcmc_samples$samps[burnin:iter,]), 1000),
                 function(x)solve_de(logistic_model, mcmc_samples$samps[burnin:iter,][x,],
                                    inits=c(N = 0.1), Tmax=120, numsteps=100))

sima <- simplify2array(simlist)
dim(sima)
```

```
## [1]  100    2 1000
```

```r
simCI <- aaply(sima, .margins = c(1,2), quantile, probs=c(0.025,0.5,0.975))
for (p in 2:ncol(simlist[[1]])){
  plot(simCI[,,3][,c(1,p)], main="", type='n', ylim=c(0,11))

  #for (i in 1:length(simlist)){
  #  lines(simlist[[i]][,c(1,p)],lty=2,col='grey')
  #}
  for (i in 1:3){
    lines(simCI[,,i][,c(1,p)], lty=c(2,1,2)[i], col=c("red"))
  }
  if (dimnames(simlist[[1]])[[2]][p] == "N") points(N_obs$time,N_obs$N_noisy)
}
```




```r
#str(out)
#lines(posterior.mean.sim, ylim=c(0,15))

#l_ply(simlist, function(x)  lines(x[,1], x[,2]))
#lines(out[,1],out[,2], col='blue')
```