Theory-driven analysis for ecological data: Confronting population models with time series data

Benjamin Rosenbaum
benjamin.rosenbaum@idiv.de

17.05.2022

Contents

We start by loading the packages `rstan` for fitting and `coda` for plotting Bayesian model output. A random seed is set for reproducability of the stochastic simulations.

```
rm(list=ls())
library("rstan")
library("coda")
set.seed(100) # random seed
```

Modeling

We simulate a time series with known parametrization (r, α). The Ricker model for a single population

$$Z_{i+1} = Z_i \cdot e^{r(1-\alpha Z_i)}$$

is coded featuring environmental and demographic stochasticity (process noise)

$$Z_{i+1} \sim \text{Poisson}\left(Z_i \cdot e^{r(1-\alpha Z_i)+\epsilon_i}\right), \epsilon_i \sim \text{Normal}(0, \sigma).$$

We simulate a measurement process, where only a fraction $p \in [0, 1]$ of the total volume or area is observed, where individuals are counted. This is realised by a Binomial distribution, and upscaling the estimate to the total volume again.

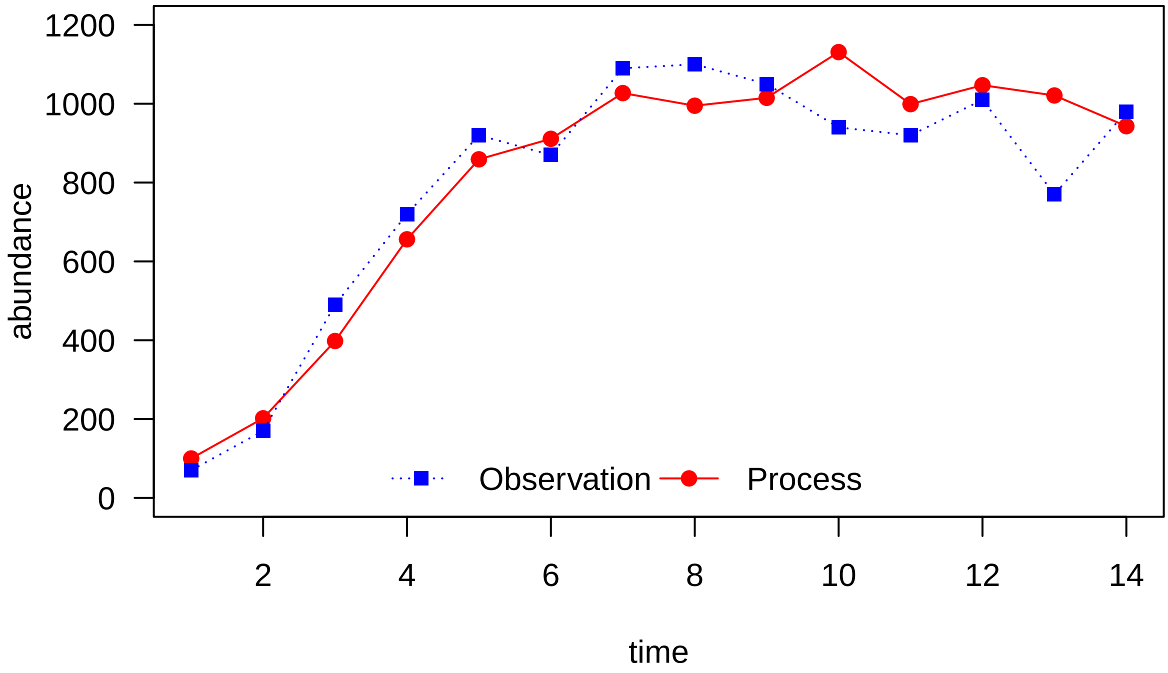
$$Y_i \sim \frac{1}{p} \cdot \text{Binomial}(Z_i, p).$$

The resulting time series $Y_1, \ldots, Y_T$ then features variance that scales with Z^2 (environmental noise ϵ) and with Z (demographic noise and observation error).

```
TT = 14 # total time: 14 days
Z = numeric(TT) # process time series

s_proc = 0.05 # process error standard deviation for environmental noise

r = 0.8 # growth rate
K = 1000 # carrying capacity
```

```
alpha = 1/K # competition

# process equation: Ricker model
Z[1]=100
for(i in 1:(TT-1)){
  Z[i+1] = Z[i]*exp(r*(1-alpha*Z[i]))       # deterministic prediction
  Z[i+1] = Z[i+1]*exp(rnorm(1, 0, s_proc)) # environmental noise
  Z[i+1] = rpois(1, Z[i+1])                # demographic noise
}

# observation: count abundances in fraction p of total volume
p = 0.1
Y = rbinom(TT,round(Z),rep(p,TT))/p
```

```
plot(1:TT, Z,
     pch = 19, col="red", ty = "o",
     xlab = "time", ylab = "abundance",
     ylim=c(0,1.2*K), las=1)
points(1:TT, Y,
       pch=15, col="blue", ty="o", lty = 3)
legend("bottom",
       legend = c("Observation", "Process"),
       pch = c(15, 19),
       col = c("blue", "red"),
       lty = c(3, 1),
       horiz=TRUE, bty="n")
```



Fitting obs error only

The model is coded as a textfile for Stan. As usual, the blocks `data`, `parameters` and `model` are used. When using the initial abundance as an additional parameter `U1`, the complete trajectory `U[i]` can be computed for given model parameters `r` and `alpha`. We assume a lognormal distribution of the observations `Y` around predictions `U` with a standard deviation parameter `s_obs`.

```
code_obs = "
data {
```

```
  int TT;
  int Y[TT];
}

parameters {
  real<lower=0> r;
  real<lower=0> alpha;
  real<lower=0> s_obs;
  real<lower=0> U1;
}

model {
  vector[TT] U; // prediction variable

  // weak priors
  r ~ exponential(1);
  alpha ~ exponential(1);
  s_obs ~ normal(0, 1);
  U1 ~ normal(100, 10);

  // predictions
  U[1] = U1;
  for(t in 2:TT){
    U[t] = U[t-1]*exp(r*(1-alpha*U[t-1]));
  }

  // observation error
  for(t in 1:TT){
    Y[t] ~ lognormal(log(U[t]), s_obs);
  }
}
"
```

For Stan, the data has to be coded as a list. Initial parameter guesses for MCMC sampling can be provided.

```
data = list(Y=Y, TT=TT)

inits = list(r=0.5,
             alpha=1e-3,
             s_obs=0.1,
             U1=data$Y[1])
```

```
fit_obs = stan(model_code = code_obs,
               data = data,
               chains = 3,
               cores = 3,
               init = rep(list(inits),3),
               iter = 4000)
```

Model output looks good.

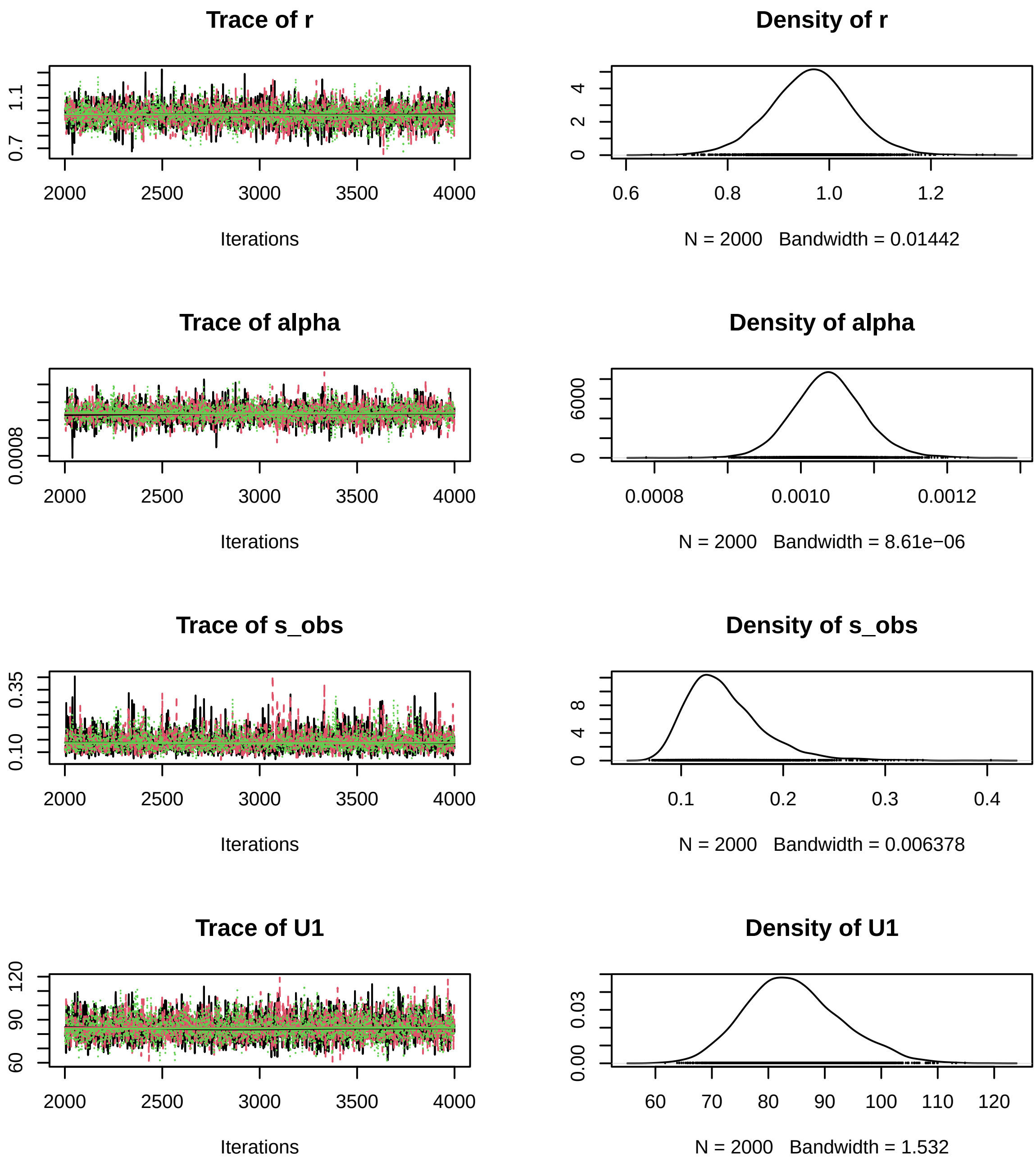
```
print(fit_obs)
```

```
## Inference for Stan model: 229e1222727d93af0cb60fab8e8695a6.
## 3 chains, each with iter=4000; warmup=2000; thin=1;
## post-warmup draws per chain=2000, total post-warmup draws=6000.
##
##          mean se_mean   sd  2.5%   25%   50%   75%  97.5% n_eff Rhat
## r        0.96    0.00 0.08  0.80  0.91  0.97  1.02   1.12  3499    1
```

```
## alpha   0.00    0.00 0.00  0.00  0.00  0.00  0.00   0.00  3846    1
## s_obs   0.14    0.00 0.04  0.09  0.12  0.14  0.16   0.24  2906    1
## U1     84.47    0.15 8.44 69.54 78.63 83.88 89.66 102.28  3277    1
## lp__   14.23    0.03 1.54 10.38 13.43 14.59 15.38  16.22  2077    1
##
## Samples were drawn using NUTS(diag_e) at Fri May 13 10:22:00 2022.
## For each parameter, n_eff is a crude measure of effective sample size,
## and Rhat is the potential scale reduction factor on split chains (at
## convergence, Rhat=1).
```

Posterior samples are transformed for plotting the model output from the `coda` package.

```
samples = As.mcmc.list(fit_obs)
plot(samples[, 1:4])
```



The competition coefficient `alpha=0.001` is estimated fairly accurate, but the growth rate `r=0.8` is overestimated in presence of process and observation error in the data.

Fitting proc error only

Here, we predict `U[i]` one-step-ahead, directly from the previous observation `Y[i-1]`, given the model parameters `r` and `alpha`. We assume a lognormal distribution of the obseravtions `Y` around these piecewise predictions `U` with a standard deviation parameter `s_proc`.

```
code_proc = "
data {
  int TT;
  int Y[TT];
}

parameters {
  real<lower=0> r;
  real<lower=0> alpha;
  real<lower=0> s_proc;
}

model {
  vector[TT] U; // prediction variable

  // weak priors
  r ~ exponential(1);
  alpha ~ exponential(1);
  s_proc ~ normal(0, 1);

  // predictions
  for(t in 2:TT){
    U[t] = Y[t-1]*exp(r*(1-alpha*Y[t-1]));
  }

  // process error
  for(t in 2:TT){
    Y[t] ~ lognormal(log(U[t]), s_proc);
  }
}
"
```

For Stan, the data has to be coded as a list. Initial parameter guesses for MCMC sampling can be provided.

```
data = list(Y=Y, TT=TT)

inits = list(r=0.5,
             alpha=1e-3,
             s_proc=0.1)
```

```
fit_proc = stan(model_code = code_proc,
                data = data,
                chains = 3,
                cores = 3,
                init = rep(list(inits),3),
                iter = 4000)
```

Model output looks good.

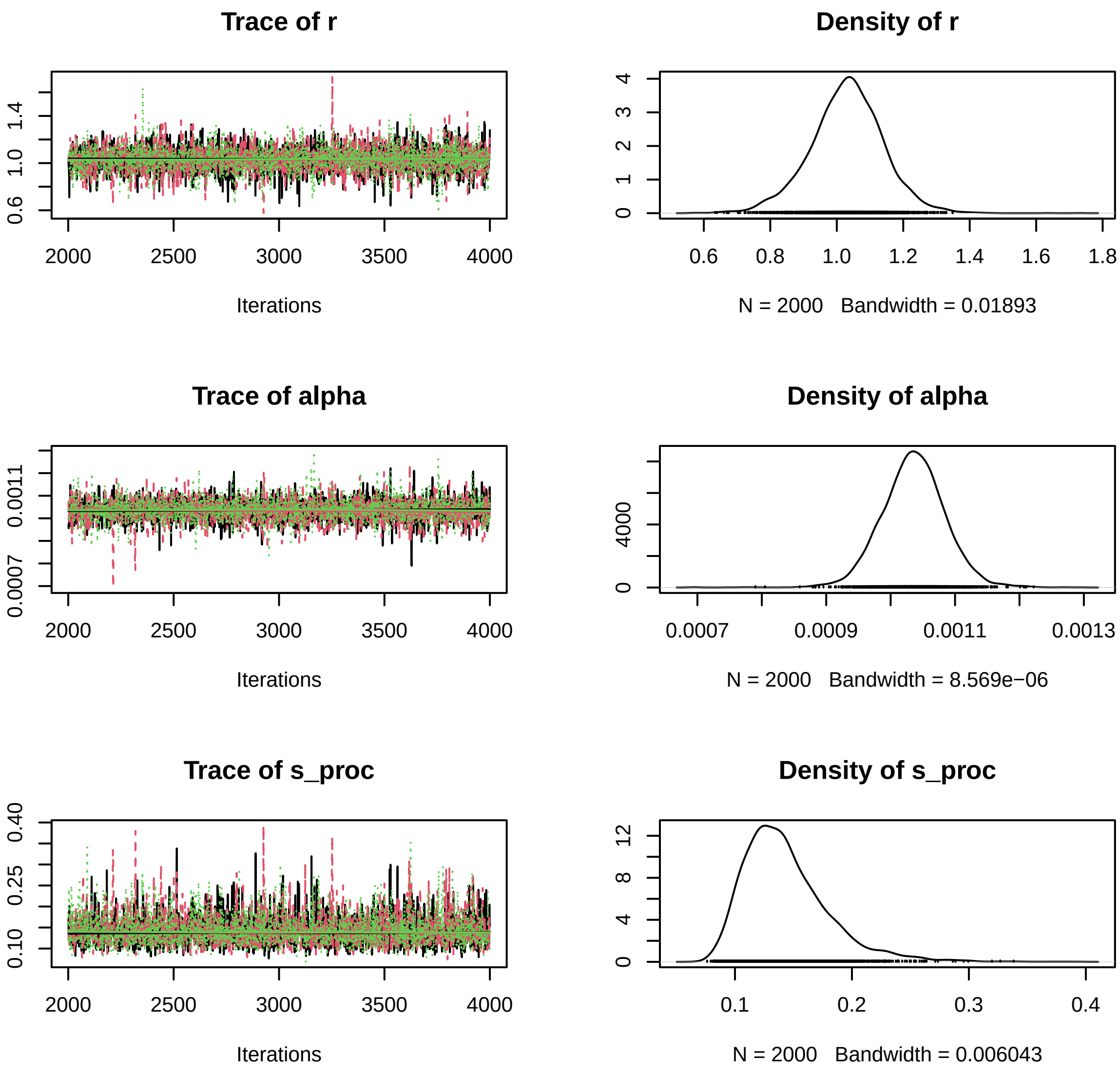
```
print(fit_proc)
```

```
## Inference for Stan model: 7557303d8a7034455a496c0b9dac6868.
## 3 chains, each with iter=4000; warmup=2000; thin=1;
## post-warmup draws per chain=2000, total post-warmup draws=6000.
```

```
##
##         mean se_mean   sd 2.5%  25%   50%   75% 97.5% n_eff Rhat
## r       1.03    0.00 0.11 0.80 0.96  1.03  1.10  1.24  3445    1
## alpha   0.00    0.00 0.00 0.00 0.00  0.00  0.00  0.00  3860    1
## s_proc  0.14    0.00 0.04 0.09 0.12  0.14  0.16  0.23  2905    1
## lp__    9.71    0.03 1.44 6.08 9.04 10.09 10.76 11.35  1845    1
##
## Samples were drawn using NUTS(diag_e) at Fri May 13 10:24:45 2022.
## For each parameter, n_eff is a crude measure of effective sample size,
## and Rhat is the potential scale reduction factor on split chains (at
## convergence, Rhat=1).
```

Posterior samples are transformed for plotting the model output from the `coda` package.

```
samples = As.mcmc.list(fit_proc)
plot(samples[, 1:3])
```



Again, growth rate `r=0.8` is overestimated in presence of process and observation error in the data.

Fitting a state-space model

If we want to to account for both process and observation error, estimated true states `Z[i]` have to be modeled as latent parameters. This time series features process error when compared to the one-step-ahead predictions `U[i]`, for which we choose a lognormal distribution with standard deviation `s_proc`. Also, observed states `Y[i]` are assumed to have observation error with a lognormal distribution with standard deviation `s_obs`.

```
code_ssm = "
data {
  int TT;
  int Y[TT];
}

parameters {
  real<lower=0> r;
  real<lower=0> alpha;
  real<lower=0> s_proc;
  real<lower=0> s_obs;
  vector[TT] Z;
}

model {
  vector[TT] U; // prediction variable

  // weak priors
  r ~ exponential(1);
  alpha ~ exponential(1);
  s_proc ~ normal(0, 1);
  s_obs ~ normal(0, 1);
  Z[1] ~ normal(100, 10);

  // predictions and process error
  for(t in 2:TT){
    U[t] = Z[t-1]*exp(r*(1-alpha*Z[t-1]));
    Z[t] ~ lognormal(log(U[t]), s_proc);
  }

  // observation error
  for(t in 1:TT){
    Y[t] ~ lognormal(log(Z[t]), s_obs);
  }
}
"
```

For Stan, the data has to be coded as a list. Initial parameter guesses for MCMC sampling can be provided. Here, we tweak a parameter of the MCMC algorithm to help with convergence. The default value for `adapt_delta` is `0.8` and we increase it to `0.99`.

```
data = list(Y=Y, TT=TT)

inits = list(r=0.5,
             alpha=1e-3,
             s_proc=0.1,
             s_obs=0.1,
             Z=data$Y)
```

```
fit_ssm <- stan(model_code = code_ssm,
                data = data,
                chains = 3,
                cores = 3,
                init = rep(list(inits),3),
                iter = 4000,
                control = list(adapt_delta=0.99))
```

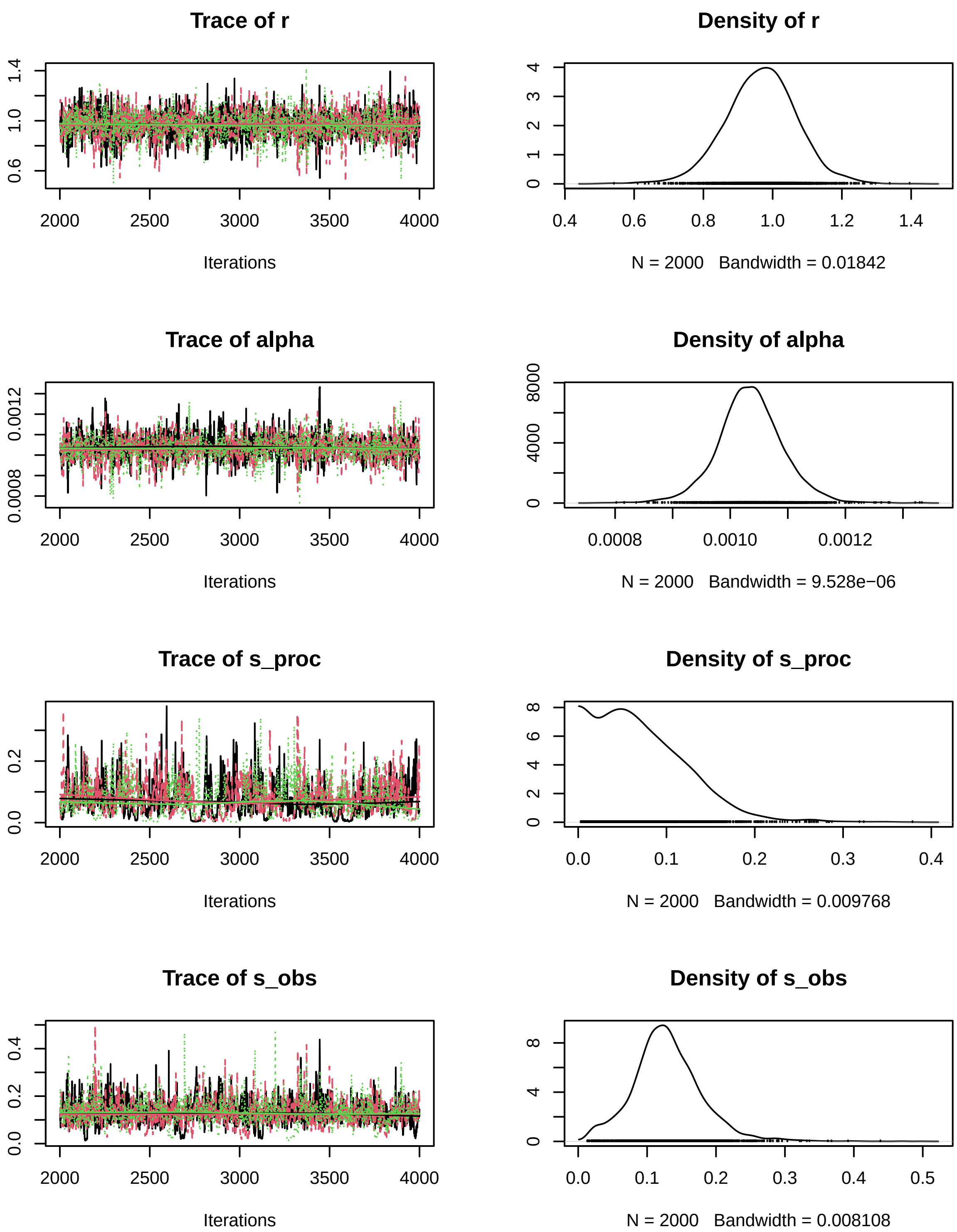
Model output looks OK.

```
print(fit_ssm)
```

```
## Inference for Stan model: 5b7a83e278793c6834409c6ebaa252d7.
## 3 chains, each with iter=4000; warmup=2000; thin=1;
## post-warmup draws per chain=2000, total post-warmup draws=6000.
##
##            mean se_mean    sd   2.5%    25%    50%     75%   97.5% n_eff Rhat
## r          0.97    0.00  0.10   0.76   0.90   0.97    1.03    1.17  1884 1.00
## alpha      0.00    0.00  0.00   0.00   0.00   0.00    0.00    0.00  1581 1.00
## s_proc     0.07    0.00  0.05   0.00   0.03   0.07    0.11    0.19   238 1.01
## s_obs      0.13    0.00  0.05   0.03   0.10   0.13    0.16    0.24   545 1.00
## Z[1]      83.05    0.28  9.11  68.38  76.19  82.30   88.94  102.85  1078 1.00
## Z[2]     196.27    0.64 18.95 164.46 183.17 194.63  207.32  238.03   889 1.00
## Z[3]     445.18    1.08 46.03 361.37 414.53 443.18  473.95  540.70  1818 1.00
## Z[4]     735.80    1.42 69.60 595.99 693.66 735.37  775.97  878.97  2400 1.00
## Z[5]     918.57    1.25 68.86 783.81 876.89 919.28  959.05 1057.56  3032 1.00
## Z[6]     935.40    2.00 72.42 797.29 890.86 935.40  979.12 1079.87  1307 1.00
## Z[7]     999.50    2.72 80.14 857.83 945.50 992.64 1047.01 1174.71   869 1.00
## Z[8]    1004.54    3.08 82.21 862.31 949.47 998.32 1053.23 1178.60   714 1.00
## Z[9]     992.57    2.13 77.39 854.23 942.35 988.33 1036.79 1155.98  1317 1.00
## Z[10]    959.89    1.42 70.57 823.75 916.69 956.88  999.86 1113.37  2463 1.00
## Z[11]    953.32    1.54 70.12 821.32 909.94 951.28  992.90 1095.17  2087 1.00
## Z[12]    980.48    1.54 71.25 844.84 935.92 979.22 1020.46 1130.62  2135 1.00
## Z[13]    910.14    3.89 86.17 746.93 849.83 916.34  969.43 1071.06   490 1.01
## Z[14]    969.55    1.46 69.06 838.28 927.37 968.08 1009.04 1115.27  2248 1.00
## lp__     -45.52    0.97 11.14 -62.76 -52.60 -47.61  -40.87  -16.99   133 1.01
##
## Samples were drawn using NUTS(diag_e) at Fri May 13 13:21:19 2022.
## For each parameter, n_eff is a crude measure of effective sample size,
## and Rhat is the potential scale reduction factor on split chains (at
## convergence, Rhat=1).
```

Posterior samples are transformed for plotting the model output from the `coda` package.

```
samples = As.mcmc.list(fit_ssm)
plot(samples[, 1:4])
```



Even with the SSM, growth rate `r=0.8` is overestimated. It looks like this single timeseries does not contain enough information to estimate the growth rate accurately. With more replications, a higher temporal sampling frequency, or a higher sampling effort (sampling more than just 10% of the volume), more accurate estimates should be possible.

Estimated true states `Z[i]` and their credible intervals can be extracted from the model output.

```
Z_med = summary(fit_ssm)$summary[5:18, "50%"]
Z_lbd = summary(fit_ssm)$summary[5:18, "2.5%"]
Z_ubd = summary(fit_ssm)$summary[5:18, "97.5%"]

plot(1:TT, Z, type="n",
```

```
     xlab = "time", ylab = "abundance",
     ylim=c(0,1.2*K), las=1)
polygon(c(1:TT,TT:1),c(Z_lbd,rev(Z_ubd)), col="lightgrey" , border=NA)
lines(1:TT, Z, pch = 19, col="red", ty = "o")
lines(1:TT, Z_med)
points(1:TT, Y,
       pch=15, col="blue", ty="o", lty = 3)
legend("bottomright",
       legend = c("Observation", "Process","Latent states"),
       pch = c(15, 19, NA),
       col = c("blue", "red", "black"),
       lty = c(3, 1, 1),
       horiz=TRUE, bty="n")
```