ICES

Estimating uncertainties with AD Model Builder

Anders Nielsen & Arni Magnusson

Important to supply uncertainty estimates?

- I can think of three reasons people give for not to presenting uncertainty estimates:
  1. Estimates are so precise that we don't need to worry about uncertainties
  2. Estimates are so uncertain that we prefer not to show uncertainties
  3. We don't know how to estimate the uncertainties, only the estimates themselves.
- The uncertainty estimates are sort of needed in the first case too, and in the last two cases the estimates themselves should be skeptically evaluated.

AD Model Builder offers three approaches

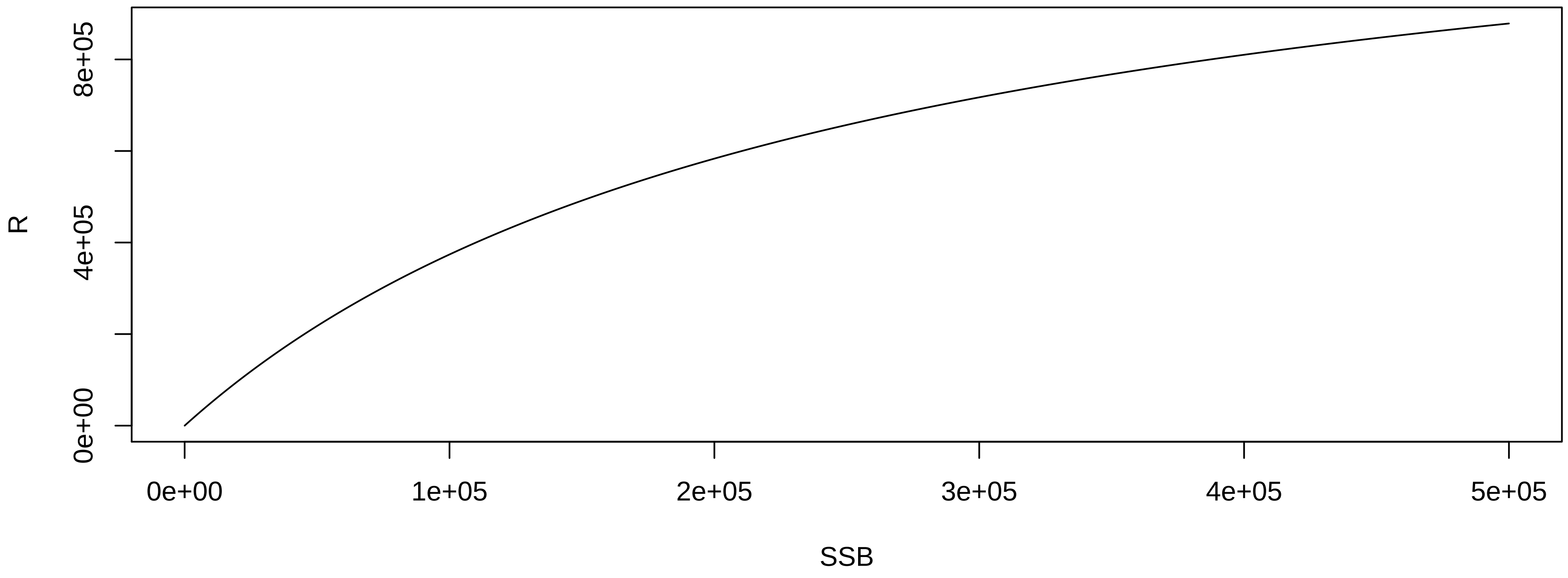
- Hessian based
- Profile likelihood
- MCMC

The Beverton-Holt stock recruitment example

- The Beverton-Holt model can be written (slightly re-parametrized) as:

$$\log R = \log(a) + \log(\mathrm{ssb}) - \log(1 + \exp(\log(b))\mathrm{ssb})$$

- From a number of observed pairs $\{\mathrm{ssb}_i, \log R_i\}$ $i = 1 \ldots n$ we want to estimate the model parameters $\log(a)$ and $\log(b)$



Maximum likelihood estimator and Hessian

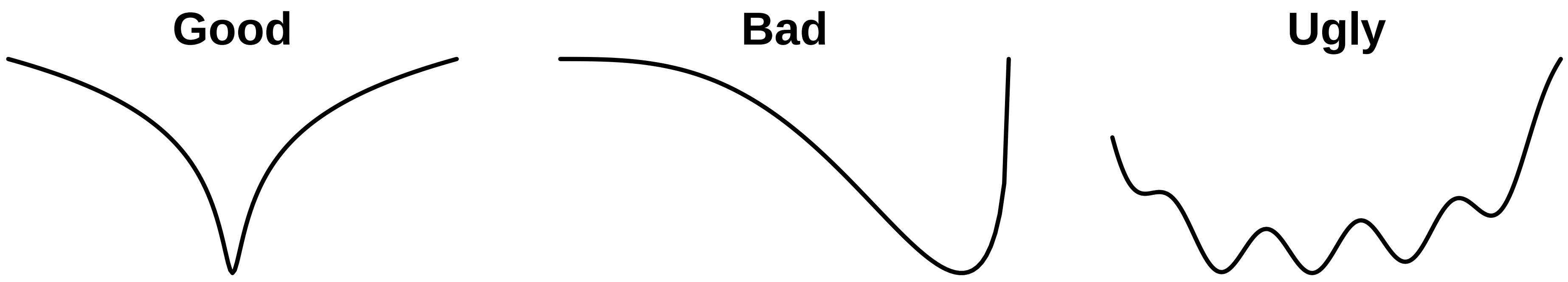
- A sensible estimate of the model parameters is to choose the values that maximize the likelihood for the actual observations.

$$\widehat{\theta} = \underset{\theta}{\operatorname{argmin}}\, \ell(y|\theta)$$

- The curvature of the negative log likelihood function gives an estimate of the variance of the maximum likelihood estimator:

$$\widehat{\operatorname{var}(\widehat{\theta})} = \left(\left. \frac{\partial^2 \ell(y|\theta)}{\partial \theta^2} \right|_{\theta=\widehat{\theta}} \right)^{-1}$$

- The matrix $\mathcal{H}(\widehat{\theta}) = \left(\left. \frac{\partial^2 \ell(y|\theta)}{\partial \theta^2} \right|_{\theta=\widehat{\theta}} \right)$ is often referred to as “the hessian matrix”
- Both the estimator and the hessian matrix are often found by numerical methods.



The delta method

- Let's be willing to assume:

$$\theta \sim N(\hat{\theta}, \Sigma)$$

- Are interested in a quantity, which is a non-linear function of θ:

$$Q = f(\theta)$$

- The linear approximation is:

$$Q \sim N\left(f(\hat{\theta}), \nabla f(\hat{\theta})^T \Sigma \nabla f(\hat{\theta})\right)$$

Likelihood for Beverton-Holt

- Here we will model the observations as:

$$\log R_i = \underbrace{\log(a) + \log(\mathrm{ssb}_i) - \log(1 + \exp(\log(b))\mathrm{ssb}_i)}_{\mu_i} + \varepsilon_i \ ,$$

  where $\varepsilon_i \sim N(0, \sigma^2)$ independent.

- or in other words:

$$\log R_i \sim N(\mu_i, \sigma^2) \ \text{, independent.}$$

- The negative log likelihood becomes:

$$\ell(\log(R_i) | \log(a), \log(b), \sigma^2) = \frac{n}{2} \log(2\pi\sigma^2) + \frac{1}{2\sigma^2} \sum_{i=1}^{n} (\log(R)_i - \mu_i)^2$$

Beverton-Holt in AD Model Builder

```
DATA_SECTION
  init_int nR
  init_int nC
  init_matrix obs(1,nR,1,nC)
  vector ssb(1,nR)
  !! ssb=column(obs,1);
  vector logR(1,nR)
  !! logR=column(obs,2);
PARAMETER_SECTION
  init_number loga;
  init_number logb;
  init_number logSigma;
  sdreport_number sigmaSq;
  vector pred(1,nR);
  objective_function_value nll;
PROCEDURE_SECTION
  sigmaSq=exp(2.0*logSigma);
  pred=loga+log(ssb)-log(1+exp(logb)*ssb);
  nll=0.5*(nR*log(2*M_PI*sigmaSq)+sum(square(logR-pred))/sigmaSq);
```

```
The logarithm of the determinant of the hessian = 12.7374
index   name        value       std.dev        1        2        3        4
    1   loga       1.8684e+00 1.3963e-01   1.0000
    2   logb      -1.2055e+01 3.3381e-01   0.9357   1.0000
    3   logSigma  -1.0963e+00 1.0426e-01  -0.0000   0.0000   1.0000
    4   sigmaSq    1.1162e-01 2.3275e-02  -0.0000   0.0000   1.0000   1.0000
```

Easy way to read this all into R

```
read.fit<-function(file){
  #
  # Function to read a basic AD Model Builder fit.
  #
  # Use for instance by:
  #
  #   simple.fit <- read.fit('c:/admb/examples/simple')
  #
  # Then the object 'simple.fit' is a list containing sub-objects
  # 'names', 'est', 'std', 'cor', and 'cov' for all model
  # parameters and sdreport quantities.
  #
  ret<-list()
  parfile<-as.numeric(scan(paste(file,'.par', sep=''),
                    what='', n=16, quiet=TRUE)[c(6,11,16)])
  ret$nopar<-as.integer(parfile[1])
  ret$nlogl<-parfile[2]
  ret$maxgrad<-parfile[3]
  file<-paste(file,'.cor', sep='')
  lin<-readLines(file)
  ret$npar<-length(lin)-2
  ret$logDetHess<-as.numeric(strsplit(lin[1], '=')[[1]][2])
  sublin<-lapply(strsplit(lin[1:ret$npar+2], ' '),function(x)x[x!=''])
  ret$names<-unlist(lapply(sublin,function(x)x[2]))
  ret$est<-as.numeric(unlist(lapply(sublin,function(x)x[3])))
  ret$std<-as.numeric(unlist(lapply(sublin,function(x)x[4])))
  ret$cor<-matrix(NA, ret$npar, ret$npar)
  corvec<-unlist(sapply(1:length(sublin), function(i)sublin[[i]][5:(4+i)]))
  ret$cor[upper.tri(ret$cor, diag=TRUE)]<-as.numeric(corvec)
  ret$cor[lower.tri(ret$cor)] <- t(ret$cor)[lower.tri(ret$cor)]
  ret$cov<-ret$cor*(ret$std%o%ret$std)
  return(ret)
}
```

Things to remember

- This is just based on a normal approximation, so it is not always appropriate. Also we may want to compute our C.I. on log scale and then transfer the quantiles the original scale to ensure that a parameter is positive.
- Variables other than the actual model parameters can be added to the output reports by declaring them as `sdreport_` in the parameter section (see e.g. sigmaSq)
- The three usual output files are called `*.par`, `*.std`, and `*.cor`

Profile likelihood

- Profile likelihood for a given parameter is easily accessible in AD Model Builder

```
DATA_SECTION
  init_int nR
  init_int nC
  init_matrix obs(1,nR,1,nC)
  vector ssb(1,nR)
  !! ssb=column(obs,1);
  vector logR(1,nR)
  !! logR=column(obs,2);
PARAMETER_SECTION
  init_number loga;
  init_number logb;
  init_number logSigma;
  sdreport_number sigmaSq;
  vector pred(1,nR);
  likeprof_number loga_pl       // NOTICE
  likeprof_number logb_pl       // NOTICE
  objective_function_value nll;
PROCEDURE_SECTION
  loga_pl=loga;                 //NOTICE
  logb_pl=logb;                 //NOTICE
  sigmaSq=exp(2.0*logSigma);
  pred=loga+log(ssb)-log(1+exp(logb)*ssb);
  nll=0.5*(nR*log(2*M_PI*sigmaSq)+sum(square(logR-pred))/sigmaSq);
```

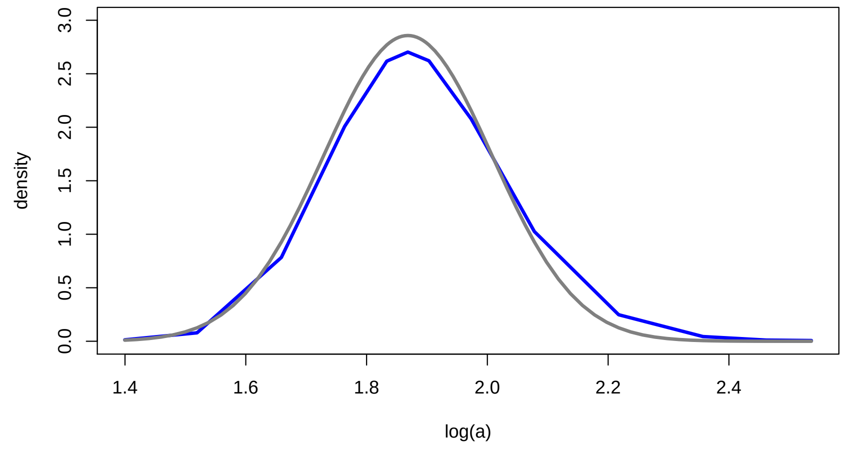
- Running the program with `./modelname -lprof`

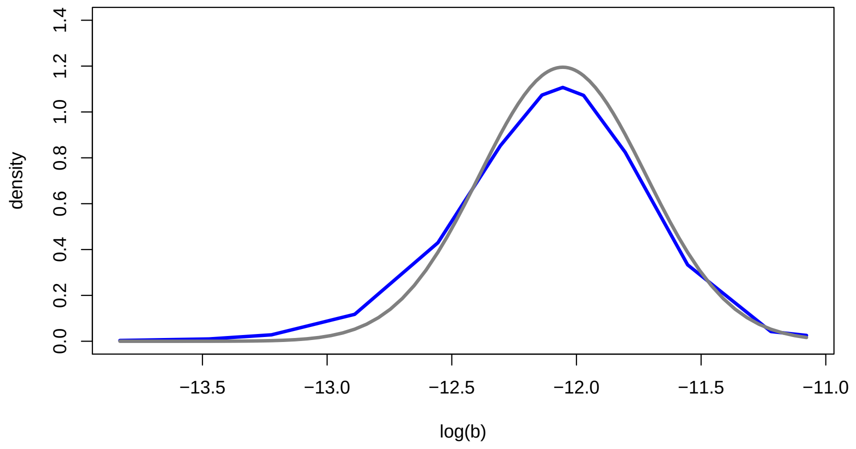
- Will produce two files `loga_pl.plt` and `loga_pl.plt`

Profile likelihood (cont.)

- Each of those files have (x,y) points of the profile density

```
loga_pl:
Profile likelihood
 1.39962 0.013531
 1.41957 0.024403
 1.43951 0.035275
 1.45946 0.046147
 1.47941 0.057019
 1.49935 0.067891
 1.5193 0.078763
 1.53925 0.179453
```

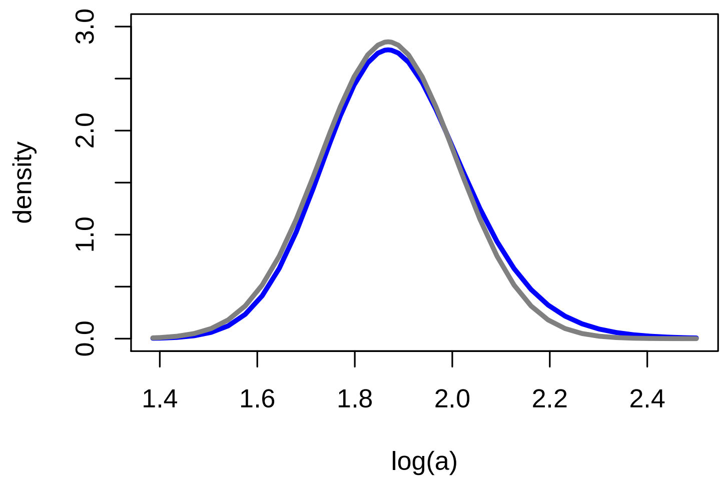


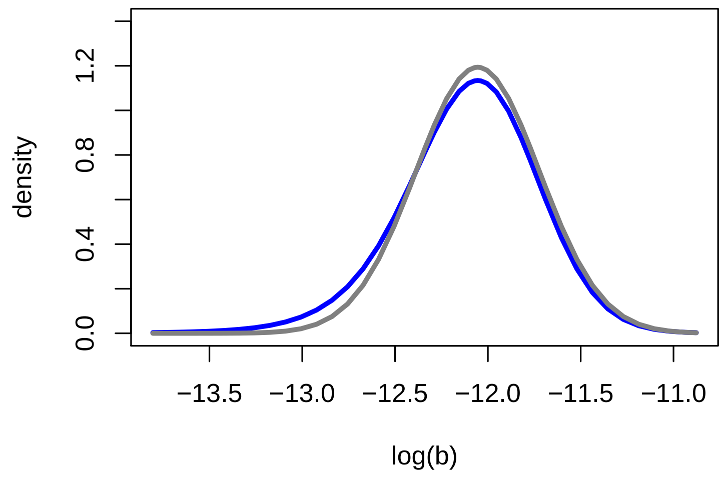


- Can we please get more support points?

Finer resolution

```
DATA_SECTION
  init_int nR
  init_int nC
  init_matrix obs(1,nR,1,nC)
  vector ssb(1,nR)
  !! ssb=column(obs,1);
  vector logR(1,nR)
  !! logR=column(obs,2);
PARAMETER_SECTION
  init_number loga;
  init_number logb;
  init_number logSigma;
  sdreport_number sigmaSq;
  vector pred(1,nR);
  likeprof_number loga_pl       // NOTICE
  likeprof_number logb_pl       // NOTICE
  objective_function_value nll;
PRELIMINARY_CALCS_SECTION       //!
  loga_pl.set_stepnumber(50);   //!
  loga_pl.set_stepsize(0.1);    //!
  logb_pl.set_stepnumber(50);   //!
  logb_pl.set_stepsize(0.1);    //!
PROCEDURE_SECTION
  loga_pl=loga;                 //NOTICE
  logb_pl=logb;                 //NOTICE
  sigmaSq=exp(2.0*logSigma);
  pred=loga+log(ssb)-log(1+exp(logb)*ssb);
  nll=0.5*(nR*log(2*M_PI*sigmaSq)+sum(square(logR-pred))/sigmaSq);
```





Is there an easy way to read the awkward plt files?

```
read.plt<-function(path){
  # Function to read a basic AD Model Builder profile files.
  #
  # Use for instance by:
  #
  #   prof <- read.plt('c:/admb/example/')
  #
  # Then the object 'prof' will be a named list with an entry
  # for each of the profiled variables with a matrix containing
  # the columns 'x', 'prof', and 'normal'.
  #
  path<-sub('/$','',path)
  filelist<-dir(path,pattern='\\.plt$')
  varnames<-gsub('\\.plt$','',filelist)
  toMat<-function(lin){
    lin<-gsub('^[[:blank:]]|[[:blank:]]$','',lin)
    mat<-matrix(as.numeric(unlist(strsplit(lin,split='[[:blank:]]'))), ncol=2, byrow=TRUE)
    return(mat)
  }
  doone<-function(file){
    lin<-readLines(paste(path,file,sep='/'))
    idx1<-grep('^Profile|^Normal',lin)+1
    idx2<-grep('Minimum',lin)-1
    M1<-toMat(lin[idx1[1]:idx2[1]])
    M2<-toMat(lin[idx1[2]:idx2[2]])
    M<-cbind(x=M1[,1],prof=M1[,2],normal=M2[,2])
    return(M)
  }
  ret<-lapply(filelist,doone)
  names(ret)<-varnames
  return(ret)
}
```

What is MCMC and which variant are we using

- Assume we have an unnormalized probability density function $\phi(\theta)$
- MCMC is a collection of methods to simulate a Markov chain $\theta_1, \ldots, \theta_N$ with an equilibrium distribution given by $\phi(\theta)$
- This is probably known to some from WinBUGS
- AD Model Builder uses what is known as a RW-MH (Random Walk Metropolis-Hastings)
- The starting point is $\hat{\theta}$ and the proposal variance is $\widehat{\mathrm{var}(\hat{\theta})}$

Example: The negative binomial

- Assume that these 15 numbers follow a negative binomial distribution:

```
13 5 28 28 15 4 13 4 10 17 11 13 12 17 3
```

- The model is coded as:

```
DATA_SECTION
  int N
  !! N=15;
  init_vector X(1,N)

PARAMETER_SECTION
  init_number logsize;
  init_bounded_number p(0,1);
  sdreport_number size;
  sdreport_number pp;
  objective_function_value nll;

PROCEDURE_SECTION
  size=exp(logsize);
  pp=p;
  nll=-sum(gammln(X+size))+N*gammln(size)+
       sum(gammln(X+1.0))-N*size*log(p)-sum(X)*log(1.0-p);
```

```
index   name     value      std.dev
    1   logsize  1.3017e+00 4.7101e-01
    2   p        2.2218e-01 8.5571e-02
    3   size     3.6754e+00 1.7312e+00
    4   pp       2.2218e-01 8.5571e-02
```

Basic use

- Simply run the model with `-mcmc N`, where N is the number of steps. For instance:

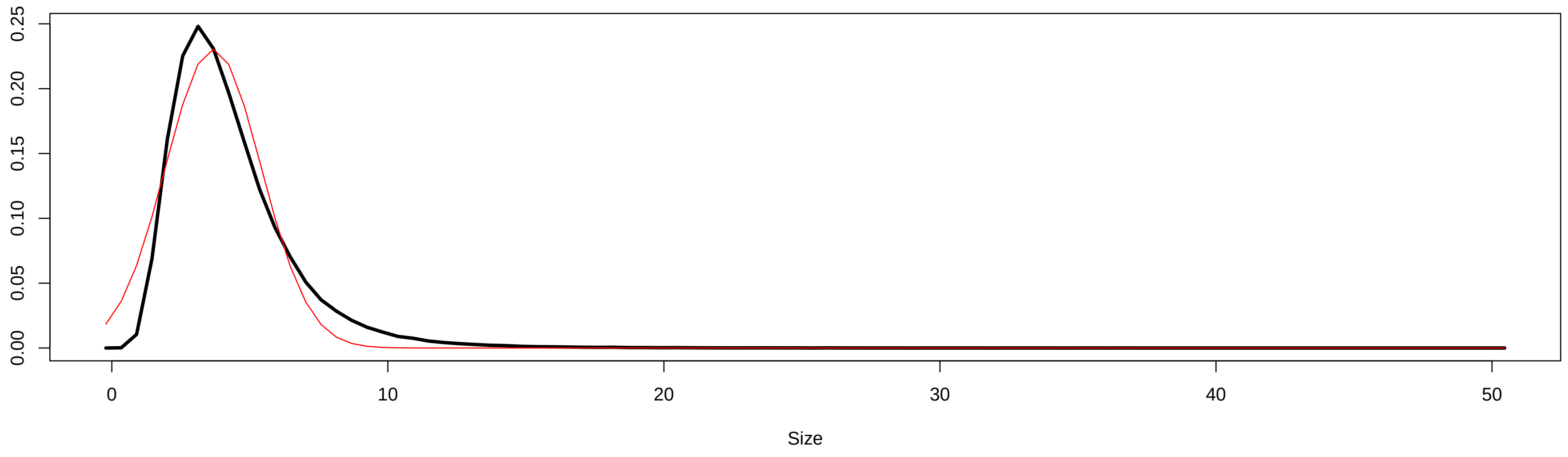
```
an@ch-pcb-an:~$./simplenbin -mcmc 10000
```

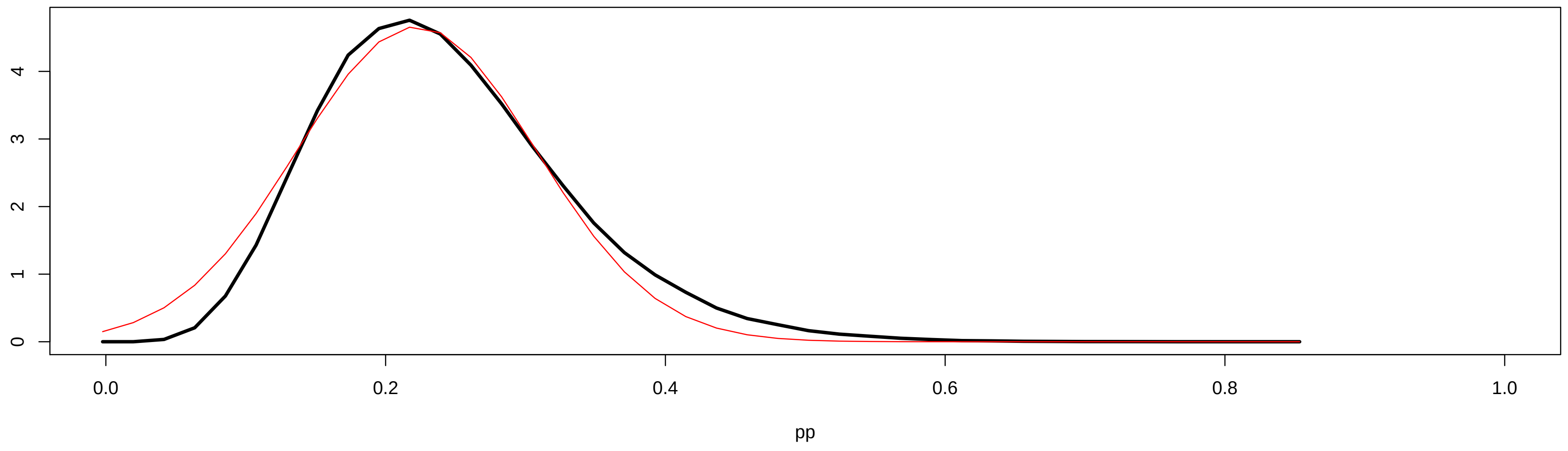
- The file `<modelname>.hst` then contains points on the simulated pdf of all sdreport variables.

```
# samples sizes
10000
# step size scaling factor
1.2
# step sizes
 0.549441 0.0223384
# means
 4.33401 0.242679
# standard devs
 4.39827 0.178819
# lower bounds
 -8 -11
# upper bounds
 34 19
#number of parameters
2
#current parameter values for mcmc restart
 0.826337 0.148048
#random nmber seed
1262173905
```

```
#size
-0.0615169 0
0.487924 0.00127402
1.03736 0.0251165
...
22.4655 0.000182003
23.015 0

#pp
-0.00304368 0
0.0192947 0.00447659
0.0416332 0.0358127
...
0.64477 0.00447659
0.667109 0
```

Useful for plotting





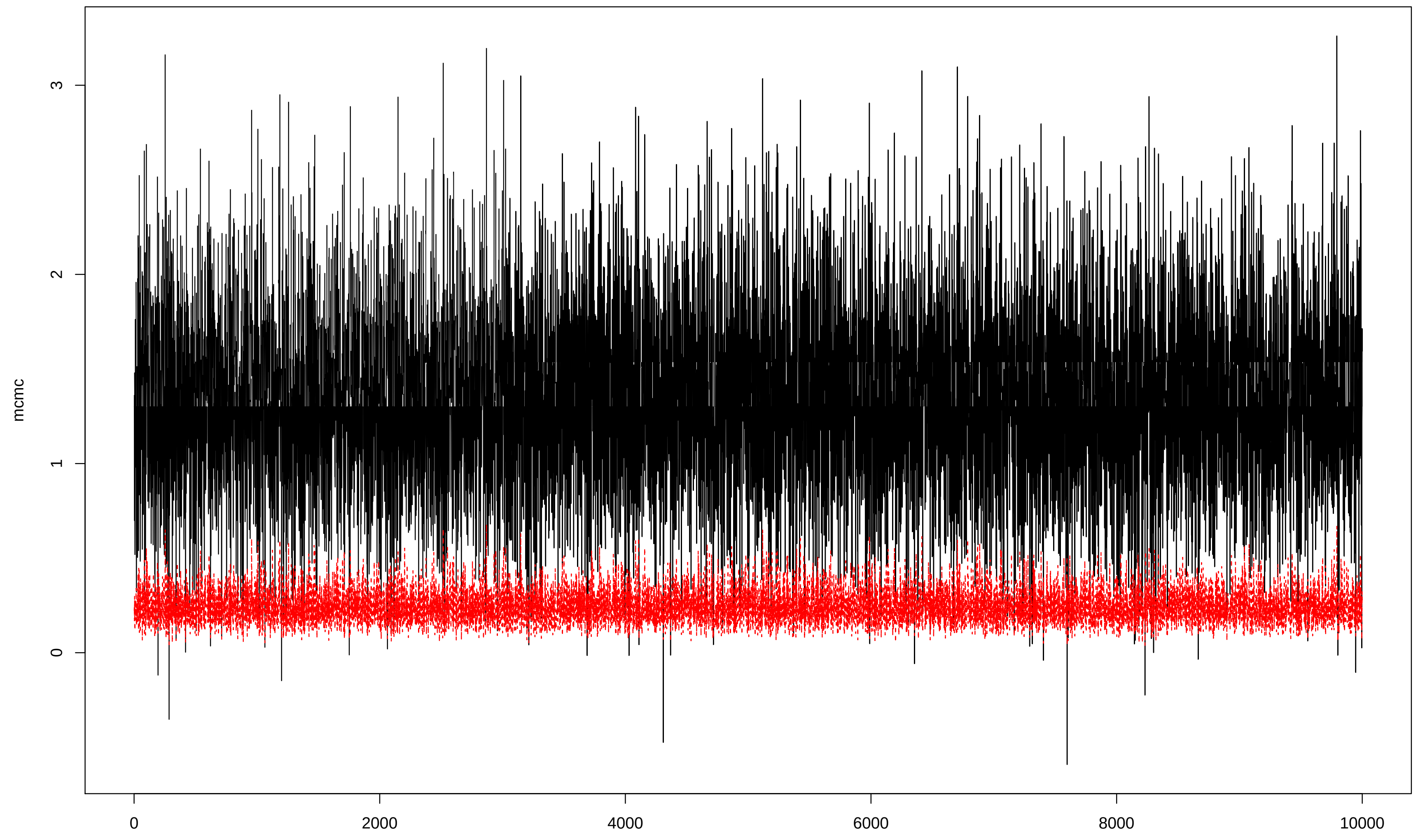
Want to study the chain?

- The chain of parameters (not sdreport variables) can be saved by:

```
an@ch-pcb-an:~$./simplenbin -mcmc 100000 -mcsave 10
```

- here the `-mcsave N` tells it to save every N'th step
- Saves to a binary file `<modelname>.psv`, which can be read into R by:

```
read.mcmc<-function(file){
  #
  # Function to read a basic AD Model Builder binary mcmc file.
  #
  # Use for instance by:
  #
  #   mcmc <- read.mcmc('c:/admb/example/simple')
  #
  # Then the object 'mcmc' will be a matrix with columns for each
  # "init_" model parameter and rows for each saved mcmc step.
  #
  fn<-paste(file,".psv",sep="")
  filen <- file(fn, "rb")
  nopar<-readBin(filen, what=integer(), n=1)
  oversize<-(file.info(fn)$size-4)/8+10
  mcmc<-readBin(filen, what=numeric(),n=oversize)
  close(filen)
  mcmc<-matrix(mcmc,byrow=TRUE, ncol=nopar)
  return(mcmc)
}
```


mcmc
0
1
2
3
0
2000
4000
6000
8000
10000

The chain of custom output

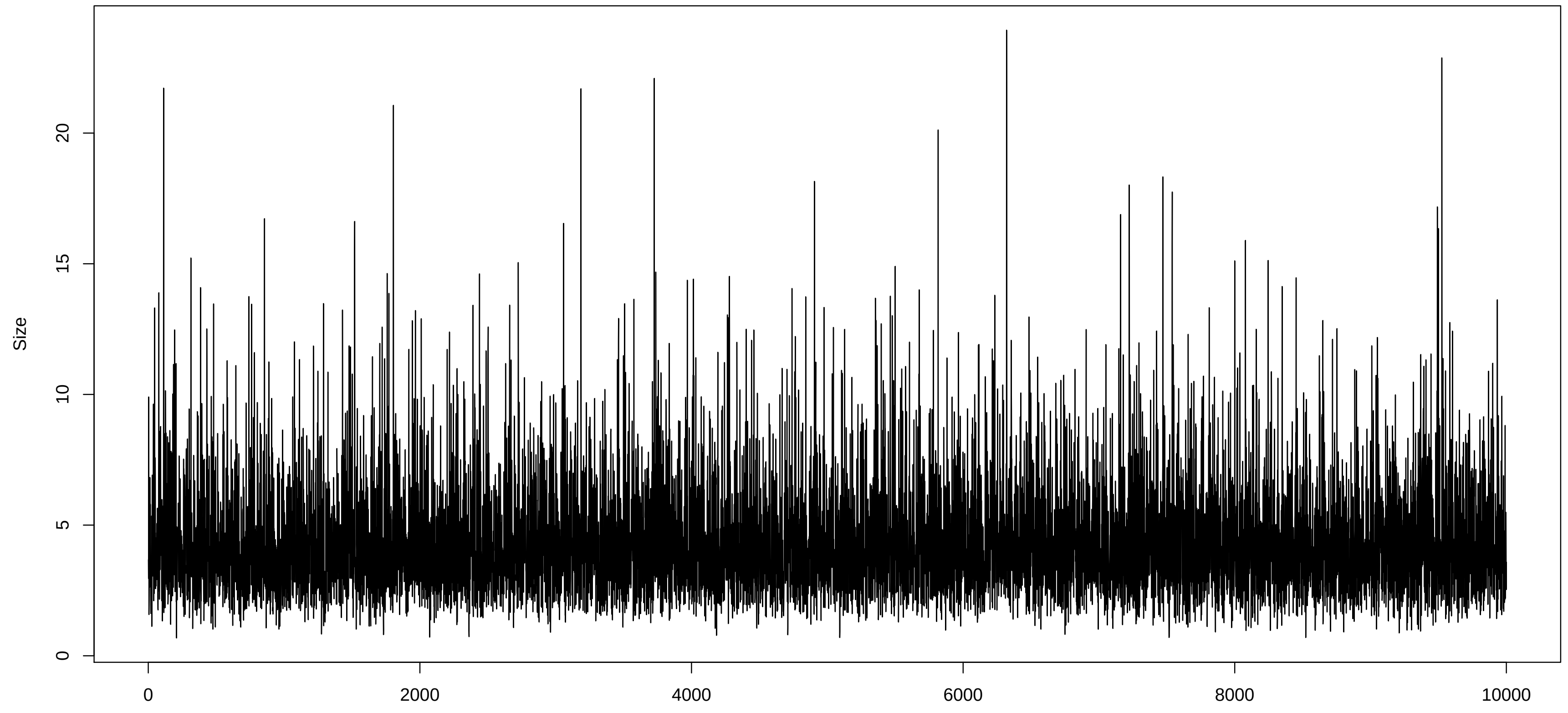
- Suppose we want the output chain of something that is not a model parameter (here ‘size’)
- Then we need to change the code a bit

```
GLOBALS_SECTION
  #include <fvar.hpp>
  ofstream sizeout("size.cha");

DATA_SECTION
  int N
  !! N=15;
  init_vector X(1,N)

PARAMETER_SECTION
  init_number logsize;
  init_bounded_number p(0,1);
  sdreport_number size;
  sdreport_number pp;
  objective_function_value nll;

PROCEDURE_SECTION
  size=exp(logsize);
  pp=p;
  nll=-sum(gammln(X+size))+N*gammln(size)+
       sum(gammln(X+1.0))-N*size*log(p)-sum(X)*log(1.0-p);

  if(mceval_phase()){
    ofstream sizeout("size.cha", ios::app);
    sizeout<<size<<"\n";
  }
```

- To run we must type:

```
an@ch-pcb-an:~$./simplenbin -mcmc 100000 -mcsave 10
an@ch-pcb-an:~$./simplenbin -mceval
```

- And then `size.cha` is produced



Exercise: Beverton-Holt comparing sequential Bayesian and integrated analysis

The results shown in the Beverton-Holt example were based only on the data file `bh.dat`

a) Sequential Bayesian

- For a similar stock in a similar area we have these estimates

```
1   loga      1.8085e+00 1.2725e-01   1.0000
2   logb     -1.2183e+01 3.2431e-01   0.9278  1.0000
3   logSigma -1.1332e+00 1.0426e-01   0       0       1
```

- Modify the program `bh.tpl` to use this as prior information.

b) Integrated analysis

- In the file `bh0.dat` we have the entire data set from the similar stock.
- Modify the program `bh.tpl` to use the both data sets to estimate $\log(a)$ and $\log(b)$.

b) Compare

- Use MCMC and plot the joint distribution of $\log(a)$ and $\log(b)$. Make one plot for the sequential Bayesian approach, and one for the integrated.

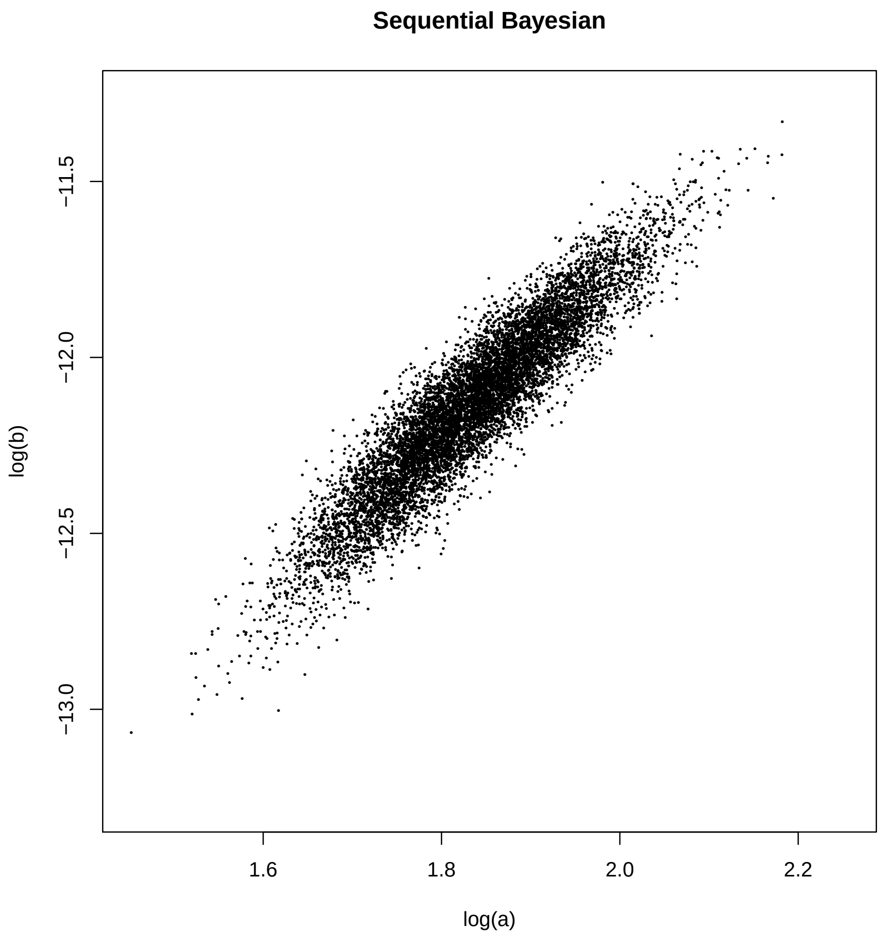
Solution

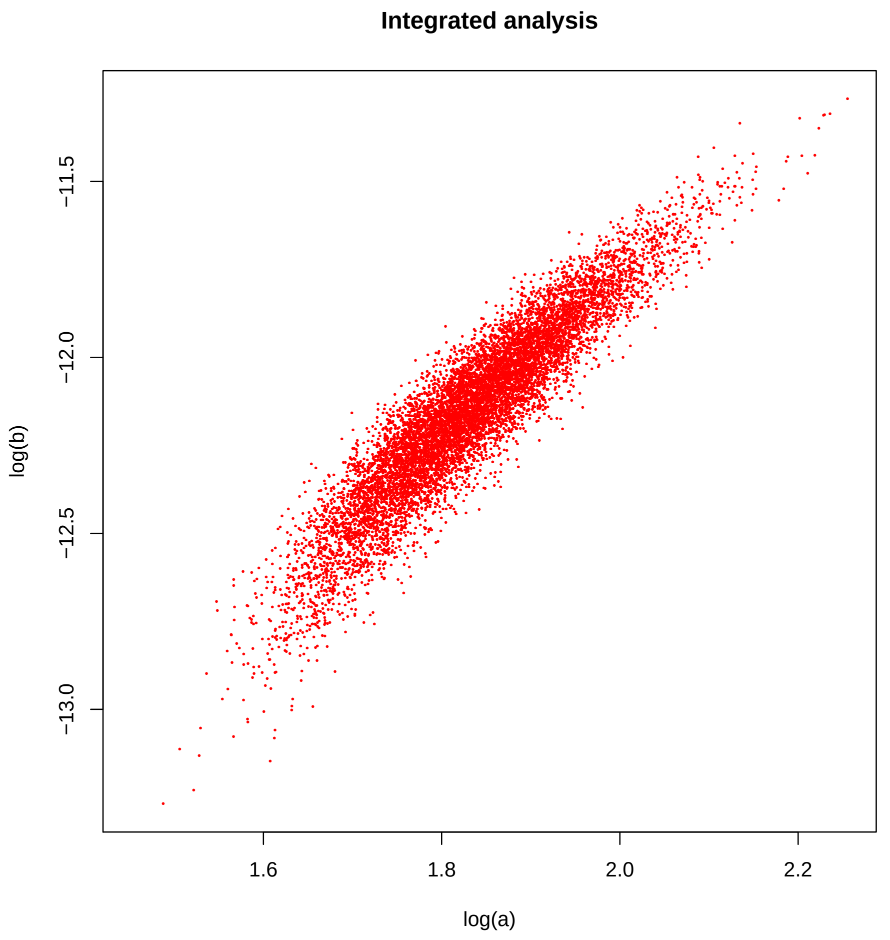
- To include the estimates and their estimated covariance as prior information the code is modified as:

```
DATA_SECTION
  init_int nR
  init_int nC
  init_matrix obs(1,nR,1,nC)
  vector ssb(1,nR)
  !! ssb=column(obs,1);
  vector logR(1,nR)
  !! logR=column(obs,2);

  init_vector priorMean(1,2)
  init_vector priorSd(1,2)
  init_matrix priorCor(1,2,1,2)
  matrix priorCov(1,2,1,2)
  !! priorCov=elem_prod(outer_prod(priorSd,priorSd),priorCor);

PARAMETER_SECTION
  init_number loga;
  init_number logb;
  init_number logSigma;
  sdreport_number sigmaSq;
  vector pred(1,nR);
  vector logab(1,2);
  vector diff(1,2);
  objective_function_value nll;
PROCEDURE_SECTION
  sigmaSq=exp(2.0*logSigma);
  pred=loga+log(ssb)-log(1+exp(logb)*ssb);
  nll=0.5*(nR*log(2*M_PI*sigmaSq)+sum(square(logR-pred))/sigmaSq);

  logab(1)=loga; logab(2)=logb;
  diff=logab-priorMean;
  nll+=0.5*(log(2.0*M_PI)*2.0+log(det(priorCov))+diff*inv(priorCov)*diff);
```

- To use both data sets the code is modified as:

```
DATA_SECTION
  init_int nR
  init_int nC
  init_matrix obs(1,nR,1,nC)
  vector ssb(1,nR)
  !! ssb=column(obs,1);
  vector logR(1,nR)
  !! logR=column(obs,2);

  !! ad_comm::change_datafile_name("bh0.dat");
  init_int nR0
  init_int nC0
  init_matrix obs0(1,nR0,1,nC0)
  vector ssb0(1,nR0)
  !! ssb0=column(obs0,1);
  vector logR0(1,nR0)
  !! logR0=column(obs0,2);

PARAMETER_SECTION
  init_number loga;
  init_number logb;
  init_number logSigma;
  sdreport_number sigmaSq;
  vector pred(1,nR);

  init_number logSigma0;
  sdreport_number sigmaSq0;
  vector pred0(1,nR0);
  objective_function_value nll;
PROCEDURE_SECTION
  sigmaSq=exp(2.0*logSigma);
  pred=loga+log(ssb)-log(1+exp(logb)*ssb);
  nll=0.5*(nR*log(2*M_PI*sigmaSq)+sum(square(logR-pred))/sigmaSq);

  sigmaSq0=exp(2.0*logSigma0);
  pred0=loga+log(ssb0)-log(1+exp(logb)*ssb0);
  nll+=0.5*(nR0*log(2*M_PI*sigmaSq0)+sum(square(logR0-pred0))/sigmaSq0);
```


Sequential Bayesian
log(b)
log(a)
−11.5
−12.0
−12.5
−13.0
1.6
1.8
2.0
2.2


Integrated analysis
log(b)
log(a)
−11.5
−12.0
−12.5
−13.0
1.6
1.8
2.0
2.2

- Identical?