Exercise 6: Stochasticity and Uncertainty

ETH Zurich Course 701-0426-00L: Modelling Aquatic Ecosystems (Schuwirth/Reichert)

13.05.2020

Goals:

• Understand the concepts of stochasticity and uncertainty and of parameter estimation described in the chapters 9 and 10 of the manuscript.
• Understand the Ornstein-Uhlenbeck process, the meaning of its parameters, how to use it to describe environmental stochasticity in driving forces or its effect on parameters, and how to propagate this stochasticity to model outputs.
• Understand the meaning of parameter uncertainty and its propagation to model outputs.
• Understand the concept of maximum likelihood parameter estimation.
• Understand the ideas underlying Bayesian inference of model parameters.
• Be able to consider environmental stochasticity and parameter uncertainty in modelling by propagating it through the simple lake plankton model using R package ecosim.
• Be able to do maximum likelihood and Bayesian parameter estimation of the simple lake plankton model based on a simple likelihood function.

Task 1: Simulation of Ornstein-Uhlenbeck processes

Study the implementation of drawing samples from Ornstein-Uhlenbeck processes using the function randou of the R package ecosim and interpret the behaviour of the solutions.

Task 2: Simulation of the lake plankton model with constant driving forces with consideration of environmental stochasticity

Study the implementation of the model that considers the effect of environmental stochasticity of four key parameters of the model and the propagation of this stochasticity to model outputs.

Task 3: Simulation of the lake plankton model with constant driving forces with consideration of parameter uncertainty

Study the implementation of the model that considers the effect of uncertainty of four key parameters of the model and the propagation of this uncertainty to model outputs.
**Task 4: Formulation of a likelihood function for the lake plankton model**

A probabilistic model is formulated as an assumed probability distribution of observations conditional on the values of the model parameters. We exemplify this concept with the simplest possible approach that assumes that the observations are distributed independently and normally, centered at the predictions of the model for the given parameters. Note that this probabilistic models has additional parameters to the deterministic model, in this case the standard deviations of the probability distributions of the observations around the deterministic model results. We assume these standard deviations to be different from one output variable to the other, but the same for the same output variable at different points in time.

The probability distribution of observations given model parameters is a function of both observations and model parameters. When used as a function of the model parameters with actual data substituted for the observations, this function is called “likelihood function”.

Study the implementation of the likelihood function as outlined above.

**Task 5: Maximum likelihood parameter estimation**

When searching for the parameters of the likelihood function with acutal data substituted for the observations, we find the parameter set that leads to the highest probability of the observed data within the parameterized model class. This approach is a very general method of point estimation of parameters. it can be complemented by estimating the uncertainty of the point estimates by confidence intervals or confidence regions. This is, however, beyond the scope of this course.

Study the implementation of maximum likelihood parameter estimation from time series of “observed” (in this case simulated) data using the R function `optim` for numerically maximizing the likelihood function implemented in task 4.

**Task 6: Bayesian parameter estimation**

Bayesian inference of model parameters combines prior information of parameters, in the form of a probability distribution, with observed data to derive an updated, posterior probability distribution of the parameters.

Study the implementation of Bayesian parameter estimation from the same “observational” data as used in task 5, based on the Metropolis Markov Chain Monte Carlo algorithm described in section 10 of the manuscript.

**Task 7: Homework: Propagating stochasticity and parameter uncertainty through the extended lake plankton model with periodic driving forces**

Transfer the analyses demonstrated in tasks 2 and 3 to the extended lake plankton model with periodic driving forces introduced in exercise 2 and interpret the results.
Questions

1. What is the difference between environmental stochasticity and uncertainty about model parameters, both from a conceptual and from an implementation point of view?

2. How can we decide whether a parameter or input should be made stochastic or uncertain?

3. What are the different characteristics of the three Ornstein-Uhlenbeck processes simulated in task 1? Can you guess the parameter values from the samples?

4. Why is it important to use transformed Ornstein-Uhlenbeck processes that lead to asymptotic lognormal distributions?

5. How can we reduce stochasticity of system and model behaviour?

6. Why is the prediction uncertainty much larger when considering zooplankton in the model?

7. Can you interpret the pattern you see in the 2d marginals from Bayesian inference?
Solution of Task 1: Simulation of Ornstein-Uhlenbeck processes

Load the package `ecosim` (and install if it is not yet installed).

```r
# load required packages:
if ( !require("ecosim") ) {install.packages("ecosim"); library(ecosim) }
```

## Loading required package: ecosim
## Loading required package: deSolve
## Loading required package: stoichcalc

Define sample sizes for stochastic simulation, uncertainty analysis due to incomplete knowledge, Maximum Likelihood parameter estimation, and Markov Chain Monte Carlo. We define two sets of parameter values that can be selected by setting the variable `didactic` either to `TRUE` or to `FALSE`. The didactic version is for demonstration purposes and requires much less computation time. However, it provides poorer results, in particular for MCMC. for `didactic` equal to `FALSE` it takes a lot of computation time, but produces the much nicer results for MCMC.

```r
# Define sample size and time domain and resolution:

didactic <- FALSE
if ( didactic ) {
  sampsize.stoch <- 5
  sampsize.unc <- 20
  sampsize.ml <- 200
  sampsize.mcmc <- 500
} else {
  sampsize.stoch <- 10
  sampsize.unc <- 20
  sampsize.ml <- 500
  sampsize.mcmc <- 10000
}
```

Open plot frame, define three Ornstein-Uhlenbeck processes by their mean, `mean`, asymptotic standard deviation, `sd`, and correlation time, `tau`, and plot three realizations of each process. Note that the third process is defined that the log of the plotted time series is an Ornstein-Uhlenbeck process. Still, the mean and the asymptotic standard deviation are defined at the non-log scale.

```r
# Draw from Ornstein-Uhlenbeck processes:

# Define time domain:

t <- 0:365
plot(numeric(0),numeric(0),xlim=c(0,365),ylim=c(0,20),xaxs="i",yaxs="i",
xlab="t",ylab=expression(theta),main="Realizations of Ornstein-Uhlenbeck Processes")

mean <- 16
sd <- 0.5
tau <- 50
abline(h=mean,lty="dotted")
lines(randou(mean=mean,sd=sd,tau=tau,t=t),lty="solid",col="black")
lines(randou(mean=mean,sd=sd,tau=tau,t=t),lty="dashed",col="blue")
lines(randou(mean=mean,sd=sd,tau=tau,t=t),lty="dotdash",col="red")
```
mean <- 10
sd  <- 1
tau <- 5
abline(h=mean,lty="dotted")
lines(randou(mean=mean,sd=sd,tau=tau,t=t),lty="solid",col="black")
lines(randou(mean=mean,sd=sd,tau=tau,t=t),lty="dashed",col="blue")
lines(randou(mean=mean,sd=sd,tau=tau,t=t),lty="dotdash",col="red")

mean <- 1
sd  <- 1
tau <- 10
abline(h=mean,lty="dotted")
lines(randou(mean=mean,sd=sd,tau=tau,t=t,log=TRUE),lty="solid",col="black")
lines(randou(mean=mean,sd=sd,tau=tau,t=t,log=TRUE),lty="dashed",col="blue")
lines(randou(mean=mean,sd=sd,tau=tau,t=t,log=TRUE),lty="dotdash",col="red")

Realizations of Ornstein–Uhlenbeck Processes
Solution of Task 2: Simulation of the lake plankton model with constant driving forces with consideration of environmental stochasticity

Note that the model definition is copied from exercise 2. We first define a named list of model parameters:

```r
# definition of model parameters:
param.mean <- list(k.gro.ALG = 0.5, # 1/d
k.gro.ZOO = 0.4, # m3/gDM/d
k.death.ALG = 0.1, # 1/d
k.death.ZOO = 0.05, # 1/d
K.HPO4 = 0.002, # gP/m3
Y.ZOO = 0.2, # gDM/gDM
alpha.P.ALG = 0.003, # gP/gDM
A = 5e+006, # m2
h.epi = 5, # m
Q.in = 5, # m3/s
C.HPO4.in = 0.04, # gP/m3
C.HPO4.ini = 0.004, # gP/m3
C.ALG.ini = 0.1, # gDM/m3
C.ZOO.ini = 0.1)
```

Next, we define the processes of growth and death of algae and zooplankton as objects of the class `process` of the package `ecosim`. Each process is defined by its name, rate, and stoichiometry. The rate is defined as an expression that can use parameters (defined for the object of class `system` below), concentrations defined in objects of class `reactor` that are part of the object of class `system`. To define the stoichiometry, a named list of expressions must be provided that identifies the substance or organism concentrations as the names and contains the stoichiometric coefficients as expressions.

```r
# definition of transformation processes
# growth of algae:
# definition of transformation processes
# growth of algae:

gro.ALG <- new(Class = "process",
   name = "Growth of algae",
   rate = expression(k.gro.ALG
       *C.HPO4/(K.HPO4+C.HPO4)
       *C.ALG),
   stoich = list(C.ALG = expression(1),
                 C.HPO4 = expression(-alpha.P.ALG))) # gDM/gDM

# death of algae:

death.ALG <- new(Class = "process",
    name = "Death of algae",
    rate = expression(k.death.ALG*C.ALG),
    stoich = list(C.ALG = expression(-1))) # gDM/gDM

# growth of zooplankton:
```
gro.ZOO <- new(Class = "process",
    name = "Growth of zooplankton",
    rate = expression(k.gro.ZOO * C.ALG * C.ZOO),
    stoich = list(C.ZOO = expression(1), C.ALG = expression(-1/Y.ZOO)))

death.ZOO <- new(Class = "process",
    name = "Death of zooplankton",
    rate = expression(k.death.ZOO * C.ZOO),
    stoich = list(C.ZOO = expression(-1)))

Next, we define the mixed box describing the epliminion of the lake as an object of the class reactor of the package ecosim.

epilimnion <- new(Class = "reactor",
    name = "Epilimnion",
    volume.ini = expression(A*h.epi),
    conc.pervol.ini = list(C.HPO4 = expression(C.HPO4.ini), C.ALG = expression(C.ALG.ini), C.ZOO = expression(C.ZOO.ini)),
    inflow = expression(Q.in*86400),
    inflow.conc = list(C.HPO4 = expression(C.HPO4.in), C.ALG = 0, C.ZOO = 0),
    outflow = expression(Q.in*86400),
    processes = list(gro.ALG, death.ALG, gro.ZOO, death.ZOO))

Finally, we combine the reactor, the parameters, and the desired output times in an object of class system of the package ecoval.

system <- new(Class = "system",
    name = "Lake",
    reactors = list(epilimnion),
    param = param.mean,
    t.out = seq(0,365,by=1))

Note that this object contains all definitions of the configuration of reactors (in this case just a single one), the processes active in each reactor, the model parameters, and the output time points. Any simulations carried out will refer to the definitions in this object, and not to the external variables that we used to set up the elements of the system.

First, we redo the simulations with constant parameters with absence or presence of zooplankton:

# Redo simulations without stochasticity and uncertainty:

system@param["C.ZOO.ini"] <- 0  # note that this reduces the model to model 91
res <- calcres(system)
To consider stochasticity in the parameters `k.gro.ALG`, `k.gro.ZOO`, `k.death.ALG`, `k.death.ZOO`, and `K.HPO4`, we first define a relative standard deviation of the parameters (for simplicity, we use the same for all parameters), a correlation time, tau, and the names of the parameters to be made stochastic:

```
# Define:
sd.rel <- 0.1
tau <- 5
names <- c("k.gro.ALG","k.gro.ZOO","k.death.ALG","k.death.ZOO","K.HPO4")
```

We then define data structures to collect the results and loop over the sampling iterations. Within each iteration, we sample from the stochastic parameters, assign them to the model system, and perform simulations without and with zooplankton. In the first iteration, we plot the sampled parameters to illustrate their behaviour and we print out the simulation time.

```
# Loop over Monte Carlo sampling and calculation of results:
```
res.envsto <- list()
res.envsto0 <- list()
par.def <- par(no.readonly=TRUE)
par(mfrow=c(2,3))
for ( i in 1:samps.size.stoch )
{
    # Make selected parameters stochastic in time:
    start.time <- proc.time()
    param <- param.mean
    for ( name in names )
    {
        mean <- param[[name]]
        param[[name]] <- randou(mean = mean,
                                sd = sd.rel*mean,
                                tau = tau,
                                t = seq(0,365,by=1),
                                log = TRUE)
        if ( i == 1 )
        {
            plot(param[[name]],type="l",xlab="t",ylab=name,ylim=c(0,2*mean))
            abline(h=mean,lty="dotted")
        }
    }
    system@param <- param
    res.envsto[[i]] <- calcres(system)
    system@param["C.ZOO.ini"] <- 0
    res.envsto0[[i]] <- calcres(system)
    if ( i == 1 )
    {
        print("computation time simple model, stochastic parameters")
        print(proc.time() - start.time)
    }
}

## [1] "computation time simple model, stochastic parameters"
## user system elapsed
##   13.59   0.00   13.61
par(par.def)
Note that, due to the fluctuations in the parameter, the integration algorithm has to take much shorter steps than with constant parameters, which results in a strong increase in the simulation time.

Finally, we plot the results:

```r
# Plot results:
plotres(res = res.envsto0,
colnames = list("C.HPO4",c("C.ALG","C.ZOO")))
```
plotres(res = res.envsto, 
colnames = list("C.HPO4", c("C.ALG", "C.ZOO")))

plotres(res = res.envsto, 
colnames = list("C.HPO4", c("C.ALG", "C.ZOO")))
Solution of Task 3: Simulation of the lake plankton model with constant driving forces with consideration of parameter uncertainty

The model is already defined from task 2 and we use the same parameters of the stochastic processes. For this reason, we can directly pass on to the loop over the Monte Carlo samples. The difference to task 2 is that within this loop, we draw constant parameters, rather than stochastic parameters to propagate their uncertainty through the model.

```r
res.parunc <- list()
res.parunc0 <- list()
for ( i in 1:sampsize.unc ) {
  # Make selected parameters uncertain:
  # -----------------------------------
  start.time <- proc.time()
  param <- param.mean
  for ( name in names ) {
    mean <- param[[name]]
    param[[name]] <- randnorm(mean = mean,
                             sd = sd.rel*mean,
                             log = TRUE)
  }
  system@param <- param
  res.parunc[[i]] <- calcres(system)
  system@param["C.ZOO.ini"] <- 0
  res.parunc0[[i]] <- calcres(system)
  if ( i == 1 ) {
    print("computation time simple model, uncertain parameters")
    print(proc.time() - start.time)
  }
}

## [1] "computation time simple model, uncertain parameters"
## user  system elapsed
## 0.37    0.00    0.38

Note that much shorter simulation time due to the use of constant parameters.

Finally, we plot again the results.

```r
plotres(res = res.parunc0,
        colnames = list("C.HPO4",c("C.ALG","C.ZOO")))
```
plotres(res = res.parunc, 
          colnames = list("C.HPO4", c("C.ALG", "C.ZOO")))
Solution of Task 4: Formulation of a likelihood function for the lake plankton model

The likelihood function is the probability of observations given parameters, seen as a function of the parameters with actual observations substituted into the function. For this reason, we first need observations.

In this didactical example, we produce faked observations by simulation, to demonstrate how one could estimate parameters from real observations also. This is useful to test whether parameter estimation recovers the true parameter values sufficiently accurately. However, typically, we would use real data to estimated model parameters.

We thus extend the parameter vector of the parameters to be inferred (we will need a vector rather than a list for use in the optimization and sampling functions in tasks 5 and 6) by the standard deviations of the observations of phosphate, algae and zooplankton. We call this parameter vector \( \text{par.ini} \), as we will use it to start the optimizer and the Markov chain. Here we use it as the “true” parameters underlying the simulated data. Note that with real data we will obviously not know the true parameters and the model structure will anyway be a simplification of reality. However, in this didactical setting, it is interesting to see to which degree we can recover the true parameters.

```r
names <- c("k.gro.ALG","k.gro.ZOO","k.death.ALG","k.death.ZOO","K.HPO4")
par.ini <- unlist(param.mean[names])
par.ini <- c(par.ini,sd.obs.HPO4=0.004,sd.obs.ALG=0.02,sd.obs.ZOO=0.02)
```

Next, we define the observation time points (two years, every 30 days), perform simulations for these time points with the model (starting at time zero), and reset the original time points in the system definition (just in case we want to see simulations at higher resolution again):

```r
t.obs <- 30*1:24
system@param <- param.mean
t.sim <- system@t.out
system@t.out <- c(0,t.obs)
res <- calcres(system)
system@t.out <- t.sim
```

We finally create our simulated observations by adding a random observation error term using the standard deviations defined above:

```r
obs <- res[-1,-1]
for ( i in 1:nrow(obs) ) {
  obs[i,"C.HPO4"] <- randnorm(obs[i,"C.HPO4"],par.ini["sd.obs.HPO4"])
  obs[i,"C.ALG"] <- randnorm(obs[i,"C.ALG"],par.ini["sd.obs.ALG"])
  obs[i,"C.ZOO"] <- randnorm(obs[i,"C.ZOO"],par.ini["sd.obs.ZOO"])
}
```

We now finally define the likelihood function. Please read it row by row and check whether you understand the comments:

```r
loglikeli <- function(par,system,obs,verbose=FALSE) {
  # negative parameter values lead to a likelihood of zero or a log likelihood of
  # minus infinity:
  if ( any(par<=0) ) return(-Inf)

  # set the parameters equal to the current values given as the first function argument
  # (keep the other parameters):
  system@param[names(par)] <- par
```
# set the start time to zero and the other output times to those with observations:
system@t.out <- c(0, as.numeric(rownames(obs)))

# calculate the deterministic results of our model:
res <- calcres(system)

# calculate the log likelihood using independent, normal distributions
# with extracting the standard deviations from the parameter vector
loglikeli <-
  sum(c(dnorm(x=obs[, "C.HPO4"], mean=res[-1, "C.HPO4"], sd=par["sd.obs.HPO4"], log=TRUE),
       dnorm(x=obs[, "C.ALG"], mean=res[-1, "C.ALG"], sd=par["sd.obs.ALG"], log=TRUE),
       dnorm(x=obs[, "C.ZOO"], mean=res[-1, "C.ZOO"], sd=par["sd.obs.ZOO"], log=TRUE)))

# print parameters and likelihood if the verbose mode was selected:
if ( verbose ) {
  print(par);
  cat("loglikeli = ", loglikeli, "\n")
}

# return the log likelihood value
return(loglikeli)

Test whether it works:
loglikeli(par=par.ini, system=system, obs=obs, verbose=TRUE)

## k.gro.ALG  k.gro.ZOO  k.death.ALG  k.death.ZOO  K.HPO4  sd.obs.HPO4
## 0.500      0.400      0.100      0.050      0.002      0.004
## sd.obs.ALG  sd.obs.ZOO
## 0.020      0.020
## loglikeli = 220.3458
## [1] 220.3458
Solution of Task 5: Maximum likelihood parameter estimation

As the likelihood function was already defined in task 4, maximum likelihood parameter estimation just consists of calling a maximizer. Note that this takes some time. Set verbose=TRUE if you want to follow the progress.

```r
# Maximum likelihood parameter estimation:
res.optim <- optim(par=par.ini,fn=loglikeli,control=list(fnscale=-1,maxit=sampsize.ml),
                    system=system,obs=obs,verbose=FALSE)
print("maximum likelihood solution:")
## [1] "maximum likelihood solution:")
print(round(res.optim$par,4))
##  k.gro.ALG  k.gro.ZOO  k.death.ALG  k.death.ZOO   K.HPO4  sd.obs.HPO4
##     0.5006    0.3963     0.1005     0.0501    0.0020    0.0047
##  sd.obs.ALG  sd.obs.ZOO
##    0.0154    0.0180
print("original parameter values:")
## [1] "original parameter values:"
print(par.ini)
##  k.gro.ALG  k.gro.ZOO  k.death.ALG  k.death.ZOO   K.HPO4  sd.obs.HPO4
##     0.500    0.4000     0.1000     0.0505    0.0020    0.0040
##  sd.obs.ALG  sd.obs.ZOO
##    0.0200    0.0200
```

Due to the noise in the data, we do not get the correct parameters back. We do not calculate confidence intervals of the parameters here, but proceed with Bayesian inference to get posterior probability distributions that describe our knowledge including its uncertainty.
Solution of Task 6: Bayesian parameter estimation

Bayesian inference combines prior information about model parameters with information gained from observed data using a given model structure to calculate results for given parameters.

For this reason, we have to define a function that returns the prior probability density for given parameters. For simplicity, we assume independent normal distributions of the parameters centered at the true value and with standard deviations of 1/4th of the mean:

\[
\text{logprior} \leftarrow \text{function}(\text{par, par.prior, sd.rel}=0.25) \{ \\
\quad \text{# the prior density is zero for negative values, its log is minus infinity:} \\
\quad \text{if} \ ( \text{any(par}<0)) \ \text{return}(-\text{Inf}) \\
\quad \text{# for non-negative parameters we sum the log densities to get the joint log density:} \\
\quad \text{return}(\text{sum(dnorm(x=par, mean=par.prior, sd=sd.rel*par.prior), log=TRUE})) \\
\}
\]

To sample from a Markov chain that represents a sample from the posterior distribution, we write a function that implements the procedure described in section 11.4 of the manuscript. Please go through the code line by line and check whether you understand the comments:

\[
\text{sample.metropolis} \leftarrow \text{function}(\text{par, loglikeli, logprior, prop.sd, sampsize, par.prior, system, obs, verbose=FALSE}) \{ \\
\quad \text{# calculate the log prior for the initial parameter values:} \\
\quad \text{logpri} \leftarrow \text{logprior}(\text{par, par.prior}) \\
\quad \text{# return with an error if it is not finite at the initial point:} \\
\quad \text{if} \ ( \text{!is.finite(logpri)}) \ { \text{cat}(\text{"*** error, initial prior is not finite\n"} \\
\quad \text{return}(\text{NA}) } \\
\quad \text{# calculate the log likelihood for the initial parameter values:} \\
\quad \text{logli} \leftarrow \text{loglikeli}(\text{par, system=system, obs=obs, verbose=verbose}) \\
\quad \text{# return with an error if it is not finite at the initial point:} \\
\quad \text{if} \ ( \text{!is.finite(logli)}) \ { \text{cat}(\text{"*** error, initial likelihood is not finite\n"} \\
\quad \text{return}(\text{NA}) } \\
\quad \text{# define a matrix to store the parameter sample and set the first row to the initial parameter values:} \\
\quad \text{parsamp} \leftarrow \text{matrix}(\text{NA, nrow=sampsize+1, ncol=length(par)}) \\
\quad \text{colnames(parsamp)} \leftarrow \text{names(par)} \\
\quad \text{parsamp}[1,] \leftarrow \text{par} \\
\quad \text{# define a counter for accepted proposals and initialize it to zero:} \\
\quad \text{accepted} \leftarrow 0 \\
\quad \text{# proceed with constructing the Markov chain:} \\
\quad \text{for}\ (\ i\ \text{in}\ 1: \text{sampsize}) \ { \\
\quad \quad \text{# sample a proposed parameter set by adding a normally distributed deviation to each parameter:} \\
\quad \quad \text{par.prop} \leftarrow \text{par} \\
\quad \quad \text{for}\ (\ j\ \text{in}\ 1: \text{length(par)}) \ { \text{par.prop}[j] \leftarrow \text{randnorm(par}[j], \text{prop.sd}[j]) \\
\quad \}
\}
\]

17
```r
# calculate the prior at the proposal:
logpri.prop <- logprior(par.prop,par.prior)

# proceed only if it is finite, otherwise reject the proposal:
if (is.finite(logpri.prop)) {
  # calculate the likelihood at the proposal:
  logli.prop <- loglikeli(par.prop,system=system,obs=obs,verbose=verbose)

  # proceed only if it is finite, otherwise reject the proposal:
  if (is.finite(logli.prop)) {
    # calculate the ratio of the posterior densities at the proposed parameter set
    # relative to the current parameter set. Note that the posterior is proportional
    # to prior times likelihood, this means that the log posterior is up to an
    # additive constant equal to the sum of the log prior and the log likelihood:
    accept.ratio <- exp(logpri.prop + logli.prop - logpri - logli)

    # accept the proposal if the posterior is larger at the proposal, otherwise only
    # with the probability equal to the ratio calculated above. This is implemented
    # by comparing the ratio with a random number drawn from a uniform distribution
    # in the unit interval:
    if (accept.ratio > runif(1)) {
      # if accepted, update parameter, log prior and log likelihood and increase the
      # acceptance counter.
      par <- par.prop
      logpri <- logpri.prop
      logli <- logli.prop
      accepted <- accepted + 1
    }
  }
}

# write the old or the accepted parameter value into the sample matrix:
parsamp[1+i,] <- par

# calculate and write the fraction of accepted proposals:
cat("acceptance fraction",accepted/sampsize)

# return the whole sample:
return(parsamp)
}
```

Now, we are ready to sample from the Markov chain. Note that this takes some time to get a sample that covers the distribution well. You can reduce the argument `sampsize` to continue and improve the sample later with a longer run.

```r
res.mcmc <- sample.metropolis(par=par.ini,loglikeli=loglikeli,logprior=logprior,
                           prop.sd=0.005*par.ini,sampsize=sampsize.mcmc,par.prior=par.ini,
                           system=system,obs=obs,verbose=FALSE)
```

## acceptance fraction 0.1401
Given the calculated sample, we can plot approximate marginal distributions of all parameters:

```r
par.def <- par(no.readonly=TRUE)
par(mfrow=ceiling(sqrt(ncol(res.mcmc)))*c(1,1))
for ( j in 1:ncol(res.mcmc) ) {
  plot(density(res.mcmc[,j],adjust=2),xlab=colnames(res.mcmc)[j],main=colnames(res.mcmc)[j])
  abline(v=par.ini[colnames(res.mcmc)[j]])
}
par(par.def)
```

Or look at the sample of two-dimensional marginals:
pairs(res.mcmc, pch=19, cex=0.3)
Solution of Task 7: Propagating stochasticity and parameter uncertainty through the extended lake plankton model with periodic driving forces

We first extend the model as we did already in exercise 2:

```r
# Extend model parameters:

param.mean.ext <- c(param.mean,
  list(beta.ALG = 0.046, # 1/degC
       beta.ZOO = 0.08, # 1/degC
       T0 = 20, # degC
       K.I = 30, # W/m2
       lambda.1 = 0.10, # 1/m
       lambda.2 = 0.10, # m2/gDM
       t.max = 230, # d
       I0.min = 25, # W/m2
       I0.max = 225, # W/m2
       T.min = 5, # degC
       T.max = 25)) # degC

# Extend growth of algae by environmental factors:

gro.ALG.ext <-
new(Class = "process",
  name = "Growth of algae extended",
  rate = expression(k.gro.ALG
    *exp(beta.ALG*(T-T0))
    *C.HPO4/(K.HPO4+C.HPO4)
    *log((K.I*I0)
      /((K.I*I0)*exp(-(lambda.1+lambda.2*C.ALG)*h.epi)))
      /((lambda.1+lambda.2*C.ALG)*h.epi)
    *C.ALG),
  stoich = list(C.ALG = 1, # gDM/gDM
                C.HPO4 = expression(-alpha.P.ALG))) # gP/gDM


gro.ZOO.ext <-
new(Class = "process",
  name = "Growth of zooplankton",
  rate = expression(k.gro.ZOO
    *exp(beta.ZOO*(T-T0))
    *C.ALG
    *C.ZOO),
  stoich = list(C.ZOO = expression(1), # gDM/gDM
                C.ALG = expression(-1/Y.ZOO))) # gP/gDM

# Re-define processes in the reactor "epilimnion":

epilimnion.ext <- epilimnion
epilimnion.ext@processes <- list(gro.ALG.ext,death.ALG,gro.ZOO.ext,death.ZOO)

# Make environmental conditions (light and temperature) time dependent:

epilimnion.ext@cond <- list(I0 = expression(0.5*(I0.min+I0.max)+
```
\[
T = \text{expression}(0.5 \times (T_{\text{min}} + T_{\text{max}}) + 0.5 \times (T_{\text{max}} - T_{\text{min}}) \times \cos(2 \times \pi / 365.25 \times (t - t_{\text{max}})))) \quad \text{# degC}
\]

# Re-define the reactor "epilimnion" in the system definition:

```r
system.ext <- system
time)
```

# Increase algal growth rate to compensate for new limitations:

```r
param.mean.ext$k.gro.ALG <- 0.8
```

# Replace parameters in the system definition:

```r
system.ext@param <- param.mean.ext
```

# Increase simulation time to more than one season:

```r
t.out = seq(0, 1461, by=1)
```

We then consider stochastic parameters as explained above in task 2:

# Loop over Monte Carlo sampling and calculation of results:

```r
sd.rel <- 0.1
tau <- 5
names <- c("k.gro.ALG", "k.gro.ZOO", "k.death.ALG", "k.death.ZOO", "K.HPO4")
```

```r
res.ext.envsto <- list()
res.ext.envsto0 <- list()
par.def <- par(no.readonly=TRUE)
par(mfrow=c(2, 3))
```

```r
for ( i in 1:sampsize.stoch ) {
    # Make selected parameters stochastic in time:
    start.time <- proc.time()
    param <- param.mean.ext
    for ( name in names ) {
        mean <- param[[name]]
        param[[name]] <- randou(mean = mean, 
                                sd = sd.rel*mean, 
                                tau = tau, 
                                t = t.out, 
                                log = TRUE)
        if ( i == 1 )
            {
                plot(param[[name]], type="l", xlab="t", ylab=name, ylim=c(0, 2*mean))
                abline(h=mean, lty="dotted")
            }
    }
}
```
system.ext@param <- param
res.ext.envsto[[i]] <- calcres(system.ext)
res.ext.envsto0[[i]] <- calcres(system.ext)
if ( i == 1 ) {
  print("computation time extended model, stochastic parameters")
  print(proc.time() - start.time)
}

## [1] "computation time extended model, stochastic parameters"
## 63.78 0.04 63.86

par(par.def)

... and plot the results:

# Plot results:

plotres(res = res.ext.envsto0, colnames = list("C.HPO4", "C.ALG", "C.ZOO"))
Then, we do the same for parameter uncertainty as done above in task 3:

```r
res.ext.parunc <- list()
res.ext.parunc0 <- list()
for (i in 1:sampsize.unc) {
  # Make selected parameters uncertain:
  start.time <- proc.time()
  param <- param.mean.ext
  for (name in names) {
    mean <- param[[name]]
    param[[name]] <- randnorm(mean = mean,
                              sd = sd.rel*mean,
                              log = TRUE)
  }
  system.ext@param <- param
  res.ext.parunc[[i]] <- calcres(system.ext)
}```
system.ext@param["C.ZOO.ini"] <- 0
res.ext.parunc0[[i]] <- calcres(system.ext)
if ( i == 1 )
  { print("computation time extended model, uncertain parameters")
    print(proc.time() - start.time)
  }
}

## [1] "computation time extended model, uncertain parameters"
## user system elapsed
##  1.85   0.00   1.85

# Plot results:
plotres(res = res.ext.parunc0,
        colnames = list("C.HPO4",c("C.ALG","C.ZOO")))

plotres(res = res.ext.parunc,
        colnames = list("C.HPO4",c("C.ALG","C.ZOO")))