Estimating Parameters of Compartment Models in SoilR:

Markus Müller* and Carlos A. Sierra†
Max Planck Institute for
Biogeochemistry

July 19, 2012

Abstract

The objective of this vignette is to provide an example on how to use SoilR to interface with package FME in order to estimate parameters of soil organic matter decomposition models using observed data. To this end we set the following task: to identify a mathematical structure that best represent some observed data of carbon stocks, fluxes and/or their isotopic composition. The target model should fulfill the following requirements.

1. It can reproduce the overall time behavior of the data.
2. Its parameters can be determined by the data with sufficient accuracy.

We will not explain FME functionality here, but strongly recommend to read the vignette for package FME. Instead, we focus here on the application to the models implemented in SoilR.

1 The data

Assume the following data have been measured.

1. The $\Delta^{14}C$ of soil respiration at some points in time.
2. The carbon content in two soil fractions measured over time.
3. The total amount of carbon entering the soil over time.
4. A time series of the atmospheric $\Delta^{14}C$ starting from 1990 to 2010.

The data used for these examples are provided as part of SoilR. Uncommenting the following code shows all datasets that come with the package.

```
> #library(SoilR)
> #data(package="SoilR")
```

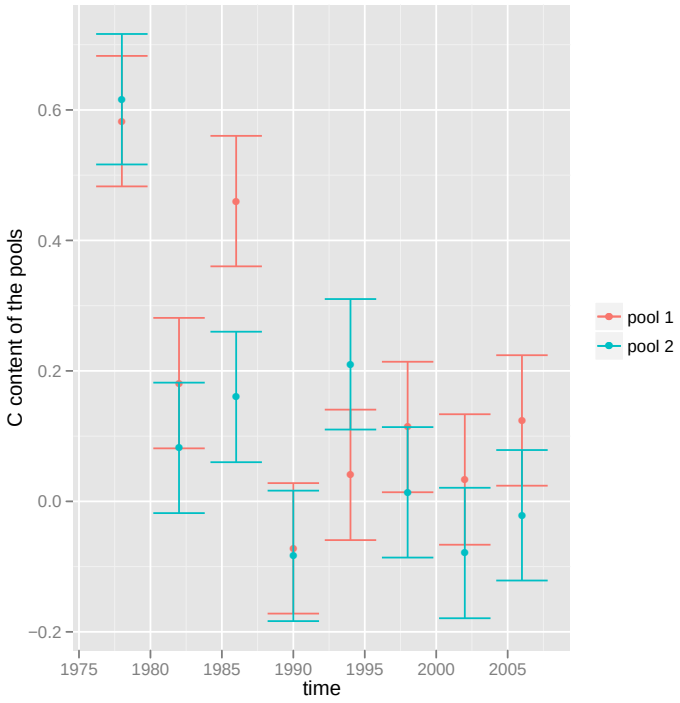
We load the following:

```
> data(CourseExample_R14)
> data(C14Atm_NH)
```

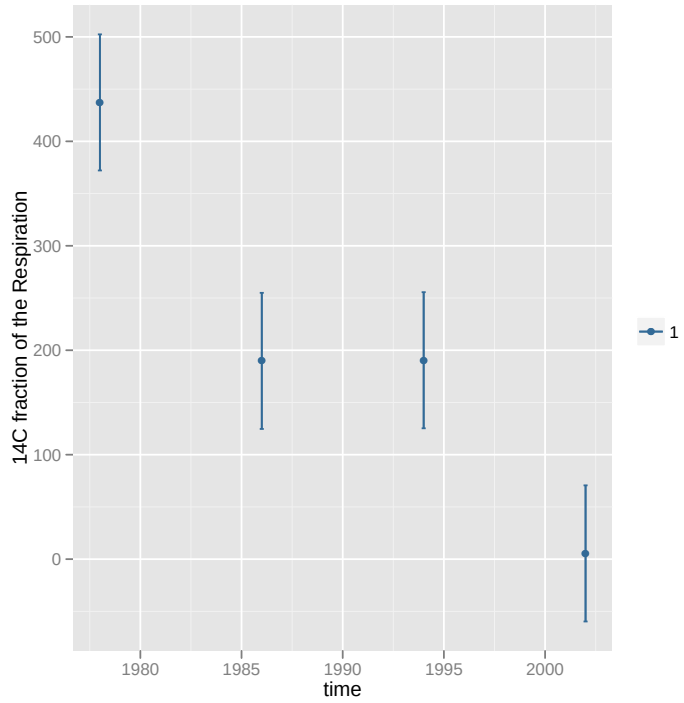
The following code plots the data.

*mamueller@bgc-jena.mpg.de
†csierra@bgc-jena.mpg.de

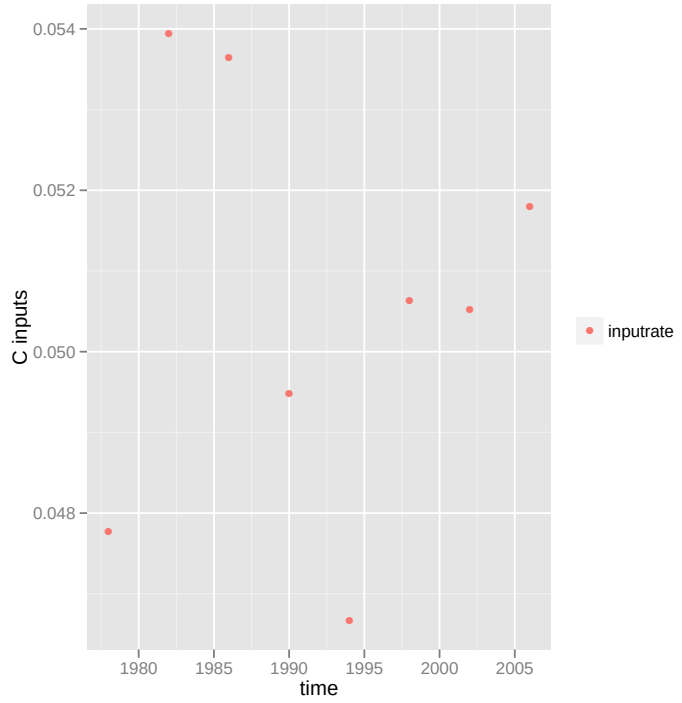
```
>       library(ggplot2)
>       data(CourseExample_R14)
>       lty1=1
>       lty2=2
>       c1="pool 1"
>       c2="pool 2"
>       yl=max(DataC[,"C1"],DataC[,"C2"])
>       ym=min(DataC[,"C1"],DataC[,"C2"])
>       p <- ggplot(data.frame(DataC))
>       p <- p+geom_point(aes(x=time,y=C1,col=c1))
>       p <- p+geom_errorbar(aes(x=time,ymin=C1-sd,ymax=C1+sd,col=c1))
>       p <- p+geom_point(aes(x=time,y=C2,col=c2))
>       p <- p+geom_errorbar(aes(x=time,ymin=C2-sd,ymax=C2+sd,col=c2))
>       p <- p+scale_y_continuous(name="C content of the pools")
>       p <- p+opts(legend.title=theme_blank())
>       p
```



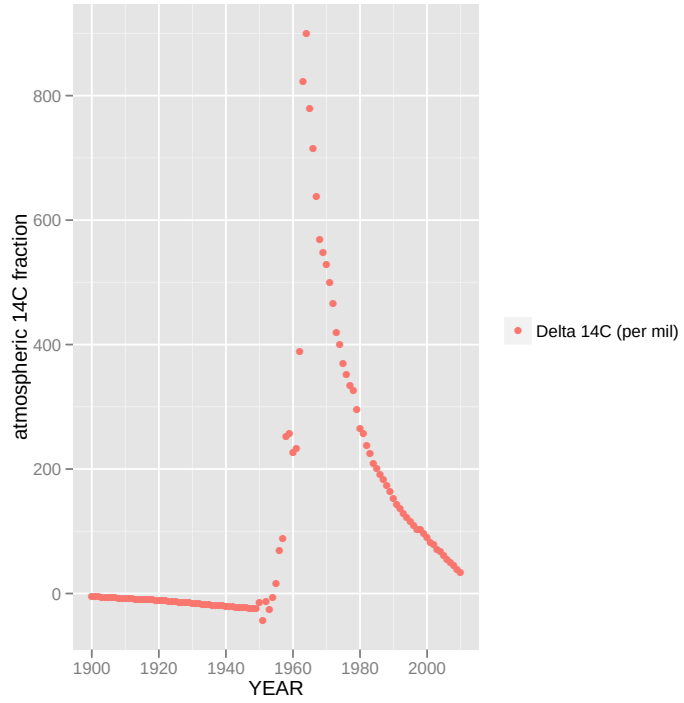
```
>       yl=max(DataR14[,"R14t"])
>       ym=min(DataR14[,"R14t"])
>       limits<- aes(ymax = R14t + sd, ymin=R14t - sd)
>       p <- ggplot(data.frame(DataR14), aes(colour=c(1), y=R14t, x=time))
>       p <- p + geom_point() + geom_errorbar(limits, width=0.2)
>       p <- p + geom_pointrange(limits)
>       p <- p+scale_y_continuous(name="14C fraction of the Respiration")
>       p <- p + opts(legend.title=theme_blank())
>       p
```



```
>       p <- ggplot(DataI)
>       p <- p+geom_point(aes(x=time,y=In,col="inputrate "))
>       p <- p+scale_y_continuous(name="C inputs")
>       p <- p+opts(legend.title=theme_blank())
>       p
```



```
>       p <- ggplot(C14Atm_NH)
>       p <- p+geom_point(aes(x=YEAR,y=Atmosphere,col="Delta 14C (per mil)"))
>       p <- p+scale_y_continuous(name="atmospheric 14C fraction")
>       p <- p+opts(legend.title=theme_blank())
>       p
```



2 Available Models

`SoilR` provides a very general approach to model soil organic matter decomposition and can therefore be used for a wide range of applications. Many models, especially those that are frequently used in the literature, have been implemented on top of the general framework making them even easier to use. To see what is in the package uncomment the following code,

```
> #?Models
```

which will show all the currently available functions for the construction of models. In our case we look for those that start with `Twop..`, which abbreviates *two pool* and end with `..14`, which marks the ^{14}C model constructors but also you may consider more general cases. The candidates are thus:

1. TwopParallelModel14
2. TwopSeriesModel14
3. TwopFeedbackModel14
4. GeneralModel14

In thoery all these models could potentially describe the data, which is our first request. A closer look at the help pages of these candidates reveals that the number of parameters to choose increases from top to bottom. Thus the difficulty to *constrain* the parameters will increase also. It is convenient to start with the `twopParallelModel14` and assume only the two decay constants of the pools as variable parameters. As we proceed we can allow more parameters to be varied and evaluate the increase in model performance.

3 Synthesis

Our first aim is to reproduce a synthetic dataset as function of the parameters to be determined. In a second step, we have to provide a measure for the disagreement between the synthetically produced data and the real data. The deviation between predictions and observations will be computed by an *objective* or *cost* function. Then, we have to try different combinations of the free parameters in order to minimize the objective function. The `FME` package will be very useful for steps two and three, while we will use `SoilR` for the first step. A look at the help page of `twopParallelModel14` shows which arguments we have to provide. We start with those that we consider fixed.

- We begin with the time steps where we want to compute the solution. Since we want to compare the output to the observed data later it is reasonable to compute values at least for the times given in our dataset.

  ```
  > library("SoilR")
  > library("FME")
  > t_start=1978
  > t_end=2008
  > #t_start=t_end-30
  > indices=(C14Atm_NH$YEAR>=t_start & C14Atm_NH$YEAR < t_end)
  > time=C14Atm_NH$YEAR[indices]
  ```

- We proceed with the initial values, which we can extract from the data.

```
> C0=as.numeric(DataC[1,c("C1","C2")])
[1] 0.5828128 0.6164349
```

- The next item is the inputrate which is part of the data set and can be passed directly.
- The parameter γ, which represents the partitioning of the inputs to the two different pools, will be held constant.

```
> gam=0.6
```

- ξ, a parameter used to describe the influence of temperature and moisture on decomposition as function of time will be held constant as well.

```
> xi=1
```

- The dataset of atmospheric $\Delta^{14}C$ values will be used directly as an argument for `FcAtm`.
- The untis of the radioactive decay constant λ must be compatible with our time unit. We compute it here from the Libby half-life given in years. `SoilR` would use the same value as default, so this preparation is really needed only if you decide to use another unit for time measurement.

```
> th=5730
[1] 5730
> #note that lambda is negative and has the unit y^-1
> lambda=log(0.5)/th
[1] -0.0001209681
```

The remaining part of the model construction we wrap into a function that depends on the unknown parameters k_1 and k_2 and the parameter `pass` only. The `pass` argument requires some explanation. It only makes sense in view of the future use of our small function with `FME`. The reason is this: If you create a model in `SoilR` the package will by default execute an number of tests on the given arguments to prevent the unintentional creation of models that are not biologically meaningful. Unfortunately `FME`'s algorithms do not care about biologically meaningful parameter combinations and would cause `SoilR` to exit with an error if by chance such a combination occurs. It might for instance happen that during the fitting procedure `FME` tries a positive value of k to fit the data, which would mean *creation* of organic matter instead of *decomposition*. The interface of `modFit` provides only one possibility to avoid this situation by setting appropriate upper and lower bounds for parameters. This would do the job with this simple example but not for all cases. *For those cases only* we introduce the `pass` facility used as a last resort to switch off the checking at model creation. If we do so we have to check the estimated parameters afterwards without this flag. We will do so at the end of this example. For the moment, note that we allow for an additional argument to turn off the internal checking by setting `pass=TRUE`.

```
> pf<-function(ks,pass=FALSE){
+     mod=TwopParallelModel14(
+             time,
+             ks,
+             C0,
+             In=DataI,
```

```
+             gam=gam,
+             xi=xi,
+             FcAtm=C14Atm_NH,
+             lambda=lambda,
+             pass=pass
+     )
+     Cs=getC(mod)
+     R14t=getTotalReleaseFluxC14CRatio(mod)
+     return(data.frame(time=time,R14t=R14t,C1=Cs[,1],C2=Cs[,2]))
+ }
```

We produce a dataset of arbitrarily chosen parameters k_1 and k_2 and plot it together with the data:

To give `FME` a tool to measure the quality of a parameter combination we need a cost function that weights the error. Since in this example we deal with two different kinds of data that have different units and have a different number of data points we have to invest some time for thought here.

1. We apply `FME`'s `modCost` twice, the first time on data arguments only but the second time with the additional argument of the first cost function. This expresses the fact that we treat errors of different observables differently.
2. The weighing is done by dividing the residuals by the error which is done for two reasons: first to get a dimensionless result and second to reduce the influence of uncertain values. The errors are assumed to be in column with name "sd" of the measured dataset.
3. We could suppress the weight of the spare $\Delta^{14}C$ values by the more frequent C measurements by rescaling the cost function according to the number of data points, which could be done with the `scaleVar` argument.

Since we now have a measure for the cost we can use fully `FME` functionality; e.g. determine how sensitive this cost is to changes in our parameters, which helps us to detect unidentifiable parameters. To do so we create the sensitivity functions. We also can estimate the approximate linear dependence (collinearity) of the two parameters which reveals parameters that have similar effect on the output and are hard to identify simultaneously. All this is `FME` functionality and is described in much more detail by `FME`'s vignettes.

We can estimate the collinearity explicitly:

```
> ident <- collin(Sfun,parset=c("k1","k2"))

  k1 k2 N collinearity
1  1  1 2            1
```

Finally we fit the parameter to the data. Before we do so we always have to think about how to prevent the creation of invalid models. The only way provided by `modFit` is to constrain parameter ranges by upper and lower bounds. Whether this will constrain `FME`'s guesses to the reasonable parameter values depends on our ability to choose the appropriate parameter set. However to give `FME` algorithms the freedom to (temporarily) test even (biologically) unreasonable values we deliberately disabled `SoilR`'s checks during the creation of models in the cost function. To demonstrate that with `pass` set to true the parameter estimation works even for biologically meaningless parameters we deliberately choose

a positive start value for one of the decay constants and also a to large parameter range to include this value. (For a valid model both decay constants would have to be smaller than zero since positive values do not refer to decomposition but "growth".) We compare the fitted curve to the curve resulting from the values that were used to create the example dataset in the first place. To this end we create again the model with the estimated parameter values, this time without the `pass` argument to use `SoilR`'s checks. So we are sure that the best numerical fit is also biologically possible. Usually we are not only interested in the best fitting parameters, but rather in the distribution of the parameters. We can estimate them with the Monte Carlo Markov Chain method that is also part of `FME`. Once again we refer to the `FME`'s vignette for details.

```
> pars=c(k1=-0.1,k2=-0.2)
> Df=pf(pars)
> c1="black"; c2="red";lty1=1;lty2=1
> plot(DataC[,"time"],
+      DataC[,"C1"],
+      col=c1,lty=lty1,
+      xlab="time",
+      ylab="C content of the pools"
+ )
> points(
+        DataC[,"time"],
+        DataC[,"C2"],
+        col=c2,
+        lty=lty2
+ )
> lines(Df$time,Df$C1,col=c1,lty=lty1)
> lines(Df$time,Df$C2,col=c2,lty=lty2)
> legend("topright",
+        legend=c("C content of pool 1","C content of pool 2"),
+        col=c(c1,c2),lty=c(lty1,lty2),bty="n")
```

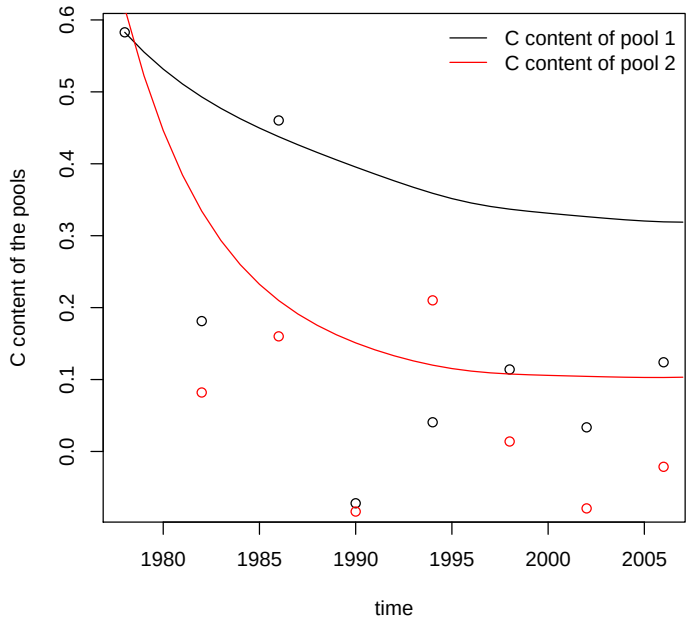


Figure 1: The output for the C stock of a model with arbitrarily chosen parameters k_1 and k_2 together with the observed data.

```
> plot(DataR14[,"time"],DataR14[,"R14t"],lty=1,col=c1,
+      xlab="Years",ylab=expression(paste(Delta^14,"C "," in respiration (permil)")))
> lines(Df$time,Df$R14t,col=c1,lty=lty1)
```

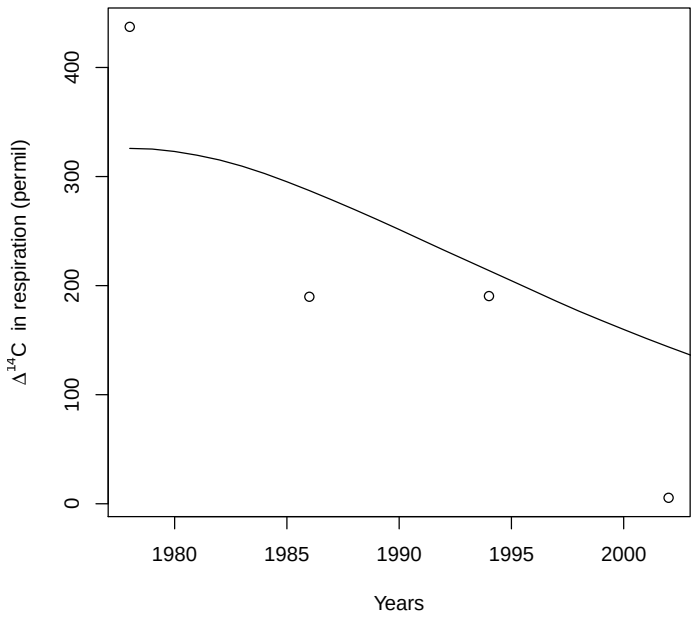


Figure 2: The modeled $\Delta^{14}C$ fraction of the respiration of a model with arbitrarily chosen parameters k_1 and k_2 together with the (measured) data
.

```
>     DfCost <- function(pars){
+         Df <- pf(pars,pass=TRUE)
+         Ccost=modCost(
+                     model=Df,
+                     obs=DataC,
+                     err="sd"
+                     #,scaleVar=TRUE
+                     )
+         return(
+                modCost(
+                        model=Df,
+                        obs=DataR14,
+                        err="sd",
+                        #,scaleVar=TRUE
+                        cost=Ccost)
+                )
+     }
>     plot(DfCost(pars),xlab="time")
```

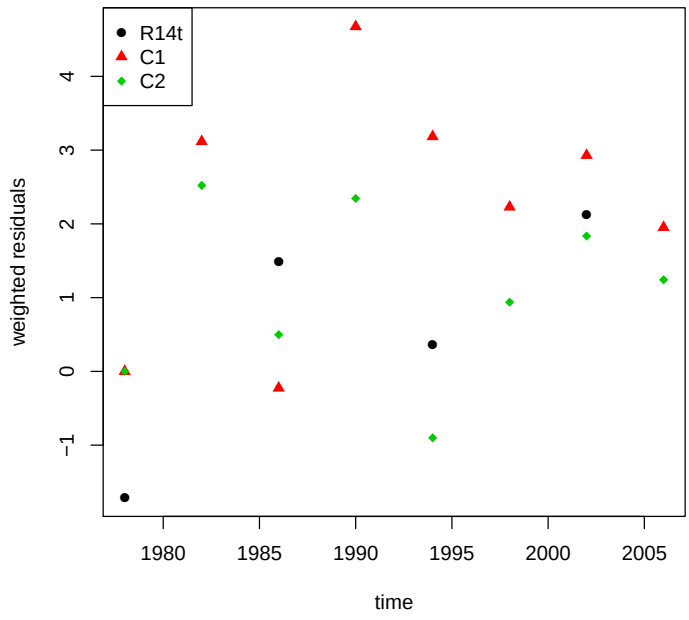


Figure 3: The (error) weighted residuals.

```
> Sfun <- sensFun(DfCost,pars)
> summary(Sfun)
```

```
   value scale   L1   L2  Mean  Min Max  N
k1  -0.1  -0.1 0.31 0.12 -0.31 -1.0   0 20
k2  -0.2  -0.2 0.42 0.16 -0.42 -1.4   0 20
```

```
> plot(Sfun,which=c("R14t","C1","C2"),xlab="time",lwd=2)
```

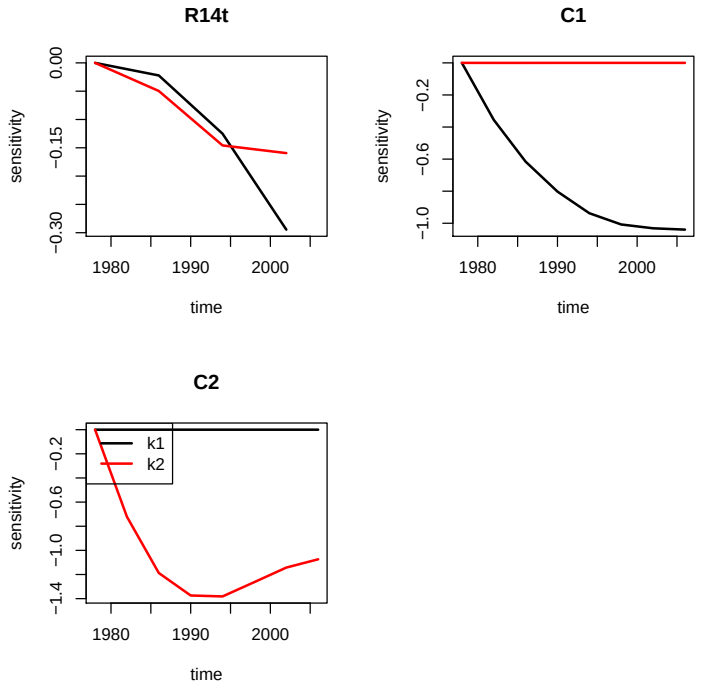


Figure 4: The plot shows the sensitivity functions of all three observables on both parameters. The result is an extreme case and shows that in a parallel model the C stock of pool 1 does not depend on k_2 and vice versa, while the overall respiration naturally depends on both variables

```
> pairs(Sfun,which=c("R14t","C1","C2"),col=c("green","blue","red"))
```

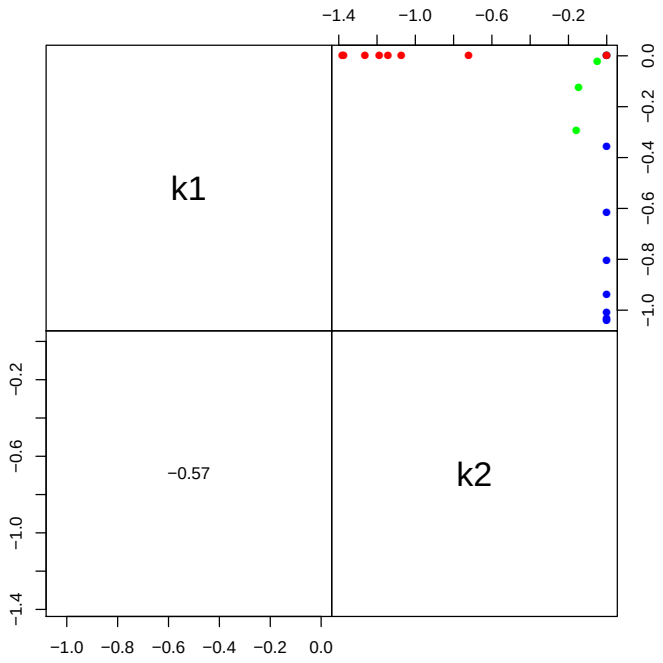


Figure 5: The pairs plot summarizes the dependency of the sensitivity function of all three observables on both parameters. It would also reveal collinearity if the points of different colors were showing a similar pattern which they fortunately do not in this example. The alignment of the blue and red clouds to the axis expresses the same lesson that we learnt from the plot of the sensitivity functions: In a parallel model the C stock of pool 1 does not depend on k_2 and vice versa

```
>     Fit <- modFit(f=DfCost,upper=c(0.2,0),p=c(0.1,-0.1))
>     print(Fit$par)

[1]  0.1999981 -0.6283194

>     plot(Fit)
>     summary(Fit)

Parameters:
     Estimate Std. Error t value Pr(>|t|)
[1,]  0.20000    0.04735   4.224  0.00051 ***
[2,] -0.62832    0.07376  -8.519 9.91e-08 ***
---
Signif. codes:  0 âĂŸ***âĂŹ 0.001 âĂŸ**âĂŹ 0.01 âĂŸ*âĂŹ 0.05 âĂŸ.âĂŹ 0.1 âĂŸ âĂŹ 1

Residual standard error: 1.329 on 18 degrees of freedom

Parameter correlation:
        [,1]    [,2]
[1,]  1.0000 -0.0758
[2,] -0.0758  1.0000
```

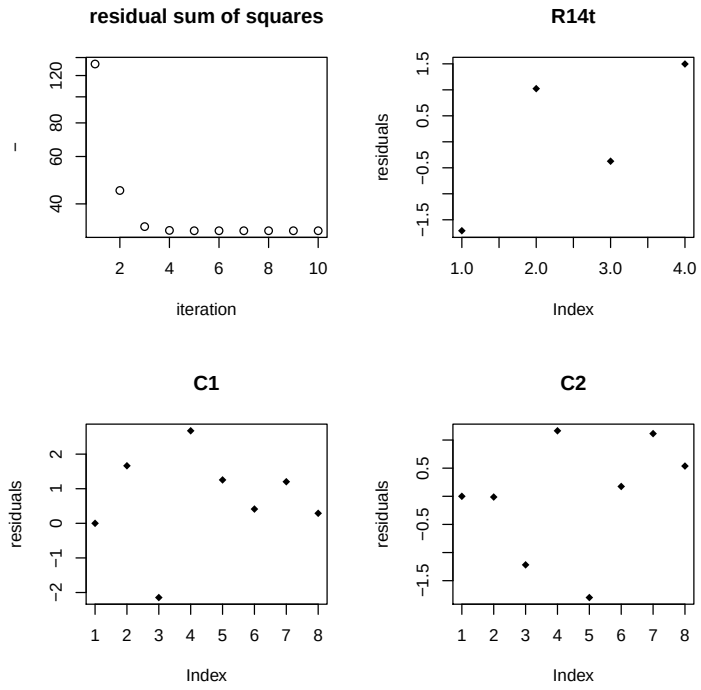
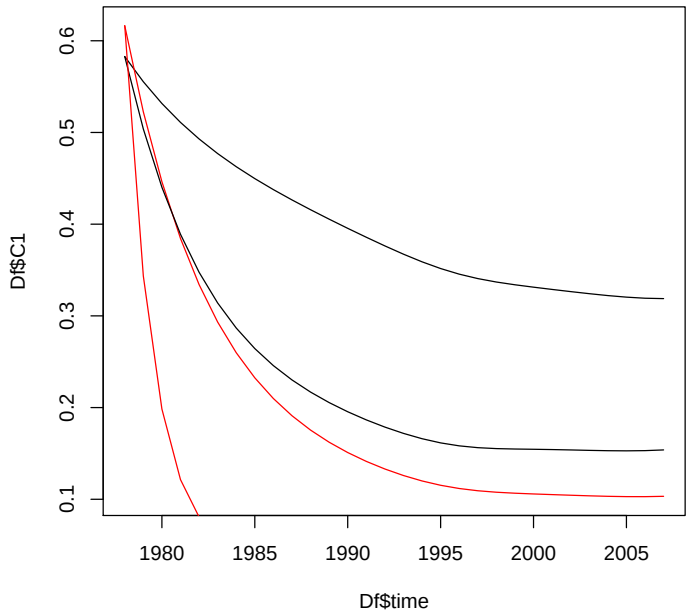
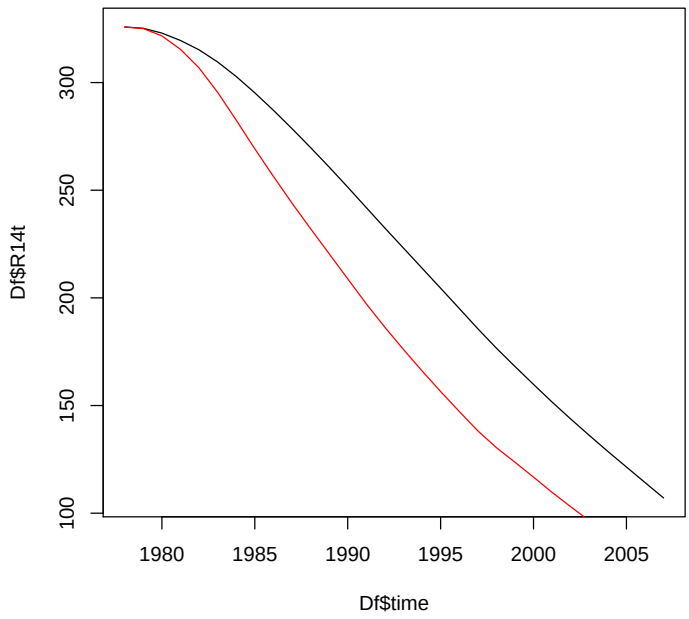


Figure 6: The plot shows how the residuals have been minimized by the fitting procedure and can be used to inspect convergence of the algorithm

```
> #pars=c(k1=-0.6,k2=-0.2)
> Dfinal=pf(Fit$par)
>     plot(Df$time,Df$C1,type="l",lty=lty1,
+          ylim=c(min(Df[,c("C1","C2")]),max(Df[,c("C1","C2")])),
+          col=c1)
>     lines(Df$time,Df$C2,lty=lty1,col=c2)
>     lines(Dfinal$time,Dfinal$C1,lty=lty2,col=c1)
>     lines(Dfinal$time,Dfinal$C2,lty=lty2,col=c2)
```



```
>     plot(Df$time,Df$R14t,type="l",lty=lty1,col=c1)
>     lines(Dfinal$time,Dfinal$R14t,lty=lty2,col=c2)
```



```
>     #var0 <- Fit$var_ms_unweighted
>     #cov0 <- summary(Fit)$cov.scaled#*2.4^2/5
>     #p=Fit$par
>     niter=500
>     t1=Sys.time()
>     MCMC  <- modMCMC(f=DfCost,niter=niter,p=Fit$par)

number of accepted runs: 476 out of 500 (95.2%)

>     t2=Sys.time()
>     print(t1-t2)

Time difference of -1.512057 mins

>     summary(MCMC)

             p1         p2
mean 0.32550783 -2.0802482
sd   0.07860243  0.7702962
min  0.18029414 -3.2766230
max  0.51559924 -0.5527030
q025 0.25994852 -2.8042067
q050 0.31139789 -2.3073376
q075 0.39304738 -1.4001887

>     plot(MCMC, Full = TRUE)
```

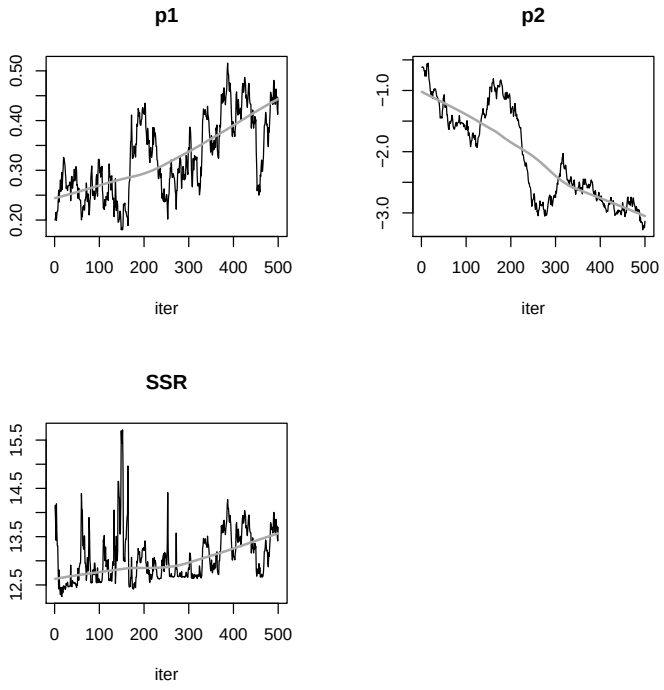


Figure 7: The plot shows the accepted values for each parameter. It can be used to identify situations where there is doubt if the collected values represent the equilibrium distribution of the chain

```
>     sR=sensRange(func=pf, parInput=MCMC$par)
>     plot(summary(sR)
+          ,xlab="Years"
+      )
```

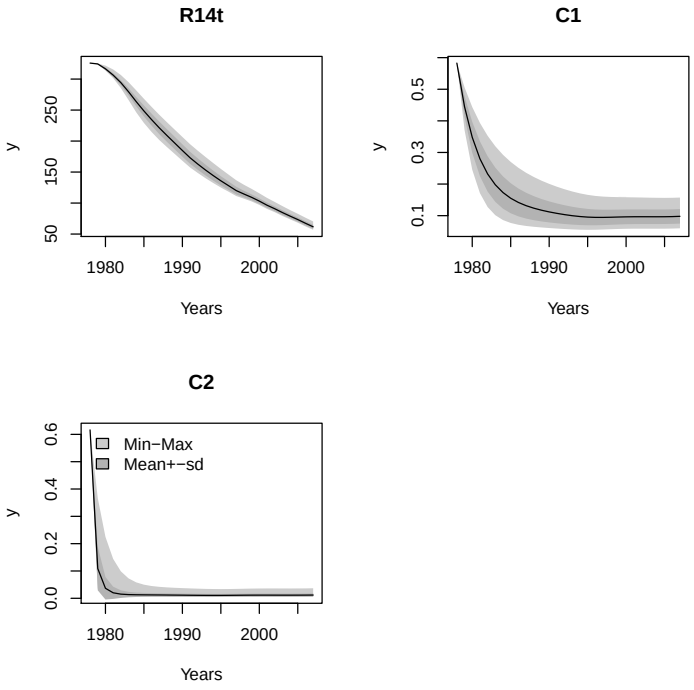


Figure 8: The plot shows the range for the model output based on the estimated distribution of the parameters

```
>     pairs(MCMC, nsample = niter/4)
```

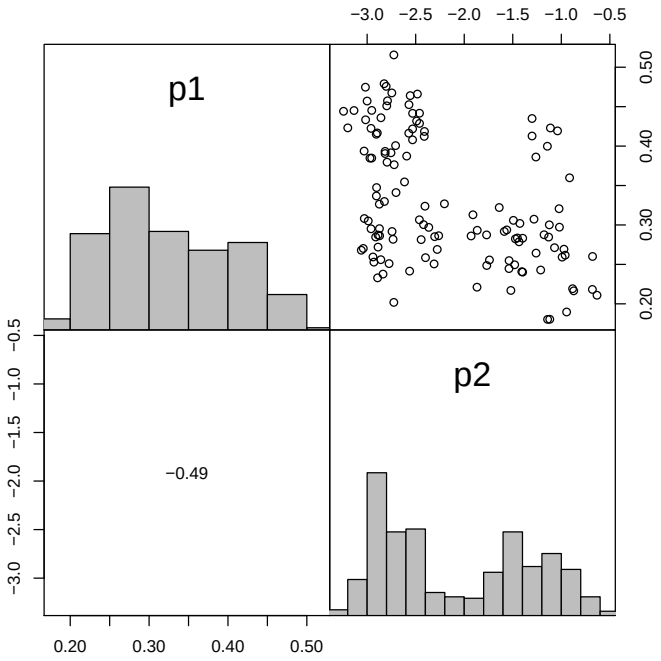


Figure 9: This plot is the analogon of the pairs plot for the sensitivity