Parameter Estimation of Compartment Models in `SoilR` Using Classical and Bayesian Optimization

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

June 25, 2019

1 Introduction

The objective of this document is to provide examples on how to use `SoilR` in combination with package `FME` to infer parameter values of soil organic matter decomposition models using observed data. Parameter estimation for dynamical systems is an advanced topic of inverse modeling and as such is far beyond the scope of this vignette. We will point to some principal questions and possible problems as they arise but this treatment will be far from comprehensive. This document also does not replace the documentation of package `FME` (Soetaert & Petzoldt, 2010) which we strongly recommend to consult. Instead, we show firstly, how a thin wrapper function makes a `SoilR` model available for the functions in `FME` and secondly how to choose the right parameterizations of `SoilR` models to meet the requirements of the `FME` algorithms. We present two examples. One is the parameterization of a two-pool model applied to a soil incubation experiment. The other example uses observed radiocarbon data from CO_2 measurements conducted at Harvard Forest, USA.

2 Example 1: A soil incubation experiment

Measurements of evolved CO_2 from incubation experiments can provide useful data for parameterizing soil organic matter decomopsition models and identify functionally distinct pools (Schadel et al., 2013). We present here data from an incubation experiment in which we measured the evolved CO_2 from a forest soil. The dataset, `eCO2`, is already included in `SoilR` and containes data from an incubation experiment with a boreal forest soil. First, we load `SoilR` into our `R` session and extract the data from the boreal site into a separate object; column names need to be renamed for consistency with `FME` .

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

```
library(SoilR)
library(FME)

## Loading required package:  rootSolve
## Loading required package:  coda

library(MASS)
library(lattice)
BorealCO2=eCO2
names(BorealCO2)<-c("time","eCO2","eCO2sd")
```

2.1 Principal identifiability of the model

Before we embroil ourselfes in technicalities we should think about the general possibility to identify the parameters from a single time line of the combined release of an yet unknown number of pools with yet unknown connections and outlets. We face several challenges, in particular we have to:

1. Find a class of models that is large enough to contain a model, capable of reproducing the observed data. If the class is to small we will "underfit" our data and the model will be "biased".

2. Find a number of parameters that is small enough to be actually determined unambigiously from the data. If we have parameters whose effects enhance or cancel the effects of other parameters the parameterized model will be "overfitted" and make (possibly extremely) misleading predictions for data not included in the training set. (In our case the predicted timeline of an overfitted model could meet all the measurements very well but be completely unreasonable in between or beyond the measurement times.)

3. Make sure that the parameters can at least be tested independently by the optimization procedure. (We refer here not to the desired independence of the parameters w.r.t (with respect to) the impact on the result as mentioned under 2. but to the shape of the set of valid parameters. For n parameters the algorithms that we will use except n ranges and assume to be able to choose *any* combination of parameters out of this n-dimensional rectangular set to form a *valid* model. E.g. an optimization algorithm that changes parameters independently should not accidentally create a model that breakes mass conservation or produces negative fluxes. `SoilR` allows many ways to specify models, including functions whose arguments include matrices whose parameters are *not* independent. Since it allways checks that the specified model is a valid compartmental system, it would actually stop the optimization procedure from trying unreasonable parameters.)

We will start with the simplest model that fulfills conditions 2. and 3. and extend it to improve the degree of accuracy until we are either satisfied or more likely can no longer guarantee 2.. Note that the search for accurate and identifyable soil models is an open research question Sierra et al. (2015). The possible answers are limited only by your igenuity and more severly by the

```
plot(BorealCO2[,1:2], xlab="Days", ylab="Evolved CO2 (mgC g-1
     soil day-1)",ylim=c(0,50))
arrows(BorealCO2[,1],BorealCO2[,2]-BorealCO2[,3],BorealCO2[,1],
       BorealCO2[,2]+BorealCO2[,3],code=3,angle=90,length=0.1)
```

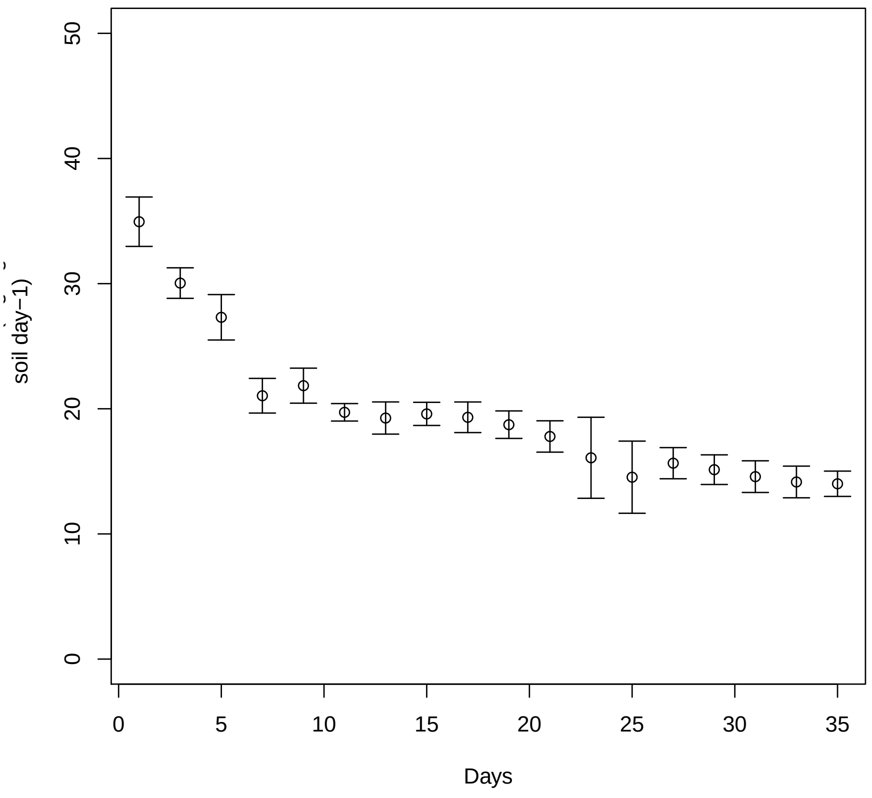

Figure 1: Cummulative evolved CO_2 from an incubation experiment with a boreal forest soil.

availability of data as the example will demonstrate. The FME function we want to use is modFit. Our first attempt to model the data is a one pool model with a constant decomposition rate k.

```
days=seq(0,35)
#C concentration * 450 g soil
Ctotal=mean(c(0.04683975, 0.04703255, 0.04687287))*450
#Changed units
BorealCO2=data.frame(
  time=BorealCO2[,1],
  BorealCO2[,2:3]*1e-06*Ctotal
)

eCO2func=function(pars){
  k1=pars[1]
  k2=1e-4*k1
  alpha=pars[2]
  gamma=.0001
  mod=TwopFeedbackModel(
    t=days
    ,ks=c(k1,k2)
    ,a21=alpha
    ,a12=0
    ,C0=Ctotal*c(gamma,1-gamma)
    ,In=0
    ,pass=TRUE
  )
  Rt=getReleaseFlux(mod)
  return(data.frame(time=days,eCO2=rowSums(Rt)))
}
```

Notice that our function, eCO2func, requires a vector of parameters pars with the values of the first decomposition rate in positions 1 and the values of the transfered proportion $\alpha_{2,1}$ in position 2. This function returns a data.frame with two columns, time in days and the sum of the cummulative release for the two pools.

The next step is to create a cost function according to FME requirements. This cost function takes as arguments a function with the model, the set of observations, and a measure of the error in the observations. The function calculates sums of squared residuals from the model output and the observed data, which can be further minimized for optimization.

```
eCO2cost=function(pars){
  modelOutput=eCO2func(pars)
  return(modCost(model=modelOutput, obs=BorealCO2, err="eCO2sd"))
}
```

This function returns an object of class modCost, which can be further used by FME . We strongly recommend to to read FME documentation for sensitivity and identifiability analyses. We will use the function modFit. It implements the optimization as an iterative process. First the cost function will be evaluated on the initial parameter values. Then the algorithm will try to guess new parameters, evaluate the cost function again and repeat this process until the cost is small enough or the number of permitted iterations exceeded. We can choose the optimization method (Levenberg-Marquardt in the example) which determines how the algorithm arrives determines the next parameters.

```
inipars=c( 1/20 ,1/100)
up=c(1,1)
lo=c(0,0)

eCO2fit=modFit(
    f=eCO2cost
    ,p=inipars
    ,method="Marq"
    ,upper=up
    ,lower=lo
)
```

To see the best set of parameter values found by the function we can type:

```
eCO2fit$par

## [1] 1.696373e-01 3.203139e-09
```

These set of parameters can be used now to run the model again and plot the obtained model against the observations

```
fitmod=eCO2func(eCO2fit$par)
```

The results from this optimization can be used for Bayesian parameter estimation with `FME` . For details about the procedure please see Soetaert & Petzoldt (2010). In our example, we need first to extract the variance from the prior optimization and used as the initial variance in the Bayesian procedure and to determine the *jump*, a value that determines how much a new parameter set is deviated from the old one. To avoid long compiling times in `SoilR` we only use 1000 iterations in this example, but this number can be much larger to guarantee convergence of the chains.

The results of the MCMC procedure can be obtained with the function `summary()`. The output gives the mean, standard deviation, min and max, and 25% quantiles for all parameter values. It also produces summary statistics for the variance of the observed variable.

```
summary(eCO2mcmc)

##                p1           p2    var_eCO2
## mean 0.171720039 4.618881e-03 1.90553e-09
## sd   0.004700158 5.383253e-03 0.00000e+00
## min  0.158077194 1.860336e-09 1.90553e-09
```

```r
plot(BorealCO2[,1:2], xlab="Days", ylab="Evolved CO2 (gC day-1)",ylim=c(0,9e-04))
arrows(BorealCO2[,1],BorealCO2[,2]-BorealCO2[,3],BorealCO2[,1],
       BorealCO2[,2]+BorealCO2[,3],code=3,angle=90,length=0.1)
lines(fitmod)
```

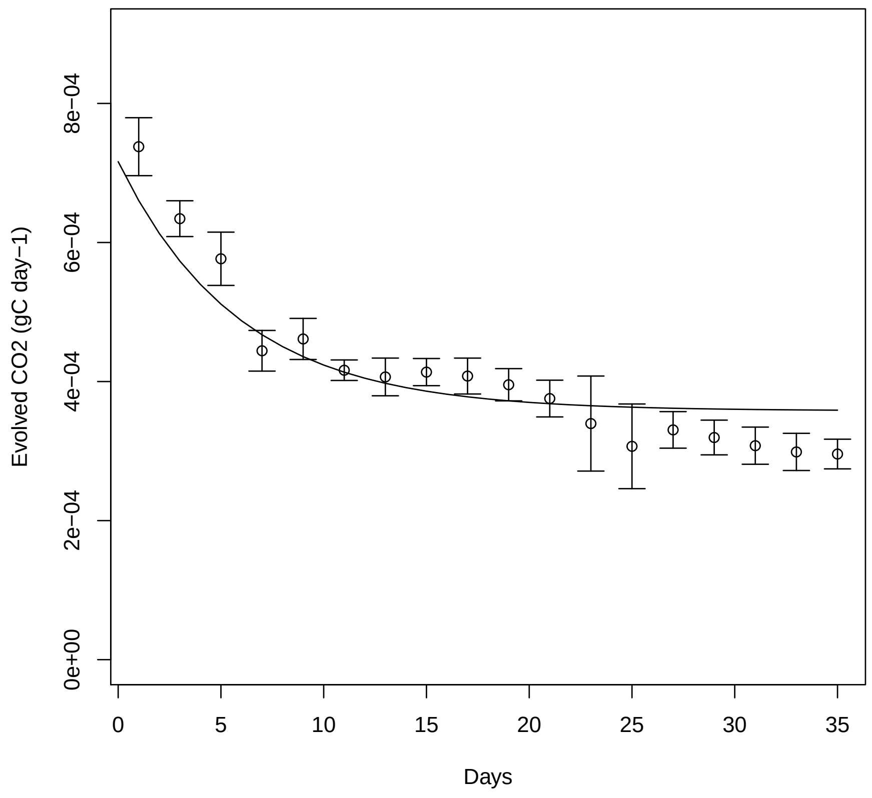

Figure 2: Best fit curve and observed data of CO2 evolved from an incubation experiment.

```
pairs(eCO2mcmc)
```

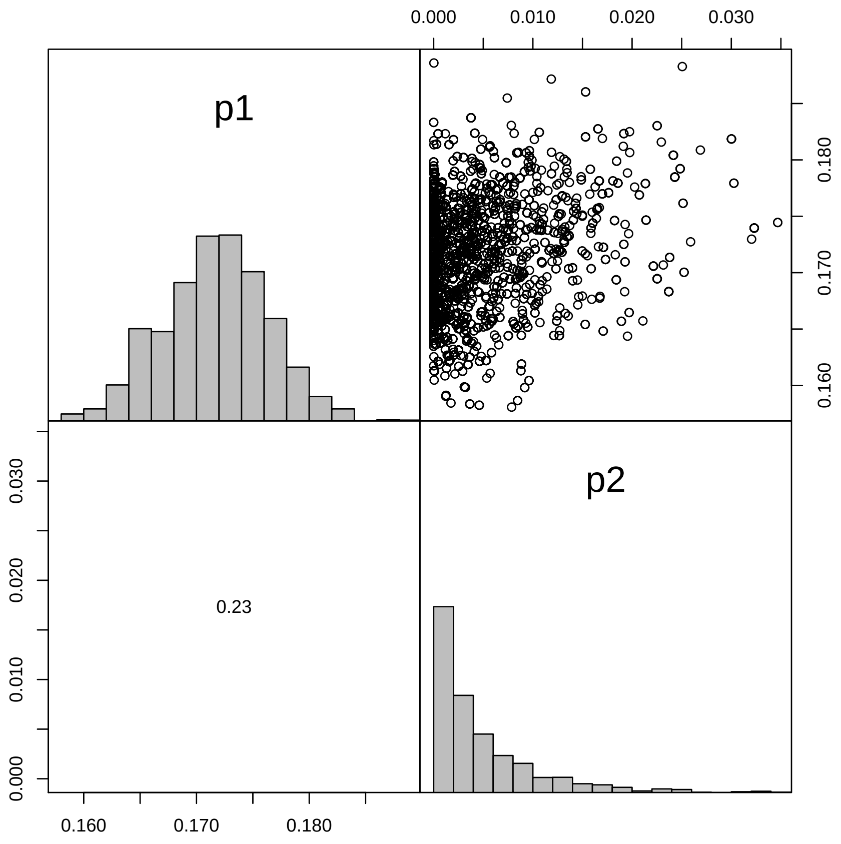

Figure 3: Histogram and scatter plots of the values obtained from the Markov chain Monte Carlo procedure.

```
## max  0.188593651 3.467826e-02 1.90553e-09
## q025 0.168401008 6.690192e-04 1.90553e-09
## q050 0.171847915 2.996371e-03 1.90553e-09
## q075 0.175018284 6.496190e-03 1.90553e-09

sR1=sensRange(func=eCO2func, parInput=eCO2mcmc$par)
```

A plot with the posterior distribution of the obtained parameter values can be obtained with function `pairs` (Figure 3)

For model prediction, it is also possible to use `FME` and function `sensRange` to obtain a graph of the model prediction with uncertainty ranges (Figure 4).

```
predRange=sensRange(func=eCO2func, parInput=eCO2mcmc$par)
plot(summary(predRange),ylim=c(0,9e-04),xlab="Days",
     ylab="Evolved CO2 (g C day-1)",main="")
points(BorealCO2)
arrows(BorealCO2[,1],BorealCO2[,2]-BorealCO2[,3],BorealCO2[,1],
       BorealCO2[,2]+BorealCO2[,3],code=3,angle=90,length=0.1)
```

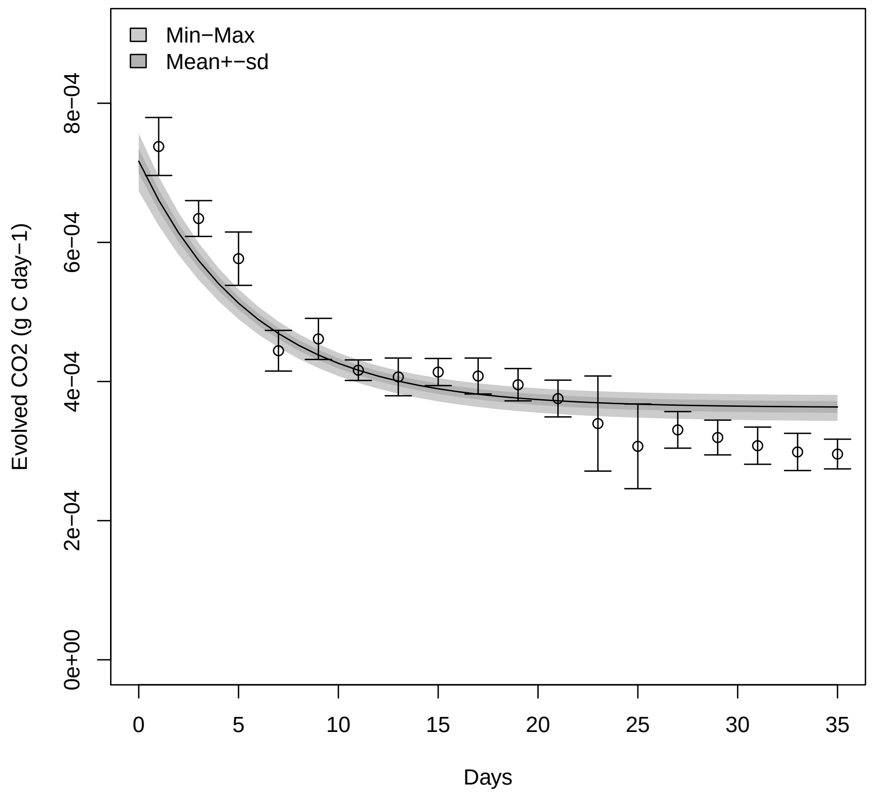

Figure 4: Model predictions using the set of parameters obtained from the MCMC procedure.

It is now obvious from this example that the workhorse of the parameter estimation procedure is the package `FME` of Soetaert & Petzoldt (2010). The main important task to learn about `SoilR` is how to set up the function that runs the model and obtains the variable of interest.

Example 2: Radiocarbon in respired CO_2

`SoilR` can also calculate the amount of radiocarbon in soils or in respired CO_2. Here, we take as an example a series of observations of radiocarbon in respired CO_2 conducted at Harvard Forest, USA. The dataset is also included in `SoilR` , and is visualized in Figure 5.

We are interested in fitting the following three-pool model to the data:

$$\frac{d\mathbf{C}(t)}{dt} = I \begin{pmatrix} \gamma_1 \\ \gamma_2 \\ 0 \end{pmatrix} + \begin{pmatrix} -k_1 & 0 & 0 \\ a_{21} & -k_2 & 0 \\ a_{31} & 0 & -k_3 \end{pmatrix} \begin{pmatrix} C_1 \\ C_2 \\ C_3 \end{pmatrix}. \tag{1}$$

where γ_1 and γ_2 are known. However the parameters $a_{21}, a_{31}, k_1, k_2, k_3$ are not independent, which is a condition for the parameter estimation with FME.

For this task, we simply need to prepare a model object in `SoilR` that can be further used by `FME` for parameter estimation. The radiocarbon content of CO_2 in the atmosphere is necessary for running the model, because it informs us about the concentration and rate of radiocarbon input to the soil. For this example we will use the dataset `C14Atm_ NH` provided with `SoilR` , but other provided datasets such as `Hua2013` can also be used.

First, we define the points in time to run the model from the atmospheric radicarbon dataset

```
time=C14Atm_NH$YEAR
t_start=min(time)
t_end=max(time)
```

To create the vector of input fluxes we need to create a new object of class `InFlux`. For our particular model, input fluxes to the C_1 and C_2 pools are created by this command

```
inputFluxes=InFlux( c(270,150,0))
```

assuming that pool 1 receives 270 gC m^2 yr^{-1} and pool 2 150 gC m^2 yr^{-1}.

The initial amount of carbon is created by aggregating the organic and mineral pools for this site reported in Sierra et al. (2012)

```
C0=c(390,220+390+1376,90+1800+560)
```

We now write a function that creates a `Model` object in `SoilR` that takes as arguments a set of parameters and returns the Δ^{14}C value of the respired carbon

```
## Warning in title(...):  font metrics unknown for Unicode
                         character U+2030
  ## Warning in title(...):  conversion failure on '()' in
          'mbcsToSbcs':  dot substituted for <e2>
  ## Warning in title(...):  conversion failure on '()' in
          'mbcsToSbcs':  dot substituted for <80>
  ## Warning in title(...):  conversion failure on '()' in
          'mbcsToSbcs':  dot substituted for <b0>
## Warning in title(...):  font metrics unknown for Unicode
                         character U+2030
  ## Warning in title(...):  conversion failure on '()' in
          'mbcsToSbcs':  dot substituted for <e2>
  ## Warning in title(...):  conversion failure on '()' in
          'mbcsToSbcs':  dot substituted for <80>
  ## Warning in title(...):  conversion failure on '()' in
          'mbcsToSbcs':  dot substituted for <b0>
  ## Warning in title(...):  conversion failure on '()' in
          'mbcsToSbcs':  dot substituted for <e2>
  ## Warning in title(...):  conversion failure on '()' in
          'mbcsToSbcs':  dot substituted for <80>
  ## Warning in title(...):  conversion failure on '()' in
          'mbcsToSbcs':  dot substituted for <b0>
```

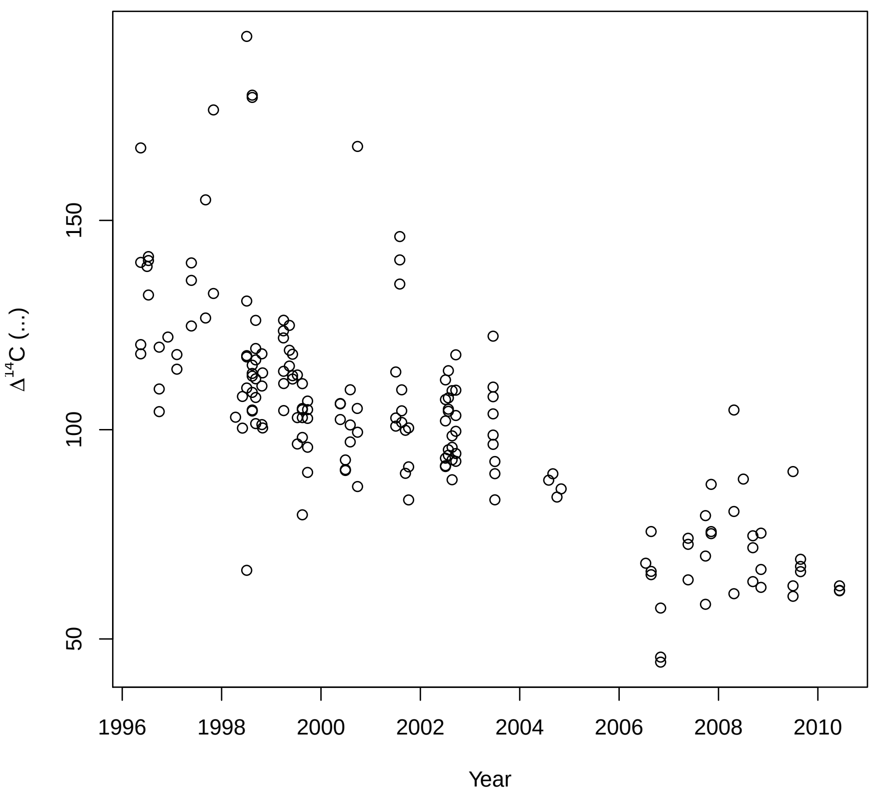

Figure 5: $\Delta^{14}C$ value of the respired CO_2 in a temperate broadleave forest at Harvard Forest, USA.

```
Fc=BoundFc(C14Atm_NH,lag=0,format="Delta14C")
np=3
Mod1<-function(ks,pass=TRUE){
  At=ConstLinDecompOp(
       internal_flux_rates=c("1_to_2"=ks[[4]],"1_to_3"=ks[[5]])
      ,out_flux_rates=c("1"=ks[[1]],"2"=ks[[2]],"3"=ks[[3]])
      ,numberOfPools = np

  )
  mod=GeneralModel_14(
        t=time,
        A=At,
        ivList=C0,
        initialValF=ConstFc(rep(0,3),"Delta14C"),
        inputFluxes=inputFluxes,
        inputFc=Fc,
        pass=TRUE
  )
  R14t=getF14R(mod)
  return(data.frame(time=time,R14t=R14t))
}
```

The observed data needs to be orginazed in a dataframe of the form

```
DataR14t=cbind(time=HarvardForest14CO2[,1],
               R14t=HarvardForest14CO2[,2],
               sd=sd(HarvardForest14CO2[,2]))
```

With all these elements ready, we can now use FME for the parameter optimization procedure. We will avoid a detailed explanation and present in the following the creation of the cost function, the initial optimization, and the final Bayesian parameter estimation.

```
## Error in modFit(f = R14tCost, p = c(0.1, 0.2, 0.3, 0.4, 0.5), lower
= rep(0, :  object 'nk' not found
## Error in eval(expr, envir, enclos):  object 'Fit' not found
## Error in summary(Fit):  object 'Fit' not found
## Error in modMCMC(f = R14tCost, p = Fit$par, niter = number_of_iterations,
:  object 'Fit' not found
```

The obtained posterior distributions of the parameters can now be assessed graphically (Figure 6). The final model with its uncertainty and how it compares to the data can now be shown(Figure 7).

```
pairs(MCMC,nsample=floor(number_of_iterations/4))

 ## Error in pairs(MCMC, nsample = floor(number_of_iterations/4)):
                              object 'MCMC' not found
```

Figure 6: Posterior parameter distributions for the parameters of the model described by equation 1. p1= k_1, p2= k_2, p3= k_4, p4= a_{21}, p5= a_{31}. Numbers in the lower diagonal indicate the correlation coefficient between parameters.

```
#The sensitivity range is calculated from the output of the MCMC
sR=sensRange(func=Mod1, parInput=MCMC$par)

 ## Error in sensRange(func = Mod1, parInput = MCMC$par):  object
                                   'MCMC' not found

par(mar=c(5,5,4,1))
plot(summary(sR),xlim=c(1950,2010),ylim=c(0,1000),xlab="Year",
     ylab=expression(paste(Delta^14,"C ","(\u2030)")),main="")

          ## Error in summary(sR): object 'sR' not found

points(DataR14t,pch=20)

## Error in plot.xy(xy.coords(x, y), type = type, ...):  plot.new
                              has not been called yet

lines(C14Atm_NH,col=4)

## Error in plot.xy(xy.coords(x, y), type = type, ...):  plot.new
                              has not been called yet
```

Figure 7: Predictions of respired radiocarbon values from the model of equation 1 versus observations. Model predictions include uncertainty range for the mean $\pm$ standard deviation, and the minimum-maximum range. Radiocarbon concentration in the atmosphere is depicted in blue.

Acknowledgements

We would like to thank Saadat Malghani for performing the laboratory incubation study. Susan E. Trumbore provided the radiocarbon data for Harvard Forest and gave important support and insights for the development of this project. Funding from the Max Planck Society.

References

Schädel, C., Y. Luo, R. David Evans, S. Fei, and S. Schaeffer. 2013 Separating soil CO_2 efflux into C-pool-specific decay rates via inverse analysis of soil incubation data. *Oecologia* 171: 721–732.

Sierra, C.A., Saadatullah Malghani and M. Müller. 2015 Model structure and parameter identification of soil organic matter models *Soil Biology and Biochemistry*, 90: 197-203.

Sierra, C.A., M. Müller, and S. E. Trumbore. 2012. Models of soil organic matter decomposition: the SoilR package, version 1.0. *Geosci. Model Dev.*, 5: 1045–1060.

Sierra, C.A., S. E. Trumbore, E. A. Davidson, S. D. Frey, K. E. Savage, and F. M. Hopkins. 2012. Predicting decadal trends and transient responses of radiocarbon storage and fluxes in a temperate forest soil. *Biogeosciences*, 9: 3013–3028.

Soetaert K. and T. Petzoldt. 2010. Inverse modelling, sensitivity and monte carlo analysis in R using package FME. *Journal of Statistical Software*, 33: 1–28.