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

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

November 14, 2023

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 first how a small wrapper function makes a `SoilR` model available for the functions in `FME` , and second, 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 First Example: A soil incubation experiment

Measurements of evolved CO_2 from incubation experiments can provide useful data for parameterizing soil organic matter decomposition 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 `incubation_experiment`, is already included in `SoilR` and contains data from an incubation experiment with a boreal forest soil. After loading `SoilR` into our `R` session we can explore its parts, extract the flux data from the boreal site into a separate object and plot it.

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

```
library(SoilR)

              ## Loading required package: deSolve
                                    ##
                   ## Attaching package: 'SoilR'
    ## The following object is masked from 'package:deSolve':
                                    ##
                              ##     euler

summary(incubation_experiment)

##                   Length Class      Mode
## eCO2              3      data.frame list
## c_concentrations 3      -none-     numeric
## soil_mass         1      -none-     numeric

BorealCO2=incubation_experiment$eCO2
plot(
        BorealCO2[,1:2]
        , xlab="Days"
        ,ylab="CO2 flux in  (mg C /g_soil/day)"
        ,ylim=c(0,50)
)
arrows(BorealCO2[,1],BorealCO2[,2]-BorealCO2[,3],BorealCO2[,1],
       BorealCO2[,2]+BorealCO2[,3],code=3,angle=90,length=0.1)
```

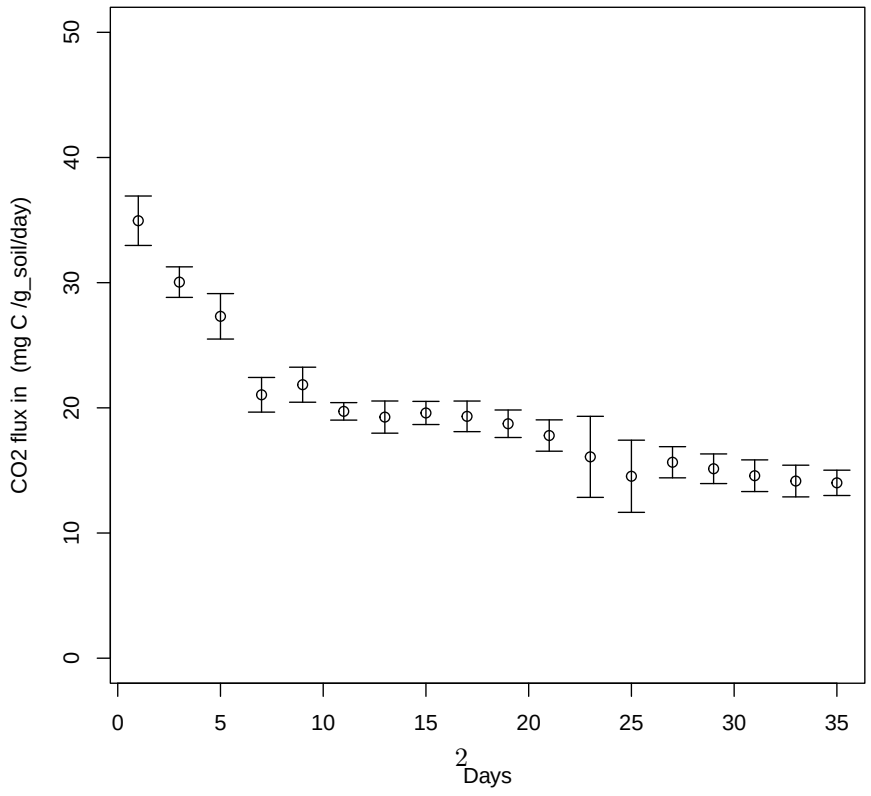

Figure 1: Respired CO_2 from an incubation experiment with a boreal forest soil.

The models in `SoilR` do not use units, but rather assume that the input data is consistent in this respect. If we want to supply the initial mass in g and the time in days, then we have to provide the flux in g per day, whereas the original data is given in mg carbon per g of soil per day. We therefore rescale the columns for the fluxes and the error.

```
Ctotal<- mean(incubation_experiment$c_concentrations) *
        incubation_experiment$soil_mass
BorealCO2<-data.frame(
         BorealCO2[,1]
        ,BorealCO2[,2:3]*1e-06*Ctotal)

# rename the columns
names(BorealCO2)<-c("time","eCO2","eCO2sd")
# create a function for later plots
plot_with_data<-function(x){
    plot(
        x=x
        ,xlab="Days"
        ,ylab="Evolved CO2 (gC/day)"
        ,ylim=c(0 ,9e-04)
        ,type='l'
    )
    points(
         BorealCO2
    )
    arrows(
        BorealCO2[,1]
        ,BorealCO2[,2]-BorealCO2[,3]
        ,BorealCO2[,1]
        ,BorealCO2[,2]+BorealCO2[,3]
        ,code=3
        ,angle=90
        ,length=0.1
    )
}
```

Before we embroil ourselves in technicalities we should think about the general possibility to identify the parameters from a single time line of the combined release of a 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 that is capable of reproducing the observed data. If the class is too 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 unambiguously 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 accept 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. For example, an optimization algorithm that changes parameters independently should not accidentally create a model that breaks 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 always checks that the specified model is a valid compartmental system, it would actually stop the optimization procedure from trying parameters that can not possibly constitute a soil model. We will choose parameterizations that avoid this situation.)

We will start with a simple model that fulfills conditions 2. and 3. and extend it to improve the degree of accuracy until we can no longer guarantee 2. Note that the search for accurate and identifiable soil models is an open research question. A recent study Sierra et al. (2015) suggests that for the possibilities to uniquely identify (structural) parameters from incubation data are severely limited. We can usually only hope to identify 2 or maybe 3 parameters. Our first attempt to model the data is therefore a very modest one pool model with a constant decomposition rate k.

2.1 One pool, one parameter model

The first `FME` function that we want to use is `modFit`. A look at its documentation (type `? modFit`) suggests that we at least have to provide a function `f` to be minimized, a vector of start values for the parameters, and in our case, a lower limit for the parameter k since we know that a negative value would create an invalid (not mass conserving) model. The function `f` will be a "costfunction" that will actually need to evaluate the model as a function of the parameter(s) at the times where we have data and compare the output to the measured data. It will return a vector of residuals (one entry for every measurement time) possibly weighted by the accuracy of our data. To help us write such a function, `FME` provides the function `modCost` to automatically create the cost function that compares the result of our model run with our data and computes the residuals weighted by the measurement errors if available. What remains to be done for us is to create a function of the parameter(s) that produces the release flux for the times in our dataset.

```
library(FME)

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

```
eCO2P1=function(pars){
    At=ConstLinDecompOp(
          out_flux_rates=ConstantOutFluxRateList_by_PoolIndex(
            list("1"=pars[[1]])
          )
         ,numberOfPools =1

    )
    mod=GeneralModel(
       t=BorealCO2[,'time']
      ,A=At
      ,ivList=c(Ctotal)
      ,inputFluxes=0
      #,pass=TRUE
    )
    Rt<-getReleaseFlux(mod)
    return(
    data.frame(time=BorealCO2[,'time'],eCO2=rowSums(Rt)))
}
```

Notice that our function, `eCO2func`, requires a (for this first example one dimensional) vector of parameters `pars` with the values of the first decomposition rate in position 1. Our function returns a `data.frame` with two columns, time in days and the sum of the release flux 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, weighted by the standard deviation of the measurement. For convenience we use the `FME` function `modCost` which returns an object of a class with the same name.

```
eCO2P1cost=function(pars){
 return(
   modCost(
     model=eCO2P1(pars)
     ,obs=BorealCO2
     ,err="eCO2sd"
   )
 )
}
```

Having defined the cost function we only need initial values (`inivars`) and (optional) lower and upper bounds (`upP1, loP1`) for the parameters to finally 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.

```
inipars=c( 1/2000 )
upP1=c(1)
loP1=c(0)

eCO2P1fit=modFit(
   f=eCO2P1cost
   ,p=inipars
   ,upper=upP1
   ,lower=loP1
)

## Error in (function (classes, fdef, mtable) : unable to find an inherited
method for function 'ConstLinDecompOp' for signature '"missing", "missing",
"ConstantOutFluxRateList_by_PoolIndex", "numeric", "missing"'
```

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

```
eCO2P1fit$par

## Error in eval(expr, envir, enclos): object 'eCO2P1fit' not found
```

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

```
plot_with_data(eCO2P1(eCO2P1fit$par))

## Error in h(simpleError(msg, call)): error in evaluating the argument
'x' in selecting a method for function 'plot': error in evaluating
the argument 'out_flux_rates' in selecting a method for function 'ConstLinDecompOp':
error in evaluating the argument 'object' in selecting a method for
function 'ConstantOutFluxRateList_by_PoolIndex': object 'eCO2P1fit'
not found
```

It is obvious that this model is too simple. The measured release flux is so small that - if all the carbon is decomposing - the decomposition rate must be also very small, indeed so small as to render the carbon almost stable. The expected exponential shape suggested by the data is not visible at all. We therefore probably can improve the fit significantly if we allow a second parameter determining how much of the total carbon is accessible for decomposition.

2.2 One pool, with two parameters

```
eCO2P1a=function(pars){
  At=ConstLinDecompOp(
      out_flux_rates=ConstantOutFluxRateList_by_PoolIndex(
        list("1"=pars[[1]])
      )
      ,numberOfPools =1
```

```
  )
  mod=GeneralModel(
     t=BorealCO2[,'time']
    ,A=At
    ,ivList=c(Ctotal*pars[2])
    ,inputFluxes=0
  )
  Rt<-getReleaseFlux(mod)
  return(data.frame(time=BorealCO2[,'time'],eCO2=rowSums(Rt)))
}

eCO2P1acost=function(pars){
  return(
    modCost(
      model=eCO2P1a(pars)
      ,obs=BorealCO2
      ,err="eCO2sd"
    )
  )
}
upP1a=c(1,1)
loP1a=c(0,0)

initPars=c(0.02, 0.01)
eCO2P1afit=modFit(
    f=eCO2P1acost
    ,p=initPars
    ,upper=upP1a
    ,lower=loP1a
)
```

```
## Error in (function (classes, fdef, mtable) : unable to find an inherited
method for function 'ConstLinDecompOp' for signature '"missing", "missing",
"ConstantOutFluxRateList_by_PoolIndex", "numeric", "missing"'
```

```
plot_with_data(eCO2P1a(eCO2P1afit$par))
```

```
## Error in h(simpleError(msg, call)): error in evaluating the argument
'x' in selecting a method for function 'plot': error in evaluating
the argument 'out_flux_rates' in selecting a method for function 'ConstLinDecompOp':
error in evaluating the argument 'object' in selecting a method for
function 'ConstantOutFluxRateList_by_PoolIndex': object 'eCO2P1afit'
not found
```

This time we also look at some estimated statistics.

```
eCO2P1afit$par
```

```
## Error in eval(expr, envir, enclos): object 'eCO2P1afit' not found
```

```
summary(eCO2P1afit)
```

```
## Error in eval(expr, envir, enclos): object 'eCO2P1afit' not found
```

According to these results we can say: Assuming a one pool model with constant decomposition rate with only a fraction of the total carbon accessible for decomposition, the best fitting model has a decomposition rate of `2.3% per day` where only `0.12%` of the total carbon are accessible for decomposition.

We can also check the estimated correlation between our two parameters. The high (anti)correlation indicates that it is difficult to identify the best model confidently since increasing the decomposition rate and decreasing the accessible fraction of carbon have similar effects on the model output (the release flux) .

The results of `modFit` can be used for Bayesian parameter estimation with `FME` 's Markov Chain Monte Carlo function `modMCMC`. In our example, we used the default values to avoid explanations about `FME` , but there are many possibilities to influence the result, which are briefly mentioned in the help (type `? modMCMC`) and explained in more detail in Soetaert & Petzoldt (2010). To avoid long runtimes, we only use 3000 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.

```
eCO2P1amcmc=modMCMC(
   f=eCO2P1acost
  ,p=eCO2P1afit$par
  ,niter=3000
  ,upper=upP1a
  ,lower=loP1a
)
```

```
## Error in eval(expr, envir, enclos): object 'eCO2P1afit' not found
```

```
summary(eCO2P1amcmc)
```

```
## Error in eval(expr, envir, enclos): object 'eCO2P1amcmc' not found
```

A plot with the posterior distribution of the obtained parameter values can be obtained with function `pairs`.

```
pairs(eCO2P1amcmc)
```

```
## Error in eval(expr, envir, enclos): object 'eCO2P1amcmc' not found
```

The strong correlation between the parameters is again apparent. We can also visualize the results of the Monte Carlos sampling by estimating the effect on the results (at different times) with function `sensRange` and plotting the results as a function of time.

```
predRange=sensRange(func=eCO2P1a, parInput=eCO2P1amcmc$par)
```

```
## Error in eval(expr, envir, enclos): object 'eCO2P1amcmc' not found
```

```
 plot_with_data(
  summary(predRange)
  )
```

```
## Error in h(simpleError(msg, call)): error in evaluating the argument
'x' in selecting a method for function 'plot': object 'predRange' not
found
```

Note that the results depend not only on our equations (as they would if we had sampled the complete parameter space with infinitely close parameter pairs), but also on the the other arguments of `modMCMC`, which for instance determine how many parameter combinations will be tested or how much they differ, or in general how well the parameter space has been sampled.

2.3 Two pool model with three parameters

As a last example we try to further improve the fit of the model by allowing a second pool with its decomposition rate as second parameter. The role of the first parameter as constant decomposition rate for the first pool does not change. The second parameter of the last model becomes the third here and decides now how the total carbon is distributed between the two pools.

```
eCO2P2=function(pars){
   At=ConstLinDecompOp(
       out_flux_rates=ConstantOutFluxRateList_by_PoolIndex(
          list("1"=pars[[1]],"2"=pars[[2]])
        )
       ,numberOfPools =2
   )
   gamma=pars[[3]]
   mod=GeneralModel(
      t=BorealCO2[,'time']
     ,A=At
     ,ivList=Ctotal*c(gamma,1-gamma)
     ,inputFluxes=c(0,0)
     ,pass=TRUE
   )
   Rt<-getReleaseFlux(mod)
   return(data.frame(time=BorealCO2[,'time'],eCO2=rowSums(Rt)))
}
#r=c(1:4,34:35)
eCO2P2cost=function(pars){
  return(
    modCost(
      model=eCO2P2(pars)
      ,obs=BorealCO2
      ,err="eCO2sd"
    )
  )
}
```

```
initPars=c(.02,.0001,0.999)
upP2=c(.1,.1,1)
loP2=c(0,0,0)
eCO2P2fit=modFit(
    f=eCO2P2cost
    ,p=initPars
    ,method="Marq"
    ,upper=upP2
    ,lower=loP2
)
```

```
## Error in (function (classes, fdef, mtable) : unable to find an inherited
method for function 'ConstLinDecompOp' for signature '"missing", "missing",
"ConstantOutFluxRateList_by_PoolIndex", "numeric", "missing"'
```

```
eCO2P2mcmc=modMCMC(
   f=eCO2P2cost
  ,p=eCO2P2fit$par
  ,niter=3000
  ,upper=upP2
  ,lower=loP2
)
```

```
## Error in eval(expr, envir, enclos): object 'eCO2P2fit' not found
```

```
plot_with_data(
    summary(
        sensRange(
            func=eCO2P2
            ,parInput=eCO2P2mcmc$par
        )
    )
)
```

```
## Error in h(simpleError(msg, call)): error in evaluating the argument
'x' in selecting a method for function 'plot': object 'eCO2P2mcmc'
not found
```

```
pairs(eCO2P2mcmc)
```

```
## Error in eval(expr, envir, enclos): object 'eCO2P2mcmc' not found
```

```
summary(eCO2P2fit)
```

```
## Error in eval(expr, envir, enclos): object 'eCO2P2fit' not found
```

We see again a high correlation between the parameters, which indicates poor identifiability. Numerical experiments (e.g. variation of the startvalues) also show that the algorithms occasionally get lost in local minima, so that including more pools or fluxes does not seem to be a very good idea. Note however that although we probably cannot hope to fit models that are structurally more complex there are infinitely more ways to choose parameters. We could for

instance define a time dependent decomposition rate whose shape depends on more than the one parameter we chose for a constant rate.

3 Second Example: 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 2.

```
plot(
    D14C~Year
    ,data=HarvardForest14CO2
    ,ylab=expression(paste(Delta^14,"C",' per mille'))
)
```

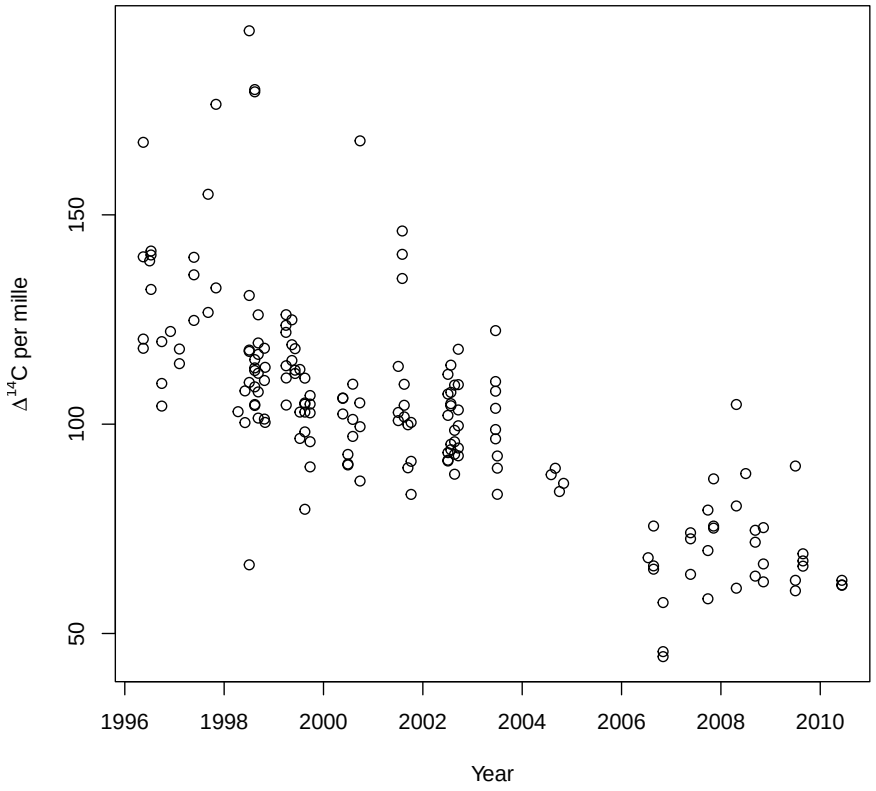

Figure 2: $\Delta^{14}C$ value of the respired CO_2 in a temperate broadleaf forest at Harvard Forest, USA.

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

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

where γ_1 and γ_2 are known. However the parameter a_{21} is not independent from k_2 , which is a condition for the parameter estimation with `FME` . We therefore formulate the model again based on the independent flux rates.

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 radiocarbon 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 `InFluxes`. For our particular model, input fluxes to the C_1 and C_2 pools are created by this command

```
inputFluxes=InFluxes( 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 can now create the complete `Model` object in `SoilR` . As before we wrap it in a function that takes as arguments a set of parameters and returns the $\Delta^{14}C$ value of the respired carbon for the desired times.

```
Fc=BoundFc(C14Atm_NH,lag=0,format="Delta14C")
Mod1<-function(ks,pass=TRUE){
    At=ConstLinDecompOp(
         internal_flux_rates=
         ConstantInternalFluxRateList_by_PoolIndex(
            list(
              "1->2"=ks[[4]]
              #,"1->3"=ks[[5]]
            )
          )
        ,out_flux_rates=ConstantOutFluxRateList_by_PoolIndex(
          list(
            "1"=ks[[1]]
            ,"2"=ks[[2]]
```

```
            ,"3"=ks[[3]]
          )
        )
        ,numberOfPools = 3

    )
    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))
}
```

To create the cost function we just slightly reorganize the observed data. Note that we do not have error bars for the measurements, so the cost function will not have to weight the residuals.

```
DataR14t=cbind(
     time=HarvardForest14CO2[,1]
    ,R14t=HarvardForest14CO2[,2]
)
#Create the cost function
R14tCost <- function(pars){
  R14t <- Mod1(pars)
  return(modCost(model=R14t,obs=DataR14t))
}
```

As before we perform an initial optimization, and the final Bayesian parameter estimation.

```
nk=4
up=rep(1,nk)
lo=rep(0,nk)
Fit <- modFit(
        f=R14tCost
        ,p=c(.01,.02,.03,.9)
        ,lower=lo
        ,upper=up
)
```

```
## Error in h(simpleError(msg, call)): error in evaluating the argument
'internal_flux_rates' in selecting a method for function 'ConstLinDecompOp':
unable to find an inherited method for function 'ConstantInternalFluxRate_by_PoolIndex'
for signature '"missing", "missing", "character", "numeric"'
```

```r
var0 <- Fit$var_ms_unweighted
```

```
## Error in eval(expr, envir, enclos): object 'Fit' not found
```

```r
cov0 <- summary(Fit)$cov.scaled
```

```
## Error in eval(expr, envir, enclos): object 'Fit' not found
```

```r
MCMC <- modMCMC(
    f=R14tCost
   ,p = Fit$par
   ,niter = 1000
   ,covscale= cov0
   ,var0 = var0
   #,wvar0 = 0
   ,lower=lo
   ,upper=up
)
```

```
## Error in eval(expr, envir, enclos): object 'Fit' not found
```

```r
pairs(MCMC)
```

```
## Error in eval(expr, envir, enclos): object 'MCMC' not found
```

```r
summary(Fit)
```

```
## Error in eval(expr, envir, enclos): object 'Fit' not found
```

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

## Error in eval(expr, envir, enclos): 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 ","per mille")),main="")

## Error in h(simpleError(msg, call)): error in evaluating the
argument 'x' in selecting a method for function 'plot': 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 3: Predictions of respired radiocarbon values from the model of equation 1 versus observations. Model predictions include uncertainty range for the mean ± 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.