Implementation of a simple ecosystem model in R

Instructor: Carlos A. Sierra
Max Planck Institute for Biogeochemistry

April 24, 2020

In this document we will implement a simple eight pool model in R using the SoilR package. See the attached documentation about details on the model.

Initial set up

The first thing to do is to download SoilR from GitHub. You can do this directly in your R session by typing

```
> install.packages("devtools")
> devtools::install_github('MPIBGC-TEE/SoilR-exp/pkg')
```

Then, load SoilR into R by running the command

```
> library(SoilR)
```

Now, we will call an external file that contains the basic implementation of the model following the SoilR syntax. This file contains the implementation of the model as a function. The name of the function is `modWengLuo`.We do not need to worry too much about the contents of this file. However, if you are interested in learning the basic workflow in SoilR you are encourage to look at the contents of this file. For the moment, we only need to tell R to load the code by writing

```
> source("https://raw.githubusercontent.com/crlsierra/lectNotes_TB/master/modWengLuo.R")
```

Notice that I am telling R to load the file from an address in GitHub. You can also save this file locally in your compauter and read it from there.

The next step is to create a series of numbers representing the times in which we want the model to run. In this case we will run the model for 500 years on a daily time-step

```
> days=seq(from=0,to=365*500)
```

In this case, we will be running our simulations for 182,500 days, but you can change this at any time, specially if the simulations are running slow in your own computer.

The inputs of carbon from photosynthetic products are defined by the value of $U = 3.37$ g C day^{-1}, which we can input as

```
> U=3.370
```

and the partitioning coefficients in the vector $\boldsymbol{b}$ can be given as

```
> b=c(0.14,0.26,0.14,rep(0,5))
```

The next step is to create the matrix $\mathbf{C}$ with the values of the coefficients that determine the proportion of C that leaves each pool, which is a diagonal matrix and can be easily created as

```
> C=diag(c(0.00258,0.0000586,0.00239,0.0109,0.00095,0.0105,
+          0.0000995,0.0000115))
```

To create the matrix with the transfer coefficients, we input the values of the coefficient first and then create the matrix as

```
> f41=0.9; f43=0.2
> f51=0.1; f52=1; f53=0.8
> f64=0.45; f65=0.275; f67=0.42; f68=0.45
> f75=0.275; f76=0.296
> f86=0.004;f87=0.01
> A=matrix(c(-1, 0, 0, 0, 0, 0, 0, 0,
+            0, -1, 0, 0, 0, 0, 0, 0,
+            0, 0, -1, 0, 0, 0, 0, 0,
+            f41, 0, f43, -1, 0, 0, 0, 0,
+            f51, f52, f53, 0, -1, 0, 0, 0,
+            0, 0, 0, f64, f65, -1, f67, f68,
+            0, 0, 0, 0, f75, f76, -1, 0,
+            0, 0, 0, 0, 0, f86, f87, -1),
+          byrow=TRUE,nrow=8,ncol=8)
>
```

The last step to solve the model is to give a vector of initial values for the amount of C in all pools. We will use this vector of initial conditions

```
> X0=c(250,4145,192,93,545,146,1585,300)
```

Simulation experiments

Now we have all elements to run the model. We will run the model a few times with different settings so we can explore different behaviors. First, we will run one simulation that will serve as the 'control', i.e. a simulation that we will compare against other simulation in which we make changes to this set of parameters. We will run the model by calling the function `modWengLuo` that we sourced at the begining of this session. The output of this model call will be stored in the object `ctrl`

```
> ctrl=modWengLuo(t=days,U=U,b=b,A=A,C=C,X0=X0,xi=1,pass=TRUE)
```

This call to `modWengLuo` only performs initial checks on the input information for the model. Now we need to ask the model for the information we want to obtain from it. To obtain the amount of C stored in the 8 different pools over time we write

```
> ctrlCt=getC(ctrl)
```

To see the output of the model we can quickly check the first results of the simulation by calling

```
> head(ctrlCt)
```

```
         [,1]     [,2]     [,3]     [,4]     [,5]     [,6]     [,7]     [,8]
[1,] 250.0000 4145.000 192.0000 93.00000 545.0000 146.0000 1585.000 300.0000
[2,] 249.8270 4145.633 192.0129 92.66023 545.1567 145.1371 1585.437 300.0042
[3,] 249.6545 4146.267 192.0258 92.32375 545.3132 144.2815 1585.872 300.0084
[4,] 249.4824 4146.900 192.0386 91.99053 545.4697 143.4334 1586.303 300.0126
[5,] 249.3108 4147.533 192.0514 91.66053 545.6260 142.5925 1586.732 300.0168
[6,] 249.1396 4148.166 192.0642 91.33372 545.7821 141.7589 1587.159 300.0209
```

This output presents the value of the amount of C for the 8 different pools (columns) for each time step (rows). Therefore, the dimension of this matrix is 8×182500. We can also ask SoilR for the amount of respiration from the eight different pools with the command

```
> ctrlRt=getReleaseFlux(ctrl)
```

The format of the output for this function is similar as the previous output.

Our next step is to run two more simulations in which we double or decrease by half the values of the elements in the matrix $\mathbf{C}$. In other words, we will look at the effect of either doubling or reducing by half the rates of carbon flow out of the pools. For example, changes in climate by 10°C can double these rates, so this is a way to explores the effects of climate change on carbon storage and CO_2 exchanges with the atmosphere. We can achieve this by modifying the value of `xi` in the function `modWengLuo`. In the case of doubling, we can run the model as

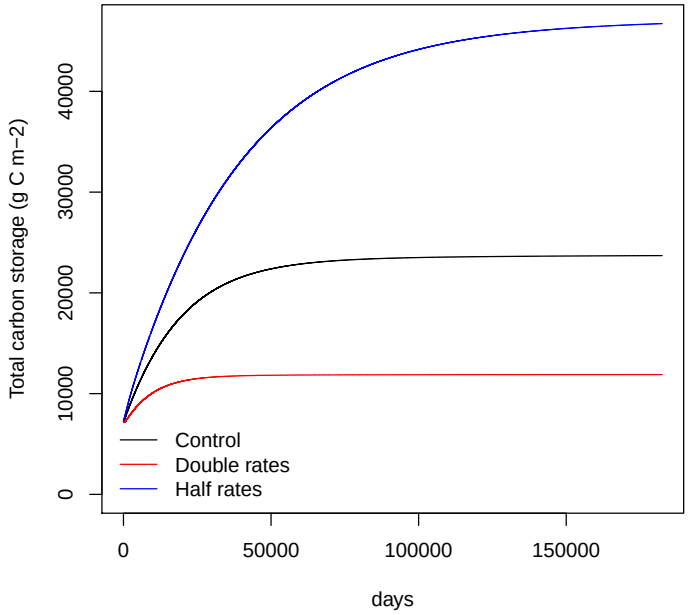
```
> ratesx2=modWengLuo(t=days,U=U,b=b,A=A,C=C,X0=X0,xi=2,pass=TRUE)
> ratesx2Ct=getC(ratesx2)
> ratesx2Rt=getReleaseFlux(ratesx2)
```

and in the case of reducing the rates by half we write

```
> ratesx05=modWengLuo(t=days,U=U,b=b,A=A,C=C,X0=X0,xi=0.5,pass=TRUE)
> ratesx05Ct=getC(ratesx05)
> ratesx05Rt=getReleaseFlux(ratesx05)
```

We can now evaluate the output graphically, plotting our results for the three simulations. Let's look first at the time evolution of the total carbon stocks. Because the output we obtained is in the form of a matrix we need to sum over all pools at each time step to obtain the total amount of carbon. We will use the `rowSums` function for this purpose

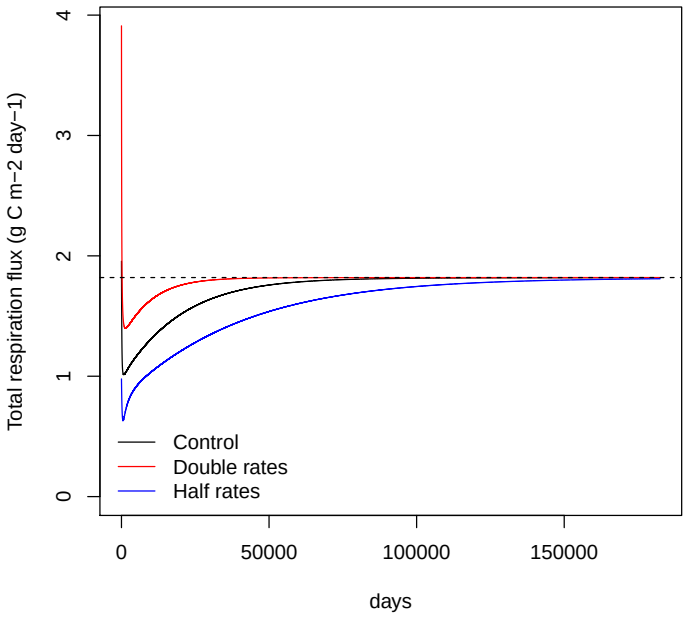
```
> plot(days,rowSums(ctrlCt),type="l",ylab="Total carbon storage (g C m-2)"
+      ,ylim=c(0,max(rowSums(ratesx05Ct))))
> lines(days,rowSums(ratesx2Ct),col=2)
> lines(days,rowSums(ratesx05Ct),col=4)
> legend("bottomleft",c("Control","Double rates","Half rates"),
+        lty=1,col=c(1,2,4),bty="n")
```



There are two important things to look at here. 1) decreasing flow rates increases the amount of carbon storage while increasing flow rates decreases the amount of carbon storage. In other words, carbon storage and flow rates are inversily related. 2) As the flow rates decrease the amount of time required to reach equilibrium increases.

We can also look at the amount of carbon respired from this ecosystem plotting the results from the three simulations

```
> plot(days,rowSums(ctrlRt),type="l",ylab="Total respiration flux (g C m-2 day-1)",
+      ylim=c(0,max(rowSums(ratesx2Rt))))
> lines(days,rowSums(ratesx2Rt),col=2)
> lines(days,rowSums(ratesx05Rt),col=4)
> legend("bottomleft",c("Control","Double rates","Half rates"),
+        lty=1,col=c(1,2,4),bty="n")
> abline(h=U*(0.14+0.26+0.14),lty=2)
```



These results also present some interesting aspects related to the structure of the model. 1) All simulations converge to a constant respiration flux independent on the value of the flow rates. Remember that at steady-state equilibrium the outputs are equal to the inputs and because in all cases the inputs are the same ($U \cdot \boldsymbol{b}$), then the three simulations converge to a common value of 1.82 g C m^{-2} day^{-1}. 2) The time required to reach equilibrium depends on the value of the flow rates. The lower the flow rates the longer it takes to reach equilibrium. 3) In the three simulations there were an initial flow of respired C associated with the initial amount of C present as initial value. As carbon starts accumulating over time, this *legacy effect* decreases.

Ages and transit times

We can calculate now the system age and the transit time distribution of this model, which are implemented in SoilR throught the function `systemAge` and `transitTime`. First we need to create a sequence of ages for plotting the results, and convert the elements of $\boldsymbol{b}$ as an R vector

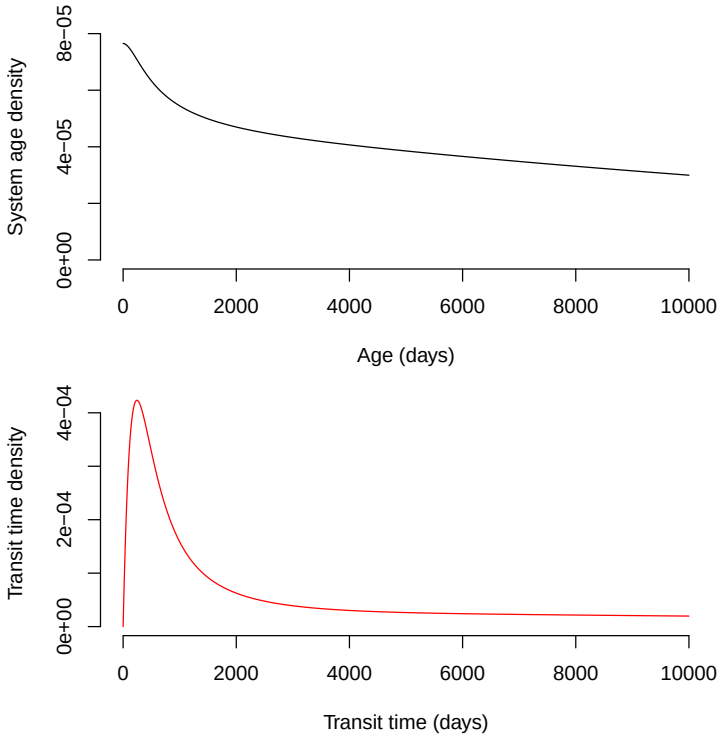
```
> tau=seq(0,10000)
> u=matrix(b,nrow=8,ncol=1)
```

To run the `systemAge` and `transitTime` we also need to multiply the **A** and **C** matrices

```
> aLW=systemAge(A=A%*%C, u=u,a=tau)
> tLW=transitTime(A=A%*%C, u=u,a=tau)
```

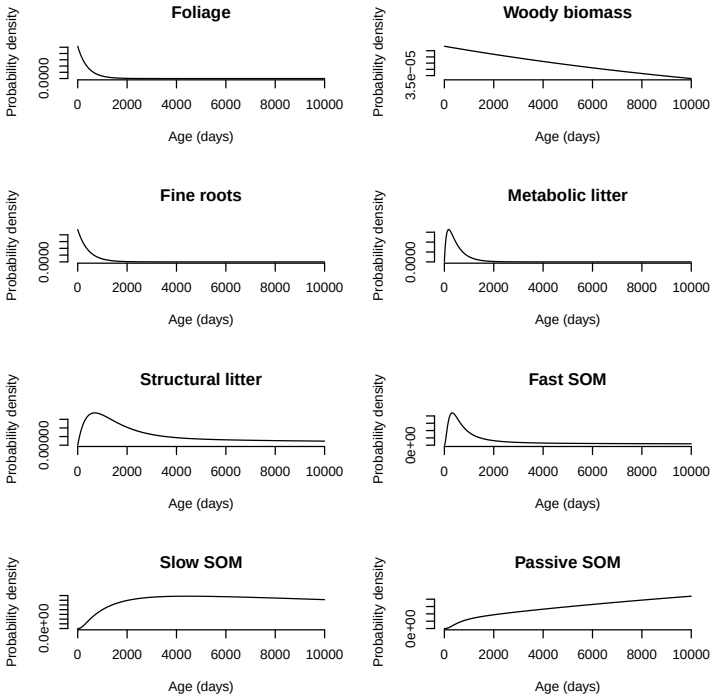
Theses functions return a list with the probability density functions, the mean values, and quantiles of the distributions. We can plot the results as

```
> par(mfrow=c(2,1), mar=c(4,4,1,1))
> plot(tau,aLW$systemAgeDensity,type="l",ylim=c(0,0.00008), xlab="Age (days)",
+      ylab="System age density", bty="n")
> plot(tau,tLW$transitTimeDensity,col=2, type="l", xlab="Transit time (days)",
+      ylab="Transit time density", bty="n")
> par(mfrow=c(1,1))
```



We can also explore the age distribution for each individual pool in the model as

```
> poolnames=c("Foliage", "Woody biomass", "Fine roots", "Metabolic litter",
+             "Structural litter", "Fast SOM", "Slow SOM", "Passive SOM")
> par(mfrow=c(4,2))
> for(i in 1:8){
+   plot(tau,aLW$poolAgeDensity[,i],type="l", main=poolnames[i],
+        ylab="Probability density", xlab="Age (days)", bty="n")
+ }
> par(mfrow=c(1,1))
```



Exercises

Now it is your turn to run your own simulation experiments and evaluate the behavior of the model under different assumptions. The following exercises would guide you through.

- Effects of disturbance on long-term carbon storage
  - Run a simulation in which you start an ecosystem from bare ground; i.e. the vector of initial conditions has only zeros.
  - Run a simulation in which the starting conditions are much higher than the steady-state values. For example, you can run the model with twice as much carbon as the steady-state value of the `ctrl` simulation.
  - Run other simulations in which your initial conditions are 75, 50, and 25% of the steady-state value of the `ctrl` simulation.
  - What can we say about the general behavior of the model? What can you infer about an ecosystem that experiences a disturbance such as fire?
- For the cases in which rates where doubled and reduced by half, calculate the mean age and the mean transit time. How did these changes in rates

affected the ages and transit times of the system? What pools changed more drastically?

- It is common practice to calculate the *turnover times* of carbon for each reservoir. These turnover times τ are defined in different ways. One definition is as the inverse of the absolute value of the eigenvalues of the product of the matrix $\mathbf{A} \cdot \mathbf{C}$, that is

$$\tau_i = \frac{1}{|\lambda_i|} \tag{1}$$

  - Calculate the eigenvalues of the matrix that result by multiplying $\mathbf{A} \cdot \mathbf{C}$. In R, matrix multiplication is performed by the operator `%*%`, and eigenvalues are calculated by the function `eigen`.
  - Take the absoulte value of these eigenvalues and divide them by 1.
  - What is the range of timescales of cycling for these model? Which ones do you think cycle fast and which ones cycle slow?
  - An alternate definition of turnover times is the inverse of the elements of the diagonal of the matrix $\mathbf{C}$. Calculate these values and compare them with the ones we previously obtained.
  - Calculate the mean age for each pool. Compare the results from the previous approaches.

- Plot the carbon stocks in the different pools for the `ctrl` simulation. For this you need to use the function `plotCPool` from SoilR. Look at the documentation of this function by typing `?plotCPool`.