Hierarchical Bayesian Modeling of CO2 Flux

Introduction

In this document, we study synthetic soil Carbon data generated by the century model. Each flux data has several replicates, corresponding to different incubation test done on the soil from the same sample. We consider a hierarchical Bayesian model, and examine the estimates as (i) the amount of input data is increased, and (ii) the parameter priors get stronger.

Century Model

The century model is proposed by Parton et al. (1988). We only focus on the three soil pools in this model which are listed below along with their expert-tuned decomposition rates:

$$\begin{aligned} &\text{pool 1: Active Soil C}, && \kappa_1 \approx 1/1.5, \\ &\text{pool 2: Slow Soil C}, && \kappa_2 \approx 1/25, \\ &\text{pool 3: Passive Soil C}, && \kappa_3 \approx 1/1000, \end{aligned}$$

where κ denotes the decay rate which is defined as 1 over the turnover.

We denote the transfer rate from pool j to pool i by r_{ij}. The transfer rates are parameterized as a ratio of the decay rate: $r_{ij} = \alpha_{ij}\kappa_j$. For the Century model, we have:

$$\begin{aligned} \alpha_{21} &= 1 - F(\mathrm{T}) - 0.004, \\ \alpha_{31} &= 0.004, \\ \alpha_{12} &= 0.42, \\ \alpha_{32} &= 0.03, \\ \alpha_{13} &= 0.45, \\ \alpha_{23} &= 0, \end{aligned}$$

where 'T' is the soil silt + clay content, and $F(\mathrm{T}) = 0.85 - 0.68 \times \mathrm{T}$. These values are expert-tuned; however, in our Bayesian framework, we estimate them from the data. For each pool, we can write the following differential equation:

$$\frac{dC_i(t)}{dt} = -\kappa_i C_i(t) + \sum_{j \neq i} \alpha_{ij}\kappa_j.$$

Combining all these differential equations into a single formula, we get:

$$\frac{dC(t)}{dt} = \begin{pmatrix} -\kappa_1 & \alpha_{12}\kappa_2 & \alpha_{13}\kappa_3 \\ \alpha_{21}\kappa_1 & -\kappa_2 & 0 \\ \alpha_{31}\kappa_1 & \alpha_{32}\kappa_2 & -\kappa_3 \end{pmatrix} C(t).$$

The total amount of Carbon in the beginning is C_{tot} which is divided among the three pools: $C(0) = (\gamma_1, \gamma_2, \gamma_3) \cdot C_{tot}$.

Simulating Synthetic Data

We simulate data according to the century model. Before presenting the `R` code, we need to clarify some details.

CO_2 flux, sampling times, and cap times

We assume that our observations are in terms of CO_2 fluxes. Theoretically, the CO_2 flux at time t is defined as $\sum_{i=1}^{3} |dC_i(t)/dt|$, i.e., the rate of CO_2 leaving the soil. In experiments, this is calculated by measuring the CO_2 emitted between a cap time, t_{cap}, and a measurement time, t_{meas}:

$$\text{flux}(t_{meas}) = \Delta CO_2/(t_{meas} - t_{cap})$$

.

Generally, the sampling times are not uniform. The CO_2 flux is sampled more frequently in the beginning. A common practice is to sample once per day for the first week, then once per week for the first month, and then once per month. Given that the scale of the turnover rates are in years, this amounts to the following measurement times:

$$t_{meas} = (\frac{1}{360}, \frac{2}{360}, \frac{3}{360}, \frac{4}{360}, \frac{5}{360}, \frac{6}{360}, \frac{7}{360}, \frac{14}{360}, \frac{21}{360}, \frac{28}{360}, \frac{60}{360}, \frac{90}{360}, \frac{120}{360}, \ldots, \frac{360}{360})$$

The cap times might vary in different experiments. Here, we assume that the cap times are halfway between previous and current measurements:

$$t_{cap} = (\frac{0.5}{360}, \frac{1.5}{360}, \frac{2.5}{360}, \frac{3.5}{360}, \frac{4.5}{360}, \frac{5.5}{360}, \frac{6.5}{360}, \frac{10.5}{360}, \frac{17.5}{360}, \frac{24.5}{360}, \frac{45}{360}, \frac{75}{360}, \frac{105}{360}, \ldots, \frac{345}{360})$$

One of the goals of this report to increase the number of data points and examine the corresponding fits.

Replications

Generally, we have several replications for each experiment (i.e., several CO_2 fluxes). We assume that these replications differ slightly in the transfer rates but have the same γ and turnover rates (given that they are from the same soil sample). The way we model the variation in the turnover rates is through a hirarchical Dirichlet model. The vector $(\alpha_{1j}^i, \alpha_{2j}^i, \alpha_{3j}^i)$ is a simplex. The superscript i refers to the i'th replicate and index j refers to the j'th pool. We assume that there are global simplex vectors $(\alpha_{1j}^0, \alpha_{2j}^0, \alpha_{3j}^0)$ such that we have the following hierarchical structure:

$$(\alpha_{1j}^0, \alpha_{2j}^0, \alpha_{3j}^0) \sim \text{Dirichlet}(1,1,1), (\alpha_{1j}^i, \alpha_{2j}^i, \alpha_{3j}^i) \sim \text{Dirichlet}(\kappa \times (\alpha_{1j}^0, \alpha_{2j}^0, \alpha_{3j}^0)),$$

where $\kappa \sim \text{Pareto}(1, 1.5)$.

R code

The R function for simulating the data is shown below. The details are described above.

```r
simulate_data_century <- function(t_meas, t_cap, init_C, num_rep) {
# INPUTS:
#   t_meas:  measurement times
#   t_cap:   cap times
#   init_C:  initial pool contents
#   num_rep: number of replications
library(deSolve)
library(gtools)
genDerivs <-function(t, Ct, params) {
  # General diff eq model: dC_dt = I(t) + A(t)*C(t)
  # INPUTS:
  #   t: time
  #   Ct: the value of the vector C at time t, C(t)
  #   params: it has two fields, params$I and params$A
  dC_dt = params$I + params$A %*% Ct;
  return(list(dC_dt));
}
m <- 3; # number of pools
C_t0 <- matrix(init_C, nrow=m);
turnover <- c(1.5, 25, 1000);
K <- 1/turnover;
I <- rep(0, m); # no input flux for century
N_t <- length(t_meas);
CO2_flux_mat <- matrix(NA, nrow = N_t, ncol = num_rep);
Alpha_rep <- array(0, c(m, m, num_rep));
Alpha <- matrix(0, m, m);
# Setting global transfer rates with expert-tuned values:
Alpha[2, 1] = 0.5;
Alpha[3, 1] = 0.004;
Alpha[1, 2] = 0.42;
Alpha[3, 2] = 0.03;
Alpha[1, 3] = 0.45;
Alpha[1, 1] = 1 - Alpha[2, 1] - Alpha[3, 1];
Alpha[2, 2] = 1 - Alpha[1, 2] - Alpha[3, 2];
Alpha[3, 3] = 1 - Alpha[1, 3] - Alpha[2, 3];
for (this_rep in 1:num_rep) {
  # Hierarchical modeling of transfer rates for replications:
  kappa <- 10;
  Alpha_rep[,1, this_rep] <-  rdirichlet(1, Alpha[,1] * kappa);
  Alpha_rep[,2, this_rep] <-  rdirichlet(1, Alpha[,2] * kappa);
  Alpha_rep[,3, this_rep] <-  rdirichlet(1, Alpha[,3] * kappa);
  Alpha_rep[1, 1, this_rep] <- 0;
  Alpha_rep[2, 2, this_rep] <- 0;
  Alpha_rep[3, 3, this_rep] <- 0;
  A <- Alpha_rep[,, this_rep] * matrix(rep(K, m), nrow = m, byrow = TRUE) - diag(K);
  params <- list(I=I, A=A);
  t0 <- 0;
  # Solving the ODE system for given parameters:
  meas_data<-ode(y = C_t0, func = genDerivs,
          times = c(t0,t_meas), parms = params);
  cap_data<-ode(y = C_t0, func = genDerivs,
                 times = c(t0,t_cap), parms = params);
  # Calculating CO2 flux
  totalC_t0 = sum(meas_data[1,2:(m+1)]);
  CO2_t_meas <- totalC_t0 - rowSums(meas_data[2:nrow(meas_data), 2:(m+1)]);
  CO2_t_cap <- totalC_t0 - rowSums(cap_data[2:nrow(cap_data),2:(m+1)]);
  CO2_flux <- (CO2_t_meas - CO2_t_cap)/(t_meas-t_cap);
  # Adding log-normal noise:
  CO2_flux_mat[, this_rep] <- exp(log(CO2_flux) + rnorm(length(CO2_flux),0,.05));
}
```

```
simulated_data <- list(N_t = N_t, t_meas = t_meas, t_cap = t_cap,
                       num_rep = num_rep, totalC_t0 = totalC_t0,
                       t0=t0, CO2_flux=CO2_flux_mat, Alpha_rep=Alpha_rep);
return(simulated_data);
}
```

Stan code

```
// Fitting the century model with partial pooling
functions {
  /**
  * ODE system for the Century model with no input fluxes.
  * @param t time at which derivatives are evaluated.
  * @param C system state at which derivatives are evaluated.
  * @param theta parameters for system.
  * @param x_r real constants for system (empty).
  * @param x_i integer constants for system (empty).
  */
    real[] century_model(real t, real[] C, real[] theta,
                         real[] x_r, int[] x_i) {
      real k[3];
      real a21;
      real a31;
      real a12;
      real a32;
      real a13;
      real dC_dt[3];
      k = theta[1:3];
      a21 = theta[4];
      a31 = theta[5];
      a12 = theta[6];
      a32 = theta[7];
      a13 = theta[8];
      dC_dt[1] = -k[1] * C[1] + a12 * k[2] * C[2] + a13 * k[3] * C[3];
      dC_dt[2] = -k[2] * C[2] + a21 * k[1] * C[1];
      dC_dt[3] = -k[3] * C[3] + a31 * k[1] * C[1] + a32 * k[2] * C[2];
      return dC_dt;
    }

  /**
  * Compute evolved CO2 from the system given the specified
  * parameters and times. This is done by simulating the system
  * defined by the ODE function century_model and then
  * calculating the rate CO2 is emmited.
  *
  * @param N_t number of times
  * @param t0 initial time
  * @param ts times
  * @param gamma partitioning coefficient
  * @param k decomposition rates
  * @param ajk transfer rates
  * @param x_r real data (empty)
  * @param x_i integer data (empty)
  * @return evolved CO2 for times ts
  */
    vector evolved_CO2(int N_t, real t0, vector ts,
                       vector gamma, real totalC_t0,
                       vector k, real a21, real a31, real a12,
                       real a32, real a13, real[] x_r, int[] x_i) {

      real C_t0[3];         // initial state
      real theta[8];        // ODE parameters
      real C_t[N_t,3];      // predicted pool content
      vector[N_t] CO2_t;    // evolved CO2 at times ts

      C_t0 = to_array_1d(gamma*totalC_t0);
      theta[1:3] = to_array_1d(k);
      theta[4] = a21;
      theta[5] = a31;
      theta[6] = a12;
      theta[7] = a32;
      theta[8] = a13;
      C_t = integrate_ode_rk45(century_model,
                              C_t0, t0, to_array_1d(ts), theta, x_r, x_i);
      for (t in 1:N_t)
        CO2_t[t] = totalC_t0 - sum(C_t[t]);
      return CO2_t;
    }
}
data {
  real<lower=0> totalC_t0;      // initial total carbon
  real t0;                      // initial time
  int<lower=0> N_t;             // number of measurement times
  int<lower=0> num_rep;         // number of replicates
  vector<lower=t0>[N_t] t_meas; // measurement times
  vector<lower=t0>[N_t] t_cap;  // cap times
  matrix<lower=0>[N_t, num_rep] CO2_flux; // measured carbon fluxes
}
transformed data {
  real x_r[0];  // no real data for ODE system
  int x_i[0];   // no integer data for ODE system
}
parameters {
  vector<lower=0>[3] turnover;  // turnover rates
  simplex[3] gamma;             // partitioning coefficients (a simplex)
  vector<lower=0>[3] sigma;     // turnover standard deviation
  real<lower=0> sigma_obs;      // observation standard deviation
  simplex[3] A1[num_rep];       // output rates from pool 1
  simplex[3] A2[num_rep];       // output rates from pool 2
  simplex[3] A3[num_rep];       // output rates from pool 3
  simplex[3] A1_g;              // global values for rates
  simplex[3] A2_g;              // global values for rates
  simplex[3] A3_g;              // global values for rates
  real<lower=1> kappa;
}
transformed parameters {
  vector<lower=0>[3] k;             // decomposition rates (1/turnover)
  matrix[N_t, num_rep] CO2_meas;    // evolved CO2 at measurement times
  matrix[N_t, num_rep] CO2_cap;     // evolved CO2 at cap times
  matrix[N_t, num_rep] CO2_flux_hat;// CO2 flux (average evolved CO2 between t_cap & t_meas)
  real<lower=0, upper=1> a21[num_rep];  // transfer rates
  real<lower=0, upper=1> a31[num_rep];
  real<lower=0, upper=1> a12[num_rep];
  real<lower=0, upper=1> a32[num_rep];
  real<lower=0, upper=1> a13[num_rep];
  k = 1 ./ turnover;
  // transfer rates are different for each replication:
  for (i in 1:num_rep) {
    a21[i] = A1[i, 2];
    a31[i] = A1[i, 3];
    a12[i] = A2[i, 1];
    a32[i] = A2[i, 3];
    a13[i] = A3[i, 1];
  }
  for (i in 1:num_rep) {
    CO2_meas[,i] = evolved_CO2(N_t, t0, t_meas, gamma, totalC_t0,
                          k, a21[i], a31[i], a12[i], a32[i], a13[i],
                          x_r, x_i);
    CO2_cap[,i] = evolved_CO2(N_t, t0, t_cap, gamma, totalC_t0,
                         k, a21[i], a31[i], a12[i], a32[i], a13[i],
                         x_r, x_i);
    CO2_flux_hat[,i] = (CO2_meas[,i] - CO2_cap[,i])./(t_meas - t_cap);
  }
}
model {
  // priors
  turnover[1] ~ normal(1.5, 0.15 * sigma[1]);
  turnover[2] ~ normal(25, 2.5 * sigma[2]);
  turnover[3] ~ normal(1000, 100 * sigma[3]);
  sigma ~ cauchy(0,1);
  sigma_obs ~ cauchy(0,.1);
  kappa ~ normal(10,5);
  for (i in 1:num_rep) {
    A1[i] ~ dirichlet(kappa*A1_g);
    A2[i] ~ dirichlet(kappa*A2_g);
    A3[i] ~ dirichlet(kappa*A3_g);
  }

  // likelihood
  to_vector(CO2_flux) ~ lognormal(to_vector(log(CO2_flux_hat)), sigma_obs);
}
```

Fitting models

```
t_meas <- c(seq(from=1/360, to=7/360, by=0.1*1/360), seq(from=14/360, to=28/360, by=0.1*7/360),
            seq(from=60/360, to=360/360, by=0.1*30/360));
t_cap <- c(0, (head(t_meas,-1)+tail(t_meas,-1))/2)
init_C <- c(10, 10, 80);
```

```r
num_rep <- 5;
data <- simulate_data_century(t_meas, t_cap, init_C, num_rep);
library(rstan);
library(tictoc);
rstan_options(auto_write = TRUE);
options(mc.cores = parallel::detectCores());
sm <- stan_model("century_hier_nonstiff.stan")
fit_all_data<-sampling(sm, data=data, iter=15, seed=1234);
```

```
## [[1]]
## Stan model 'century_hier_nonstiff' does not contain samples.
##
## [[2]]
## Stan model 'century_hier_nonstiff' does not contain samples.
##
## [[3]]
## Stan model 'century_hier_nonstiff' does not contain samples.
##
## [[4]]
## Stan model 'century_hier_nonstiff' does not contain samples.
```

```r
half_data <- simulate_data_century(t_meas[seq(1,length(t_meas),2)],
                                   t_cap[seq(1,length(t_meas),2)], init_C, num_rep);
fit_half_data<-sampling(sm, data=half_data, iter=15, seed=1234);
```

```
## [[1]]
## Stan model 'century_hier_nonstiff' does not contain samples.
##
## [[2]]
## Stan model 'century_hier_nonstiff' does not contain samples.
##
## [[3]]
## Stan model 'century_hier_nonstiff' does not contain samples.
##
## [[4]]
## Stan model 'century_hier_nonstiff' does not contain samples.
```

```r
quarter_data <- simulate_data_century(t_meas[seq(1,length(t_meas),4)],
                                      t_cap[seq(1,length(t_meas),4)], init_C, num_rep);
fit_quarter_data<-sampling(sm, data=quarter_data, iter=15, seed=1234);
```

```
## [[1]]
## Stan model 'century_hier_nonstiff' does not contain samples.
##
## [[2]]
## Stan model 'century_hier_nonstiff' does not contain samples.
##
## [[3]]
## Stan model 'century_hier_nonstiff' does not contain samples.
##
## [[4]]
## Stan model 'century_hier_nonstiff' does not contain samples.
```

```r
tenth_data <- simulate_data_century(t_meas[seq(1,length(t_meas),10)],
                                      t_cap[seq(1,length(t_meas),10)], init_C, num_rep);
fit_tenth_data<-sampling(sm, data=tenth_data, iter=15, seed=1234);
```

```
## [[1]]
## Stan model 'century_hier_nonstiff' does not contain samples.
##
## [[2]]
## Stan model 'century_hier_nonstiff' does not contain samples.
##
## [[3]]
## Stan model 'century_hier_nonstiff' does not contain samples.
##
## [[4]]
## Stan model 'century_hier_nonstiff' does not contain samples.
```

```r
saveRDS(fit_all_data,"all_data.rds")
saveRDS(fit_half_data,"half_data.rds")
saveRDS(fit_quarter_data,"quarter_data.rds")
saveRDS(fit_tenth_data,"tenth_data.rds")
```