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 analyze Bayesian models with no pooling, partial pooling, and complete pooling.

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:

pool 1: Active Soil C, $\kappa_1 \approx 1/1.5$,

pool 2: Slow Soil C, $\kappa_2 \approx 1/25$,

pool 3: Passive Soil C, $\kappa_3 \approx 1/1000$,

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:

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

Observation error

From empirical studies we know that the noise standard deviation is approximately 30%–40% of the value of the flux. We simulate this by a log-normal (multiplicative) noise as follows:

$$\text{observed flux} = \log\left(\exp(\text{actual flux}) + \tau\right),$$

where $\tau \sim \mathcal{N}(0, 0.5)$. Therefore, with 95% chance observed flux will be between $\exp(-0.5) = .6$ and $\exp(0.5) = 1.6$ of the actual value.

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 $(\alpha^i_{1j}, \alpha^i_{2j}, \alpha^i_{3j})$ 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 $(\alpha^0_{1j}, \alpha^0_{2j}, \alpha^0_{3j})$ such that we have the following hierarchical structure:

$$(\alpha^0_{1j}, \alpha^0_{2j}, \alpha^0_{3j}) \sim \text{Dirichlet}(1,1,1), (\alpha^i_{1j}, \alpha^i_{2j}, \alpha^i_{3j}) \sim \text{Dirichlet}(\kappa \times (\alpha^0_{1j}, \alpha^0_{2j}, \alpha^0_{3j})),$$

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.

```
simulate_data_century <- function(t_meas, t_cap, init_C, num_rep) {
# Simulating data using the century model
# 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);
for (rep in 1:num_rep) {
  Alpha <- matrix(0, m, m);
  Alpha_rep <- 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];
  # Hierarchical modeling of transfer rates for replications:
  kappa <- 100;
  Alpha_rep[,1] <-  rdirichlet(1, Alpha[,1] * kappa);
  Alpha_rep[,2] <-  rdirichlet(1, Alpha[,2] * kappa);
  Alpha_rep[,3] <-  rdirichlet(1, Alpha[,3] * kappa);
  Alpha[1, 1] <- 0;
  Alpha[2, 2] <- 0;
  Alpha[3, 3] <- 0;
  A <- Alpha * 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[,rep] <- exp(log(CO2_flux) + rnorm(length(CO2_flux),0,.5));
}
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);
return(simulated_data);
}
```

Model specification in Stan

We can fit the simulated flux replications in three ways:

1. Complete pooling: fitting a single model to all the replications, i.e., estimating only a single set of parameters for all models;
2. No pooling: fitting a model separately to each replication, i.e., estimating the model parameters independently for each replication;
3. Partial pooling: fitting a hierarchical Bayesian model which estimates parameters jointly for all replications and allows for variation between replicates.

These models are similar in many aspects. We start by explaining the portions of the Stan code they all share.

Functions

We define two functions. The first one, called `century_model`, takes in the model parameters and calculates the derivatives. The second function, `evolved_CO2`, uses the function `century_model ans Stan's stiff solver`integrate_ode_bdf‘ to calculate the total evolved (cumulative) CO_2 at time t.

```
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_bdf(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

The data is also the same for all the models.

```
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
}
```

Other parts of the stan model are different for different models. In the following, we explain each model in its own section.

Complete pooling

For the model with complete pooling, we need to define a single set of transfer rate parameters, as all the replications share the same set of parameters. We define simplex vectors `A1`, `A2`, and `A3` and then assign the transfer rates to their elements. The flux is derived by subtracting the total evolved carbon at times `t_meas` and `t_cap`.

```
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;                // output rates from pool 1
  simplex[3] A2;                // output rates from pool 2
  simplex[3] A3;                // output rates from pool 3
}
transformed parameters {
  vector<lower=0>[3] k;         // decomposition rates (1/turnover)
  vector[N_t] CO2_meas;         // evolved CO2 at measurement times
  vector[N_t] CO2_cap;          // evolved CO2 at cap times
  vector[N_t] CO2_flux_hat;     // CO2 flux (average evolved CO2 between t_cap & t_meas)
  real<lower=0, upper=1> a21;   // transfer rates
  real<lower=0, upper=1> a31;
  real<lower=0, upper=1> a12;
  real<lower=0, upper=1> a32;
  real<lower=0, upper=1> a13;
  k = 1 ./ turnover;
  // transfer rates are the same for all replications:
  a21 = A1[2];
  a31 = A1[3];
  a12 = A2[1];
  a32 = A2[3];
  a13 = A3[1];
  CO2_meas = evolved_CO2(N_t, t0, t_meas, gamma, totalC_t0,
```

```
                          k, a21, a31, a12, a32, a13,
                          x_r, x_i);
  CO2_cap = evolved_CO2(N_t, t0, t_cap, gamma, totalC_t0,
                          k, a21, a31, a12, a32, a13,
                          x_r, x_i);
  CO2_flux_hat = (CO2_meas - CO2_cap)./(t_meas - t_cap);
}
```

In the model block, we assign normal distributions to turnovers with mean equal to the expert-tuned variables. We believe that the estimated values should not be too far from the expert-tuned values; so, we set the standard deviations to be 1/10'th of the mean times a Cauchy random variable. The observation noise is modelled as log-normal with a Cauchy standard deviation.

```
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);

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

No pooling

With noo pooling, we estimate a separate set of transfer rates for each replication. Thus, we have simplex arrays `A1`, `A2`, and `A3`, each with the length `num_rep`. The values of `CO2_meas`, `CO2_cap`, and `CO2_flux` is calculated separately for each replication.

```
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
}
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);
  }
}
```

The prior part of the model block is the same as before, but the likelihoos is modified to account for the no-pool constraint.

```
model {
  ...

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

Partial pool with a hierarchical model

The model specification is similar to the no-pool case with the difference that we now have a set of global trannsfer rates which connects the parameters for different replications. The transformed parameters block is the same as no-pool case and parameters and model block change as follows:

```
parameters {
  ...
  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 parameter {
  ...
}
model {
  ...
  kappa ~ pareto(1,1.5);
  for (i in 1:num_rep) {
    A1[i] ~ dirichlet(kappa*A1_g);
    A2[i] ~ dirichlet(kappa*A2_g);
    A3[i] ~ dirichlet(kappa*A3_g);
  }
  ...
}
```

Fitting the models

```
num_rep <- 5;
t_meas <- c(seq(from=1/360, to=7/360, by=1/360), seq(from=14/360, to=28/360, by=7/360),
            seq(from=60/360, to=360/360, by=30/360));
t_cap <- c(seq(from=0.5/360, to=6.5/360, by=1/360), seq(from=10.5/360, to=24.5/360, by=7/360),
           seq(from=45/360, to=345/360, by=30/360));
init_C <- 1E3*c(.1, .1, .8);
data <- simulate_data_century(t_meas, t_cap, init_C, num_rep);

library(rstan);
rstan_options(auto_write = TRUE);
options(mc.cores = parallel::detectCores());
fit_pool <- stan("century_pool.stan", data=data, iter=50, seed=1234); # complete pooling
fit_nopool <- stan("century_nopool.stan", data=data, iter=50, seed=1234); # no pooling
fit_hier <- stan("century_hier.stan", data=data, iter=50, seed=1234); # partial pooling
```

```
print(fit_pool, c("a21"))
```

```
Inference for Stan model: century_pool.
4 chains, each with iter=50; warmup=25; thin=1;
post-warmup draws per chain=25, total post-warmup draws=100.

    mean se_mean  sd 2.5%  25%  50%  75% 97.5% n_eff Rhat
a21  0.3    0.03 0.2 0.02 0.15 0.25 0.42  0.72    36 1.11

Samples were drawn using NUTS(diag_e) at Tue Dec 20 15:00:45 2016.
For each parameter, n_eff is a crude measure of effective sample size,
and Rhat is the potential scale reduction factor on split chains (at
convergence, Rhat=1).
```

```
print(fit_nopool, c("a21"))
```

```
Inference for Stan model: century_nopool.
4 chains, each with iter=50; warmup=25; thin=1;
post-warmup draws per chain=25, total post-warmup draws=100.

       mean se_mean   sd 2.5%  25%  50%  75% 97.5% n_eff Rhat
a21[1] 0.31    0.05 0.21 0.02 0.16 0.27 0.44  0.77    20 1.11
a21[2] 0.30    0.03 0.20 0.04 0.13 0.28 0.43  0.77    42 1.04
a21[3] 0.31    0.03 0.20 0.05 0.14 0.26 0.45  0.73    51 1.09
a21[4] 0.31    0.03 0.22 0.01 0.13 0.29 0.45  0.76    58 1.03
a21[5] 0.27    0.02 0.19 0.02 0.12 0.22 0.39  0.64    69 1.00

Samples were drawn using NUTS(diag_e) at Tue Dec 20 15:07:43 2016.
For each parameter, n_eff is a crude measure of effective sample size,
and Rhat is the potential scale reduction factor on split chains (at
convergence, Rhat=1).
```

```
print(fit_hier, c("a21"))
```

```
Inference for Stan model: century_hier.
4 chains, each with iter=50; warmup=25; thin=1;
post-warmup draws per chain=25, total post-warmup draws=100.
```

```
       mean se_mean   sd 2.5%  25%  50%  75% 97.5% n_eff Rhat
a21[1] 0.08    0.03 0.13    0 0.00 0.02 0.12  0.51    15 1.22
a21[2] 0.06    0.02 0.07    0 0.01 0.05 0.10  0.26    22 1.24
a21[3] 0.07    0.03 0.13    0 0.00 0.02 0.08  0.34    17 1.26
a21[4] 0.07    0.02 0.09    0 0.00 0.05 0.11  0.28    14 1.29
a21[5] 0.07    0.03 0.12    0 0.00 0.03 0.08  0.35    16 1.32

Samples were drawn using NUTS(diag_e) at Tue Dec 20 15:13:10 2016.
For each parameter, n_eff is a crude measure of effective sample size,
and Rhat is the potential scale reduction factor on split chains (at
convergence, Rhat=1).
```