R vs. Julia in MCMC

> Lijun Wang
>
> 29 August, 2018

Let me use the Example 7.1.1 of [Robert and Casella (2013)] to compare the speed of R and Julia in MCMC.

> **Example 7.1.1 Bivariate Gibbs sampler.** Let the random variables X and Y have joint density $f(x,y)$, and generate a sequence of observations according to the following:
>
> Set $X_0 = x_0$, and for $t = 1, 2, \ldots,$ generate
>
> $$\begin{aligned} Y_t &\sim f_{Y|X}(\cdot|x_{t-1}), \\ X_t &\sim f_{X|Y}(\cdot|y_t), \end{aligned} \tag{7.1.1}$$
>
> where $f_{Y|X}$ and $f_{X|Y}$ are the conditional distributions. The sequence (X_t, Y_t), is a Markov chain, as is each sequence (X_t) and (Y_t) individually. For example, the chain (X_t) has transition density
>
> $$K(x, x^*) = \int f_{Y|X}(y|x) f_{X|Y}(x^*|y) dy,$$
>
> with invariant density $f_X(\cdot)$. (Note the similarity to Eaton's transition (4.10.9).)
>
> For the special case of the bivariate normal density,
>
> $$(X, Y) \sim \mathcal{N}_2\left(0, \begin{pmatrix} 1 & \rho \\ \rho & 1 \end{pmatrix}\right), \tag{7.1.2}$$
>
> the Gibbs sampler is
>
> Given y_t, generate
>
> $$\begin{aligned} X_{t+1} \mid y_t &\sim \mathcal{N}(\rho y_t,\ 1-\rho^2), \\ Y_{t+1} \mid x_{t+1} &\sim \mathcal{N}(\rho x_{t+1},\ 1-\rho^2). \end{aligned} \tag{7.1.3}$$
>
> The Gibbs sampler is obviously not necessary in this particular case, as iid copies of (X, Y) can be easily generated using the Box–Muller algorithm (see Example 2.2.2). ||

We can implement this example with the following code. It is worth noting that I try to keep the same form between this two programming language as much as possible. For example, Julia does not support arbitrary mean and variance in its Gaussian sampling function `randn()`, while R can directly realize in `rnorm()`, but we both adopt the linear transformation of Gaussian distribution.

R

```r
bigibbs <- function(T, rho)
{
    x = numeric(T+1)
    y = numeric(T+1)
    for (t in 1:T){
        x[t+1] = rnorm(1) * sqrt(1-rho^2) + rho*y[t]
        y[t+1] = rnorm(1) * sqrt(1-rho^2) + rho*x[t+1]
    }
    return(list(x=x, y=y))
}
## example
res = bigibbs(2e6, 0.5)
```

Julia

```julia
function bigibbs(T::Int64, rho::Float64)
    x = ones(T+1)
    y = ones(T+1)
    for t = 1:T
        x[t+1] = randn() * sqrt(1-rho^2) + rho*y[t]
        y[t+1] = randn() * sqrt(1-rho^2) + rho*x[t+1]
    end
    return x, y
end

## example
x, y = bigibbs(Int64(2e6), 0.5)
```

Results

The running environment is as follows:

- System: Ubuntu 18.04 (Windows subsystem for Linux)
- Processor: Intel(R) Core(TM) i7-6700 CPU @ 3.40GHz x 8
- Memory: 16 GiB
- R version: 3.4.4 (2018-03-15)
- Julia version: 1.0.0 (2018-08-08)

In terminal, use `time` command to get their running time:

```
weiya $ time julia toy_gibbs.jl

real    0m0.410s
user    0m0.422s
sys     0m0.406s
weiya $ time Rscript toy_gibbs.R

real    0m6.506s
user    0m6.406s
sys     0m0.094s
```

Obviously, in our toy example, Julia outperforms much than R, nearly 16 times. Try another number of iterations, the results are similar.

Moreover, we can use `PyPlot` to plot in Julia v1.0, such as the histogram in this toy example: