How to make R simulations faster

László Varga

2016/2017, spring semester

To achieve major success in making **R** codes run faster, two options are available:

- writing some computing-intensive parts of the code (big loops) in C++, and combine it with **R**, using the Rcpp package and/or the packages depending on it
- parallel programming – using all of the cores of our computer

The main person on working **R** and C++ integration is Dirk Edelbuettel.

For further reading:

- Edelbuettel, Francois: *Rcpp: Seamless **R** and C++ Integration.* Journal of Statistical Software, 2011.
- Edelbuettel: *Seamless **R** and C++ integration with Rcpp.* Springer, 2013.
- Edelbuettel: Rccp attibutes, `http://dirk.eddelbuettel.com/code/rcpp/Rcpp-attributes.pdf`
- `http://dirk.eddelbuettel.com`

1.) Let us write an **R** function to calculate the nth Fibonacci number! How much time does it take calculate the 32nd one? Let us write a C++ function also and import to **R** with the Rcpp package! How much speed-up can you experience?

2.) After Rémillard and Scaillet (2009), the proposed test statistic to test if two copulas are the same or not:

$$S_{n,m} = \left(\frac{1}{n}+\frac{1}{m}\right)^{-1} \cdot \left[\frac{1}{n^2}\sum_{i=1}^{n}\sum_{j=1}^{m}\prod_{s=1}^{d}(1-U_{is,n}\vee U_{js,n}) + \frac{1}{m^2}\sum_{i=1}^{m}\sum_{j=1}^{m}\prod_{s=1}^{d}(1-V_{is,m}\vee V_{js,m}) - \right.$$
$$\left. - \frac{2}{nm}\sum_{i=1}^{n}\sum_{j=1}^{m}\prod_{s=1}^{d}(1-U_{is,n}\vee V_{js,m})\right],$$

where

- $u \vee v = \max(u, v)$;
- n, m: the size of the samples, respectively;
- U, V: pseudo-observations of the samples, respectively.

Let us write an **R** function and a C++ function to calculate the test statistic! Let us simulate for different sample sizes the speed-up of the code! We can use the temperature data in the file *resi.Rdata*.

3.) GARCH process – ML-estimation

The Gaussian quasi-likelihood function, conditional on the $x_{1-q}, \ldots, x_0, \tilde{\sigma}^2_{1-p}, \ldots, \tilde{\sigma}^2_0$ initial values, is

$$L_n(\theta) = L_n(\theta; x_1, \ldots, x_n) = \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi\tilde{\sigma}^2_t}} e^{-\frac{x_t^2}{2\tilde{\sigma}^2_t}}.$$

where the $(\tilde{\sigma}^2_t)_{t\geq 1}$ are recursively defined by the following equation:

$$\tilde{\sigma}^2_t = \tilde{\sigma}^2_t(\theta) = \omega + \sum_{i=1}^{q}\alpha_i x^2_{t-i} + \sum_{j=1}^{p}\beta_j \tilde{\sigma}^2_{t-j}(\theta)$$

The QMLE of θ is defined as

$$\hat{\theta}_n = \underset{\theta\in\Theta}{\operatorname{argmax}}\ L_n(\theta).$$

To maximize the Gaussian likelihood function, we have to minimize the following function:

$$I_n(\theta) = \frac{1}{n}\sum_{t=1}^{n} l_t(\theta), \quad \text{where} \quad l_t(\theta) = \frac{x_t^2}{\tilde{\sigma}_t^2(\theta)} + \log(\tilde{\sigma}_t^2(\theta)).$$

Its weighted likelihood bootstrap version is the following (the τ values are proper random weights):

$$I_n^*(\theta) = \frac{1}{n}\sum_{t=1}^{n} l_{nt}^*(\theta), \quad \text{where} \quad l_{nt}^*(\theta) = \tau_{nt}\left(\frac{x_t^2}{\tilde{\sigma}_t^2(\theta)} + \log(\tilde{\sigma}_t^2(\theta))\right)$$

The weighted bootstrap QMLE of the parameter θ is defined as the solution $\hat{\theta}_n^*$ of

$$\hat{\theta}_n^* = \underset{\theta\in\Theta}{\operatorname{argmax}}\ I_n^*(\theta).$$

a.) Write an **R** and a C++ code to calculate the weighted bootstrap QMLE of a given sample. Examine the speed-up of the code for different sample sizes!

b.) Write an **R** code, using parallel computing to generate 16 samples from a GARCH(1,1) process with sample size 1000, then calculate the QML-estimates and 100 pieces of weighted bootstrap QML-estimates with polynomial weights!