5. Multithreaded Graphics and Iterative Parallel Algorithms

Chuanhai Liu

Department of Statistics, Purdue University

2016

Table of Contents

5.1 More SupR additions

SupR command mode???

- `where` displays the location where this command is invoked.

5.2 SupR graphics

- R graphics is problematic in the traditional single threaded R environment, especially with big data.

  > For example, when the graphics is doing redrawing, the R system will not be available to interact with the user.

- While alternative approaches can be considered later, the current SupR graphics is simply implemented with a single but separate thread, call the *graphics* server.

5.2 SupR graphics: basics

```
> ?graphics.server
```

With '...' standing for the arguments that can be passed to the `X11()` function, the `new.thread(X11(...))` call starts a new thread. We call this newly created thread the *graphics server*.

The graphics package "`package:graphics`" on the search list is replaced with a new environment, named "`package:[graphics]`". This new environment consists of new active bindings that return the actual functions for the graphics server. For other threads, they return a function to redirect the calls to the graphics server.

This makes it possible that after starting the X11 graphics devices, the graphics server handles all function calls with the functions in the attached `package:graphics` package (on the search path). Thus, graphics calls are synchronized because the function evaluations are effectively carried out by the single graphics server.

5.2 SupR graphics: examples

Before starting the graphics server:

```
> search()
 ...  [3] package:graphics ...
> bindingIsActive("plot", as.environment("package:graphics"))
[1] FALSE
> plot
function (x, y, ...)
UseMethod("plot")
<environment:  namespace:graphics>
> new.thread(X11())
[1] "GraphicsServer_1"
>
```

You can also inspect interval structures of these objects with, for example, the following function calls:

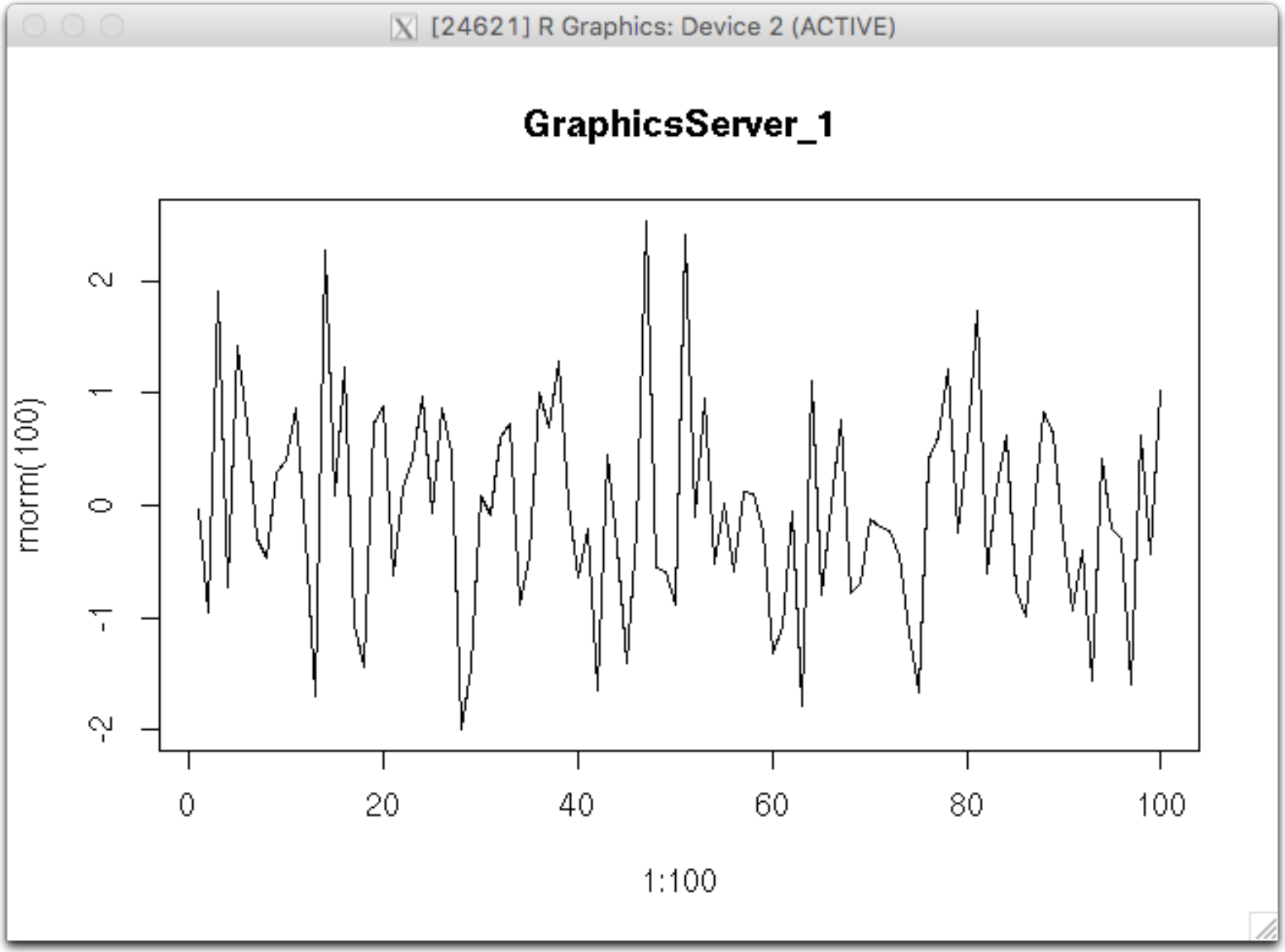
- `.Internal(inspect(plot))`
- `.Internal(address(plot))`
- `get.activeBinding(plot, as.environment("package:[graphics]"))`

5.2 SupR graphics: examples

After starting the graphics server:

```
> current.thread()
[1] "__MAIN__"
> search()
 ...  [3] package:[graphics] ...
> bindingIsActive("plot", as.environment("package:[graphics]"))
[1] TRUE
> plot
function (...)
{
    call = match.call()
    .Internal(thread.gcall(call, parent.frame()))
}
<environment:  package:graphics>
> plot(1:100, rnorm(100), type="l", main={
        print(plot); current.thread()})
function (x, y, ...)
UseMethod("plot")
<environment:  namespace:graphics>
>
```

5.2 SupR graphics: examples



5.3 Iterative parallel algorithms

Iterative Algorithms The development of iterative algorithms, such as Newton-Raphson, Conjugate gradient, Quasi-Newton, Expectation-Maximization (EM), and Markov chain Monte Carlo, had been fundamental for the best part of the last century. It would continue to be the key for the success of scientific investigation with large data and realistic but complex statistical models.

Parallel Algorithms With currently increasing interest in big data analysis and intensive methods such as Machine Learning, development of intelligent parallel algorithms is expected to be an important focus in years to come.

Motivating example In what follows, we discuss a parallel EM algorithm for a simple example and show how it can be implemented in SupR, a multi-threaded computing environment.

5.4 The EM algorithm: incomplete Gaussian data

It would make more sense to discuss the problem of analyzing multivariate observations with the multivariate Gaussian model. But the computational problem remains if we keep in mind the setting of complete-data model.

To be more specific, suppose that we have an incomplete sample, X_1, ..., X_n. By incomplete, we mean that some of X_i's are missing. Formally, we can represent the data by

$$(X_i, M_i) \qquad i = 0, ..., n$$

where M_i is a dichotomous variable as the missing-data indicator, i.e., X_i is missing if $M_i = 1$ and X_i is observed if $M_i = 0$.

The complete-data model to consider is

$$X_i \overset{iid}{\sim} N(\mu, 1) \qquad (i = 1, ..., n; \text{for some } \mu \in R)$$

and the missing data mechanism is assumed to *ignorable*.

Under such assumptions, it is easy to see that the maximum likelihood estimate of μ is the sample mean of the fully observed values. The EM algorithm discussed below works for more general multivariate cases where closed-form solutions are typically not available.

5.4 The EM algorithm: toward implementation

The EM algorithm iterates between an E-step and an M-step until no any significant improvements can be achieved without running forever.

E-step computes the Expected complete-data log-likelihood given the observed data, with the current fitted model for the missing-data (r.v.).

M-step updates the parameter estimate by maximizing the Expected complete-data log-likelihood function computed in the E-step.

For the above incomplete-normal-sample problem, we have

E-step computes the Expectation of the complete-data sufficient statistics

$$S(\mu_t) = \sum_i E(X_i|\mu = \mu_t, X_{obs}) = \sum_{i:M_i=0} X_i + \sum_{i:M_i=1} \mu_t$$

M-step updates the parameter estimate by computing

$$\mu_{t+1} = S_{\mu_t}/n$$

5.4 The EM algorithm: single process implementation

See the online example.

5.6 Multithreaded graphics for big data analysis

TO DO

5.7 Exercises

1. Try EM on the robit model!
2. For students who are familiar with MCMC, try it by modifying the EM example.