Regular Expressions in R

Paul E. Johnson[1]

[1]Center for Research Methods and Data Analysis

2018

KU

Outline

KU

Outline

What for?

Regular expressions are used for

1. Matching: Identifying patterns that exist in strings
2. Editing and re-arranging matched patterns in strings

What for?

- Some users know "shell globs" for pattern matching, e.g. "*.docx" represents all files that end in "docx".
- Regular expressions are a different pattern matching system that allows much more delicate filtering.

- Regular expressions are Jedi light sabres compared to shell globs

General Purpose Terminology

- "grep": the GNU regular expression parser
- Regular expressions are heavily used Perl, Python, awk, sed, and other interactive languages used in the Web
- More-or-less uniform, some variations among implementations

General Purpose Terminology

- R (R Core Team, 2017) includes functions that use regular expressions. Today we focus on functions named
  - `grep()` : identifying the presence of a pattern
  - `gsub()` : replacing a matched pattern
- We will only go about 25% below the surface of this (deep deep) water.

Regular Expression Symbol Highlights

- `.` matches any character
- `*` a quantifier, meaning any number of times. "`.*`" is any character, any number of times
- `^` the beginning of a string
- `$` the end of a string

Other special symbols in regex: ( ) [ ] - + \

Outline

grep is for finding values

- grep: "GNU regular expression parser"
- In a terminal, I use grep all the time to filter text.
- R's implementation is an R function `grep()`
- primary arguments

  pattern a regular expression character string

  x the input character vector in which matches are to be found

- Return will be the number of each matching item

simple example 1

```
x <- c("Dieter-Charles", "Charles", "Charlie",
    "Charles-William", "Charlene")
```

- Which ones end with "Charles"?

```
grep("Charles$", x)
```

```
[1] 1 2
```

- I want the ones that begin with "Charles"

```
grep("^Charles", x)
```

```
[1] 2 4
```

simple example 1 ...

- You want the names, not the positions in the vector? I forgot to mention this argument:

  value return the values of matching items

```
grep("^Charles", x, value = TRUE)
```

```
[1] "Charles"          "Charles-William"
```

The back slash problem

- The backslash is a special symbol, it is used in "escape combinations" like

  \n new line
  \t tab character

- These are known as "escape sequences". Any "\" is seen as the beginning of an escape sequence by the R interpreter.
- In regular expressions, some symbols have special meaning, so they have to be escaped by "\" when we want to use them literally.
- Ex: If quotation marks are needed inside a quoted string, for example, we "escape" the middle quotes

```
aname <- "Fred \"The Hammer\" Williamson"
aname
```

```
[1] "Fred \"The Hammer\" Williamson"
```

The back slash problem ...

- Easier way to type that in uses single quotes, but you see end result is same

```
aname <- 'Fred "The Hammer" Williamson'
aname
```

```
[1] "Fred \"The Hammer\" Williamson"
```

- The single backslash is not allowed to appear in text strings unless it is performing a special purpose of escaping something. So, if you want to type in "\" literally, it must escape itself!
  - "\" is entered by "\\"!

```
x <- c("Dieter\\Charles", "Friend/Foe")
```

- Note, that the double backslash appears when we `print()` the string, but not when we `cat()` it.

The back slash problem ...

```
print(x)
```

```
[1] "Dieter\\Charles" "Friend/Foe"
```

```
cat(x)
```

```
Dieter\Charles Friend/Foe
```

- The cat output indicates that the "real" string just has one backslash, but the display of it shows two because the first one is escaping the second one.
- If you want to match the items that have the literal backslash, get ready for "`\\\\`"

```
grep("\\\\", x)
```

```
[1] 1
```

The back slash problem ...

- Why 4? Well,
  - R is an interpreter, so you need 2 escapes for R
  - R is passing this to the GNU regex engine, and it needs 2 escapes as well

Other grep parameters

fixed	if TRUE, turns off regex matching, uses literal character matching
ignore.case	capitalization does not count
perl	use the regex style of the Perl program
invert	return the non-matching items

fixed parameter helps on the backslash problem

- Turning off regular expressions means we don't need 4 slashes anymore, just 2

```
grep("\\", x, fixed = TRUE)
```

```
[1] 1
```

- Why do we still need 2? All character strings in computers always have "\n" or "\t" for newline or tab, those are not regular expression escapes.
- If we include just "\", the system will say we did not supply "n" or "t" to finish what we started.
- If we say "\\", it is interpreted as one literal backslash.
- The "why do I need four backslashes to match one backslash?" problem is one of the FAQs for many programs.

Quiz

```
cities <- c("Dallas Texas", "Denver Colorado",
    "Austin Texas", "Salem Oregon", "Salem
    Massachusetts", "California Pennsylvania",
    "Long Beach California", "San Francisco
    California", "Texas Missouri", "Nevada
    Missouri", "St. Louis Missouri", "Truth or
    Consequences New Mexico", "Charleston South
    Carolina", "Charleston North Carolina")
```

If you ask "which elements are from the state of Texas", note that a fixed pattern match will not find them.

Quiz ...

```
grep("Texas", cities, fixed = TRUE, value = TRUE)
```

```
[1] "Dallas Texas"   "Austin Texas"   "Texas Missouri"
```

- Can you find a regular expression to find cities in state of Texas?

```
grep("______", cities)
```

Texas, As If you would want that

```
grep("Texas$", cities, value = TRUE)
```

```
[1] "Dallas Texas" "Austin Texas"
```

A real life pattern matching problem

- We have a data set with a lot of variables, here are the column names, in a vector

```
cnames <- c("ID", "Q1", "encounter", "forms",
    "ar_physinj.n", "ar_illness.n",
 "ar_chronic.n", "ar_job.n", "ar_hunger.n",
    "ar_STI.n", "ar_sa.n",
 "ar_UTI.n", "ar_abuse.n", "ar_dental.n",
    "ar_drugalc.n", "ar_suicide.n",
 "ar_chronrun.n", "ar_truancy.n",
    "ar_sysinvolve.n", "ar_menthealth.n",
 "ar_tattoos.n", "ar_other.n", "rf_pov.n",
    "rf_homeless.n", "rf_famdys.n",
 "rf_control.n", "rf_addiction.n",
    "rf_physdis.n", "rf_cogdis.n",
 "rf_race.n", "rf_LGBTQ.n", "rf_undoc.n",
    "rf_lang.n", "rf_pregnancy.n",
```

A real life pattern matching problem ...

```
"rf_dropout.n", "rf_running.n", "rf_sex.n",
   "rf_abuse.n", "rf_fincontrol.n",
"rf_ssnetworks.n", "rf_other.n", "ar_physinj.l",
   "ar_illness.l",
"ar_chronic.l", "ar_job.l", "ar_hunger.l",
   "ar_STI.l", "ar_sa.l",
"ar_UTI.l", "ar_abuse.l", "ar_dental.l",
   "ar_drugalc.l", "ar_suicide.l",
"ar_chronrun.l", "ar_truancy.l",
   "ar_sysinvolve.l", "ar_menthealth.l",
"ar_tattoos.l", "ar_other.l", "rf_pov.l",
   "rf_homeless.l", "rf_famdys.l",
"rf_control.l", "rf_addiction.l",
   "rf_physdis.l", "rf_cogdis.l",
"rf_race.l", "rf_LGBTQ.l", "rf_undoc.l",
   "rf_lang.l", "rf_pregnancy.l",
```

A real life pattern matching problem ...

```
"rf_dropout.l", "rf_running.l", "rf_sex.l",
   "rf_abuse.l", "rf_fincontrol.l",
"rf_ssnetworks.l", "rf_other.l", "ar_physinj.s",
   "ar_illness.s",
"ar_chronic.s", "ar_job.s", "ar_hunger.s",
   "ar_STI.s", "ar_sa.s",
"ar_UTI.s", "ar_abuse.s", "ar_dental.s",
   "ar_drugalc.s", "ar_suicide.s",
"ar_chronrun.s", "ar_truancy.s",
   "ar_sysinvolve.s", "ar_menthealth.s",
"ar_tattoos.s", "ar_other.s", "rf_pov.s",
   "rf_homeless.s", "rf_famdys.s",
"rf_control.s", "rf_addiction.s",
   "rf_physdis.s", "rf_cogdis.s",
"rf_race.s", "rf_LGBTQ.s", "rf_undoc.s",
   "rf_lang.s", "rf_pregnancy.s",
```

A real life pattern matching problem ...

```
"rf_dropout.s", "rf_running.s", "rf_sex.s",
   "rf_abuse.s", "rf_fincontrol.s",
"rf_ssnetworks.s", "rf_other.s", "cr_strongfam",
   "cr_mentors",
"cr_faithcom", "cr_culturecom", "cr_employment",
   "cr_edaccess",
"cr_houseaccess", "cr_pathimm", "cr_physhealth",
   "cr_menthealth",
"cr_insurance", "cr_govprog", "cr_other",
   "or_protocol", "or_collab",
"or_lawenforce", "or_hotline", "or_training",
   "or_other", "sector",
"chronrun", "truancy", "region", "arak", "arub",
   "arakUTI", "Q57", "Q59",
"Q61", "Q63", "Q65", "Q19", "Q21", "Q23", "Q25",
   "Q27")
```

Here are the assignments

- Use a regular expression to pull out the variable names that begin with "ar" and end in ".n", ".s", and ".l"
- Find the variables that have "chronrun" in the middle after "_" and before ".s", ".n", or ".l" (hint: this is a literal period, not a regex "this . means anything" period, so it will need to be escaped "\\.").

Fill in the blanks

- Use a regular expression to pull out the variable names that end in ".n", ".s", and ".l"

```
grep("______", _____, value = TRUE)
```

My Answer 1

- I needed those as columns in a data frame, so lets try

```
reslt1 <- data.frame(
    n = grep("^ar_.*\\.n$", cnames, value = TRUE),
    s = grep("^ar_.*\\.s$", cnames, value = TRUE),
    l = grep("^ar_.*\\.s$", cnames, value = TRUE))
head(reslt1, 15)
```

```
                n               s               l
1     ar_physinj.n    ar_physinj.s    ar_physinj.s
2     ar_illness.n    ar_illness.s    ar_illness.s
3     ar_chronic.n    ar_chronic.s    ar_chronic.s
4         ar_job.n        ar_job.s        ar_job.s
5      ar_hunger.n     ar_hunger.s     ar_hunger.s
6         ar_STI.n        ar_STI.s        ar_STI.s
7          ar_sa.n         ar_sa.s         ar_sa.s
8         ar_UTI.n        ar_UTI.s        ar_UTI.s
9       ar_abuse.n      ar_abuse.s      ar_abuse.s
10     ar_dental.n     ar_dental.s     ar_dental.s
11    ar_drugalc.n    ar_drugalc.s    ar_drugalc.s
12    ar_suicide.n    ar_suicide.s    ar_suicide.s
13   ar_chronrun.n   ar_chronrun.s   ar_chronrun.s
```

My Answer 1 ...

```
15  14     ar_truancy.n     ar_truancy.s     ar_truancy.s
    15 ar_sysinvolve.n ar_sysinvolve.s ar_sysinvolve.s
```

My Answer 2

```
reslt2 <- grep("_chronrun\\.", cnames, value =
    TRUE)
reslt2
```

```
[1] "ar_chronrun.n" "ar_chronrun.l" "ar_chronrun.s"
```

- And we could use that to pull columns from the `hts` data frame:

```
hts[reslt2]
```

Outline

gsub

- `gsub` is for replacing character strings
- `gsub("pattern", "replacement", x)`

City & State names, again

```
cities <- c("Dallas Texas", "Denver Colorado",
    "Austin Texas", "Salem Oregon", "Salem
    Massachusetts", "California Pennsylvania",
    "Long Beach California", "San Francisco
    California", "Texas Missouri", "Nevada
    Missouri", "St. Louis Missouri", "Truth or
    Consequences New Mexico", "Charleston South
    Carolina", "Charleston North Carolina")
```

Suppose you want to divide this into 2 variables, city names and state names.

The complications:

- names have spaces in them, so I can't choose all of the states by taking the last name.
- ideas?

Here's what I thought of before I asked you

- Step 1. Lets replace the state names with abbreviations.
- Step 2. Cut out the abbreviations to get the city names by themselves.
- Step 3. Keep only last 2 characters to get the State names.
- Step 4. Put the full names back in.

Step 1. Insert Abbreviated State names

- Easy to replace one state name at end by typing

```
gsub("Texas$", "TX", cities)
```

```
 [1] "Dallas TX"                          "Denver Colorado"
 [3] "Austin TX"                          "Salem Oregon"
 [5] "Salem Massachusetts"                "California Pennsylvania"
 [7] "Long Beach California"              "San Francisco California"
 [9] "Texas Missouri"                     "Nevada Missouri"
[11] "St. Louis Missouri"                 "Truth or Consequences New
     Mexico"
[13] "Charleston South Carolina"          "Charleston North Carolina"
```

- But I don't want to type each state one by one
- R has variables "state.name" and "state.abb" built into the session

```
head(state.name)
```

```
[1] "Alabama"    "Alaska"     "Arizona"    "Arkansas"   "California"
    "Colorado"
```

Step 1. Insert Abbreviated State names ...

```
head(state.abb)
```

```
[1] "AL" "AK" "AZ" "AR" "CA" "CO"
```

- We can go one by one, through those names, with a for loop.

Step 1 Detour. named vector

- In the first workshop using this guide, the most surprising/unfamiliar items was the way we use a named vector.
- The following will name the items in `state.name`, using the short 2-letter abbreviations as the names.

```
names(state.name) <- state.abb
```

- Now the abbreviations and full names are tied together. Inspect the first few

```
head(state.name)
```

```
        AL         AK         AZ         AR         CA
                   CO
"Alabama"  "Alaska"  "Arizona"  "Arkansas" "California"
    "Colorado"
```

- Now I can retrieve a state's long name by using the index name from the abbreviation

Step 1 Detour. named vector ...

```
state.name["TX"]
```

```
     TX
"Texas"
```

Step 1 Resumes, with a for loop

- The index, i, will be the abbreviations.
- The gsub inside the loop scans each item, looking for the long state name given by `state.name[i]`, and replacing it with the short name from `i`.

```
cities2 <- cities
## I'll iterate on the abbreviations
for(i in state.abb) {
     cities2 <- gsub(state.name[i], i, cities2)
 }
```

Understanding Checkpoint

- To test your understanding of the for loop, try these things.
  1. Insert " `print()` " statements into the loop to cause verbose output from each step, such as `print(i)`, `print(state.name[i])`, or `print(cities2)`.
  2. Insert this " `browser()` " at the beginning of the for loop. This will stop the calculation so you can inspect the variables interactively. (Remember, to exit from browser, type "Q" return)

Step 1. There is an error, however

- Inspect the output, especially item 10

```
cities2
```

```
 [1] "Dallas TX"                  "Denver CO"                  "Austin TX"
 [4] "Salem OR"                   "Salem MA"                   "CA PA"
 [7] "Long Beach CA"              "San Francisco CA"           "TX MO"
[10] "NV MO"                      "St. Louis MO"               "Truth or
    Consequences NM"
[13] "Charleston SC"              "Charleston NC"
```

```
cities2[10]
```

```
[1] "NV MO"
```

- My state-name-matching gsub() usage was not smart enough. I need to restrict its work to the end of the string. We need to add a "$" to the end of `state.name` in the target.

Step 1. Correction for the state name error

- In Summer 2017, we tried 2 equivalent approaches

```
## Keep cities safe, write on a copy cities2
cities2 <- cities
## Method 1
## Puts the target match correction into the for
    loop
for(i in state.abb) {
     ## target is a string, only has one element
     target <- paste0(state.name[i], "$")
     cities2 <- gsub(target, i, cities2)
 }
## Appears correct:
cities2
```

Step 1. Correction for the state name error ...

```
 [1] "Dallas TX"                  "Denver CO"                  "Austin TX"
 [4] "Salem OR"                   "Salem MA"                   "California
     PA"
 [7] "Long Beach CA"              "San Francisco CA"           "Texas MO"
[10] "Nevada MO"                  "St. Louis MO"               "Truth or
     Consequences NM"
[13] "Charleston SC"              "Charleston NC"
```

```
##
## Here is method 2. We start over,
## Creating a target VECTOR before the for loop.
## After this, target is a vector of corrected
    matches
cities2 <- cities
target <- paste0(state.name, "$")
## names were lost, so re-apply them
names(target) <- names(state.name)
## inspect
head(target, 5)
```

Step 1. Correction for the state name error ...

```
          AL           AK           AZ           AR           CA
"Alabama$"    "Alaska$"    "Arizona$"   "Arkansas$" "California$"
```

```
for(i in state.abb) {
     ## target is a vector, choose the i'th one
     cities2 <- gsub(target[i], i, cities2)
 }
cities2
```

```
 [1] "Dallas TX"           "Denver CO"           "Austin TX"
 [4] "Salem OR"            "Salem MA"            "California
     PA"
 [7] "Long Beach CA"       "San Francisco CA"    "Texas MO"
[10] "Nevada MO"           "St. Louis MO"        "Truth or
    Consequences NM"
[13] "Charleston SC"       "Charleston NC"
```

Step 1. Consider kutils::mgsub

- We found ourselves writing that for loop very often.
- To streamline, we made a simple function `mgsub()` (as in "multi-gsub") in the `kutils` package.
- It does the exact same work in one line of code
- Inputs are
  1. the vector of target regular expressions that will be replaced
  2. the vector of replacements, must be same length as target
  3. the string vector in which replacements are to be made.
- Example usage

```
library(kutils)
target <- paste0(state.name, "$")
mgsub(target, state.abb, cities)
```

Step 1. Consider kutils::mgsub ...

```
 [1] "Dallas TX"                  "Denver CO"                  "Austin TX"
 [4] "Salem OR"                   "Salem MA"                   "California
     PA"
 [7] "Long Beach CA"              "San Francisco CA"           "Texas MO"
[10] "Nevada MO"                  "St. Louis MO"               "Truth or
    Consequences NM"
[13] "Charleston SC"              "Charleston NC"
```

Step 2. Cut out the abbreviated state names from the ends

```
head(cities2, 5)
```

```
[1] "Dallas TX" "Denver CO" "Austin TX" "Salem OR"  "Salem MA"
```

```
cities3 <- gsub(" ..$", "", cities2)
head(cities3, 5)
```

```
[1] "Dallas" "Denver" "Austin" "Salem"  "Salem"
```

We deleted a space (" ") and any two characters ("..") at the end ("$").

Step 3. Cut out city names to get state names

- This seems a bit too easy after all of that work.
- Abbreviated state names, called `stabbs` here, results from replacing everything up to the last space with “”

```
stabbs <- gsub(".* ", "", cities2)
stabbs
```

```
 [1] "TX" "CO" "TX" "OR" "MA" "PA" "CA" "CA" "MO" "MO" "MO" "NM" "SC"
     "NC"
```

- This exploits a regular expression default, “greedy matching”. The match goes as far as logically possible, so it picks up all of the spaces until the last space.

Step 4. Create full state names from abbreviations

- The project requires full state names, not just abbreviations.
- Use the named `state.name` vector to pull out the long names, one for each abbreviation

```
stnames <- state.name[stabbs]
stnames
```

```
             TX              CO              TX              OR
                             MA
       "Texas"      "Colorado"          "Texas"        "Oregon"
           "Massachusetts"
             PA              CA              CA              MO
                             MO
"Pennsylvania"   "California"    "California"      "Missouri"
          "Missouri"
             MO              NM              SC              NC
    "Missouri"    "New Mexico" "South Carolina" "North Carolina"
```

Put that together in data.frame to finish

```
reslt <- data.frame(address = cities,
                    cities = cities3, st.abb = stabbs,
                    st.name = stnames)
head(reslt)
```

```
                   address      cities st.abb        st.name
1             Dallas Texas      Dallas     TX          Texas
2          Denver Colorado      Denver     CO       Colorado
3             Austin Texas      Austin     TX          Texas
4             Salem Oregon       Salem     OR         Oregon
5     Salem Massachusetts       Salem     MA Massachusetts
6 California Pennsylvania  California     PA   Pennsylvania
```

How did we do this without RE?

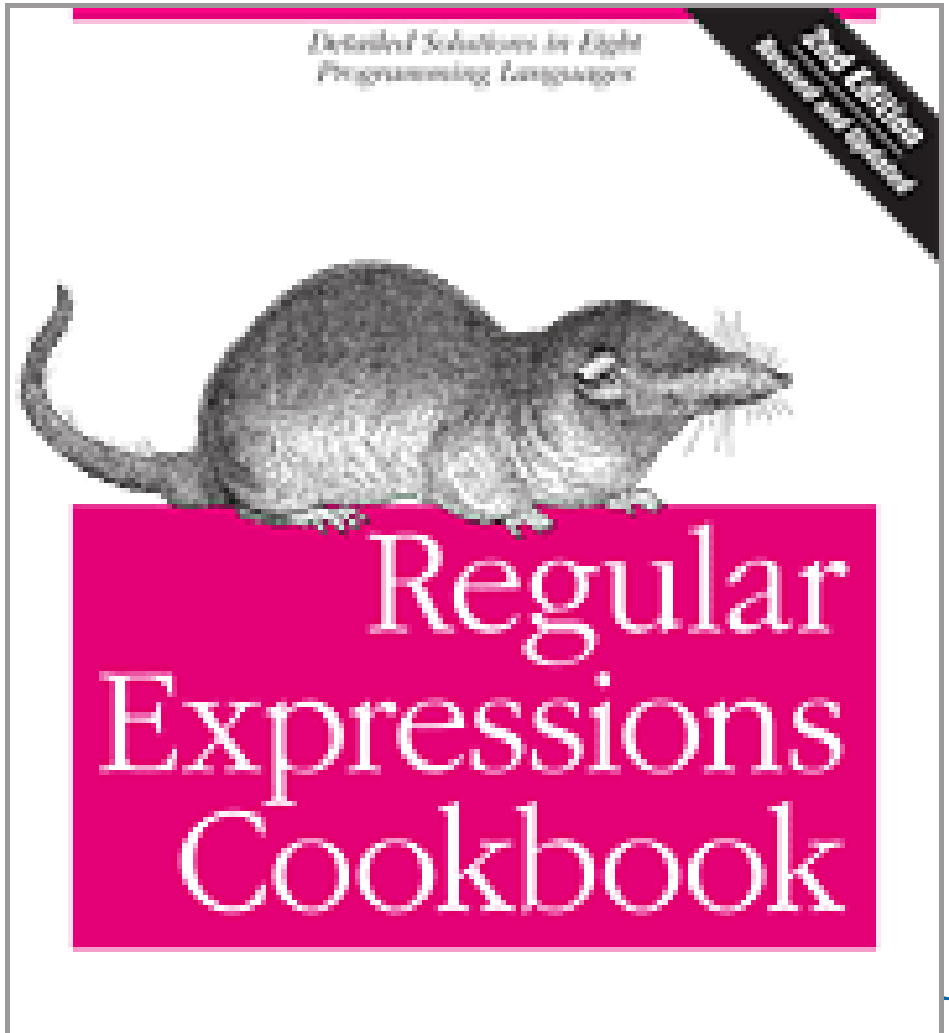
- We have gotten this wrong by trying to split on both spaces and commas with the R function “strsplit”.

Outline

How did we do this without RE?

- Don't be too intimidated by regular expressions
- Don't let your life be consumed by them either

Levithan (2012) strike the right balance for me



References

Levithan, Steven, J. G. (2012). *Regular Expressions Cookbook*. O'Reilly Media.

R Core Team (2017). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria.

Session

```
sessionInfo()
```

```
R version 3.6.0 (2019-04-26)
Platform: x86_64-pc-linux-gnu (64-bit)
Running under: Ubuntu 19.04

Matrix products: default
BLAS:   /usr/lib/x86_64-linux-gnu/atlas/libblas.so.3.10.3
LAPACK: /usr/lib/x86_64-linux-gnu/atlas/liblapack.so.3.10.3

locale:
 [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C
     LC_TIME=en_US.UTF-8
 [4] LC_COLLATE=en_US.UTF-8     LC_MONETARY=en_US.UTF-8
     LC_MESSAGES=en_US.UTF-8
 [7] LC_PAPER=en_US.UTF-8       LC_NAME=C                  LC_ADDRESS=C
[10] LC_TELEPHONE=C             LC_MEASUREMENT=en_US.UTF-8
    LC_IDENTIFICATION=C

attached base packages:
[1] stats     graphics  grDevices utils     datasets  methods   base

other attached packages:
[1] kutils_1.69
```

Session ...

20

```
loaded via a namespace (and not attached):
 [1] compiler_3.6.0 plyr_1.8.4     tools_3.6.0    foreign_0.8-71
     lavaan_0.6-3   Rcpp_1.0.1
 [7] mnormt_1.5-5   pbivnorm_0.6.0 xtable_1.8-4   zip_2.0.2
     openxlsx_4.1.0 stats4_3.6.0
```