Fake Foreign Interface (FFI-2)

Paul E. Johnson[1] [2]

[1]Department of Political Science

[2]Center for Research Methods and Data Analysis, University of Kansas

2012

Outline

Outline

What this Lecture is About

- Some R packages masquerade as usages of the foreign function interface.
- They follow this approach. From R:
    - Write a text file of program syntax for the other program
    - Use R commands to interact with the operating system
        - Call a compiler on that syntax if necessary
        - Run the program in a separate shell
        - That other program must write results on disk
    - Use R to harvest the results from the disk file
- Examples: OpenBUGS (BUGS code), SabreR (Fortran), MPlusAutomation

Outline

paste and paste0: Combine text with values

- Read ?paste
  - creates text strings that can be written into files
  - can interleave text with values from the R session

```
> A ← 18
> B ← 0.1
> myStr ← paste("A is: ", A, "B is: ", B)
> myStr
[1] "A is: 18 B is: 0.1"
```

paste and paste0: Combine text with values

- Default paste will insert one blank space between arguments
  - can be controlled by argument sep="" (sep equals quoted nothing!)
  - or the new paste0 function, which is the same as paste, except it does not insert spaces.
- paste is vectorized. For example:

```
> B ← c(0.1, 0.2, 0.3)
> paste("Combining A:", A, "and B:", B)
 [1] "Combining A: 18 and B: 0.1" "Combining A:
    18 and B: 0.2"
 [3] "Combining A: 18 and B: 0.3"
```

gsub: for replacing text strings

- Read ?gsub. gsub can replace a string with a value in a text variable.

```
> str1 ← "mary had a little tiger"
> gsub("tiger", "lamb", str1)
[1] "mary had a little lamb"
```

- The second argument need not be a character string. It can also be an R variable name.

gsub: for replacing text strings

- Consider a tedious usage of paste:

```
> myStr ←paste("A is: ", A, "And A is still:",
     A, "after we look at A, which is", A)
> myStr
[1] "A is:  18 And A is still: 18 after we look
     at A, which is 18"
```

- That's a silly example, but many projects that write code for Mplus or SAS need to get some value and put it in text in several different places.

gsub: for replacing text strings

- Consider this less tedious alternative, which uses a "marker" VALA for the value of A

```
> myStr2 ← "A is: VALA And A is still: VALA
    after we look at A, which is VALA"
> myStr3 ← gsub("VALA", A, myStr2)
> myStr3
[1] "A is: 18 And A is still: 18 after we look
    at A, which is 18"
```

- Generally, if a paste command repeatedly has to insert the same value, I believe it is smarter to use gsub than it is to write tedious paste code.

cat: for writing files

- Read ?cat
- because ... is first, all arguments after that must be NAMED.
- file = “some-good-name.txt”
  - file must be a valid character string for the output file name
- append = TRUE or FALSE
  - append arguments controls whether an existing file will be erased and replaced when cat runs.
- sep = “\n”
  - can simplify the entry of information by inserting a blank line after each element.
- I’ve never needed to change the fill or label options.

cat: for writing files

- Simplest possible example:

```
> cat("Hello, my name is \n Paul. Who \n are \n
    you", file="practice.txt")
```

- The output file will look like this:

```
Hello, my name is
 Paul. Who
 are
 you
```

- Note
  - Effect of "\n" (new line)
  - indentation of lines 2-4 due to spaces I typed in after "\n" (possibly an accident on my part!)

Two more convenient ways to get new lines

- Method 1: use sep="\n"

```
cat("Hello, my name is", "Paul. Who","are","you
    ", sep="\n", file="practice2.txt"))
```

- The output file will look like this:

```
Hello, my name is
Paul. Who
are
you
```

Two more convenient ways to get new lines

- Method 2:

```
myMessage ← 'Hello, my name is
Paul. Who
are
you'
cat(myMessage, file = "practice3.txt")
```

- The output file will be identical to the result from Method 1.
- In my opinion, this approach is more readable.

Outline

We want to write an Mplus script file

Here's an example of an Mplus command file that receives a file called "rundata.dat".

```
TITLE:
  Alternative Fit Indices

DATA:
  FILE IS "rundata.dat";

VARIABLE:
  NAMES ARE y1-y6 cluster;
  USEVARIABLES ARE y1-y6;

MODEL:
  F1 BY y1-y3*.7;
  F2 BY y4-y6*.7;
```

We want to write an Mplus script file ...

```
  y1-y6*.51;
  F1@1.0;
  F2@1.0;
  F1 WITH F2*.3;

OUTPUT:
  STDYX;
```

Write that From R

The work of writing that file from R is quite simple, of course, if nothing needs to be changed.

Make a giant quoted string variable, including the carriage returns in the obvious way, and cat it out.

```
mpprog ← '
TITLE:
   Alternative Fit Indices;

DATA:
  FILE IS "rundata.dat";

VARIABLE:
  NAMES ARE y1−y6 cluster;
  USEVARIABLES ARE y1−y6;
```

Write that From R ...

```
MODEL:
   F1 BY y1-y3*.7;
   F2 BY y4-y6*.7;
   y1-y6*.51;
   F1@1.0;
   F2@1.0;
   F1 WITH F2*.3;

OUTPUT:
  STDYX;
"

cat(mpprog, file = "myMplus.inp")
```

Note: No need for “\n” when the string is multi-line.

That is MUCH better than a long set of cat commands

One of my students figured out that this "works", and then he taught all of the other students to follow the same tedious style. This ugly style of coding is surprising attractive to R novices:

```
myFile <- "MyMplusCode.inp"
cat("TITLE: \n", file = myFile)
cat("Alternative Fit Indices; \n", file = myFile,
    append = TRUE)
cat("DATA: \n", file = myFile, append = TRUE)
cat("  FILE IS \"rundata.dat\"; \n", file = myFile,
     append = TRUE)
cat("VARIABLE: \n", file = myFile, append = TRUE)
```

That is MUCH better than a long set of cat commands ...

```
cat("  NAMES ARE y1-y6 cluster;\n", file = myFile,
    append = TRUE)
cat("  USEVARIABLES ARE y1-y6;\n", file = myFile,
    append = TRUE)

cat("MODEL: \n", file = myFile, append = TRUE)
cat("  F1 BY y1-y3*.7; \n", file = myFile, append =
     TRUE)
cat("  F2 BY y4-y6*.7; \n", file = myFile, append =
     TRUE)
cat("  y1-y6*.51; \n", file = myFile, append = TRUE
    )
cat("  F1@1.0; \n", file = myFile, append = TRUE)
cat("  F2@1.0; \n", file = myFile, append = TRUE)
```

That is MUCH better than a long set of cat commands ...

```
cat("  F1 WITH F2*.3; \n", file = myFile, append =
    TRUE)

cat("OUTPUT:\n", file = myFile, append = TRUE)
cat("  STDYX;\n", file = myFile, append = TRUE)
,
```

- Note: “\n” is required in this style
- This is annoying, tedious and difficult to maintain, in my opinion.
- If you really do want to treat each line of code separately, be less odious:

That is MUCH better than a long set of cat commands ...

- use only one cat function
- use the sep = "\n" argument, like so:

```
myFile ← "MyMplusCode.inp"
cat("TITLE:",
"  Alternative Fit Indices;",
" ",
"DATA:",
"  FILE IS \"rundata.dat\";",
" ",
"VARIABLE:",
"  NAMES ARE y1-y6 cluster;",
"  USEVARIABLES ARE y1-y6;",
"  ",
"MODEL:",
```

That is MUCH better than a long set of cat commands ...

```
"    F1 BY y1-y3*.7;",
"    F2 BY y4-y6*.7;",
"    y1-y6*.51;",
"    F1@1.0;",
"    F2@1.0;",
"    F1 WITH F2*.3;",
"    ",
"OUTPUT:",
"   STDYX;", sep = "\n", file = myFile)
```

Complications arise in "adjusting" the output file

- Suppose the user's R code generates 100s of data files, named "data-1.dat", "data-2.dat" ...
- So we need to write separate Mplus command files, one for each data file
- That means we need to edit the above code, to replace the word "rundata.dat" with "data-1.dat" or "data-2.dat" and so forth. And we need a unique name for each of those Mplus files.
- In the following example, I'll just demonstrate code that works for one particular run (will be obvious to reader how to write a for loop that does same for all values of "run").

My suggested method for "splicing in" data-7.dat

Necessary to use one paste to combine the text string with the run number.

Necessitates breaking the text block of the program into separate pieces.

Can write in one large cat command, but seems more manageable to do this:

```
run ← 7

mpprog1 ← '
TITLE:
  Alternative Fit Indices;

DATA:
```

My suggested method for "splicing in" data-7.dat ...

```
  FILE IS '

mpprog2 <- paste("\"data-", run, ".dat\"; \n", sep
   = "")

mpprog3 <- '
VARIABLE:
  NAMES ARE y1-y6 cluster;
  USEVARIABLES ARE y1-y6;

MODEL:
  F1 BY y1-y3*.7;
  F2 BY y4-y6*.7;
  y1-y6*.51;
  F1@1.0;
```

My suggested method for "splicing in" data-7.dat ...

```
  F2@1.0;
  F1 WITH F2*.3;

OUTPUT:
  STDYX;
',

outFileName ← paste("myMplus-", run, ".inp", sep =
     "")
cat(mpprog1, file = outFileName)
cat(mpprog2, file = outFileName, append = TRUE)
cat(mpprog3, file = outFileName, append = TRUE)
```

I believe gsub usage is cleaner

Insert a “marker” UUUfn in place of “rundata.dat”.

Use gsub to replace that after.

This is used in hpcexamples: Ex09-MplusRunall-2

```
mplusinp ← 'TITLE:
  Alternative Fit Indices

DATA:
  FILE IS \"UUUfn\";

VARIABLE:
  NAMES ARE y1−y6 cluster;
  USEVARIABLES ARE y1−y6;

MODEL:
```

I believe gsub usage is cleaner ...

```
   F1 BY y1-y3*.7;
   F2 BY y4-y6*.7;
   y1-y6*.51;
   F1@1.0;
   F2@1.0;
   F1 WITH F2*.3;

OUTPUT:
   STDYX;
'

mplusinp <- gsub("UUUfn", flist[i], mplusinp)
mpinpname <- paste(flist[i],".inp", sep="")
cat(mplusinp, file=mpinpname)
```

What if there are many changes to be written?

- If the mplus code has one R variable in many locations, don't write separate paste() statements for each one.
- Put a text maker on each one
- Then USE gsub !

Make A LOT of changes

Suppose a use wants to replace “y6” by “y8”.?

Put in a marker with each “y6”.

```
mplusinp ← 'TITLE:
  Alternative Fit Indices

DATA:
  FILE IS \"UUUfn\";

VARIABLE:
  NAMES ARE y1−UUUy6 cluster;
  USEVARIABLES ARE y1−UUUy6;

MODEL:
  F1 BY y1−y3∗.7;
```

Make A LOT of changes ...

```
  F2 BY y4–UUUy6*.7;
  y1–UUUy6*.51;
  F1@1.0;
  F2@1.0;
  F1 WITH F2*.3;

OUTPUT:
  STDYX;
'

mplusinp ← gsub("UUUy6", "y8", mplusinp)
mplusinp ← gsub("UUUfn", flist[i], mplusinp)
mpinpname ← paste(flist[i],".inp", sep="")
cat(mplusinp, file=mpinpname)
```

.Call

.Call