

EmpoweR your Application with R!

Sudipta Basu

Pravin Holkar

Introduction

Make our application so dynamic that it goes beyond the scope of its specifications?

Is it Possible?

- Is it only a pipedream or is it really possible?
- For example, consider the following three scenarios

Scenario 1

We have an application which mirrors an oncology trial, which simulates time to event data using the exponential distribution.

But a user wants to simulate data using Weibull distribution which is not available in the software.

Scenario 2

An application provides various types of output including some plots and tables.

But the user may want to create some other plots not provided by the application.

Scenario 3

Our application provides output in some particular format.

But the user may want the report in some other formats.

Can we achieve this in an automated manner
without modifying application code?

Solution

The answer is :

YES!

We can achieve these and lot more by harnessing the **Power of R** through our application.

R Application Program Interface

- R provides an API to access its functions from C code.

MODULE 1

MODULE 2

MODULE 3



Roadmap

- Access R data types from C code and create R variables
- Manage memory allocation and de-allocation.
- Call R function from C code using this API

R Internals

- This API is known as the R Internals
- It consists of **R.dll** which contains the corresponding functionalities
- Also allows access to **Rinternals.h** listing down all the R Internals functionalities

APIs in R Internals

- **R_tryEval** : Executes R statements from C code
- **Rf_allocVector** : Allocating memory to a vector
- **R_GlobalEnv** : The global R environment
- **Rf_getAttrib** : Gets a variable attribute from R

SEXP

- Fundamental structure defined in C.
- Encapsulates all data types in R.
- Short for “symbolic expression”.
- Different subtypes of SEXPs like INTSXP, REALSXP, STRSXP, LGLSXP, VECSXP and more.
- Subtypes can be accessed using data accessors.

Loading R

- Execute this code to load R in your environment

```
char *Args[] = {"R", "--silent"};
```

```
Rf_initEmbeddedR(sizeof(Args) /  
sizeof(Args[0]), Args);
```

Unloading R

- At the very end, execute the following code to unload R from your environment

```
Rf_endEmbeddedR ( 0 ) ;
```

Data Types

At the heart of every C function in the R API are conversions between R data types and C data types.

R Data Type	Description	C Data Types
integer	Specifies integer vectors or scalars	int, int*, int[]
numeric	Specifies floating point vectors or scalars	double, double*
logical	Boolean variables taking one of the two values TRUE / FALSE	bool
character	Strings in R	char

Advanced Data Types

R Data Type	Description	C Data Types
matrix	Collection of data elements of the same type arranged in a two-dimensional rectangular layout	int[], int**, double[], double**, etc.
list	Collection of data elements of the same type arranged in a two-dimensional rectangular layout	User defined type (linked lists, struct, void*)
data.frame	A data table capable of storing different types of data	User defined type (void*)

Creating R Variables

- R code for allocating a vector of integers of size 2

```
N <- integer()  
N[1] <- 4  
N[2] <- 7
```

- C code for allocating a vector of integers of size 2

```
SEXP N;  
N = Rf_allocVector(INTSXP, 2);  
PROTECT(N);  
  
INTEGER(N)[0] = 4;  
INTEGER(N)[1] = 7;
```

Retrieving R Variables

- R code Retrieving integers from the vector

```
nVal <- N
```

- C code Retrieving integers from the vector

```
int nVal[2];  
int nLen;  
  
nVal = INTEGER(N);  
nLen = LENGTH(N);  
  
UNPROTECT(1);
```

Creating R Variables

- R code for allocating a vector of integers of size 2

```
N <- integer()  
N[1] <- 4  
N[2] <- 7
```

- C code for allocating a vector of integers of size 2

```
SEXP N;  
N = Rf_allocVector(INTSXP, 2);  
PROTECT(N);  
  
INTEGER(N)[0] = 4;  
INTEGER(N)[1] = 7;
```

Creating R Variables

- R code for allocating a vector of integers of size 2

```
N <- integer()  
N[1] <- 4  
N[2] <- 7
```

- C code for allocating a vector of integers of size 2

```
SEXP N;  
N = Rf_allocVector(INTSXP, 2);  
PROTECT(N);  
  
INTEGER(N)[0] = 4;  
INTEGER(N)[1] = 7;
```

Retrieving R Variables

- R code Retrieving integers from the vector

```
nVal <- N;
```

- C code Retrieving integers from the vector

```
int nVal[2];  
int nLen;  
  
nVal = INTEGER(N);  
nLen = LENGTH(N);  
  
UNPROTECT(1);
```

Retrieving R Variables

- R code Retrieving integers from the vector

```
nVal <- N;
```

- C code Retrieving integers from the vector

```
int nVal[2];  
int nLen;  
  
nVal = INTEGER(N);  
nLen = LENGTH(N);  
  
UNPROTECT(1);
```

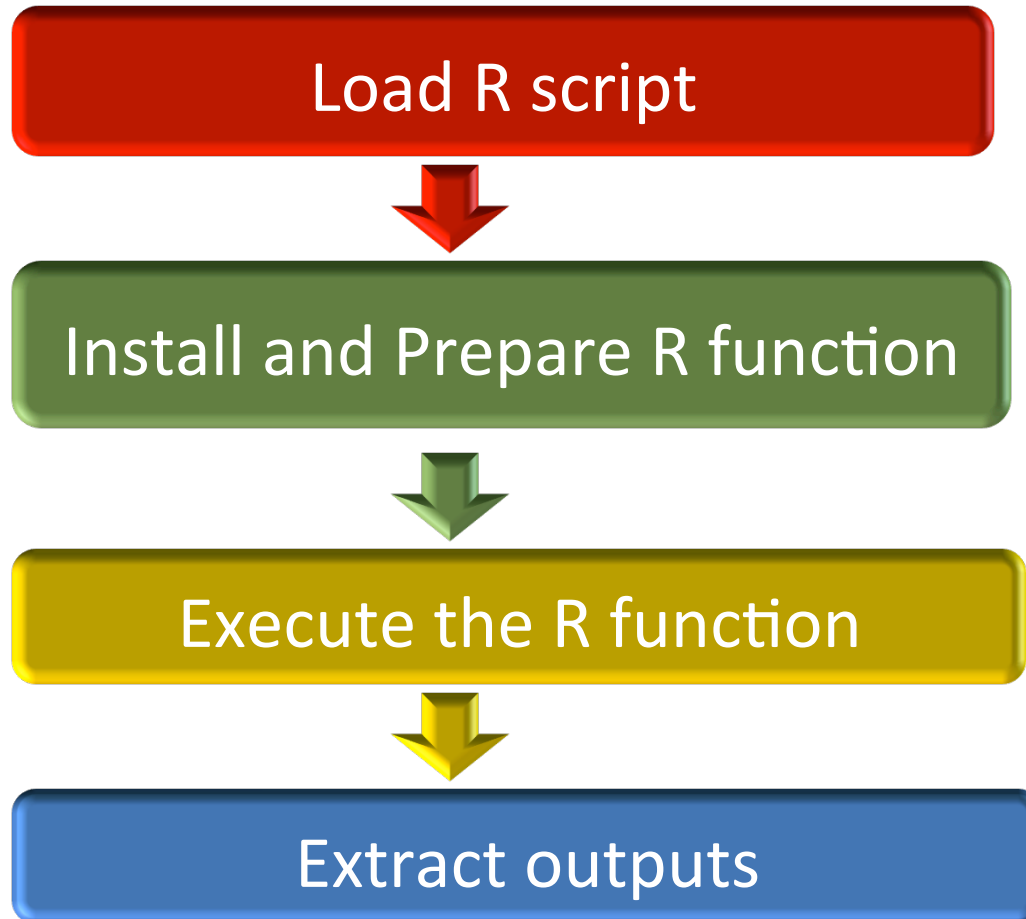
Garbage Collection



Garbage Collection

- A method to release the unused memory used in many languages
- The PROTECT() call tells R that an object is in use and shouldn't be deleted.
- We need to unprotect every protected object by calling UNPROTECT().

Calling R from C



Scenario 1: Weibull Distribution

```
GenWeibull <- function(Count, TrtID, HazardRates)
{
  time <- c()

  for(m in 1:Count) {
    j <- TrtID[m]

    time[m] <- rweibull(n = 1, shape = 2, scale = 1 /
      HazardRates[j+1])
  }

  return(SurvivalTime = as.double(time))
}
```

Loading the R script from C

Load an R
script

Install and
Prepare R
function

Execute R
function

Extract
outputs
from R

We use the R command **source** to load the script file that contains **GenWeibull()**

```
R_tryEval (Rf_lang2 (install ("source"),  
mkString ("Weibull.R")), R_GlobalEnv,  
&nError) ;
```

Installing the R function

Load an R script

Install and Prepare R function

Execute R function

Extract outputs from R

- To pass the function to R, we create a linked list
- We pass the function name to R as the first node of that linked list

```
SEXP NextArg = ArgList =  
Rf_allocList(ArgCount+1);  
  
SET_TYPEOF(ArgList, LANGSXP);  
SETCAR(ArgList, install("GenWeibull"));
```

Preparing the Arguments

Load an R script

Install and Prepare R function

Execute R function

Extract outputs from R

- Next, we create an SEXP object for the function argument to R

```
ArgArr[0] = Rf_allocVector(INTSXP, 1);  
INTEGER(ArgArr[0])[0] = Count;
```

Passing the Arguments

- After that, we create a linked list of function arguments

```
NextArg = CDR(NextArg) ;  
SETCAR(NextArg, ArgArr[0]) ;
```

Load an R
script

Install and
Prepare R
function

Execute R
function

Extract
outputs
from R

Passing the other Arguments

Load an R
script

**Install and
Prepare R
function**

Execute R
function

Extract
outputs
from R

We then pass the other two arguments `TrtID` and `HazardRates` to the R function in a similar manner

Executing R Function

Load an R
script

Install and
Prepare R
function

**Execute R
function**

Extract
outputs
from R

```
RetVal= R_tryEval(ArgList,  
R_GlobalEnv, &nErr);
```

Extracting R Output

Load an R
script

Install and
Prepare R
function

Execute R
function

Extract
outputs
from R

```
SEXP names = Rf_getAttrib(RetVal,  
R_NamesSymbol);  
  
SEXP data = VECTOR_ELT(RetVal, 0);  
  
memcpy(Time, REAL(data),  
length(data) * sizeof(double));
```

Weibull Data

Load an R
script

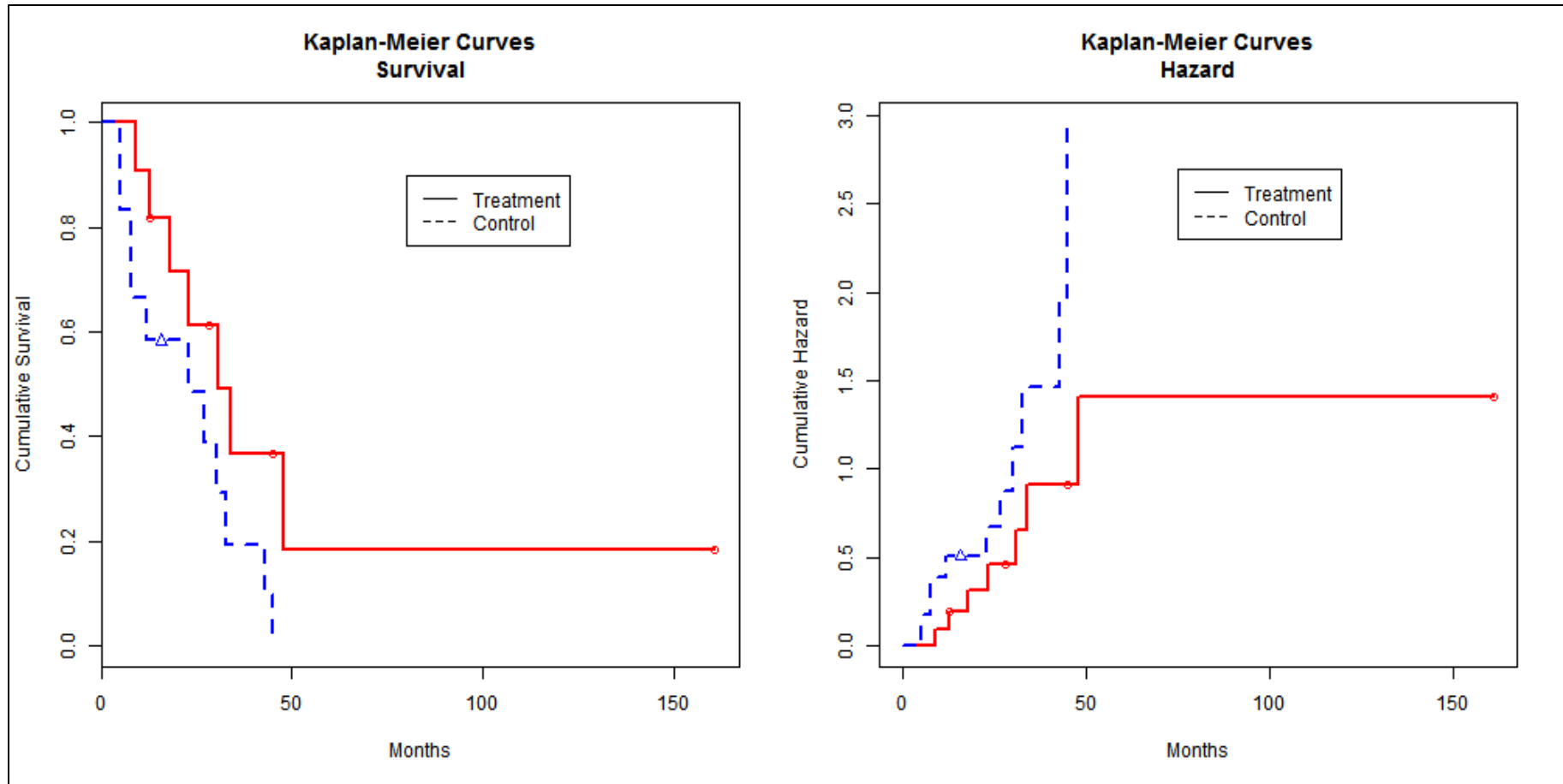
Install and
Prepare R
function

Execute R
function

Extract
outputs
from R

Patient ID	Time to Event
1	46.027
2	19.920
3	27.784
4	67.754
5	2.802
6	14.687
7	9.783
8	46.434
9	28.576
10	31.156

Scenario 2: Kaplan Meier Plot



Scenario 3: Customized Reports

This is an R Markdown document. Markdown is a simple formatting syntax for authoring HTML, PDF, and MS Word documents. For more details on using R Markdown see <http://rmarkdown.rstudio.com>.

Here, we have got the time to event data for 23 patients. A snapshot of the data gives us a glimpse into its variable values. Also, this data has 3 columns namely, time, status and trtid. A better idea about these variables can be obtained from the summary of the data.

time	status	trtid
9.163	1	1
12.9	1	1
12.91	0	1
18.04	1	1
23.19	1	1
28.1	0	1

Table: Time-to-event Data

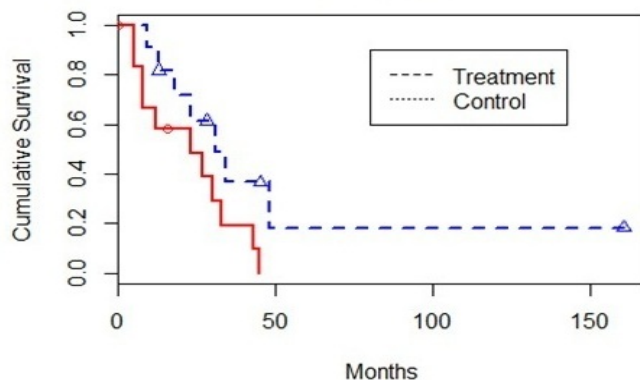
time	status	trtid
Min. : 0.00	Min. :0.00	Min. :0.000
1st Qu.: 11.15	1st Qu.:0.75	1st Qu.:1.000
Median : 23.15	Median :1.00	Median :1.500
Mean : 28.23	Mean :0.75	Mean :1.458
3rd Qu.: 33.13	3rd Qu.:1.00	3rd Qu.:2.000
Max. :160.89	Max. :1.00	Max. :2.000

Table: Variable Summary of Time-to-event Data

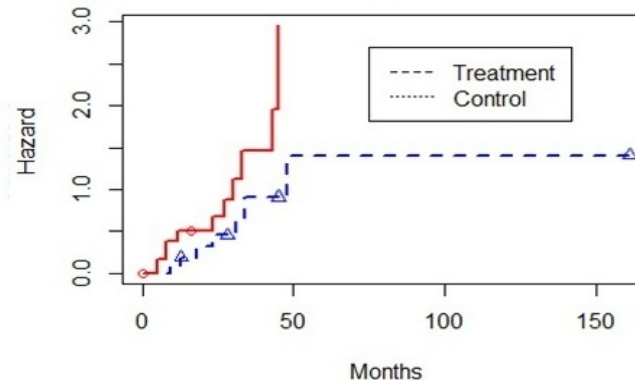
With this data, we have generated 2 Kaplan Meyer plots. One is for Events, while the other is for Survival corresponding to the control and the active treatment.

Loading required package: splines

**Kaplan-Meier Curves
Survival**



**Kaplan-Meier Curves
Hazard**



Limitations

- We have no control over what user does in R code
- The API provided by R cannot be used in a multithreaded environment
- Speed may be slower compared to the case in which the logic had been directly coded in application
- The process is dependant on the version of R user is using

Summary

- It makes an application significantly more powerful, flexible and dynamic.
- It gives more choice and power to the user by tapping into the gigantic repertoire of R.
- No need to spend extra resources or time to access seemingly limitless R functionalities.
- This technique has been used in East[®]

References

1. ***R Internals: R-ints.pdf, R-ints.html*** kept in
RHome\doc>manual
2. ***Advanced R - Hadley Wickham:***
<http://adv-r.had.co.nz/C-interface.html>
3. ***Native Interfaces for R*** - Seth Falcon, 2010

Any Questions ?



Thank You!