RSiena: Design

R Routines

R.M. Ripley

Department of Statistics
University of Oxford

2010

Outline

1. Introduction
2. Some points about R
3. R routines of RSiena
4. Printing
5. Interface to C++
6. Effects
7. Estimation /Simulation
8. Building
9. Parallel runs
10. Profiling

RSiena

Consists of

- **R functions**
  - GUI (using the package TCL/TK) for input or display
  - Robbins Monro algorithm
  - Printing
  - Everything else but the simulation code
- **C++ functions**
  - Simulation code: for execution speed
  - Interface: copy data from R to C++, return results
  - Interface is discussed in detail here
- Other files, required or permitted parts of an **`R package`**

R packages

For details, see *Lecture 10* of on-line course

- Well-defined source structure (details in *Writing R Extensions*)
- Standard method for build
- Distributed as **`.tar.gz`** for Linux, binaries for Windows or Mac
- Binaries do not contain the source: one can access the R source but not other languages from within R.
- **`install.packages`** function builds on Linux, not usually on Windows or Macs
- Help files written in **`.Rd`** format, converted to **`tex`**, **`text`** or **`html`**
- Packages distributed via **`CRAN`** must pass **`checks`**

Object Oriented programming in R

For details, see *Lecture 8* of on-line course

- Classes: **`S3`** (informal) or **`S4`** (formal).
- **`S4`** classes have a well-defined structure. Very tedious in development.
- **`S3`** classes cannot be relied upon to match the structure expected, so some defensive code is necessary. But easy to change, and easy for users to examine and change too.
- Only S3 classes defined in RSiena
- **`Matrix`** package contains S4 classes
- Methods: functions which adapt to the class of the argument
- Standard behaviour if one types the name of an object at the command prompt in R is to call the appropriate **`print`** method
- RSiena classes (may) have **`print`**, **`summary`**, **`xtable`** (to create **`tex`**) methods
- Code to handle sparse matrices in RSiena uses S4 methods

Namespaces

For details see *Lecture 10* of on-line course.

- Avoids name clashes between packages.
- Possible that RSiena could have some object with the same name as something later on the search path.
- This would lead to unpredictable errors, depending on the order of loading packages, and changes in packages.
- With a namespace we only have to worry about the exported, visible names clashing.
- All names could be disambiguated using a prefix of **`RSiena::`**, but this is rather tedious.
- So: namespaces are optional, but RSiena uses one

Namespaces: consequences

- Only exported objects are visible to the user
- Can access others using **`RSiena:::`** or **`getAnywhere`** or **`getFromNamespace`**.
- Alters the search path (*details*)
- Must import **`Matrix`** package (because search path is altered, and **`Matrix`** redefines some basic R functions (e.g. **`print`**!))
- Need to import **`Matrix`** creates a delay on loading RSiena : but no further delay to use sparse matrices!
- Complicates alterations to a built package: can only alter the non-exported functions without rebuilding, using **`fixInNamespace`**
- May be possible to remove by renaming the **`NAMESPACE`** file and adding **`library(Matrix)`** commands where appropriate. But probably need to alter dll loading too. (*details*)

Documentation

- R help files: displayed within R by e.g. **`?RSiena`**, **`?siena07`**
- Can create aliases: **`?xxx`** can be linked to a particular help file
- **`pdf`** file containing all the help files is on **`CRAN`**
- Manual for RSiena is stored in **`pdf`** format in the package and can be accessed via the **`Help`** button on the GUI or via **`RShowDoc("s_man400",package="RSiena")`**
- One day we may have **`vignettes`** which contain **`R`** code and the corresponding output (automatically updated!).
- Source documentation: I have written code to produce an **`R`** data frame, csv file or LaTeX table, detailing calls/called by/source file/type. An example, minimally edited in Excel, is shown here.
- We use Doxygen system to produce very good source documentation for C++ . See RSienaDeveloper.pdf for details of how to get this documentation.

R Source documentation: details

- Each function contains a comment at the start. Begins with **`##@`**
- e.g. **`##@xxx yyy zzz`**
- where **`xxx`** is name of function, **`yyy`** is type of function, **`zzz`** is notes about function
- For internal functions, use
  `##@xxx internal fn`
  where **`fn`** is the (non-internal) function in which defined.
- Enclosing functions of internal functions need new parameter: **`getDocumentation=FALSE`** and at start of function proper:

```
if (getDocumentation)
   return(getInternals())
```


R Source documentation: producing it

- The function uses **`xtable`** and **`codetools`** and writes various intermediate files.
- Output produced by
  `R -vanilla < RSienaRDocumentation.r`
- or, within R with RSiena loaded,
  `RSiena:::getRSienaDocumentation(rdir)`
  where **`rdir`** is the path of the RSiena R source.

Functions

- All arguments are passed by value.
- To allow update of arguments, I frequently use

```
z <- myfn(z, ...)
```

where

```
myfn <- function(z, ...)
{
    ....
    z
}
```

More details (*here*)

Global variables: the problem

- Variables which may apparently be used before they are defined are highlighted in the **`check`** procedure.
- Although not a fatal failure, I found several mistypings by working through the list, so it is worth removing them
- The check is not very clever, so will not pick up elements of an object which do not yet exist, or check objects created with **`assign`** statements.
- Tcl/tk routines need many globals in order to communicate with call back routines.

Global variables: two solutions

1. Make routines internal to others: the outer function can define the globals. Used in **siena01Gui**, where there are many call backs. Makes source files unwieldy, and difficult to document automatically.
2. Create functions which encapsulate the variables. Set and retrieve the variables by function calls. Used for **Report**, **UserInterrupt** flags, and **FRANstore** which allows data to be passed to other processors when using more than one. e.g.
   FRANstore(f)
   sets the value and
   f <- FRANstore()
   retrieves the value.

The code for the second method is rather unintuitive: I suggest you adapt a copy rather than trying to understand it!

Source control systems

- We use 2! One, at Nuffield and one at R-forge. (Note: Not Rforge)
- R-forge will automatically build and make available to users, provided all is OK
- R-forge has two packages: RSiena and RSienaTest
- Access to R-forge is available to all
- Read access to R-forge is possible from any computer with web access.
- Nuffield one is easier to use, although it does not insist on a log entry or allow the update of log entries when I forget!
- Easiest way to use them is via **tortoiseSvn** on Windows, which has an interface with Windows Explorer (although some of you don't use that!)
- I have used **svnX** on Mac.

Visible R objects of RSiena

To list: **library(RSiena); ls("package:RSiena")**

Estimation	**siena07**, **siena08**, **iwlsm**, **sienaTimeTest**
Data creation	**sienaDataCreate**, **sienaGroupCreate**, **sienaDataCreateFromSession**
Model creation	**allEffects**, **getEffects**, **includeEffects**, **includeInteraction**, **includeTimeDummy**, **sienaModelCreate**, **setEffect**, **sienaDataConstraint**
GUI interface	**siena01Gui**
Utilities	**coCovar**, **coDyadCovar**, **sienaNet**, **sienaNodeSet**, **sienaCompositionChange**, **sienaCompositionChangeFromFile**, **varCovar**, **varDyadCovar**
Print	**print01Report**
Data objects	**s501**, **s502**, **s503**, **s50a**, **tmp3**, **tmp4**
Windows only	**installGui**
Documentation	**effectsDocumentation**

Adding visible data objects to RSiena

The visible data objects were created by

- Reading the data files into R
- Saving the complete set:
  save(s501, s502, s503, s50a, tmp3, tmp4, file="RSiena.rda")
- Moving the file "RSiena.rda" to the data sub-directory of the package directory structure.
- Adding the names to the list of exports in **NAMESPACE** file
- Creating a help page for each object in the **man** sub-directory (otherwise the **CHECK** process will fail).

To add another, simply read it into R, and then save all the data objects to the **data** sub-directory. Add the new name to the **NAMESPACE** file and create a help page for it in the **man** sub-directory: you can use **prompt** to create a skeleton help file. Then commit everything, remembering to complete a log entry!

Adding other objects

allEffects This object defines the effects for each type of dependent variable or covariate. You can add rows or columns to it to define new effects or change the behavior of existing ones. Make changes by editing **data/allEffects.csv** which is read in at package creation time. The read command is in the file **data/allEffects.R** (This is a clever feature of R.)

R function Write the function, put it in the **R** sub-directory of the package, **add** it to the repository and then **commit** it. If it is to be visible to users, also add it to the **NAMESPACE** file and create a help file.

Adapting **siena07**

Suppose you want to just simulate, and collect the returned statistics and simulations for later use. siena07 returns the simulations.

```
mynet1 <- sienaNet(array(c(s501, s502, s503),
    dim=c(50, 50, 3)))
mydata <- sienaDataCreate(mynet1)
myeff <- getEffects(mydata)
mymodel <- sienaModelCreate(nsub=0, n3=100)
ans <- siena07(mymodel, data=mydata, effects=myeff,
     returnDeps=TRUE)
or
ans <- siena07(sienaModelCreate(nsub=0, n3=100),
    data=mydata, effects=myeff, returnDeps=TRUE)
```

ans contains the statistics and **ans$sims** the simulated values for the network.

Adding C functions

- Write the function, and test it thoroughly.
- Put it in an appropriate place in the **src** sub-tree.
- Make sure **R CMD INSTALL siena** still works. If you have altered any header files, you will find **R CMD build siena** helpful to clear out all the old object files.
- If you have extended the **src** structure, you need to alter **Makevars.win** or **Makevars** to include the new subdirectory
- Make the corresponding change in the other Make file and in the cleanup files.
- Please run **R CMD build** followed by **R CMD check** on the built tarball.
- When very sure, add and commit the files.

Utility routines

- Create the basic level **RSiena** objects, by adding *attributes* of **class** and any other necessary information to **R** objects
- Do some validation of the object
- Necessary so that **sienaDataCreate** knows what the objects are
- Do not call other RSiena functions
- Called by **siena01Gui** when the **Apply** button is clicked
- Can be used directly

RSiena Data Classes

Look slightly different in R and C++

sienaNodeSet	Actor set, used to distinguish nodes in data sets with multiple or two-mode networks.
sienaNet	A single dependent variable, (i.e. network or behavior variable, all waves)
coCovar	Constant covariate
coDyadCovar	Constant dyadic covariate
varCovar	Varying covariate
varDyadCovar	Varying dyadic covariate
sienaCompositionChange	List of changes, entry for each node.
siena	Data for a single project
sienaGroup	A list of siena objects, with global attributes, used for multi-group projects
sienaModel	Contains the fitting options.
sienaEffects	Data frame of effects.
sienaGroupEffects	Data frame of effects for a group object.

RSiena Other Classes

sienaFit	Returned by **`siena07`**. Currently contains (almost) everything from the estimation. Will be slimmed down later.
sienaTimeTest	Returned by **`sienaTimeTest`**.
iwlsm	Returned by **`iwlsm`**.
sienaMeta	Returned by **`siena08`**

Printing: Basic points

- Print to file or screen?
  - Only printed report from **`siena07`** is the ***`projname`*`.out`** file.
  - Other reporting available in Siena3 is here suppressed or sent to console.
  - R print methods print to the console.
  - Console output can be redirected using **`sink()`**.
  - **`print.sienaMeta`** and **`print.sienaEffects`** have option to print to a named file.
- **`Report`** function
  - Used as utility from within estimation and reporting routines.
  - Has options to **`open`** or **`close`** the output file.
  - Can suppress the output file completely.
  - Will suppress or send to console output not designed for the output file depending on arguments **`verbose`** and **`silent`**.

Initial report printing

Description of the project in a new output file is produced by the function **`print01Report`**.

- Called by **`siena01Gui`** (via **`sienaDataCreateFromSession`**) when the **`Apply`** button is clicked
- Called by **`sienaDataCreateFromSession`**
- Can be called directly with parameters: data object and corresponding effects object.
- Calls **`printInitialDescription`**, **`sienaGroupCreate`** (as it is convenient always to have a group object), **`Report`**
- **`printInitialDescription`** calls **`sienaGroupCreate`** (probably superflous), **`getNetworkStartingVals`**, along with the utilities **`Report`** and **`Heading`**

siena07 estimation reports

Reporting of the estimation is in several parts:

- **`InitReports`** called by **`siena07`**
- **`DataReport`** called in **`initializeFRAN`**
- **`PrintReport`** called in the final call to **`simstats0c`** or **`maxlikec`**.
- Uses the utilities **`Heading`** and **`Report`** and **`PrtOutMat`**, along with **`sienaGroupCreate`**
- **`print`** method for **`sienaFit`** object will display the fitted parameters and standard errors.
- **`summary`** method for **`sienaFit`** objects will display more details of the results.

Interface to C++

- In four source files in the **`src`** directory.
  1. **`siena07internals.cpp`**
  2. **`siena07models.cpp`**
  3. **`siena07setup.cpp`**
  4. **`siena07utilities.cpp`**
- Called from R functions **`simstats0c`**. and **`maxlikec`**.
- Details of how to do it are in *Lecture 9* of on-line course.
- I suspect that the course material does not cover all aspects: more details are in *Writing R Extensions*
- Some extra, now unused, routines persist.

Interface to C++: Output

- To output text on the console from C++, use **`Rprintf`**, as other functions will not produce visible output within R .
- To print an R object within C++, use **`PrintValue`**.
- **`Rprintf`** is used for debugging output.
- Utilities exist to print out ministeps by creating an R data frame and using **`PrintValue`**.
- To stop because of an error, use the **`error`** function.
- Standard C++ exceptions (Used within the C++) are converted to calls to the **`error`** function by means of the function **`Rterminate`**.

Effects

- Effects are defined in a data frame, produced by the function **`getEffects`**.
- The columns of the data frame are defined in the help page (**`?getEffects`** or **`?sienaEffects`**)
- Altering the columns **`include`**, **`fix`**, **`test`**, **`initialValue`**, **`parm`**, allows selection, fixing, testing, and setting of initial values for effects. (Utility functions are provided for this.)
- Other columns are used to communicate with C++.
- Effects are constructed by combining rows from the object **`allEffects`** with the parts of the siena or sienaGroup data object.
- Two columns, **`effectFn`** and **`statisticFn`** are designed to allow effects defined by the user in terms of R functions, but are not currently in use.

Other Effects Columns

name	Dependent variable which "owns" the effect
effectName	used for effect name on reports
functionName	used for function (statistic) name on reports
shortName	used to identify the type of effect for C++
interaction1/2	used to identify other object(s) on which effect depends, covariate or another dependent variable
type	one of **rate**, **eval**, **endow**
basicRate	boolean, **TRUE** if a basic rate effect
randomEffects	not currently used
functionType	distinguishes rate effects from others
period	for rate effects only
rateType	for rate effects only, **structural** or **covariate**
untrimmedValue	used for report if the initial value has been trimmed
netType	**oneMode**, **behavior** or **bipartite**
groupName	used in validation for multi-group projects
group	used to distinguish rate effects for different groups

Adding new effects

- Add appropriate lines to the **allEffects** object by editing **allEffects.csv**.
- Make **shortName** different from any other.
- If the effect fits in with the current structure, create the required C++ functions and add the **shortName** to the **EffectFactory**
- If the effect does not fit in the current structure, may need to alter the functions **effects** (in **siena07setup.cpp**), **updateParameters** and **getStatistics** (in **siena07internals.cpp**). And other parts of the C++.

Current structure

columns	uses (only) netType, name, shortName, parm, interaction1, interaction2, initialValue, type, group, period, rateType
non-basic rate	names are hard coded in **getStatistics** with call to corresponding score function
non rate	• **effectInfo** object is created using above columns. • Refer to Krists' documentation...

siena07 — Estimation

- More or less copies structure of Siena 3
- Simplifies it a little
- R does not have a **goto**
- **siena07** is the main routine, but calls **robmon**
- **robmon** does the Robbins Monro part:
  - 10 iterations of phase 1, then if using finite differences, check to see **epsilon** is OK
  - If not, restart
  - If OK, finish phase 1
  - Estimate derivative
  - If not OK, restart
  - If OK, do phase 2
  - If OK, do phase 3
  - Estimate derivatives and standard errors

More details in **RSIENAspec.pdf**

simstats0c/FRAN/maxlikec

- First call **`initializeFRAN`** reformats data and sends it to C++, stores addresses and data in **`FRANstore`** object.
- Simulation calls retrieve the data, do one simulation, and return statistics, scores (maybe), random number seeds, simulated networks(maybe), time if conditional.
- For maximum likelihood estimation the chains are stored on the model object in C++ between iterations.
- Final call does not call C (object destruction is done using R **`finalizer`** calls). Nullifies the addresses of the objects.
- With multiple processes, **`initializeFRAN`** is called one extra time in each process to set up the data.
- More details in **`simstats0c.pdf`**

Recreating the package

1. If using Windows, install Rtools.exe from `http://www.murdoch-sutherland.com/Rtools/`.
2. Obtain the current source.
3. In a command prompt window, navigate to the directory above the siena tree.
4. Check that Rtools is at the start of your path, by typing `set` and looking at the entry for PATH or path. (Windows only!)
5. Type
   `R CMD build siena`
6. You will probably need to either add R to your path, or use 'path-to-R.exe' in the command above.
7. When this one is OK, type
   `R CMD INSTALL RSiena\_1.0.n.tar.gz`
   (where `n` is adjusted to match the file you have just created.)
8. You now should have a new version of the package ready for use.

Alternative for repeated changes

- In a command prompt window, navigate to the directory above the siena tree.
- Type `R CMD INSTALL siena`
- This will compile all the C++, then source all the R , then create the help files.
- If you change one file and do it again, only the relevant parts will be redone, plus the help.
- Beware, it does not always work for changes to the C++. Sometimes you need to remove the RSiena.dll or all the .o files. You can do both by running `R CMD build siena`, but then you have to start again.

Parallel running against siena3324

1. In Siena3 source file rangen.pas, edit the lines at the bottom so that version = 5 and version2 = 6.
2. In s_base.pas, add, after
   ```
   if (version=5) then
       begin
       taumin := -log(rando(version))/lamsumtot;
   ```
   about line 5350
   ```
   if (nxa + nza = 1) then
        hstar := 1
   else
   ```
3. Build the project
4. Set the random seed to something non-zero in the .MO file.
5. Select the option to use standard starting values in the .MO file.
6. Use the argument parallelTesting=TRUE in the call to siena07.
7. I think that one or two other changes are needed to the source...

Profiling

If you are interested in knowing where RSiena spends its time, you can examine the R part by:

```
Rprof()
siena07(...)
Rprof(NULL)
summaryRprof()
```

For the C part it is a lot harder.

Profiling in C

- I seriously recommend use of a Mac, with the software **`Instruments`** (Time profiler) or **`shark`**.
- If this is not possible, try using **`SienaProfile.exe`**
- Works for networks, behavior variables, covariates. 1 group only.
- Run siena07 with the extra argument profileData=TRUE.
- This creates a file called data.txt. Move this to the directory src.
- In R, run the code, assuming the effects object is called myeff and you have nothing valuable called tmp:

```
tmp <- myeff[myeff$include,c(1,4,7,13,14,5,6,17,
    21,23,16)]
tmp[tmp==""] <- NA
write.table(tmp, 'effects.txt', quote=FALSE,
    row.names=FALSE,
col.names=FALSE, na="NA")
```

- This creates a file called effects.txt. Move this to the directory src.

SienaProfile.exe, continued

- In the src directory,

```
make -f makefile.profile clean
make -f makefile.profile
SienaProfile.exe
gprof SienaProfile.exe gmon.out > gprof.out
more gprof.out
```

- You will need to adjust the RHOME line in makefile.profile to reflect the location of R on your system.
- Google for “gprof” to find out how to understand the output!

There is an alternative on linux (oprofile), but it did not work last time I tried it (any Statistics department installation...)

Some standards

- Please, please, please, use spaces e.g. "x <- 1" not "x<-1"
- Please do not make **any** line in **any** source file longer than 80 characters. This includes **`.Rd`** files and LaTeXfiles.
- I have reused many variable names from Siena3: apart from these please make names intelligible to the casual reader.
- Restrict usage of **`tcl/tk`** to those parts which are shipped with R for Windows.
- Somes notes about coding standards for R can be found in *R Internals manual*).
- Occasionally I check that the C++ code will compile on obscure and fussy compilers.
- For building official tar balls C++ files should have LF-only line endings. We achieve this by using the property **`eol-style`** in the svn.

We have some more detailed rules: see the documentation.

A few further oddments

- The document **`RSienaDeveloper.pdf`** contains more detail about building and testing etc. and is kept up to date.
- I have a sort-of-functioning automatic parallel testing system, which runs through all the effects one by one, and pulls off the relevant lines to a text file for comparison using diff. Not quite ready to describe how to use it, but please ask if you want to try it out.

THE END