R 语言摘记（R-lang, Part I）

2009/07/06 at 12:23 PM ·Filed under R/S-plus ·Tagged data structure

R 语言某些地方可能和 C 类似，但是用过一段时间你就会发现其实它和 Lisp 这类 functional language 才是一家人，它们都允许对“语言”本身进行操作。因此为了更好的理解 R，我们其实要理解 R 的解释器怎么看待我们的输入的，即看看词法和语法分析。在 R 里面有 typeof() 函数（看看为什么叫对“语言本身操作”），可以返回输入里面一些元素的类型，这和 mode() 返回的是不一样的

- NULL 就是 NULL，没有 type 也没有可更改的属性，可以用 is.null() 检测。
- symbol 变量名，任意 R 对象的名字通常是一个 symbol，可以通过 as.name() 或者 quote() 创建。
- pairlist 主要是内部使用的 pairlist 对象，如 .Options 里面的，处理起来和 list 类型类似。
- closure 函数，含有三个成员，正规参数 list，body 和一个 environment，参数列表里面是 symbol 或者 symbol=value 的形式，或者是 …。body 是 parsed 的 R 表达式，也可以是单独一个 symbol 或者常数。函数的 enviroment 就是函数被调用时创建的上下文环境。这三个部分我们可以用 formals()、body() 和 environment() 获得和赋值。
- environment 环境，可以认为 environment 含有两部分，一个是 frame，即 symbol-value 对，用于查询，一个是指向上一级 environment 的指针。我们可以用 emptyenv() 创建空的 environment，使用 baseenv() 获得最上面一级的 environment。通过 ls() 可以列出某个 environment 里面的 symbols，get()/assign() 获得该 symbol 的值或者对它进行赋值。通过 parent.env() 获得上一级 environment。
- promise，是 lazy evaluation 产生的对象，它含有三个部分，值、表达式和对应的 environment，当函数调用后，首先进行参数匹配，然后其值实际上尚未获得，直到需要获得的时候才在对应的 environment 里面将 expression evaluate，产生值供以后使用。可以用 substitute() 函数（这是因为它是 special 类型，因此这个不会直接 evaluate 这个 promise 的 expression 部分）将 expression 部分抽取出来，这样程序员其实既能访问 expression 本身也可以获得其值，比如 plot() 函数里面就利用了这一点，默认的 sub、xlab、ylab 都是这样设定的。另有 delayAssign() 函数可以创建 promise 对象。但是实际上没有 R 代码能检测一个对象是不是 promise。
- language，R 语言的结构。一般分为 call、expression 和 name。不过这里的 expression 应该和下面的区分。
- special，不对参数取值的内部函数。
- builtin，对参数取值的内部函数。和 special 类似，可以认为是一种函数，但是是没有那三个部分的，可以通过 typeof() 将三者区分开来，比如 is.null() 就是 builtin，lm() 就是 closure，special 的一个例子就是前面的 substitute()。

- char，数量形式的字符串对象，仅在内部使用。
- logical，保存 bool 值的 vector。
- integer，保存整型的 vector。
- double，双精度浮点 vector。
- complex，复数类型的 vectoer。
- character，字符类型的 vector。
- … 可变长度参数，可以通过 list() 将其转换成为 list，然后就可以用 R 代码访问了。
- any 对任意类型都能匹配上的特殊类型，很少使用，如 as.vector(x, “any”) 表达的意思是说不需要做类型转换。
- expression，表达式，特指语法正确的一个或者多个 R 表达式，与 language 不同之处主要是 expression 可以看作是 language 中 expression 的 list，语法正确，且已经被 parse 了，只是没有 evaluate，通常可以用 eval() 函数执行，而 language 对象也许已经被 evaluate 过了。
- list 就是 list。
- bytecode，内部使用的 bycode 类型。
- externalptr，指向外部对象的指针。
- weakref，weak reference 对象。
- raw，含有数据的 vector，是什么不关心。
- S4 是一个 S4 对象。

这就是 R 解释器看到的基本元素。我们可以比较一下 mode()、typeof() 和 storage.mode() 判别类型的不同之处。R 中除了 NULL 以外，所有的对象都有属性，前面我们也知道可以用 attributes() 和 attr() 访问或者添加属性。前面我们也知道 matrix 或者 array 只是 vector 含有 dim 属性，另外可选的属性还有 dimnames。属性的作用就是形成 R 里面的类结构。下面我们举出一些常见的属性：

- names，这个常见于 list 对象，表示每一个元素都是一个单独的 vector 或者 list，可以用于索引，比如 ["key"] 和 $key 方式。
- dim，用于实现 array 和 matrix，储存方式和 Matlab 一样，也是 column-major。
- dimnames 用于在输出的时候更加漂亮，这是一个字符串的 list。
- class，是一个字符串，用于实现 OOP 里面的类方法，但是与 C++ 等语言不同，R 不做类型检查，，需要转换的时候一般 as.* 就好了。
- tsp 属性是时间序列的一些相关信息，如开始、终结、频率等。

一个对象被复制的时候其属性是否应该被复制是一个复杂的问题。常见的规则有标量函数应该保持属性，两元操作应保证较长的操作数的属性，取子集应该修改 names、dim 或者 dimnames 等属性，但是保留其他的，对子集赋值应该保持属性不变，转换应该丢弃属性。我们前面接触到的 factor 是含有 levels 和 class 为 factor/ordered 的 vector，另外还可能有 contrast 属性。另外 data.frame 我们也知道

是 list 的一种，除了有 names 以外，还可以有 row.names。

下面我们来看看 R 是如何 evaluate 表达式的。常数的值是已知的，对变量赋（常数）值时，对应的 symbol 和 value 被添加到对应的 environment 里面，引用时通过查询获得其值。函数调用是通过 search() 给出的环境搜索获得。有两种函数需要区分，我们通过 class() 获得的类名以及通过 class() <- 修改类名对应的是两个不同的函数，后者其实调用的是 class<-() 这个函数。运算符对应着函数，一般是写成 `op`()，如 1+1 对应 `+`(1, 1)。R 里面的运算符除了 +-*/ 还有 ^（指数），%% 取模，%/% 整数除法，%*% 矩阵乘法，%o% 外积，%x% Knronecker 成绩，%in% 匹配，逻辑运算里面常用的 !<><=>= & |（& 和 && 不同在于后者不是 vectorized）以外，赋值有 <- 和 ->，list 的操作 $ 也是一个运算符。索引使用的 `[` 和 `[[` 对应的 <- 版本也是函数。

前面讲 flow control 的时候没有提到 switch，它更像一个函数，switch(statemane, list)，如果 statement 的值是整数，如果是 list 长度以内的则返回 list 里面对应项的值，否则返回 NULL。又或者 statement 是字符串，则会对 list 里面的 symbol-value pair 进行匹配。

函数调用会产生所谓的 call stack，这个结构也就产生了 environment 的树状结构。我们可以用 sys.* 函数访问这个 call stack，如 sys.call() 返回当前（或者通过 which 参数表示更上几个层次的）函数，sys.frame() 返回当前 environment 的 frame，sys.function() 返回的是当前函数，sys.parent(0 返回的是上级 environment，对应还有复数版本，比如 sys.functions() 就是获得调用栈里面所有函数。传递参数使用的 symbol=value 形式也可以用 missing() 函数判断（这样就不用写成那种形式了）。（给非 primitive 函数）传递参数时，如果是给定参数，则在调用函数的 evaluation frame 就已经 evaluate 了，而如果是使用的默认参数，则是进入到被调用函数里面才 evaluate。这也就是所谓的传值（相对于传引用而言），在函数内部改变参数不会影响主调函数里面的变量。因此如果需要访问未经 evaluate 的表达式，就需要用 substitue() 函数获得对应的 promise。

R 语言摘记（R-lang, Part II）

2009/07/06 at 2:36 PM ·Filed under R/S-plus ·Tagged debug, functional programming, object-oriented programming

R 的 OOP 实现是比较怪异的，它依赖于 .Generic、.Class、.Method 和 .Group 这四个对象，对象使用的方法（generic function）依赖于 UseMethod() 和 NextMethod() 这两个 dispatching 函数。一般我们需要一个接口，如 mean() 函数，对于 vector 和 data.frame 应该是两个不同的函数实现，因此我们需要定义一个 generic function，然后为不同的类实现各自的版本。

实现 generic function 的时候，我们就需要使用前面所说的对象和方法了。如 mean 就应该定义为 function(x, …) UseMethod(“mean”)，那么在调用 mean() 的时候，.Class 会被设置为 class(x)，这往往是一个类的继承列表，最前面的是 class hierarchy 最下面的，.Generic 被设置为调用的方法名 mean，然后我们根据这个去搜索对应的函数，搜索的次序是寻找 mean.class 这样的函数，class 从最具体的类开始，如果没找到，就会寻找 mean.default()，我们可以通过 getAnywhere() 看看系统自己的 mean()，不难发现 mean() 本身是一个 generic function，为 vector 使用的是 package:base 里面的 mean.default()，而为 data.frame 调用的则是 package:base 中的 mean.data.frame()。
需要注意的是，UseMethod() 之后函数并不返回到 generic function，因此在 generic function 调用 UseMethod() 之后的语句没有任何效果。前面另外两个对象，.Method 一般是被调用的方法名，另外如果是通过内部调用的可能有 .Group 对象。

为了实现所谓的继承类函数调用，如子类的对象被 generic function 调用后，可能需要同类的某些其他方法，这时使用 NextMethod(generic, obj) 调用会根据当前 .Generic 或指定的 generic 和 .Class 寻找对应的函数调用。方法存在一定的分组，现在没有语句可以操作分组，但是常用的三个分组有 Math、Summary 和 Ops。在对应的方法里面会自动设定 .Group 对象。

下面我们将讨论 R 语言里面 functional programming 的一部分，重要的函数就是 quote()，这是一个 special 对象，它的作用就是产生没有 evaluated 的 language 对象，比如 quote(1+2) 返回的是一个 language 对象，其 mode 是 call，这种对象可以用类似 list 的方式操纵，如果里面有 =，如 exp<-quote(plot(x = age))，则可以用 exp$x 获得 age，又或者通过 exp[[ ]] 这种方式。我们可以用 as.name() 把一些字符串转换成为对应的 symbol 替换某些 expression 里面的操作。

可见使用 quote() 之后，expression 已经被 parse 过了，可以用 deparse() 将其还原为字符串。这个最常见的应用就是 plot() 里面的 xlab 这类，我们用 substitute 获得 promise 里面的 expression，然后 deparse() 成为字符串就可以画到图上了。

我们可以用 eval() 执行一段 expression，但是需要指定 environment，而使用 eval.parent() 可以在当前 frame 里面执行。与 quote() 不同的是 expression()，这产生的是 expression 对象，因此只能对应一个表达式，且 eval() 之后还是 exzpression。

在一个函数里面通过 sys.call() 可以获得对应的 environment，这常在调试中应用。更常用的是 match.call()，这样获得的是匹配好的参数列表，那么当我们需要从这个函数调用一个参数一样的函数就比较方便。我们通过操纵 language 对象改变其调用函数即可。

R 与 OS 交互的函数通过 Sys 开头的函数，如 Sys.get/putenv()，Sys.get/putlocale()，Sys.time() 等。对文件操作可以用 file.copy()、file.rename() 等。使用 .C() 或

者 .Fortran() 我们可以使用编译好的 .so 文件，而 .Call() 和 .External() 允许我们用编译好的 C 代码操纵 R 里面的对象。

R 的异常处理比较简单，主要有 stop()、warning() 和 on.exit()机制，这跟原来 Matlab 的 error、warning 很像，只是 Matlab 后面提供了 try catch 机制。R 的调试主要是通过 browser() 和 debug()/undebug()，trace()/untrace()。

R 语言摘记（R-intro, Part III）

2009/07/05 at 10:59 PM ·Filed under R/S-plus ·Tagged linear regression

R 里面有一种特殊的表达模型结构的方式，即 response ~ input op input op …，这里 op 可以用诸如 + – * / ^ 等符号，但是表达的意思完全不同，如 x1 + x2 表示使用两个输入特征 x1 和 x2 进行回归，而 – 则表示不使用某个 input，常数项可以用 1 表示，一般 + 表示的是 additive model，而使用 * 和 / 表达 non-additive model，另外可用 poly() 表达多项式 feature 以及 Error() 表达误差项。使用 : 表示 tensor product。下面我们来看看使用这种表达式进行回归的例子。我们另外知道调用 R 进行数据分析的方式一般是，通过 lm()/glm() 等函数决定模型的种属，如是线性模型还是广义线性模型还是别的什么，然后通过 data 决定分析的数据，模型通过前面的表达式决定，可以用专门的函数来处理获得的模型，如 anova() 进行方差分析，plot() 进行绘制等，另外可以用 update() 函数获得在原模型上做修改的模型结果。如果是一般的优化目标，可以用 nlm() 进行拟合。其他的模型，比如混合模型，可以用 nlme 包中的 lme() 和 nlme() 函数；局部回归模型可以用 stats 包里面的 loess() 函数；鲁棒回归可以用 MASS 包里面的 lqs()；additive 模型可以用 acepack 或者 mda、gam、mgcv 里面的函数。另外一些决策树算法也有一些对应的包，如 rpart 和 tree。下面我们结合 Hastie 的那本 The Elements of Statistical Learning 书中的例子来分析数据。
第一个例子是 prostate，见该书 3.2.1 节

```
# read the data and preprocess
prostate <- read.table( "../../datasets/prostate.data", head=TRUE ) ;
attach( prostate ) ;
prostate$lcavol  <- scale( lcavol ) ;
prostate$lweight <- scale( lweight ) ;
prostate$age     <- scale( age ) ;
prostate$lbph    <- scale( lbph ) ;
prostate$svi     <- scale( svi ) ;
prostate$lcp     <- scale( lcp ) ;
prostate$pgg45   <- scale( pgg45 ) ;
prostate$gleason <- scale( gleason ) ;

# corrlation
corre <- cor( prostate[1:7] ) ;

# linear regression for lpsa
lm.pro <- lm( lpsa ~ lcavol + lweight + age + lbph
```

```
            + svi + lcp + gleason + pgg45 + 1,
            data=prostate, subset=train ) ;
summary( lm.pro )

# remove the insignificant predictors
lm.prol <- lm( lpsa ~ lcavol + lweight + lbph + svi + 1,
            data=prostate, subset=train ) ;
summary( lm.prol )

# compare the two models
anova( lm.pro, lm.prol )

# predict
Y <- subset( prostate, subset=!train )
Y.predict <- predict( lm.pro, newdata=Y )
var( Y.predict - Y$lpsa )

# the end
detach( prostate ) ;
```

可见使用 R 对数据拟合的一般形式。下面我们使用 MASS 的 ridge regression 和 glmnet 的 lasso 进行回归，这也很简单

```
lm.ridge.pro <- lm.ridge( lpsa ~ lcavol + lweight + age + lbph
            + svi + lcp + gleason + pgg45 + 1,
            data=prostate, subset=train, lambda = 0.1 ) ;
```

但是 glmnet 并不是使用 formula 确定的关系，因此写起来和其他语言的风格类似。这里有几个注意点：

- 使用了 attach() 以后，如果对 data.frame 的数据要更该，前缀还是不能缺省的，这里如果去掉了，后面可能会出问题。
- lm() 返回的是一个对象，它包含了很多信息，我们往往使用某些函数提取这些信息，另外使用 predict() 的时候，newdata 必须和原来的 data.frame 结构一样。
- 有意思的是，用户自己写库的时候如何处理 formula 呢？感觉上直接声明输入而把形式包含在程序内部更加方便一些。

R 语言摘记（R-admin）

2009/07/05 at 5:35 PM ·Filed under R/S-plus ·Tagged installation, management

系 统默认会加载一些 package，这可以用环境变量 R_DEFAULT_PACKAGES 定义，也可以在 .Rprofile 里面设置，我们可以用 getOptions(“defaultPackages”) 获得。这个 getOptions() 函数还可以设置编辑器，浏览器等 R 里面使用的相关工具。

我们一般使用 library() 或者 require() 载入一个 package，函数 .First.lib() 将在载入 package 的时候被调用，.Last.lib() 将在 detach() 该 package 时被调用。能通过该命

令载入的 package 需要在所谓的 library path 里面，我们可以用 .Library 返回默认搜索路径，一般是 /usr/lib/R/library，我们可以在用户 .Rprofile 里面通过 .libPaths() 添加 R 的 library 路径，另外也可以使用环境变量 R_LIBS_USER 设定。

有意思的是如何为 R 添加一些 package 呢？这个其实和 deb 系统类似，一方面我们可以用命令行 R CMD INSTALL -l /path/to/library pkgs 来进行安装（只能指定下载下来的 package），还可以用 R 的函数 install.packages() 安装（可以从网络安装）指定的 package，我们通常也需要解决依赖关系，另外，我们可以用 setRepositories() 指定安装 package 的镜像站点。安装后的 package 还可以用 update.package() 进行升级。

例如，The Elements of Statistical Learning 一书里面有这些 packages， 除了 MART 和 PRIM 以外都可以在 r-cran 里面获得，但是 debian 的 gnu-r 里面并没有这些 package，我们在用户目录里面进行安装。首先建立 ~/Rlibrary 目录，然后我们使用命令安装 glmnet 这个 package，我们看见它依赖与 matrix，这是 debian 所带有的，我们使用 $ R CMD INSTALL -l ~/Rlibrary/ glmnet_1.1-3.tar.gz 这样会在 ~/RLibrary 里面创建 glmnet 目录，对包里面的代码进行编译，最后连同文档一起放在这个目录中。又比如说 mda，我们可以直接在 R 环境下，install.packages(c(”mda”, “lasso2″, “lars”, “gbm”, “princurve”, “bootstrap”), “~/Rlibrary”, dependencies=TRUE) 最后我们编辑 ~/.Rprofile 保证这个 library 能被 R 自动的搜索，添加一行 .libPaths(new=”~/Rlibrary”) 即可。

R 语言摘记（R-intro）

2009/07/03 at 7:24 PM ·Filed under R/S-plus ·Tagged data structure

寻 求帮助，使用 help(solve)，?solve 和 help(”solve”)是一样的，如果需要搜索可以用 help.search(solve) 或者 ??solve。另外使用 help.start() 可以打开网页版的帮助，这个功能倒是和 Matlab 的 doc 有几分相似。

使用 source() 和 sink() 命令可以将 R 的输入输出重新定向，比如从一个文件中读入命令就是 source(“my-file.R”)，而将输出导入到文件则是 sink(“output.result”)。对于一个 workspace 里面的数据，可以用 objects() 或者 ls() 列出，rm(x, y, z) 删掉某些变量，值得注意的是这里直接写变量名，都没有引号，这是和 R 语言的规范有关系的（与 lisp 挺像的）。结束一个 session 的时候可以将 workspace 的内容存储在工作目录下，作为 .RData 文件，下次在该目录使用 R 打开新的 R 对话时会自动的读入。存储 workspace 可以用 save.image()，或者更加细致的 save() 命令。

R 的赋值使用 ->、<- 或者 assign(variable, content) 形式。主要的数据类形式 vector、objects、factor、array、matrix、list 和 data frame。vector 这个和一般的 Matlab 数组

类似，但是和 Matlab 的运算不同的是，它不需要参与运算的两者长度一样，不同长度的元素会按照最长的那一个重复。

常用生成 vector 的函数有 seq() 类似 matlab 的 linspace 或者 1:2:10 这种类型的等差数列，rep() 是类似 repmat 的作用，但是可以将整体重复或者将每个元素重复。和 Matlab 类似，R 也拥有 logical vector（支持 !、& 等逻辑操作），vector 里面的 NA 表示 missing value。判断 NA 使用 is.na()，这个与 Matlab 的 is 系列函数类似。注意 is.na(x) 与 x == NA 的结果是不同的。另外有一些计算会产生 NaN，这和 NA 有一定的不同，但是 is.na() 对两者不加区分，而 is.nan() 可以区分两者。

R 的字符串 vector 和 matlab 类似，都是用现成的结构（R 使用 vector）存放的，R 支持单引号或者双引号，也支持 C 语言中的转意字符，如 \n、\t 等。如果需要取一个 vector 的一部分出来，与 Matlab 类似可以用 logical vector，或者一个整数 vector，但是和 matlab 不同的是，索引使用 []，而不是 ()，另外允许对返回值使用索引，比如 (1:10)[seq(1, 10, by=2)] 是允许的。另外如果 index 出现负数，表示除掉这些绝对值后的 index 剩下的 index。与 Matlab 类似，索引的 vector 可以作为左值。不同的是 R 的 vector 有 names 属性，这意味着每个元素还可以通过字符串来进行索引（需要用 names(myvector)<-c(…) 赋值）。

R 里面的原子结构（atomic structure）是 logical、numeric、complex、character 和 raw，其他的 objects 都是这些 atom 的组合，我们也把原子的种类称为 mode，比如 vector 是相同的 atom 组成的；而 list 就是为了形成各种结构的嵌套结构，所以每个元素可以是不同的 mode。我们可以用 mode(obj) 获得一个对象的 mode，另外一个属性是 length()。一般来说，我们可以通过 attributes() 获得一个对象的属性。结构的互相转换，常使用 as.*() 函数，如 as.character()、as.integer()。

值得注意的是，访问某个对象的属性和修改该属性都使用同一个函数，只是后者加上赋值，如 length(x) <- 3 可以将 x 的长度变成 3，即自动的抛弃其他的元素，一般的使用 attr(objective, “property”) 获得某个对象的属性。每个对象有自己的 class，这可以用 class() 获得，使用 unclass() 可以暂时消除起 class 信息。

所谓的 factor 就是所谓的 categorical variable，这分为 ordered 以及 unordered，如创建一个 string vector，通过 factor 构造出一个 unordered factor，我们可以用 levels 获得其标签，如果我们用 class 查看会发现这是 factor，但是用 mode 看却是 numeric。一个 factor 和一个 vector 的组合常被称为 ragged array，因为每个子类的长度不一定相同。我们常用 taaply(vector, factor, fun) 来对每个子类的数据进行计算。ordered factor 使用 ordered() 创建，两者大致看来只是后者会依照 factor 排序，但是在 regression 里面两者是区别对待的。

matrix 和 array 都是含有 dim 属性的 vector，因此建立一个 vector，然后对其 dim 属性赋值就变成 matrix 或者 array，matrix 是两维的，更高维的就是 array，如 x<-runif(24)，dim(x) <- c(3, 😎这产生的是 3×8 的 matrix，我们可以通过 class 看见，其

mode 仍然是 numeric。对 array 的索引和 Matlab 一样，可以用 [,] 的形式，只是用空参数表示全部的（Matlab 是 :），另外 R 支持用 array 来作索引，如 3×4 的 matrix 可以用 nx2 的 matrix 取其中 n 个元素，每行是一个索引位置。可见 R 里面的 matrix 或者 array 的数据储存并没有改变，只是通过 dim 属性给它添加了额外的索引方式。可以用 array() 和 matrix() 更方便的创建我们需要的矩阵、数组。如我们需要产生一个 N 个样本 c 类的 assignment matrix，A <- matrix(0, N, c); idx <- cbind(1:N, labels); A[idx] <- 1 ; 其中 cbind() 将两个 matrix 依照列拼接，类似的还有 rbind（这分别相当于 Matlab 里面的 [;] 与 [,]）。当 vector 和 marix/array 混合在一起进行计算的时候，短的 vector 补长，必须拥有相同的 dim。将 matrix/array 转换成为 vector 只需要 as.vector() 或者直接用 c() 即可。

常用的矩阵操作有元素对元素的加法、减法和乘法，直接使用 + – * 即可，另外有矩阵乘法用 %*%（vector 作矩阵乘法的时候会产生较小的矩阵，比如两个 vector 相乘，产生的是标量而不是一个矩阵），外积（其实是 Kronecker 乘积）%o% 或者 outer()。不过 outer() 是更广义的函数，除了可以做 %o% 运算，还可以将任意函数作用在类似的结构上。矩阵的转置使用 aperm，这个操作实际上是把 array 的维数互换，因此 aperm(X, c(2 1)) 等价于转置了，更简洁的是使用 t(X)。crossprod(X, y) 等价于 X’y，而 diag() 和 Matlab 中函数类似。求解线性系统可以用 solve(A, b)，如果写 solve(A) 返回的是 A 的逆。使用 eigen() 可以求出特征值，这返回的的一个是 $value，一个是 $vectors，这种结构将在后面介绍。奇异值使用 svd()，它返回 $d $u $v 三个部分。lsfit() 可以计算最小二乘拟合。qr() 计算对应的 QR 分解。

为了统计一些频率，比如 contingency table，都会使用 table() 函数，比如对 factor 的统计，如果是实值的可以用 cut() 函数将其值离散化为 ordered factor，如果有多个 因素，则对应的 contingency table 也是用 table(factor1, factor2) 获得的。

前面我们看见了 $ 这种写法，这就是 list，这一般使用 list() 创建，list 的元素可以有用来索引的 key，这可以用 listvar$key 获得其内容，也可以使用 listvar[["key"]] 获得，通过 [[i]] 这样可以遍历所有 key，注意 [i] 将和 Matlab 对 cell array 使用 (i) 类似，获得的不是第 i 个位置的内容，而是对应位置上的 list。对 list 使用 attributes() 将会返回 names，这是这个 list 所有的 key 组成的 key，如果没有 key，对应的是一个空字符串。这也是 R 默认返回多个返回值的方法。

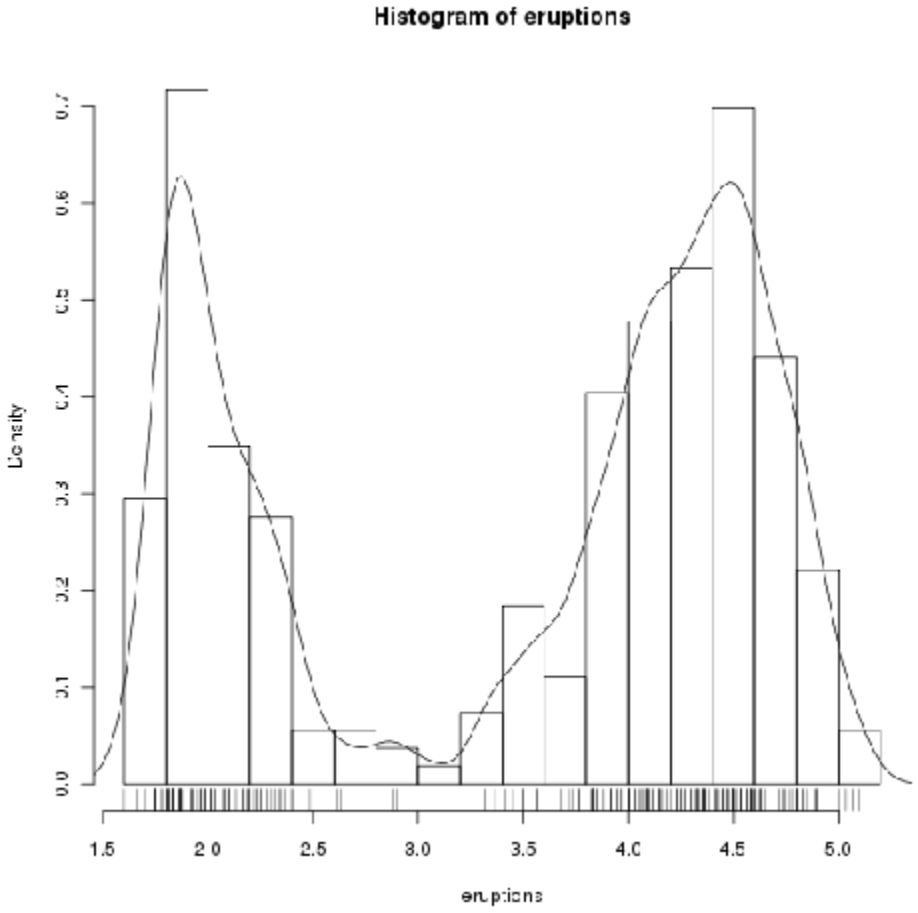
所谓的 data.frame 也是一种 list，但是要求每个子 list 里面都是长度一样的 vector，其实 data.frame 是处理数据最佳的方式，相当于每一列都是一个特征的数据矩阵。通常我们可以用 data.frame() 函数产生，这其实和 list() 函数完全一样。为了处理某个 data.frame 方便，可以用 attach()/detach() 进入到某个 data.frame 的 scope 里，这时就可以免去前面 data$ 的修饰符了，对一般的 list 也可以使用类似的方式。我们要对 R 寻找变量的方式熟悉的话就知道，使用 search() 返回的是搜索变量和函数的 scope，一般 R 的基本函数在 package:base 里面，全局变量在 .GlobalEnv 里面，attach() 的 list 也会添加到这个 vector 里面。最常见读入外部数据的函数是 read.table()，这也是创建 data.frame 的最方便的方式，另外还有一个 scan()。R 自己

也带了很多数据供做实验，可以用 data(dataset) 读入，还可以用某个特定的 package 读入数据。对这种数据一般都可以在一个 spreadsheet 里面编辑，这只需要 edit() 那个 data.frame 就可以了。

R 语言摘记（R-intro, Part II）

2009/07/03 at 10:20 PM ·Filed under R/S-plus ·Tagged flow control

R 里面有很多分布的分布函数、密度函数、产生随机数的函数以及分位点函数，一般都是以 d、p、r、q 开头的，比如 runif() 就是产生均匀分布随机数的函数。获得一个一元样本的信息的最简单的方式是通过 summary() 函数，这个和 finvenum() 产生的最小最大以及其他三个四分位点。可以用 hist() 进行可视化，density() 进行核密度估计，之后可以通过 lines() 和 rug() 将对应的画在图上。下面是一个例子



另外，ecdf() 返回经验似然函数，qqnorm()，qqplot() 和 qqline() 是为了将样本与正态分布比较画的所谓 Q-Q plot。另外可以用 boxplot 比较两个样本之间的 5 点，或者用 t.test() 进行 t 检验，var.test() 进行方差检验，wilcox.test() 进行 Mann-Whitney 检验，ks.text() 进行 Kolmogorov-Smirnov 检验。

R 里面用 {} 将语句组成小组，常见的控制结构有 if(exp 1) exp 2 else exp 3，for(var in vector) exp，repeat exp，while(condition) exp，对循环可以用 next 和 break 进行跳转。

函数的写法是 function_name <- function(arg list) exp，其中最后一个 exp 作为整个 exp list 的返回值，也是这个函数的返回值。另外可以用 %something% 定义二元操作符，如 %o% 我们就可以用 outer() 来定义 %o% <- function(a, b) outer(a, b, "*")。注意有名字的参数可以通过 = 传递，其他的参数用 … 传递，函数里面通常赋值不会影响到函数外，如果希望对全局变量或者函数的静态变量赋值，需要用 <<- 或者 assign() 函数。

如果一个变量输出的时候不希望有列说明、行说明，需要用 dimnames() 将他的 dimnames 设置为空串。函数可以嵌套定义，如果在内层的函数需要访问外面的变量，可以用 <<- 赋值，同时这样可以实现多个非共享的数值。

R 启动的时候会寻找 R_PROFILE 指向的初始化文件，若没找到则执行 /etc/Rprofile，另外在执行目录下的 .Rprofile 将被执行，还有用户 ~/.Rprofile。在这些 profile 里面定义的 .First() 函数将被执行，R 退出的时候执行 .Last() 函数。我们通常可以设定 options()，读入库 library() 等。

R 里面的 class 的方法可以用 methods() 获得，同时用 getAnywhere() 可以获得关于某个类实现的某个方法的位置和内容，我们可以用 ls(name="xxx") 获得某个 package、list 里面的内容。后面我们会学习如何使用 R 的 OOP。和 C++ 类似，R 也有 namespace 的概念，就是前面 search() 获得的列表，我们可以用 :: 表示某个 package 里面的函数，::: 是用来访问某些隐藏对象的。

这里介绍一下常用的 R 作图概念，首先要知道设备，比如默认的在屏幕上绘制是 X11()（Windows 和 Mac OS 是别的），然后如果要输出到文件，需要用 png()、postscript() 或者 pdf() 等，调用一次就会打开一个设备，因此调用完后记得 dev.off()，如果需要多个设备，可以用 dev.set() 或者 dev.next()/dev.prev() 切换，如果已经画出来了，可以用 dev.print()/dev.copy() 复制到新的设备上，如图片文件。画图命令有比较高层次的，也有比较低级的。高级的主要是 plot() 可以绘制两个变量的关系。如果是多个变量之间的关系，可以使用 coplot() 或者 pairs()，另外前面提到的 qq 系列，hist()，dotchart()、image()、contour、persp()，这些函数常用的参数有，add 表示是否覆盖，log 表示是否使用对数坐标，type 是画图种类，x/ylab 是坐标轴标注，main 标题，sub 子标题。

比较 low-level 的作图命令有 points() 画点，lines() 画线，text() 标注文本，abline() 用于为当前图片添加一根直线（比如表示拟合结果什么的，这样不用声明起始点坐标），polygon() 画多边形，legend() 添加图示，可以看看 plotmath 的帮助获得如何在图片上添加数学符号，Hershey 字体，类似于 Matlab 的 ginput，R 有 locator()。

par() 可以设置默认的画图参数，如果是临时更改，应该直接更改 plot() 这些函数的 options。常用的 option 有 pch（point character）、lty（line type）、lwd（line width）、col（color）、col.axis、col.axis、col.main、col.sub、font（1 正体，2 粗体，3 斜体，4 粗斜体，5 符号）、font.*、adj（adjustification，对齐）、cex（character expansion）、lab（坐标轴刻度）、las（纵轴是否旋转标签）、mgp（坐标轴位置）、tck（刻度长度）、x/yaxs（坐标轴样式）、 mai/r（边距）、mfcol/mfrow（多幅子图的位置）或者用 mfg 或 fig 设定。