

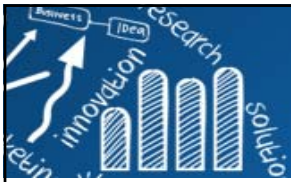
R/C interfaces: .C 和.Call

吳漢銘

國立臺北大學 統計學系



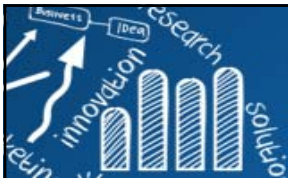
<http://www.hmwu.idv.tw>



- 從R呼叫系統指令
 - GUI 應用程式
 - 主控台應用程式命令
 - DOS內建指令

- 從R利用.C或 .Call 呼叫C副程式
 - 如何傳送參數從R到C
 - 如何從C得到回傳值
 - 如何編譯C程式碼
 - 如何從R呼叫C程式碼並執行

- R/.C/.Call的執行時間比較



從R呼叫系統指令: GUI 應用程式

3/44

語法:

```
system(command, intern = FALSE, ignore.stderr = FALSE,  
        wait = TRUE, input = NULL, show.output.on.console = TRUE,  
        minimized = FALSE, invisible = TRUE)
```

- 指令執行結束且輸出完成，結果才可被看到。
- GUI 應用程式要設定 `invisible = FALSE` 才會出現畫面。

範例:

```
system("notepad myfile1.txt", wait = TRUE, invisible = FALSE)  
system("notepad myfile2.txt", wait = FALSE, invisible = FALSE)  
system(paste('\"C:/Program Files/Internet Explorer/IEEXPLORE.EXE\"',  
            '-url cran.r-project.org'), wait = FALSE, invisible = FALSE)
```

從R呼叫系統指令: 主控台命令 (console commands)

範例:

```
system("net use")
system("net view")
```

```
R Console
> system("net use")
會記錄新的網路連線。

狀態      本機      遠端      網路
-----
OK        Z:        \\163.13.226.230\software Microsoft Windows Network
命令執行成功。

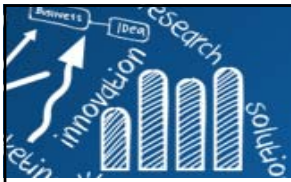
> system("net view")
伺服器名稱      說明
-----
\\B2D            Linux Samba
\\BULLETIN       bulletin server (Samba, Ubuntu)
\\COMSERV        comserv server (Samba, Ubuntu)
\\HANKPC         hankpc
\\MATH-S409
\\MATH-SA103-PC01
\\MATH-SA104-02
\\MATH-USER
\\MEDIACAST      mediacast server (Samba, Ubuntu)
\\MINYI
\\WCHAN
命令執行成功。

> |
```

範例: 從Rgui去編譯.C/.Call要呼叫的C程式碼

```
system("R CMD SHLIB MatrixProduct2.c")
```

NOTE: 即使設定 `intern = TRUE` 且/或 `show.output.on.console = TRUE`, 需要使用者輸入的主控台命令程式不會運作。



範例:

```
#NOT RUN
system("dir", wait = TRUE, invisible = FALSE)

system(paste(Sys.getenv("COMSPEC"), "/K", "dir > list.txt"))
system("notepad list.txt", wait = FALSE, invisible = FALSE)
```

CMD

/C

執行字串中所描述的命令然後結束命令視窗

/K

執行 字串中所描述的命令然後保留命令視窗

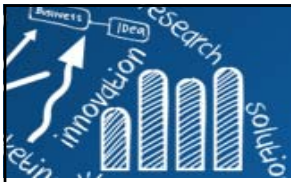
其它常用指令:

```
dir()
dir(pattern = "c$")
dir(pattern = "[.]c$")
```

為什麼要從R呼叫其它程語言(例如 C語言)?

- 優點
 - 有些程式以C編寫，執行速度會較R快。
 - 已經有C的程式碼可利用。
- 缺點
 - 編寫C程式所花的時間通常比編寫R的時間長，而且較不易除錯。
 - C程式碼較R程式碼通常不太容易了解。

```
R R Console
> eigen
function (x, symmetric, only.values = FALSE, EISPACK = FALSE)
{
  x <- as.matrix(x)
  if (!is.null(dimnames(x)))
    dimnames(x) <- list(NULL, NULL)
  n <- nrow(x)
  if (!n)
    stop("0 x 0 matrix")
  if (n != ncol(x))
    stop("non-square matrix in 'eigen'")
  complex.x <- is.complex(x)
  if (!complex.x && !is.double(x))
    storage.mode(x) <- "double"
  if (!all(is.finite(x)))
    stop("infinite or missing values in 'x'")
  if (missing(symmetric))
    symmetric <- isSymmetric.matrix(x)
  if (!EISPACK) {
    if (symmetric) {
      z <- if (!complex.x)
        .Call("La_rs", x, only.values, PACKAGE = "base")
      else .Call("La_rs_cmplx", x, only.values, PACKAGE = "base")
    }
  }
}
```



從R呼叫C的指令有三種

7/44

- **.C**
 - 針對不太了解R程式碼的使用者所設計。
 - 使用上很直觀。
 - 參數(argument)型別有限制。
 - 參數輸入的檢查是在R裡完成。
 - 沒有回傳值，但可以改變參數內容當成回傳值。
- **.Call**
 - 針對已了解R程式碼的使用者所設計。
 - 准許有多種的參數型別。
 - 可以有回傳值。
- **.External**
 - 針對已了解R程式碼的使用者所設計。
 - 以單一參數傳送，由所呼叫的程式語言解譯。
 - 可以有回傳值。

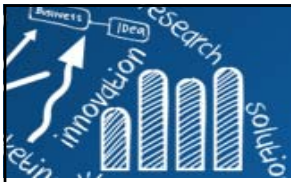
NOTE:

寫程式容易程度

.C > .Call > .External

執行效率

.C < .Call < .External

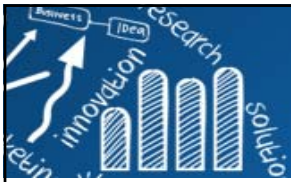


`.C(name, ..., NAOK = FALSE, DUP = TRUE, PACKAGE, ENCODING)`

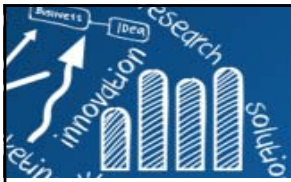
- **name**
 - C副程式名稱或屬於"NativeSymbolInfo", "RegisteredNativeSymbol" or "NativeSymbol"類別的物件名稱
- ...
 - 傳送給C副程式的參數。
- **NAOK**
 - TRUE: 任何參數中含有NA or NaN or Inf 值，皆會傳送到C副程式。
 - FALSE: 參數中含有NA or NaN or Inf 值，視做錯誤，R停止執行。
- **DUP**
 - TRUE: 參數會在傳送之前重覆複製一份。
- **PACKAGE**
 - 使用PACKAGE中的**name**程式。
 - 可避免其它相同**name**的副程式所覆寫。例如: 使用 **PACKAGE="base"** for symbols linked in to R.
- **ENCODING**
 - 指定字元資料的編碼。

.C/.Call 三步驟

1. 寫好C程式: `myCcode.c`。
2. 以R CMD 編譯C程式: `R CMD SHLIB myCcode.c`
3. 在R中呼叫C程式並執行:
 - `dyn.load("myCcode.dll")`
 - `.C("mySquare", ...)` 或
 - `.Call("mySquare", ...)`



- 執行.C和.Call的程序相同。
- 在Windows下的工具需安裝RTools。
- C程式碼以R CMD SHLIB編譯完成的程式以「動態連結程式庫」(Dynamic-link library, DLL) 檔案存在。
- 單一C程式檔: `R CMD SHLIB myCcode.c`
- 多個C程式檔:
 - `R CMD SHLIB myCcode.c myobj1.o myobj2.o` 或
 - `R CMD SHLIB -o myprogram *.c *.o`
- 在R中，以`dyn.load()`呼叫dll
 - `dyn.load("myCcode.dll")` 或
 - `dyn.load("myprogram.dll")`
- 在R中, 執行程式`.C("myfunction",...)`
- 在R中，以`dyn.unload()`卸載dll。



範例 1: (1) 寫好C程式

- 利用.C計算一數列(長度大於1)每個元素的平方。(如果此數列長度小於或等於1，請印出錯誤。)

myCcode.c

```
#include <R.h>

void mySquare(double *y, double *x, int *size){
    int i;
    int n = *size;
    if(n <= 1){
        Rprintf("Error n = %d <= 1\n", n);
    }
    for(i=0; i < n; i++){
        y[i] = x[i]*x[i];
    }
}
```

範例 1: (2) 以R編譯C程式

R CMD SHLIB myCcode.c

(1) 命令提示字元

```

C:\WINDOWS\system32\cmd.exe
Microsoft Windows XP [版本 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

D:\Web-R\01-主題\04-Call_C\code\ex1>dir
磁碟區 D 中的磁碟是 新增磁碟區
磁碟區序號: 1CC4-61FF

D:\Web-R\01-主題\04-Call_C\code\ex1 的目錄

2010/08/22 下午 12:51 <DIR>      .
2010/08/22 下午 12:51 <DIR>      ..
2010/08/22 下午 12:51             228 myCcode.c
2010/08/22 下午 12:49             212 myRcode.R
                2 個檔案             440 位元組
                2 個目錄      22,567,940,096 位元組可用

D:\Web-R\01-主題\04-Call_C\code\ex1>R CMD SHLIB myCcode.c
making myCcode.d from myCcode.c
gcc -std=gnu99 -IC:/PROGRA~1/R/R-28~1.1/include -O3 -Wall -c myCcode.c -o
myCcode.o
windres --preprocessor="gcc -E -xc -DRC_INVOKED" -I C:/PROGRA~1/R/R-28~1.1/inclu
de -i myCcode_res.rc -o myCcode_res.o
gcc -std=gnu99 -shared -s -o myCcode.dll myCcode.def myCcode.o myCcode_res.o
-LC:/PROGRA~1/R/R-28~1.1/bin -lR

D:\Web-R\01-主題\04-Call_C\code\ex1>

```

Makedeps
myCcode.c
myCcode.d
myCcode.dll
myCcode.o
myCcode_res.o
myCcode_res.rc
myRcode.R

或者(2) in R

```

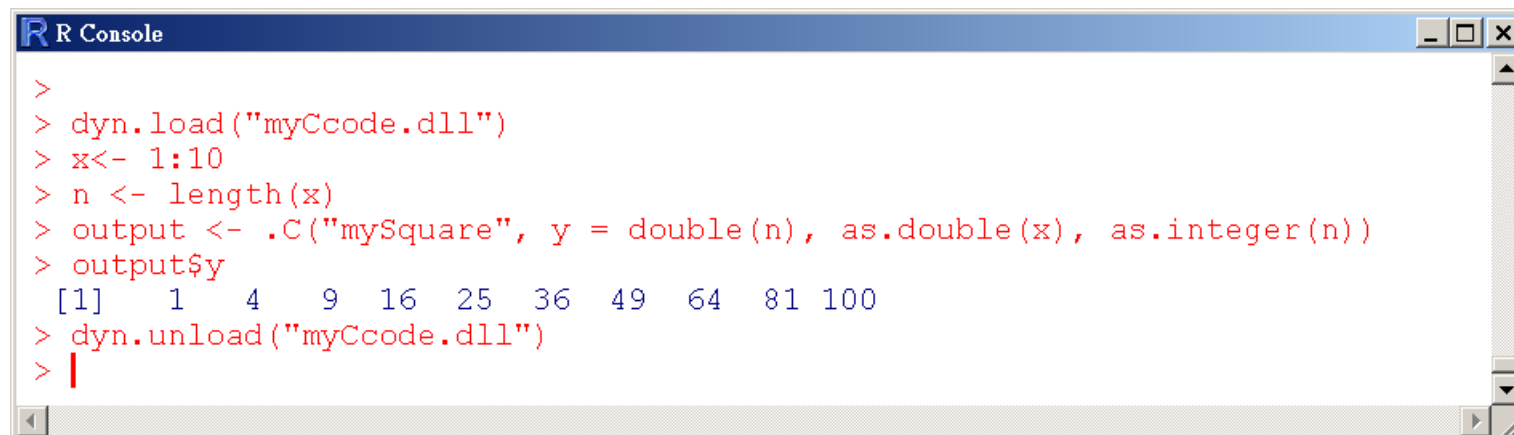
> getwd()
[1] "D:/Web-R/01-主題/04-Call_C/code/ex1"
> dir()
[1] "myCcode.c" "myRcode.R"
> system("R CMD SHLIB myCcode.c")
gcc -I"C:/PROGRA~1/R/R-211~1.1/include"
-O3 -Wall -std=gnu99 -c myCcode.c -o
myCcode.o
gcc -shared -s -static-libgcc -o myCcode.dll
tmp.def myCcode.o -LC:/PROGRA~1/R/R-
211~1.1/bin -lR
> dir()
[1] "myCcode.c" "myCcode.dll" "myCcode.o"
"myRcode.R"
>

```

範例 1: (3) 在R中，呼叫C程式執行

```
dyn.load("myCcode.dll")  
.C("funName", ...)
```

```
dyn.load("myCcode.dll")  
x <- 1:10  
n <- length(x)  
output <- .C("mySquare", y = double(n), as.double(x), as.integer(n))  
output$y  
dyn.unload("myCcode.dll")
```



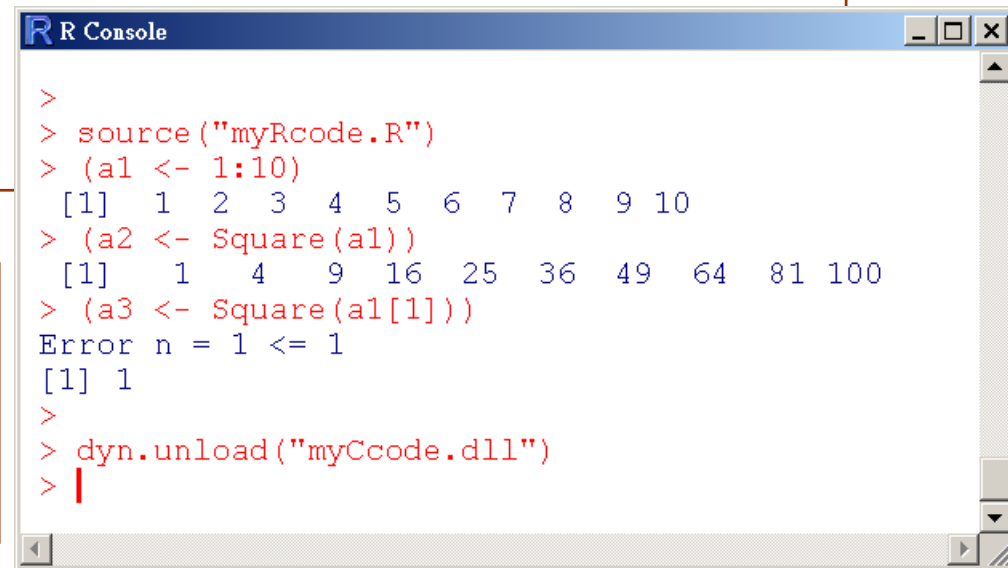
```
R Console  
>  
> dyn.load("myCcode.dll")  
> x<- 1:10  
> n <- length(x)  
> output <- .C("mySquare", y = double(n), as.double(x), as.integer(n))  
> output$y  
[1] 1 4 9 16 25 36 49 64 81 100  
> dyn.unload("myCcode.dll")  
> |
```

範例 1: 或者(4) 編寫R程式呼叫C程式並執行

myRcode.R

```
dyn.load("myCcode.dll")
Square <- function(x){
  n <- length(x)
  result <- .C("mySquare", y = double(n), as.double(x),
as.integer(n))
  result[["y"]]
  #or
  #result$y
}
```

```
source("myRcode.R")
(a1 <- 1:10)
(a2 <- Square(a1))
(a3 <- Square(a1[1]))
dyn.unload("myCcode.dll")
```



```
R Console
>
> source("myRcode.R")
> (a1 <- 1:10)
[1] 1 2 3 4 5 6 7 8 9 10
> (a2 <- Square(a1))
[1] 1 4 9 16 25 36 49 64 81 100
> (a3 <- Square(a1[1]))
Error n = 1 <= 1
[1] 1
>
> dyn.unload("myCcode.dll")
> |
```

寫C程式之注意事項

- 標頭檔必需有 `#include<R.h>` 。
- 沒有main function 。
- C副程式的回傳型別需為**void** (亦即沒有回傳值) 。
- 所有參數型別是以call by reference的方式傳送。
 - 故在C裡必需pass 一個指標(pointer) 給一個數字或向量 。
- 在C程式裡中使用**Rprintf**將結果輸出在R console 。
- C副程式計算結果可藉由參數回傳 。
- 如果參數型別為矩陣，則需轉成向量，而且維度需明確給定 。

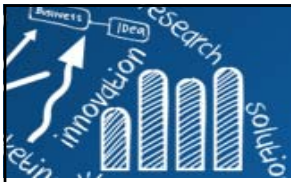
寫R程式之注意事項

- R函式的參數需以C副程式裡的順序傳送。
- 需作型別轉換，例如: `as.double`, `as.integer`。
- 需分配一塊空間給回傳結果。
- 矩陣參數型別是以向量的方式傳送。
- `result <- .C(...)`:
 - `result`為一個表列物件(a list of objects)。
 - 以`result[[number]]`或`result[["name"]]`或`result$name`存取

```
> .C("mySquare", y = double(n), as.double(x), as.integer(n))
$y
 [1]  1  4  9 16 25 36 49 64 81 100

[[2]]
 [1] 1 2 3 4 5 6 7 8 9 10

[[3]]
 [1] 10
```

R 和 .C/.Call 的參數型別對應

■ In C:

```
void mySquare(double *y, double *x, int *size){
```

■ In R:

```
.C("mySquare", y = double(n), as.double(x), as.integer(n))
```

R	.C/.Call
integer	int *
numeric	double * 或 float *
complex	Rcomplex *
logical	int *
character	char **
raw	unsigned char *
list	SEXP
other	SEXP

範例 2: 矩陣相乘

Ordinary Matrix Product

If $A \in R^{n \times m}$, and $B \in R^{m \times p}$, then $Z = AB \in R^{n \times p}$,
where

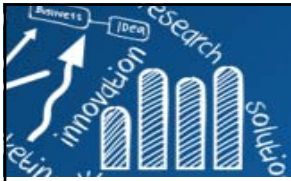
$$z_{ij} = \sum_{k=1}^m a_{ik} b_{kj}.$$

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1m} \\ a_{21} & a_{22} & \cdots & a_{2m} \\ \vdots & & & \\ a_{i1} & a_{i2} & \cdots & a_{im} \\ \vdots & & & \\ a_{n1} & a_{n2} & \cdots & a_{nm} \end{bmatrix}$$

$$B = \begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1j} & \cdots & b_{1p} \\ b_{21} & b_{22} & \cdots & b_{2j} & \cdots & b_{2p} \\ \vdots & & & & & \\ b_{m1} & b_{m2} & \cdots & b_{mj} & \cdots & b_{mp} \end{bmatrix}$$

$$\mathbf{a}_{im} = \mathbf{A}[\mathbf{i} * \mathbf{nc} + \mathbf{m}]$$

$$\mathbf{b}_{mj} = \mathbf{B}[\mathbf{m} * \mathbf{nc} + \mathbf{j}]$$



範例 2: (1) MatrixProduct1.c

19/44

```
#include <R.h>

void MatrixProduct1(double *A, int *Anrow, int *Ancol,
    double *B, int *Bnrow, int *Bncol, double *out){

    int i, j, k, tmp;
    int Anr = *Anrow;
    int Anc = *Ancol;
    int Bnr = *Bnrow;
    int Bnc = *Bncol;

    if(Anc != Bnr){
        Rprintf("Dimension Error \n");
    }else{
        for(i=0; i < Anr; i++){
            for(j=0; j < Bnc; j++){
                tmp = 0;
                for(k=0; k < Anc; k++){
                    tmp = tmp + A[i*Anc+k]*B[k*Bnc+j];
                }
                out[i*Bnc+j] = tmp;
            }
        }
    }
    return;
}
```

範例 2: (2) R CMD SHLIB MatrixProduct1.c

(3) 執行

```

dyn.load("MatrixProduct1.dll")

X <- matrix(1:12, nrow=3, ncol=4)
Y <- matrix(1:12, nrow=4, ncol=3)

XX <- as.double(as.vector(t(X)))
YY <- as.double(as.vector(t(Y)))
Xnr <- as.integer(nrow(X))
Xnc <- as.integer(ncol(X))
Ynr <- as.integer(nrow(Y))
Ync <- as.integer(ncol(Y))

result <- .C("MatrixProduct1", XX, Xnr, Xnc, YY, Ynr, Ync,
             out=as.double(rep(0, nrow(X)*ncol(Y))))
Z <- matrix(result$out, ncol=Ync, byrow=T)
print(Z)

X%*%Y

dyn.unload("MatrixProduct1.dll")

```

```

> X
      [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12
> Y
      [,1] [,2] [,3]
[1,]    1    5    9
[2,]    2    6   10
[3,]    3    7   11
[4,]    4    8   12

```

```

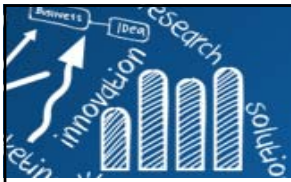
> XX
[1]  1  4  7 10  2  5  8 11  3  6  9 12
> YY
[1]  1  5  9  2  6 10  3  7 11  4  8 12

```

```

> print(Z)
      [,1] [,2] [,3]
[1,]   70  158  246
[2,]   80  184  288
[3,]   90  210  330
>
> X%*%Y
      [,1] [,2] [,3]
[1,]   70  158  246
[2,]   80  184  288
[3,]   90  210  330

```



練習 1: 一個C檔案，兩個C副程式

21/44

- 寫一C程式" `myMatrixManipulation.c`"，具兩個副程式：
 - 一副程式作「兩矩陣之相乘」。
 - 另一副程式作「兩矩陣之相加」。

```
void myMatrixPlus(double *A, double *B, int *nrow, int *ncol,  
                 double *out){  
    ...  
}
```

- 重覆範例 2 之程序。

```
> result <- .C("myMatrixPlus", XX, XX, Xnr, Xnc,  
              out=as.double(rep(0, nrow(X)*ncol(X))))  
> Z <- matrix(result$out, ncol=Xnc, byrow=T)  
> print(Z)
```

	[,1]	[,2]	[,3]	[,4]
[1,]	2	8	14	20
[2,]	4	10	16	22
[3,]	6	12	18	24

練習 2: 兩個C檔案

- 寫出單一「矩陣之轉置」之C副程式並存檔於 "myMatrixManipulation2.c" 中:
- 同時編兩個C程式: `R CMD SHLIB *.c`
- 重覆範例 2 之程序。

```
> result <- .C("myMatrixTranspose", XX, Xnr, Xnc,
  out=as.double(rep(0, nrow(X)*ncol(X))))
> Z <- matrix(result$out, ncol=Ync, byrow=T)
> print(Z)
      [,1] [,2] [,3] [,4]
[1,]    1    2    3    4
[2,]    5    6    7    8
[3,]    9   10   11   12
```

範例 3: 回傳值

(1) mySum.c

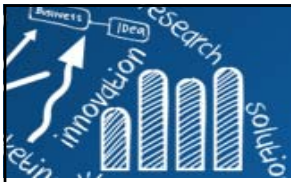
```
#include <R.h>
void sum(int *a, int *b, int *result){
    int x = *a;
    int y = *b;
    *result = x + y;
}
```

(2)

```
R CMD SHLIB mySum.c
```

(3)

```
> dyn.load("mySum.dll")
> a <- 5
> b <- 6
> out1 <- .C("sum", as.integer(a), as.integer(b), as.integer(0))
> out1
[[1]]
[1] 5
[[2]]
[1] 6
[[3]]
[1] 11
> out2 <- .C("sum", c = as.integer(a), as.integer(b), out = as.integer(0))
> out2
$c
[1] 5
[[2]]
[1] 6
$out
[1] 11
> out3 <- .C("sum", a = as.integer(a), b = as.integer(b), out = as.integer(0))
> out3
$a
[1] 5
$b
[1] 6
$out
[1] 11
```



範例 4：不同參數向量型別

(1) myDiffType.c

```
void myDiffType(int *i, double *d, char ** c, int *l){  
  
    i[0] = 2;  
    d[0] = 2.4;  
    c[0] = "A";  
    l[0] = 0;  
  
}
```

(2)

```
R CMD SHLIB myDiffType.c
```

(3.1)

```
> dyn.load("myDiffType.dll")  
> i <- 1  
> d <- 1.4  
> c <- "a"  
> l <- "TRUE"
```

(3.3)

```
> (i <- 1:5)  
[1] 1 2 3 4 5  
> (d <- seq(length=5, from=1, to=5))  
[1] 1 2 3 4 5  
> (c <- c("a", "b", "c"))  
[1] "a" "b" "c"  
> (l <- c("TRUE", "FALSE"))  
[1] "TRUE" "FALSE"
```

(3.2)

```
.C("myDiffType", ii = as.integer(i), dd = as.double(d), cc = as.character(c),  
ll = as.logical(l))
```


.Call語法

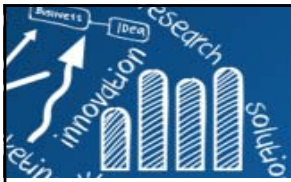
.Call(name, ..., PACKAGE)

- C程式碼較複雜
- 參數是以一系列的R物件傳送。
- 參數以巨集(macros)方式被存取。
- 需引用標頭檔 **<Rinternals.h>** 來使用巨集。
- 需回傳一個R物件(結構為 SEXP)。
- 任何物件由R傳送到C都必是唯讀的。



使用.Call而不使用.C的優點

- 可以在C副程式裡建立並處理R物件。
 - 可以設定物件的屬性，例如向量的名稱，矩陣的維度。
 - 可以存取其它參數型別，例如expressions, raw type。
- 可從C副程式裡呼叫(執行)R函式。
- 可從C副程式裡回傳 R物件。
- 處理資料遺失值較容易。
- .Call有較少的參數(物件)複製
 - 呼叫.C時，在傳送參數給C副程式之前，R物件會被複製一份。
 - 當執行完.C後，這些物件會再被複製一次給R，當成.C的回傳物件。



範例 5: 利用.Call 計算一向量元素總和

27/44

(1) myVecOperation.c

```
#include <R.h>
#include <Rinternals.h>
#include <Rmath.h>

SEXP myVecSum(SEXP Rvec){
    int i, n;
    double *vec, value = 0;
    vec = REAL(Rvec);
    n = length(Rvec);
    for(i = 0; i < n; i++){
        value+= vec[i];
    }
    Rprintf("The value is:
    %4.3f \n", value);
    return R_NilValue;
}
```

(2) R CMD

```
R CMD SHLIB myVecOperation.c
```

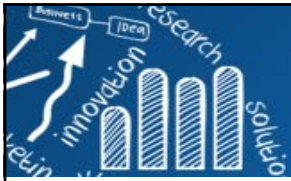
(3) in R

```
dyn.load("myVecOperation.dll")
.Call("myVecSum", rnorm(10))
```

(4) in R , 加上判別

```
dyn.load("myVecOperation.dll")
VecSum <- function(vec){
    if(!is.vector(vec)){
        stop("vec must be a vector!")
    }
    if(!is.real(vec)){
        vec <- as.real(vec)
    }
    .Call("myVecSum", vec)
}

VecSum(rnorm(10))
```



- SEXP:
 - R所定義的參數別結構，為S Expression的縮寫。
 - .Call裡所使用的C副程式必需接受和回傳SEXP。
 - 需強制轉換SEXP到合適的型別。
- 如果沒有回傳物件，則加上「**return R_NilValue**」。
- **vec = REAL(Rvec)**
 - 宣告**vec** 為一個指標，指向**Rvec**的實數部份。
 - 因為可以用**vec[0]**存取資料，不需用**REAL(Rvec)[0]**。
 - 改變**vec**，即會改變**Rvec**。
- 分配一塊空間給向量的語法，需引入**Rinternals.h**，也可以利用**Rdefines.h**得到同樣的效果。

Rdefines.h & Rinternals.h

Source Code : R-2.xx.x.tar.gz

```

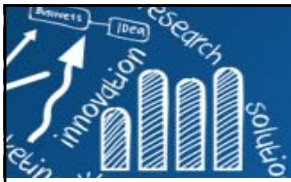
D:\Web-R\Source_Win\R-2.11.1\R-2.11.1\src\include\Rdefines.h - EmEditor
File Edit Search View Macro Tools Window Help
43 ↓
44 ↓
45 /*↓
46 * Added some macros defined in S.h from Splus 5.1↓
47 */↓
48 ↓
49 #define NULL_USER_OBJECT      R_NilValue↓
50 ↓
51 #define AS_LOGICAL(x)         coerceVector(x,LGLSXP)↓
52 #define AS_INTEGER(x)        coerceVector(x,INTSXP)↓
53 #define AS_NUMERIC(x)        coerceVector(x,REALSXP)↓
54 #define AS_CHARACTER(x)       coerceVector(x,STRSXP)↓
55 #define AS_COMPLEX(x)        coerceVector(x,CPLXSXP)↓
56 #define AS_VECTOR(x)          coerceVector(x,VECSXP)↓
57 #define AS_LIST(x)            coerceVector(x,VECSXP)↓
58 #define AS_RAW(x)             coerceVector(x,RAWSXP)↓
59 ↓
60 #define IS_LOGICAL(x)         isLogical(x)↓
61 #define IS_INTEGER(x)         isInteger(x)↓
62 #define IS_NUMERIC(x)         isReal(x)↓
63 #define IS_CHARACTER(x)       isString(x)↓
64 #define IS_COMPLEX(x)         isComplex(x)↓
65 /* NB: is this right? It means atomic or VECSXP or EXPRSXP
66 #define IS_VECTOR(x)          isVector(x)↓
67 /* And this cannot be right: isVectorList(x)? */↓
68 #define IS_LIST(x)            IS_VECTOR(x)↓
69 #define IS_RAW(x)             (typeof(x) == RAWSXP)↓
70 ↓
C++ Ln 60, Col 36 繁體中文 (Big5)

```

```

D:\Web-R\Source_Win\R-2.11.1\R-2.11.1\src\include\Rinternals.h - EmEditor
File Edit Search View Macro Tools Window Help
848 #define allocSExp             Rf_allocSExp↓
849 #define allocVector           Rf_allocVector↓
850 #define any_duplicated         Rf_any_duplicated↓
851 #define any_duplicated3       Rf_any_duplicated3↓
852 #define applyClosure          Rf_applyClosure↓
853 #define arraySubscript        Rf_arraySubscript↓
854 #define asChar                 Rf_asChar↓
855 #define asCharacterFactor      Rf_asCharacterFactor↓
856 #define asComplex             Rf_asComplex↓
857 #define asInteger             Rf_asInteger↓
858 #define asLogical              Rf_asLogical↓
859 #define asReal                 Rf_asReal↓
860 #define asS4                   Rf_asS4↓
861 #define classgets              Rf_classgets↓
862 #define coerceVector           Rf_coerceVector↓
863 #define conformable            Rf_conformable↓
864 #define cons                   Rf_cons↓
865 #define copyMatrix             Rf_copyMatrix↓
866 #define copyMostAttrib         Rf_copyMostAttrib↓
867 #define copyVector             Rf_copyVector↓
868 #define countContexts         Rf_countContexts↓
Cannot find Rf_coerceVector below the current posi C++ Ln 861, Col 32 繁體中文 (Big5)

```



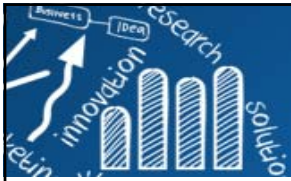
範例 6: 矩陣相乘使用.Call

MatrixProduct2.c

印出矩陣之C副程式

```
#include <R.h>
#include <Rinternals.h>

void printMatrix(double *Mat, int nr, int nc){
    int i, j;
    for(i=0;i<nr;i++){
        for(j=0;j<nc;j++){
            Rprintf("%3.1f ", Mat[i*nc+j]);
        }
        Rprintf("\n");
    }
}
```



範例 6: 矩陣相乘使用.Call (1)

31/44

兩矩陣相乘之C副程式

```
SEXP MatrixProduct2(SEXP A, SEXP B, SEXP isPrint){

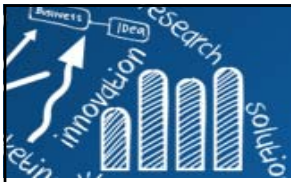
    int *dimA, *dimB;
    double *aptr, *bptr ;

    dimA = INTEGER(coerceVector(getAttrib(A, R_DimSymbol), INTSXP));
    PROTECT(A = coerceVector(A, REALSXP));
    aptr = REAL(A);

    dimB = INTEGER(coerceVector(getAttrib(B, R_DimSymbol), INTSXP));
    PROTECT(B = coerceVector(B, REALSXP));
    bptr = REAL(B);

    int *IsPrint = LOGICAL_POINTER(AS_LOGICAL(isPrint));

    SEXP out;
    double *outptr;
    PROTECT(out = allocMatrix(REALSXP, dimA[0], dimB[1]));
    outptr = REAL(out);
```



範例 6: 矩陣相乘使用.Call (2)

```
int Anr, Anc, Bnr, Bnc;
int i, j, k, tmp;

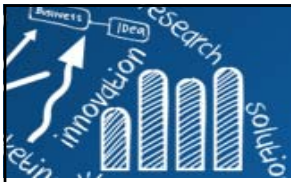
Anr = dimA[0]; Anc = dimA[1];
Bnr = dimB[0]; Bnc = dimB[1];

if(IsPrint[0]){
    Rprintf("Dimension of A matrix: %d x %d\n", Anr, Anc);
    printMatrix(aptr, Anr, Anc);
    Rprintf("Dimension of B matrix: %d x %d\n", Bnr, Bnc);
    printMatrix(bptr, Bnr, Bnc);
}

for(i=0; i<Anr; i++){
    for(j=0; j<Bnc; j++){
        tmp = 0;
        for(k=0; k<Anc; k++){
            tmp = tmp + aptr[i*Anc+k] * bptr[k*Bnc+j];
        }
        outptr[i*Bnc+j] = tmp;
    }
}

UNPROTECT(3);
return(out);
}
```

(承上頁...)



範例 6: 矩陣相乘使用.Call (3)

(1) R CMD SHLIB MatrixProduct2.c

(2)

```
> (A <- matrix(1:9, nrow=3))
      [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9
> (B <- matrix(1:6, nrow=3))
      [,1] [,2]
[1,]    1    4
[2,]    2    5
[3,]    3    6
> A%*%B
      [,1] [,2]
[1,]   30   66
[2,]   36   81
[3,]   42   96
```

(3)

```
> dyn.load("MatrixProduct2.dll")
> .Call("MatrixProduct2", A, B, FALSE)
      [,1] [,2]
[1,]   22   64
[2,]   28   76
[3,]   49  100
```

注意相乘結果

(4)

```
> .Call("MatrixProduct2", A, B, TRUE)
Dimension of A matrix: 3 x 3
1.0 2.0 3.0
4.0 5.0 6.0
7.0 8.0 9.0
Dimension of B matrix: 3 x 2
1.0 2.0
3.0 4.0
5.0 6.0
      [,1] [,2]
[1,]   22   64
[2,]   28   76
[3,]   49  100
```

注意相乘結果

範例 6: 矩陣相乘使用.Call (4)

寫一R函式呼叫.Call

```
MatrixProduct2 <- function(X, Y, isPrint){  
  Xmat <- matrix(as.vector(t(X)), ncol=ncol(X), byrow=F)  
  Ymat <- matrix(as.vector(t(Y)), ncol=ncol(Y), byrow=F)  
  ans <- .Call("MatrixProduct2", Xmat, Ymat, isPrint)  
  Z <- matrix(ans, ncol=ncol(Y), byrow=T)  
  Z  
}
```

```
> MatrixProduct2(A, B, TRUE)  
Dimension of A matrix: 3 x 3  
1.0 4.0 7.0  
2.0 5.0 8.0  
3.0 6.0 9.0  
Dimension of B matrix: 3 x 2  
1.0 4.0  
2.0 5.0  
3.0 6.0  
      [,1] [,2]  
[1,]   30   66  
[2,]   36   81  
[3,]   42   96
```

← 正確相乘結果

當 SEXP 為整數或整數向量時...

- `SEXP myCsrc(SEXP myValue, ...)`
 - `int value = INTEGER_Value(myvalue);`
- `SEXP myCsrc(SEXP myintVec, ...)`
 - `PROTECT(myintVec = AS_INTEGER(myintVec));`
 - `int *ImyintVec = INTEGER_POINTER(myintVec);`
- 若SEXP為實數，則將上述`int` 替換成`double`，`INTEGER` 替換成 `NUMERIC` 即可。
- `PROTECT(myintVec = AS_INTEGER(myintVec));`
 - 在C程式裡產生的R物件必需呼叫PROTECT巨集使用在指向該物件的指標。
 - 這是告訴R，該物件正在使用，不能destroyed。
- `UNPROTECT(n);`
 - 「保護」是stack-based的，故`UNPROTECT(n)`是取消n個被保護的物件。
 - 需與PROTECT同個數。

當 SEXP 為字串時...

- in C
 - `SEXP myCsrc(SEXP myChar, ...)`
 - `PROTECT(myChar = AS_XHARACTER(myChar));`
 - `char *CmyChar;`
 - `CmyChar = R_alloc(strlen(CHAR(STRING_ELT(myChar, 0))), sizeof(char));`
 - `strcpy(CmyChar, CHAR(STRING_ELT(myChar, 0)));`
- in R
 - `myString <- "this is a string"`
 - `.Call("myCsrc", myString, ...)`

範例 7: 比較執行時間 : R

```
MatrixProduct3 <- function(A, B){
  Anr <- nrow(A)
  Anc <- ncol(A)
  Bnr <- nrow(B)
  Bnc <- ncol(B)
  out <- matrix(0, nrow=Anr, ncol=Bnc)
  for(i in 1:Anr){
    for(j in 1:Bnc){
      for(k in 1:Bnr){
        out[i,j] = out[i,j] + A[i,k]*B[k,j]
      }
    }
  }
  out
}
```

```
> MatrixProduct3(A, B)
      [,1] [,2]
[1,]   30   66
[2,]   36   81
[3,]   42   96
```

```
runSpeed.R <- function(n){
  system.time({
    for(i in 1:n){
      Z <- MatrixProduct3(A, B)
    }
  })
}
```

```
> runSpeed.R(10000)
      user  system elapsed
 2.53    0.00    2.53
```

範例 7: 比較執行時間 : .Call

```
runSpeed.Call1 <- function(n){  
  system.time({  
    for(i in 1:n){  
      ans <- .Call("MatrixProduct2", A, B, FALSE)  
    }  
  })  
}
```

```
> runSpeed.2(10000)  
user  system elapsed  
0.14   0.00   0.14
```

```
runSpeed.Call2 <- function(n){  
  system.time({  
    for(i in 1:n){  
      MatrixProduct2(A, B, FALSE)  
    }  
  })  
}
```

```
> runSpeed.Call2(10000)  
user  system elapsed  
0.92   0.00   0.92
```

範例 7: 比較執行時間 : .C

```
runSpeed.C <- function(n){
  system.time({
    for(i in 1:n){
      ans <- .C("MatrixProduct1",
               as.numeric(t(A)), nrow(A), ncol(A),
               as.numeric(t(B)), nrow(B), ncol(B),
               out=numeric(nrow(A)*ncol(B)))$out
      Z <- matrix(ans, ncol=ncol(B), byrow=T)
    }
  })
}
```

```
> runSpeed.C(10000)
   user  system elapsed
  0.90   0.00   0.91
```



制作R套件時，使用.C/.Call/.External的注意事項

- 複製 *.c 到 `myPackage/src/`
- 因為 `myPackage` 不會自動載入編譯好的C程式碼，故
 - 新增加一個檔案 `zzz.R` 到 `myPackage/R`，內容為

```
.First.lib <- function(lib, pkg){  
  library.dynam("myPackage", pkg, lib)  
}
```

- 呼叫.C/.Call時，增加參數 `PACKAGE="myPackage"`

```
.Call("MatrixProduct2", Xmat, Ymat, isPrint,  
      PACKAGE="myPackageName")
```


Stack overflow的問題: Cstack_info()

- 回報C stack 的大小及使用的狀況(if available).
- 當R啟動時，C stack的使用狀況就會被紀錄。
- 假使stack大小不足，則記憶體對應會出錯。

語法

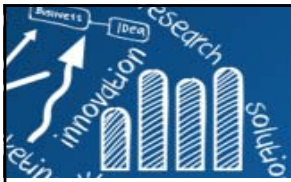
Cstack_info()

- size:
 - The size of the stack (in bytes), or NA if unknown.
- current:
 - The estimated current usage (in bytes), possibly NA.
- direction
 - 1 (stack grows down, the usual case) or -1 (stack grows up).
- eval_depth
 - The current evaluation depth (including two calls for the call to Cstack_info).

```
> Cstack_info()
      size      current  direction eval_depth
10485760      5344         1           2
```

.C/.Call的一般建議

- 先寫純R程式碼，再使用.C，最後再使用.Call。
 - R命令檔若有迴圈，通常執行時間較久。
 - 以.C/.Call方式編寫C程式碼，並在R的環境執行，同時利用了C的執行效率，及R的高階指令。
- 有用到特別的函數，例如分配函數或隨機數，則需引入<Rmath.h>



(1) Rnorm.c

```
#include <R.h>
#include <Rmath.h>

void Rnorm(){
  int i;
  double mu, sigma, *x;
  mu = 0;
  sigma = 1;

  x = (double *) R_alloc(10, sizeof(double));

  GetRNGstate();
  for(i = 0; i < 10; i++){
    x[i] = rnorm(mu, sigma);
    Rprintf("x: %f ", x[i]);
  }
}
```

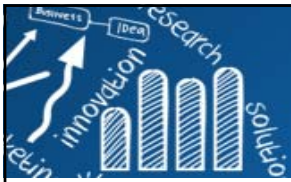
(2) R CMD

```
R CMD SHLIB Rnorm.c
```

(3) in R

```
dyn.load("Rnorm.so")
set.seed(10)
out <- .C("Rnorm")
dyn.unload("Rnorm.so")
```

請參考 http://www.math.ncu.edu.tw/~chenwc/R_note/index.php?item=call_R



Call R function from C/C++ Program (standalone)

44/44

```
void main(){
    ...
    out = fopen("Ch14.R", "w");
    ...
    fprintf(out, "X=rnorm(%d,%f,%f)\n", N, mu, sd);
    ....
    system("C:/progra~1/R/R-2.5.1/bin/R CMD BATCH Ch14.R");
    ...
}
```

C call R ?!

(1) Rnorm.c

(2) 編譯，連結並執行: `make, gcc`

- Build R as a shared library (libR.dll) and link against the c library.
- May also requires "Rmath.dll", "Rblas.dll", and "Rlapack.dll" for Windows.
- Embedding R in Other Applications:
<http://developer.r-project.org/embedded.html>

```
#include <Rmath.h>

int main(){
    int i;
    double mu, sigma, *x;
    mu = 0;
    sigma = 1;

    x = (double *) malloc(10);
    for(i = 0; i < 10; i++){
        x[i] = rnorm(mu, sigma);
        printf("X: %f ", x[i]);
    }
    return 0;
}
```

請參考 http://www.math.ncu.edu.tw/~chenwc/R_note/index.php?item=standalone