INTERFACE R/C

Alicia STOTZ

Statistique et Génome, Université d'Evry (Carène Rizzon)
Origine, structure et évolution de la biodiversité, MNHN (Jérôme Sueur)

31 Mai 2012

Projet

Package SEEWAVE développé au sein du MNHN, par Jérôme Sueur, Thierry Aubin et Caroline Simonis.

$\rightarrow$ Environ cent fonctions qui permettent la synthèse et l'analyse de différents types de sons.

BUT:

- Réaliser les calculs en extérieur de l'environnement `R`
- Récupérer les résultats dans `R`

$\rightarrow$ Temps de calcul amélioré

Sommaire

Sommaire

Le language C

Qu'est-ce que c'est ???

Le language C

Qu'est-ce que c'est ???

- Le plus célèbre et le plus utilisé

Le language C

Qu'est-ce que c'est ???

- Le plus célèbre et le plus utilisé
- Language de bas niveau

Le language C

Qu'est-ce que c'est ???

- Le plus célèbre et le plus utilisé
- Language de bas niveau
- Contient beaucoup de types

Le language C

Qu'est-ce que c'est ???

- Le plus célèbre et le plus utilisé
- Language de bas niveau
- Contient beaucoup de types
- Language impératif

Sommaire

Pourquoi avoir recours à une interface R/C

Pourquoi avoir recours à une interface R/C

- Utiliser les librairies présentes dans C

Pourquoi avoir recours à une interface R/C

- Utiliser les librairies présentes dans C
- Améliorer le temps de calcul par rapport à `R` sur des données importantes ($>$ Mo)

Pourquoi avoir recours à une interface R/C

- Utiliser les librairies présentes dans C
- Améliorer le temps de calcul par rapport à `R` sur des données importantes ($>$ Mo)
- Gestion efficace de la mémoire

Sommaire

Les librairies nécessaires

```
# include <R.h>

# include <Rinternals.h>

# include <Rmath.h>

# include <Rdefines.h>
```

Sommaire

Fonction d'appel possibles dans R

Fonction .C

Utilisée pour des arguments "primitifs" (vector, double, integer)

Fonction .Call

Utilisée pour des arguments plus complexes telles que les listes ou les matrices

Exemple :

```
> ?.Call
.Call(.NAME, ..., PACKAGE)
```

Sommaire

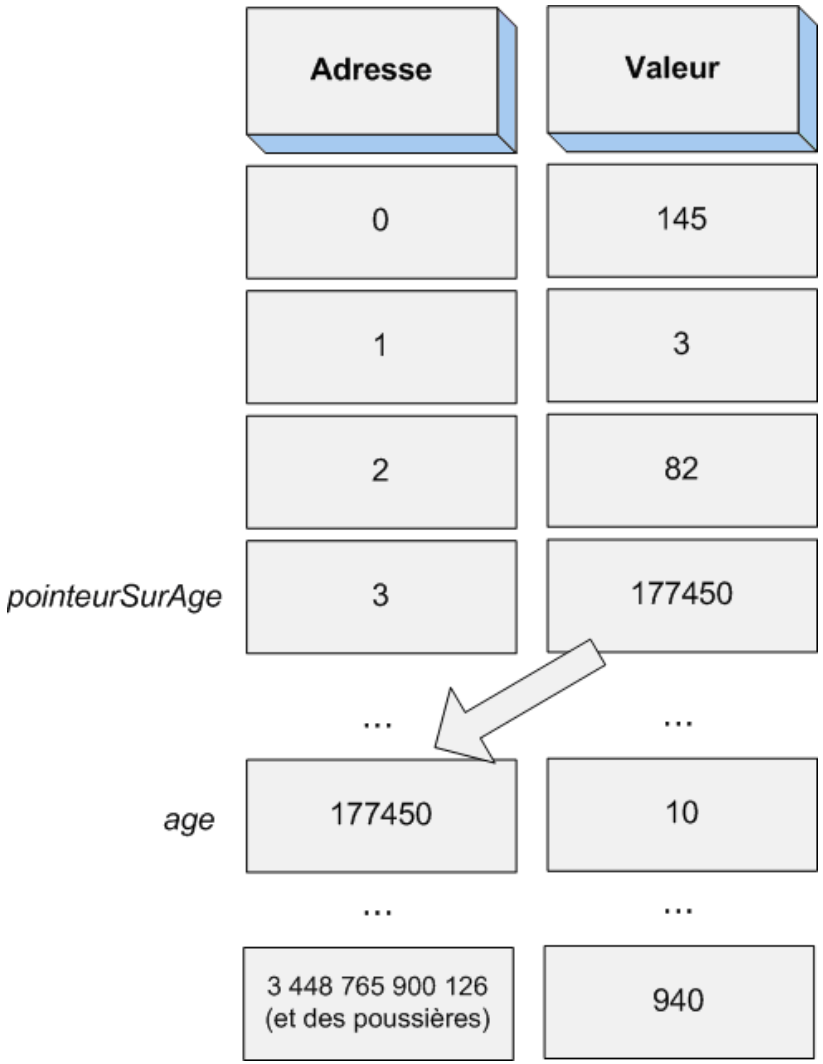
Programme C pouvant être appelé par la fonction .C

Petit exemple: Hello World !!!

```c
# include <R.h>

void hello (int *n)
{
    int i;
    for (i = 0; i < *n; i++)
    {
        printf ("hello World !\n");
    }
}
```

Les pointeurs

Figure: Mémoire, adresses et pointeurs (site du zero - languageC)



Fonction .C

La compilation

```
R CMD SHLIB nomprogramme.c
```

```
   > R CMD SHLIB helloC.c
```

Dans le terminal:

```
    gcc -std=gnu99 -I/usr/local/R-2.15.0/lib64/R/include
-DNDEBUG  -I/usr/local/include    -fpic  -g -O2  -c helloC.c
-o helloC.o
    gcc -std=gnu99 -shared -L/usr/local/lib64 -o helloC.so
 helloC.o -L/usr/local/R-2.15.0/lib64/R/lib -lR
```

Appel fonction .C

Et dans R...

```
> dyn.load("helloC.so")
> source("Rcode.R")
```

Le script R:

```
helloR <- function (n)
{
      .C("hello",as.integer(n))
}
```

Sortie R:

```
> helloR(3)
    Hello world !!
    Hello world !!
    Hello world !!
    [[1]]
    [1] 3
```

Programme C pouvant être appelé par la fonction .C

Les différents types d'objets

```
# include <R.h>
void type ( double *d, char **c, int *l)
{

      d[0] = 25.56;\\Numerique

      c[1] = "z";   \\charactere

      l[0] = 0;     \\logique
}
```

Appel fonction .C

Les différents types d'objets (suite)

```
> dyn.load("typeC.so")

> d <- seq(length =3, from = 1, to = 2)
> c <- c("a", "b", "c")
> l <- c("TRUE", "FALSE")

> out <- .C("type", d1 = as.numeric(d),
    c1 = as.character(c), l1 = as.logical(l))
> out
d1
  [1] 25.56  1.50  2.00
c1
  [1] "a" "z" "c"
l1
  [1] FALSE FALSE
```

Sommaire

.Call version améliorée du .C

Avantages

- Passer un objet `R` dans C
- Créer un objet `R` dans C
- Manipuler un objet `R` dans C
- Retourner un objet `R` de C
- Appeler une fonction `R` de C

Inconvénient

Plus difficile à manipuler et à utiliser

Programme C pouvant être appelé par la fonction .Call

Exemple fonction moyenne

```
SEXP moyenne (SEXP Rvec)
{
    int i, tailleVecteur = 0;
    double *vec = NULL;
    double valeur = 0;

    Rvec = coerceVector (Rvec, REALSXP);
    vec = REAL(Rvec);
    tailleVecteur = length(Rvec);

    for ( i = 0; i < tailleVecteur; i ++)
    {
        valeur = valeur + vec[i];
    }
    printf("la moyenne est egale a : %f\n",
    valeur/tailleVecteur);
return (R_NilValue);
}
```

Programme C pouvant être appelé par la fonction .Call

Explications...

- SEXP est une structure définie dans R. Représente des simples expressions. L'appel d'une fonction avec .Call va retourner un SEXP.

Programme C pouvant être appelé par la fonction .Call

Explications...

- SEXP est une structure définie dans R. Représente des simples expressions. L'appel d'une fonction avec .Call va retourner un SEXP.
- Ici la fonction ne retourne rien: **return** R_NilValue

Programme C pouvant être appelé par la fonction .Call

Explications...

- SEXP est une structure définie dans R. Représente des simples expressions. L'appel d'une fonction avec .Call va retourner un SEXP.
- Ici la fonction ne retourne rien: **return** R_NilValue
- Elle prend en argument un vecteur de type Réel:
  Rvec = coerceVector (Rvec, REALSXP)

Programme C pouvant être appelé par la fonction .Call

Explications...

- SEXP est une structure définie dans R. Représente des simples expressions. L'appel d'une fonction avec .Call va retourner un SEXP.
- Ici la fonction ne retourne rien: **return** R_NilValue
- Elle prend en argument un vecteur de type Réel:
  Rvec = coerceVector (Rvec, REALSXP)
- Pour pouvoir le manipuler: création d'un pointeur qui cible notre vecteur (vec = REAL(Rvec);

Programme C pouvant être appelé par la fonction .Call

Explications...

- SEXP est une structure définie dans R. Représente des simples expressions. L'appel d'une fonction avec .Call va retourner un SEXP.
- Ici la fonction ne retourne rien: **return** R_NilValue
- Elle prend en argument un vecteur de type Réel:
  Rvec = coerceVector (Rvec, REALSXP)
- Pour pouvoir le manipuler: création d'un pointeur qui cible notre vecteur (vec = REAL(Rvec);
- On parcours juste le vecteur et on calcule la moyenne:

```
for ( i = 0; i < tailleVecteur; i ++)
{
    valeur = valeur + vec[i];
}
printf("la moyenne est egale a : %f\n",
     valeur/tailleVecteur);
```

Appel fonction .Call

La compilation, utilisation

```
> library(MASS)
> data(crabs)
> crabsRW<- crabs[,4]

> dyn.load("meanNul.so")
> .Call("moyenne", crabsRW)
la moyenne est egale a : 12.738500
NULL
```

La valeur de retour est nul, on ne peut pas la manipuler mais...

Programme C pouvant être appelé par la fonction .Call

... Et avec une valeur de retour ?

```
SEXP Rmoyenne;
double *pointeurRmoyenne = NULL;

PROTECT(Rmoyenne = allocVector(REALSXP, 1));
pointeurRmoyenne = REAL(Rmoyenne);

*pointeurRmoyenne = valeur/tailleVecteur;

UNPROTECT(1);
return(Rmoyenne);
```

Dans R:

```
> dyn.load("mean.so")
> moy <- .Call("moyenne", crabsRW)
la moyenne est egale a : 12.738500
> moy
[1] 12.7385
```

Programme C pouvant être appelé par la fonction .Call

Les matrices

Programme C pouvant être appelé par la fonction .Call

Les matrices

- Matrice en entrée

```
double *matrixptr

PROTECT(MATRIX = coerceVector(MATRIX, REALSXP));
matrixptr = REAL(MATRIX);
```

Programme C pouvant être appelé par la fonction .Call

Les matrices

- Matrice en entrée

```
double *matrixptr

PROTECT(MATRIX = coerceVector(MATRIX, REALSXP));
matrixptr = REAL(MATRIX);
```

- Matrice en sortie

```
SEXP matriceC (SEXP MATRIX){
  SEXP dimMatrix;
  dimMatrix = getAttrib(MATRIX, R_DimSymbol);

  nrow = INTEGER (dimMatrix)[0];
  ncol = INTEGER (dimMatrix)[1];

  PROTECT(outMatrix = allocMatrix(REALSXP, nrow,
      ncol));
  outmatrixptr = REAL(outMatrix);
```

Programme C pouvant être appelé par la fonction .Call

Les matrices (suite)

Programme C pouvant être appelé par la fonction .Call

Les matrices (suite)

- Remplissage d'une matrice

```
for (j = 0; j < ncol; j++)
{
    for (i = 0; i < nrow; i++)
    {
        outmatrixptr[ i + nrow * j] =  20 *
          matrixptr[i + nrow * j];
    }
}
```

Programme C pouvant être appelé par la fonction .Call

Les listes

```
SEXP retList;
SEXP names;

PROTECT( retList = allocVector (VECSXP, 2));
PROTECT( names = allocVector (STRSXP, 2));

SET_STRING_ELT(names, 0, mkChar ("vec"));
SET_STRING_ELT (names, 1, mkChar ("matrix"));

setAttrib(retList, R_NamesSymbol, names);
```

Programme C pouvant être appelé par la fonction .Call

Les listes (suite)

```
    PROTECT(matrix = allocMatrix(REALSXP, nrow, ncol));
    matrixptr = REAL(matrix);

    PROTECT ( vec = allocVector (REALSXP, nrow));
    vecptr = REAL(vec);

    SET_VECTOR_ELT (List, 0, vec);
    SET_VECTOR_ELT (List, 1, matrix);

return retList;
```

Appel fonction .Call

Les listes: exemple

```
> dyn.load("linemean.so")

> vecteur <- .Call("mean",matrice)

> vecteur$vec
[1] 2 3

> vecteur$matrix
     [,1] [,2]
[1,]    1    3
[2,]    2    4
```

Intérêt de l'interface R/C

Appel des librairies

Le code C:

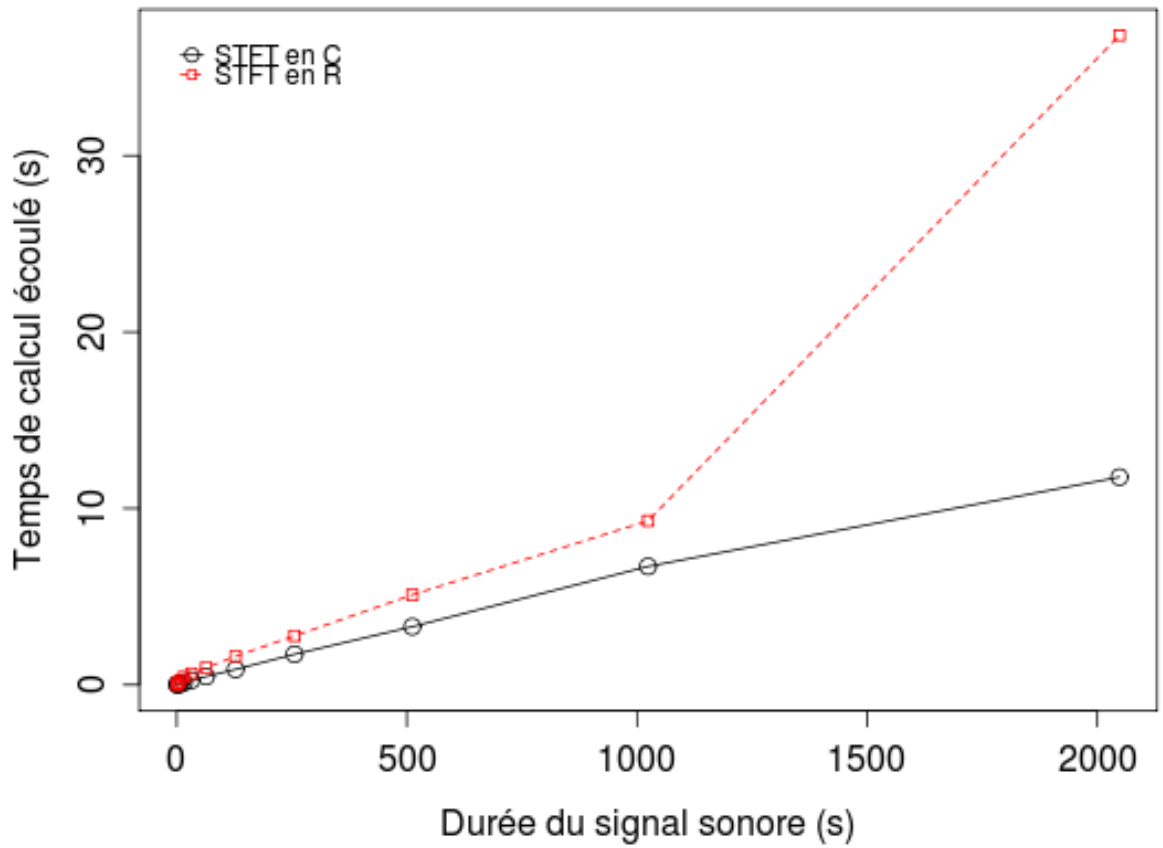
```
# include <stdio.h>

# include <stdlib.h>

# include <fftw3h.h>

# include <sndfile.h>
```

La compilation :

```
R CMD SHLIB fftwR.c -lsndfile -lfftw3
```

Intérêt de l'interface R/C

Figure: Différence dans le temps d'exécution entre R et C pour le calcul de la STFT



Références

- Image pointeur, site du zero - language C
- An Introduction to the .C Interface to R (**Roger D.Peng et Jan de Leeuw, Aout 2008**)
- R/Cpp: Interface Classes to simplify Using R Object in C++ (**J.J Java, D.P Gaile et K.F Manly, juillet 2007**)
- Calling C function from R using .C and .Call (**S. hojsgaard, janvier 2010**)
- Calling C code from R, an introduction (**S.Blay, octobre 2004**)
- Using .Call in R (**B. Caffo**)

Merci...