Building an R Package with Fortran (or C)

Seray Cetin, BSc.

23. Jänner 2019, FH Wr. Neustadt

1 Einleitung

Allgemein beim Programmieren werden Funktionen implementiert, deren Aufgaben es sind immer wieder benötigte Codezeilen nicht doppelt oder dreifach abzutippen. Der Programmierer erleichtert sich mit den Funktionen seine Arbeit indem er sie nur noch aufrufen muss (eine Zeile statt vielleicht 10). Bei größeren Programmen bzw. Projekten kann es schnell passieren, die Übersicht der Funktionen zu verlieren.

Eine Lösung für dieses Problem beziehungsweise ein eleganterer Weg wäre es ein eigenes Package zu erstellen bzw. zu implementieren, indem alle persönlichen Funktionen des Programmierers, die er/sie immer wieder für seine Projekte verwendet, gesammelt werden. Die Vorteile eines selbst erstellten Package sind: [Sch13]

- Einmaliges implementieren
- immer wieder verwendbare Methoden
- für alle Projekte übertragbar
- eigene persönliche Dokumentation

2 Warum C oder Fortran in R verwenden?

R ist bei iterativen Algorithmen, die oft wiederholte Schleifen benötigen (bspw. Markov-Chain-Monte-Carlo-Verfahren) nicht ganz effizient. Um die Geschwindigkeitsvorteile von C oder Fortran in R auszunutzen, werden die Rechenaufwendigen Schleifen in C oder Fortan ausgeführt. [Gey]

3 Implementierung

3.1 Wichtige Punkte bei der Umsetzung

- „.Call“ Interface verwenden, da es modern/neu ist, wenig overhead hat und sicherer als .C, .Fortran oder .External ist. [Adl18]

- Kommunikation zwischen R und C muss mittels SEXP (Simple EXPression) erfolgen, weil R nur SEXP versteht. Bei der Kommunikation zwischen C, C++ oder Fortran muss dies unbedingt nicht sein. Daten die zwischen R und C mittels .Call-Funktion übermittelt werden, müssen als SEXP definiert werden. [Adl18]
- Es ist einfacher Fortran-Code über C aufzurufen, aber dabei nur Unterprogramme und keine Funktionen. Funktionen innerhalb eines Fortran Code sind in Ordnung, aber Fortran-Code das über C aufgerufen wird, für eine R-Funktion, sollte eine Unterfunktion (Subroutines) sein. [Adl18]
- „Writing R Extension“ beachten [Adl18]

3.2 Packages in R erstellen (mit Fortran)

Ein eigenes Paket, mittels Fortran, zu erstellen ist ganz simpel. In diesem Kapitel wird dies in 5 Schritten erklärt. Als Beispiel wird die Lower Loss Cost (LLC) berechnet.

3.2.1 Schritt 1: Fortran Code

Der einfachste Weg in Fortran etwas zu implementieren ist mittels „module“. Diese erlaubt Funktionen und Unterfunktionen von anderen Dateien einfacher aufzurufen. Für unser einfaches Beispiel sieht der Fortran Code folgendermaßen aus:

Listing 1: Fortran Code [Adl18]

```
module fortloop
  use, intrinsic :: iso_c_binding
  implicit none
  private
  public :: llc_f

  contains
    subroutine llc_f(x, n, l, a, llc) bind(C, name = "llc_f_")
      real(kind = c_double), intent(in) :: l, a
      integer(kind = c_int), intent(in), value :: n
      real(kind = c_double), intent(in), dimension(n) :: x
      real(kind = c_double), intent(out) :: llc
      integer :: i

      llc = 0.0_c_double
      do i = 1, n
        llc = llc + max(0.0_c_double, min(x(i) - a, l))
      end do
    end subroutine llc_f
end module fortloop
```

Die Unterfunktion `llc_f` erhält ein Vektor `x` mit der Länge `n`, ein Limit `l`, die Messung `a` und den Rückgabewert `llc`. Die Verwendung von `iso_c_binding` ermöglicht es dem Kodierer, den Namen des aufgerufenen C-Programms in der Unterfunktion mittels `bind` anzugeben. Der `F77_` Prefix in C fügt dem eingebundenen Namen einen Unterstrich hinzu, in unserem Fall lautet dieser `llc_f_`. Das `iso_c_binding` beinhaltet auch build-it Datentypen, diese sollten für eine bessere Zusammenarbeit zwischen den unterschiedlichen Programmteilen hilfreich sein. [Adl18]

Standardmäßig wird die Zuordnung der Datentypen mit den Anfangsbuchstaben der Variablen übernommen, dies wird auch implizite Typenanweisung genannt. Durch die implizite Typenanweisung können Mögliche Fehler entstehen, so ist es sinnvoll diese komplett auszuschalten, dies geschieht mit folgendem Befehl: `implicit none`. [Wik11]

3.2.2 Schritt 2: C Code

3.2.3 C-to-Fortran

Die erste Funktion des C-Programms ist das Aufrufen der Fortran-Funktion:

Listing 2: C-to-Fortran Code [Adl18]

```
void F77_NAME(llc_f)(double *x, int n, double *l, double *a, double *ret);
```

Diese Funktion übergibt die Variable an die Fortran-Funktion. Dies ist auch die Funktion, die von Fortran mit einem zusätzlichen Unterstrich versehen wird. Die Variablen werden als Referenzen übergeben und sind deshalb auch, in C, als Zeiger zu übermittelt. 2003 wurde das Schlüsselwort `value` in Fortran eingeführt, welches das übergeben von Werten ermöglicht. In unserem Beispiel wurde die Variable n in Fortran mit value deklariert, deshalb muss diese Variable nicht als Zeiger übermittelt werden. [Adl18]

3.2.4 C-to-R

Die nächste Funktion ist, die die von R aus aufgerufen wird:

Listing 3: C-to-R Code [Adl18]

```
extern SEXP c_llc_f(SEXP x, SEXP l, SEXP a){
  const int n = LENGTH(x);
  SEXP ret;
  PROTECT(ret = allocVector(REALSXP, 1));
  F77_CALL(llc_f)(REAL(x), n, REAL(l), REAL(a), REAL(ret));
  UNPROTECT(1);
  return(ret);
}
```

Diese Funktion wird benötigt für die Kommunikation zwischen R und C. Die Variablen (x, l, a), die von R übergeben werden, müssen als SEXP (Simple EXPression) definiert werden. Variable die nicht von R kommen müssen nicht als

SEXP definiert werden. Der Rückgabewert ret muss ebenfalls als SEXP deklariert werden, da dieser Wert an R zurückgeliefert wird. `PROTECT` und `UNPROTECT` sind erforderlich, damit R den Speicher (Memory) nicht freigibt bevor das Programm damit fertig ist. [Adl18]

Mittels `F77_CALL` wird die `F77_NAME` aufgerufen, dabei ist es wichtig das die Funktion angemessen mittels SEXP typisiert ist. Solange alle Variablen den Typen double haben, werden sie als REAL() übergeben. [Adl18]

3.2.5 Schritt 3: R Code

In R wird die C-Funktion mit .Call aufgerufen:

Listing 4: R Code [Adl18]

```
LLC_f2 <- function(x, l, a) {
  if (!is.double(x)) {storage.mode(x) <- 'double'}
  if (!is.double(l)) {storage.mode(l) <- 'double'}
  if (!is.double(a)) {storage.mode(a) <- 'double'}
  .Call(c_llc_f, x, l, a)
}
```

Die vom Benutzer aufgerufene Package-Funktionen (bspw. im Terminal) wird in R kodiert. Hier ist es wichtig das die Datentypen nochmals überprüft werden, da es sonst Probleme mit dem C-Code auftauchen kann. Nach der Überprüfung werden die Werte an die C-Funktion mit dem Interface .Call weitergeleitet.

3.2.6 Schritt 4: Benötigte Pakete in C hinzufügen bzw. einbinden

Hier sind zwei Sachen wichtig:

1. die Deklaration der Interface-Methoden
2. die Registrierung der Unterfunktionen (Subroutines)

Listing 5: C Code Package [Adl18]

```
static const R_CallMethodDef CallEntries[] = {
  {"c_llc_f",    (DL_FUNC) &c_llc_f,   3}
  {NULL,          NULL,                0}
};

void R_init_newPackage (DllInfo *dll) {
  R_registerRoutines(dll, NULL, CallEntries, NULL, NULL);
}
```

In der ersten Funktion `R_CallMethodDef` werden alle Funktionen/Methoden definiert die mit .Call aufgerufen werden und die zweite Funktion registriert die einzelnen Unterfunktionen (Subroutines) des Fortran Programms. Bei

der zweiten Funktion muss bei der Benennung darauf geachtet werden, dass nach `R_init_` der Name des Packages geschrieben werden muss, da sonst die Registrierung der Methoden nicht funktionieren kann. (in unserem Beispiel: `R_init_newPackage`). [Adl18]

3.2.7 Schritt 5: die restlichen Dokumente des Packages bearbeiten

Übrig bleiben nur noch die Dokumentationen der einzelnen Funktionen und das Ausfüllen der DESCRIPTION- und der NAMESPACE-Datei. Beim NAMESPACE ist es wichtig folgende Zeile einzufügen, wenn die Registration in C erfolgen soll: useDynLib(newPackage, .registration=TRUE) Weiters können zusätzlich die README, die NEWS und die LICENSE Dateien hinzugefügt bzw. bearbeitet werden. [Adl18]

4 Conclusio

Ein Package in R zu implementieren, welches auch Fortran-Code enthält, ist nicht so kompliziert, wie es am Anfang erscheint. Das Beispiel zusammengefasst haben wir haben eine R-Funktion, die eine C-Funktion aufruft, die wiederum eine andere C-Funktion aufruft, welches den Fortran Code ausführt. Zwischen R und Fortran steht C, welches mit beiden Programmiersprachen Problemlos kommunizieren kann. Der Sinn des Ganzen ist es die Stärken und Vorteile von C und Fortran für die Berechnungen in R auszunutzen.

Literatur

[Adl18] Adler, Avraham: *The Need for Speed Part 1: Building an R Package with Fortran (or C).* `https://www.avrahamadler.com/2018/12/09/the-need-for-speed-part-1-building-an-r-package-with-fortran/`. Version: 2018

[Gey] Geyer, Charles: *Calling C and Fortran from R.* `http://users.stat.umn.edu/~geyer/rc/`

[Sch13] Schneider, Martin: *Professioneller R Code – einfach eigene Pakete erstellen und dokumentieren.* `https://blog.eoda.de/2013/12/11/professioneller-r-code-einfach-eigene-pakete-erstellen-und-dokumentieren/`. Version: 2013

[Wik11] *Fortran: Fortran 95: Datentypen.* `https://de.wikibooks.org/wiki/Fortran:_Fortran_95:_Datentypen`. Version: 2011