R, Other Languages
and Object-Oriented Programming

John M. Chambers

November 23, 2010
1. The benefits of multilingualism -- For serious research or data analysis, consider using more than one.

   Two recent developments that may help:

2. A new interface language? -- Easier and more powerful interface when speed is needed.

3. Object-oriented programming in R -- The natural programming model for many requirements.
The Benefits of Multilingualism

Jared Diamond

Multilingualism—the ability to understand and speak several languages—is exceptional in the United States but common elsewhere, especially in small-scale traditional societies. For instance, once while I was camped with some New Guinea Highlanders conversing simultaneously in several local languages, I asked each man to name each language in which he could converse. It turned out that everyone present spoke at least 5 languages, and the champion was a man who spoke 15. What are the cognitive effects of such multilingualism? Recent studies (1–5) show that children raised bilingually develop a specific type of cognitive benefit during infancy, and that bilingualism offers some protection against symptoms of Alzheimer’s dementia in old people.

Bilingual education is politically controversial in the United States. Even immigrants whose native language is not English often believe that their children should learn only English and will be confused by learning two languages simultaneously. Until the 1960s, research seemed to show that bilingual children acquired language more slowly than monolingual children and achieved smaller vocabularies. But other variables correlated with bilingualism in those early studies, such as schooling and parental socioeconomic status, confounded their interpretation. More recent studies, comparing subjects matched for those other variables, have found bilinguals and monolinguals to be largely similar in cognition and language processing (6–8).

The clearest difference identified by these studies involves an advantage that bilinguals have over monolinguals, rather than a disadvantage. Our minds are assaulted by varied sights, sounds, and other external sensory inputs, plus thoughts and profoceptive inputs (9). To succeed in doing anything at all, we must temporarily inhibit 99% of those inputs and attend to just 1% of them, and the appropriate choice varies with the circumstances. That selective attention involves a set of processes, termed executive function, that reside in the prefrontal cortex and develop especially over the first 5 years of life (9).

Competing inputs. A typical moment in the life of a typical Science reader, who is simultaneously processing various sensory inputs with his eyes, ears, nose (the smell of the food), skin touch receptors (the friend’s handshake, the briefcase being held), and proprioceptors (the cramp in his leg), and having competing thoughts. Depending on the circumstances, any one of those stimuli or thoughts may warrant full attention. Recent studies suggest that multilingual people may have an advantage over monolinguals in sifting and managing these distracting stimuli.

As a result, multilinguals have constant unconscious practice in using the executive function system. Recent studies assess this ability by assigning to subjects game-like tasks designed to be confusing, either because the task rules change unpredictably, or because the task presents misleading cues that must be ignored (1–3, 7, 8). For instance, children are shown cards depicting either a rabbit or a boat, colored either red or blue, with or without a star. If the card has a star, the children must sort cards by color; if a star is absent, they must sort cards by the object depicted. It turns out that monolingual and bilingual subjects are equally successful if the rule remains the same from trial to trial (e.g., “sort by color”), but multilinguals have more difficulty than bilinguals at accommodating to a switch in rules. Although success at these games won’t by itself make one rich or happy, our lives are full of other misleading information and rule changes. If bilinguals’ advantage over monolinguals in these games also applies to real-life situations, that could be useful for bilinguals.

While this superior executive function has been reported for bilinguals of all ages, results for the youngest and the oldest subjects are of particular interest. Kovacs and Mehler (4, 5) tested confusing game tasks on “monolingual” infants and “crib bilingual” infants—that is, infants reared from birth to hear and eventually to speak two languages, because mother and father speak to the infant in different languages. It might seem meaningless to describe infants who cannot speak as monolingual or bilingual. Actually, infants learn to discriminate the sounds of the language or languages heard around them and to ignore sound distinctions not heard around them. For instance, Japanese infants lose, and English infants retain, the ability to discriminate
The Benefits of Multilingualism

Jared Diamond

Multilingualism—the ability to understand and speak several languages—is exceptional in the United States but common elsewhere, especially in small-scale traditional societies. For instance, once while I was camped with some New Guinea Highlanders conversing simultaneously in several local languages, I asked each man to name each language in which he could converse. It turned out that everyone present spoke at least 5 languages, and the champion was a man who spoke 15. What are the cognitive effects of such multilingualism? Recent studies (1–5) show that children raised bilingually develop a specific type of cognitive benefit during infancy, and that bilingualism offers some protection against symptoms of Alzheimer’s dementia in old people.

Bilingual education is politically controversial in the United States. Even immigrants whose native language is not English often believe that their children should learn only English and will be con-

Competing inputs. A typical moment in the life of a typical Science reader, who is simultaneously processing various sensory inputs with his eyes, ears, nose (the smell of the food), skin touch receptors (the friend’s handshake, the briefcase being held), and proprioceptors (the cramp in his leg), and having competing thoughts. Depending on the circumstances, any one of those stimuli or thoughts may warrant full attention. Recent studies suggest that multilingual people may have an advantage over monolinguals in sifting and managing these distracting stimuli.
The Benefits of Multilingualism

• All that software may include the tool(s) you need.

• You can do efficient computations when you need them,

• and also do large-scale (distributed) computations.

• You can deal with the actual data sources,

• and, of course, express statistical ideas.
The Benefits of Multilingualism

• All that software may include the tool(s) you need. (R, Python, Java, C++, Javascript,.....)

• You can do efficient computations when you need them,

• and also do large-scale (distributed) computations.

• You can deal with the actual data sources,

• and, of course, express statistical ideas.
The Benefits of Multilingualism

• All that software may include the tool(s) you need. (R, Python, Java, C++, Javascript,.....)
• You can do efficient computations when you need them, (C++, C, Fortran, Perl, Python, ...)
• and also do large-scale (distributed) computations.

• You can deal with the actual data sources,
• and, of course, express statistical ideas.
The Benefits of Multilingualism

• All that software may include the tool(s) you need. (R, Python, Java, C++, Javascript, …)

• You can do efficient computations when you need them, (C++, C, Fortran, Perl, Python, …)

• and also do large-scale (distributed) computations. (Java, C#, Map/reduce, …)

• You can deal with the actual data sources,

• and, of course, express statistical ideas.
The Benefits of Multilingualism

• All that software may include the tool(s) you need. (R, Python, Java, C++, Javascript,.....)
• You can do efficient computations when you need them, (C++, C, Fortran, Perl, Python, ...)
• and also do large-scale (distributed) computations. (Java, C#, Map/reduce,....)
• You can deal with the actual data sources, (DBMS, Excel, ...)
• and, of course, express statistical ideas.
The Benefits of Multilingualism

• All that software may include the tool(s) you need. (R, Python, Java, C++, Javascript,.....)

• You can do efficient computations when you need them, (C++, C, Fortran, Perl, Python, ...)

• and also do large-scale (distributed) computations. (Java, C#, Map/reduce,....)

• You can deal with the actual data sources, (DBMS, Excel, ...)

• and, of course, express statistical ideas. (R, Matlab, ...)

The Benefits of Multilingualism

• All that software may include the tool(s) you need. *(R, Python, Java, C++, Javascript, …)*
• You can do efficient computations when you need them, *(C++, C, Fortran, Perl, Python, …)*
• and also do large-scale (distributed) computations. *(Java, C#, Map/reduce, …)*
• You can deal with the actual data sources, *(DBMS, Excel, …)*
• and, of course, express statistical ideas. *(R, Matlab, …)*
Two Recent Developments

• The **Rcpp** interface between **R** and **C++**
  
  A new interface language?

• Object-Oriented Programming in **R**
  
  The **C++**, etc. model in **R**, uniform and complete.
Algorithm Interface

ABC: general
(FORTRAN)
algorithm

XABC: FORTRAN subroutine to provide interface between ABC & language and/or utility programs

XABC (INSTR, OUTSTR)

Input INSTR →

"X"
"Y"

Pointers/Values
Argument Names or Blank
Algorithm Interface

ABC: general (FORTRAN) algorithm

XABC: FORTRAN subroutine to provide interface between ABC & language and/or utility programs

5/5/76
The Interface Language

FUNCTION fit(
    x/MATRIX/
    y/REAL/
)

if(NROW(x)!=LENGTH(y))
    FATAL(Number of observations in x and y must match)

STRUCTURE(  
    coef/REAL,NROW(x)/
    resid/LIKE(y)/
)

call lsfit(x,NROW(x),NCOL(x),y,coef,resid)
RETURN(coef,resid)
END
# The Evolving Interface (to C, Fortran)

<table>
<thead>
<tr>
<th>Version of S</th>
<th>Interface</th>
<th>Paradigm</th>
</tr>
</thead>
<tbody>
<tr>
<td>Versions 1 and 2</td>
<td>Interface Language</td>
<td>A language to program an S “function” (arguments &amp; other data); implemented by compilation into a Fortran subroutine.</td>
</tr>
<tr>
<td>(1976-1988)</td>
<td>.C, .Fortran</td>
<td>Calls to subroutines in C or Fortran that are “independent” of S structures, though limited in the types of arguments accepted.</td>
</tr>
<tr>
<td>Version 3 and later, and R</td>
<td>.Call, .External</td>
<td>Calls to subroutines in C that manipulate arguments and create data as S objects, via C macros and utilities.</td>
</tr>
</tbody>
</table>
Interfaces for high-performance

• Goal: to get close to machine speed for critical computations and/or to use existing code in C, Fortran, C++.
• Current interfaces are limited and/or tricky to use.
• We never returned to an “interface language” customized for R.
• Enter Rcpp. A new interface language?
Rcpp Interface from R to C++

- A CRAN package written by Dirk Eddelbuettel, Romain François & others.
- Goal: to use functions and classes from C++ in R.
- C++ code is simpler, less error-prone than C for .Call().
- Large and growing collection of R-like expressions that still have C++-level efficiency.
- C++ techniques to further improve speed.
- Potential interface to general C++ classes.
Example: Function in C/C++

Discrete Convolution:

\[ z_i = \sum_j x_j * y_{i-j} \]

From *Writing R Extensions* manual.

Hard to write efficiently in R, better in C or C++.

The challenge is to interface the C or C++ code to R, and also to make the code reliable.
C code for the `.C()` interface

```c
void convolve(double *a, int *na, double *b, int *nb, double *ab) {
    int i, j, nab = *na + *nb - 1;

    for(i = 0; i < nab; i++)
        ab[i] = 0.0;
    for(i = 0; i < *na; i++)
        for(j = 0; j < *nb; j++)
            ab[i + j] += a[i] * b[j];
}
```

- Natural in C (almost) but requires significant preparation in R for the interface.
- Tends to be space-inefficient.
- Limited data types: does not generalize to more complex data structures and computations.
C code for the `.Call()` interface

```c
#include <R.h>
#include <Rdefines.h>

SEXP convolve2(SEXP a, SEXP b)
{
  int i, j, na, nb, nab;
  double *xa, *xb, *xab;
  SEXP ab;

  PROTECT(a = AS_NUMERIC(a));
  PROTECT(b = AS_NUMERIC(b));
  na = LENGTH(a); nb = LENGTH(b); nab = na + nb - 1;
  PROTECT(ab = NEW_NUMERIC(nab));
  xa = NUMERIC_POINTER(a); xb = NUMERIC_POINTER(b);
  xab = NUMERIC_POINTER(ab);
  for(i = 0; i < nab; i++) xab[i] = 0.0;
  for(i = 0; i < na; i++)
    for(j = 0; j < nb; j++) xab[i + j] += xa[i] * xb[j];
  UNPROTECT(3);
  return(ab);
}
```
• The C code using R-related macros is more complicated and (worse) error-prone.

• It does improve efficiency, if it works.

• In principle any R object can be used or constructed, but the C code to do so becomes impractical quickly.

• Enter Rcpp
C++ code for `.Call in Rcpp

```c++
#include <Rcpp.h>
using namespace Rcpp;

RcppExport SEXP convolve3cpp(SEXP a, SEXP b) {
    NumericVector xa(a);
    NumericVector xb(b);
    int n_xa = xa.size(), n_xb = xb.size();
    int nab=n_xa+n_xb-1;
    NumericVector xab(nab);

    for(int i=0; i<n_xa; i++)
        for (int j = 0; j < n_xb; j++)
            xab[i + j] += xa[i] * xb[j];

    return xab;
}
```

- **Types** (NumericVector, NumericMatrix) are directly related to R, as in the Interface Language!
- But, in C++ the type space is totally extensible.
- **Rcpp** also interfaces to C++ classes.
- Many features for efficiency or R-like expressions.
R and Object-Oriented Programming

- **Functional Methods and Classes.** Methods are an extension of function definitions; flexibility is added while the user continues to use (generic) functions. (R)

- **Methods Encapsulated in Classes.** Methods belong to classes of objects; persistent objects are used and modified by invoking methods on the object. (C++, Java, Python, ...; and now R)

Two complementary programming paradigms; sometimes one is clearly better, sometimes either works, with different goals.
Example: Statistical Models with OOP

Functional Methods and Classes.

```r
myModel <- fitModel(data, glmModel(formula), ...)  
# Dispatch on both data and specification
qqnorm(residuals(myModel))
```

Methods Encapsulated in Classes.

```r
myModel <- modelFitter$new(data)  
# Saves a reference to the data
myModel$model <- glmModel(formula)
newFit <- myModel$fit()
```

Either paradigm works well here, emphasizing either functionality or interaction.
Reference Classes for Object-Oriented Programming in \texttt{R}.

- Reference classes are defined in \texttt{R} with fields and methods:

  ```r
  modelFitter <- setRefClass("modelFitter",
    fields = c("data", "model"),
    methods = .......)
  ```

- As with \texttt{C++}, etc. (but unlike earlier \texttt{R}) objects use references, and methods are stored in the class:

  ```r
  myModel <- modelFitter$new(data)
  myModel$model <- glmModel(formula)
  newFit <- myModel$fit()
  ```
Reference Classes for Object-Oriented Programming in \textbf{R}.

- Reference classes behave like classes in \texttt{C++}, \texttt{Java} and many other languages--NOT like \texttt{R} classes.
- But, you can use these classes and program for them with ordinary \texttt{R} expressions and functions.
- An extension to core \texttt{R} (from October, 2010)
- In addition to pure \texttt{R} programming we can embed interfaces to \texttt{C++}, etc. in reference classes to extend them smoothly in \texttt{R}.
- \texttt{Rcpp} is the first example of such an interface.
Planning computing projects to include interfacing to other systems is part of the tradition of S and R, and may be essential for challenging projects.

Recent extensions promise to allow more natural programming in the idiom of standard OOP languages.

In particular, a new interface to C++ may bring us a more natural language for efficient implementations (home-crafted or imported).