A Pascal-based Approach towards Statistical Computing

F. Katritzke[1] W. Merzenich[1]

R.-D. Reiss[2] M. Thomas[2]

[1] Department of Electr. Eng. and Computer Science, University of Siegen

[2] Mathematical Department, University of Siegen

September 16, 2002

Abstract

Most statistical programming languages (like S, R or XploRe) are dynamically typed and employ an interpreter for program execution. However, the interactivity supported by that approach is usually not required for libraries which are not modified by the user. We discuss the related performance problems and suggest the usage of a strongly typed language that includes the data manipulation facilities provided by statistical languages.

We illustrate our approach by comparing the coding techniques and runtime performance of our prototypical implementation of a Pascal based statistical language with a traditional one. An outlook on a client/server environment is given where components written in our language can be utilized in various frontends.

1 Introduction

Statistical analyses require the use of a programmable environment if tasks beyond the limited facilities of a menu system must be performed. Common general purpose languages like C or Fortran are inconvenient for implementing statistical algorithms which often require the manipulation of complete data sets. Statistical languages, like S [1], R [4] or XploRe [5], have been developed with the following characteristics:

- The handling of data sets with arbitrary lengths is supported whereby memory management is automatically performed.
- Operators and functions can be applied to complete data sets without requiring explicit loops.
- Parts of data sets can be extracted and easily manipulated.

- These languages are generally applicable, i.e., they provide the usual control structures and subroutine mechanisms of general purpose languages.

Statistical languages are usually scripting languages offering the user an interactive dialog allowing commands that are interpreted and immediately executed. Subroutines (macros) are implemented by using the same statements that are employed in an interactive analysis. Chambers [2] points out that the resulting smooth migration path from interactive usage to serious programming is important to reduce the start-up costs for new users.

Because it is not necessary to declare types of variables and formal arguments of subroutines, these languages are often called typeless. This approach can be better characterized as "dynamically typed", because each value in these languages is associated with a data type. However, any type checking is postponed until it is actually required and the type of a variable may change during the execution of a program.

Such a behavior is necessary in interactive languages, because it would be unacceptable for a user to declare all variables before starting a session. There are also some advantages when subroutines are implemented without declaring types of formal parameters and results. As a simple example, we consider the generic function for adding numbers in the S/R notation, namely,

```
add <- function (x, y) x + y.
```

This function can be used to add values of any type, for which the `+`-operator is defined. However, more complex algorithms usually require an explicit check of the data types of the actual arguments. Thus, true generic functions are usually possible in special cases only. One should note that the templates of the strongly typed C++ language provide similar facilities.

2 Problems with Interpreted Languages

In the preceeding lines, we described the advantages of interpreted, dynamically typed languages, yet one should be aware of two serious drawbacks. Firstly, these languages only perform semantic checks (e.g., for the existence of identifiers) when the code is actually executed which leaves many possibilities for typing errors to be undetected.

Secondly, type checks for each operation must be performed at run-time. This must be done not only once, but every time an instruction is executed. When performing simple operations like the addition of two real numbers, most of the time for the operation will be wasted on verifying the types of the arguments, even when it is clear that — because of the design of the algorithm — no other types may occur. While the additional overhead is negligible for interactive execution of commands typed on a command line, it has a serious impact on routines that require intensive calculations.

Interpreted statistical systems usually offer the inclusion of external subroutines written in a compiled language like C or Fortran as a solution to that

problem. Such an approach is inconvenient in various ways: a recompilation of a module is required for every target platform. Also, the developer of a module must be familiar with technical details like parameter passing conventions and the usage of host-specific development systems. Error handling is problematic. In addition, general purpose languages are not well suited for performing statistical computing, which is the main reason for using a special statistical language in the first place.

Ousterhout [10] distincts between high-level scripting languages and low-level system programming languages. He suggests to use scripting languages to combine components that are implemented in a system programming language. However, there is not yet a system programming language suited for statistical computing.

Therefore, we implement a strongly typed language that is based on a general purpose language (namely, Pascal) with the extensions required for statistical computing. Our language provides vector and matrix operations like other common statistical languages. Moreover, all variables, parameters and return types of subroutines must be declared. As a result, the language can be compiled and executed efficiently. Such a language cannot be applied as an interactive immediate language for command line execution, but it serves well for the implementation of statistical components.

In the next section, we outline some details of our language, called StatPascal, and measure its runtime performance. Then, we describe shortly the design of a prototype of a component integrating environment where StatPascal components are used within a menu system or a graphical programming environment. This allows their visual combination, as alternatives to a textual scripting language.

3 StatPascal

In this section, we describe some basic concepts of StatPascal and its implementation and compare it with other statistical languages.

3.1 Overview of StatPascal

Because a statistical programming language should provide a complete language kernel with only a few extensions that are specific for statistical computing, see Huber [6], it is natural to base a new statistical language on an existing general purpose language. We used Pascal [7] because of the clearness of its syntax and the simplicity of its compiler. StatPascal supports most of the standard Pascal constructs including a unit concept for modular programming.

However, standard Pascal is not well suited for dealing with data sets making some extensions to the language necessary. In particular, we introduce two new data structures called vector and matrix, which may be considered as one- and two-dimensional arrays. In contrast to the standard array structure there is no need to declare a maximum size. These structures can also be used in arithmetic

expressions and as arguments and return types of functions. The example in the next section shows their applicability in vector-oriented techniques that are similar to the ones present in other statistical languages.

The StatPascal compiler generates a code for an abstract stack machine which is interpreted at runtime. Such an approach has been frequently used in Pascal implementations [8]. It has the following advantages:

1. The compiler and the runtime environment can be easily ported to another platform.
2. The resulting binaries are not system dependent.
3. The language can be easily included into a host system. Access to the host system may be implemented by introducing some predefined functions generating special operation codes for the abstract machine. An object-oriented implementation facilitates the required enhancements.

Of course, it would be possible to generate a machine code for the host processor or to output source code of a system programming language like C. However, because vector and matrix operations are performed by single instructions of the virtual machine, a reasonable runtime performance is already achieved for vector oriented programs. For example, the StatPascal expression

```
y := log (rev (sort (x)))
```

which yields the logarithms of all values in a vector in descending order, is translated to just a few instructions of the virtual stack machine:

```
loadval     -6, 1
vload               ; load x on stack
vfsrt               ; sort
vrev                ; rev
vlog                ; log
loadptr     -7, 1
vstore              ; store y from stack
```

Although we do not present further details of the virtual machine, one can see that the nested vector expressions may be evaluated in-place on a calculator stack facilitating an efficient execution. Nevertheless, performance is unsatisfactory if the vectorized extensions are not used.

We now demonstrate the similarities between StatPascal and other systems (in particular, R) and measure their performance.

3.2 StatPascal and Other Systems

We implemented the Hill estimator (an estimator for the reciprocal shape index of the tail of a distribution) in R/S, StatPascal and C. The Hill estimator is given by

$$\hat{\alpha}_{n,k} = \frac{1}{k}\sum_{i=1}^{k} \log \frac{x_{n-i+1:n}}{x_{n-k:n}}, \quad k = 1, \ldots, n-1,$$

where $x_{n-k:n} \leq ... \leq x_{n:n}$ are the $k+1$ largest values of data $x_1, ..., x_n$.

The following implementations expect a univariate data vector $(x_1, \ldots, x_n)$ and return a vector of the Hill estimates $(\hat{\alpha}_{n,1}, \ldots, \hat{\alpha}_{n,n-1})$. We start with the R/S version.

```
hill <- function(x) {
  x <- log(rev(sort(x[x > 0])))
  n <- length(x)
  cumsum(x)[ - n]/(1:(n - 1)) - x[-1]
}
```

The StatPascal equivalent is given next.

```
function hill (x: realvector): realvector;
var n: integer;
begin
  x := log (rev (sort (x [x > 0])));
  n := size (x);
  return (cumsum (x)) [1..(n-1)] / (1..(n-1)) - x [2..n]
end;
```

The C version was written carefully eliminating loops whenever possible. It consists of 78 lines, including an implementation of the quicksort algorithm.

One can see that R and StatPascal provide similar vectorized operations. Especially, no explicit loop is required to evaluate the estimator. There is a significant difference in runtime performance. Table 1 shows the execution times for 10000 calls to the Hill estimator, applied to a newly simulated data set in each call (we used R 1.4, StatPascal and the GNU C compiler version 2.95.3 with optimization `-O3` under Linux). The runtime for StatPascal also includes the compilation of the program, while R had already been started and parsed the routines.

Sample Size	R	StatPascal	C	StatPascal (loops)	Pascal
20	2.8	0.26	0.14	2.3	0.17
50	4.08	0.55	0.35	6.5	0.45
100	6.28	1.1	0.71	14.0	0.94
500	26.7	5.4	3.7	82.3	5.0
1000	54.9	10.9	7.7	175	10.5

Table 1: Time (in seconds) for 10000 evaluations of the Hill estimator.

StatPascal clearly outperforms the R system, at the cost of only a small declarative overhead in the implementation of the estimator. The increased

performance is important for long running simulations and interactive visualizations. However, the performance of the virtual machine is bad if the vector oriented extensions are not used. The last two columns of Table 1 show the execution time of an unoptimized standard Pascal implementation of the Hill estimator (not utilizing any vectorized expressions), under StatPascal and the GNU Pascal compiler.

4 A Component Integrating Statistical Environment

As a descendant of a classical system programming language, StatPascal is not suited for interactive use. The combination of StatPascal components by using an interactive operating system shell seems unattractive as a statistical environment. We therefore need some other means to provide interactivity. In this section, we present a prototype of a client/server based component integrating statistical environment (called Risktec) where StatPascal programs are utilized.

We use CORBA [9] as a middleware to manage the communication between clients and servers. The system is, therefore, open for the provision and usage of components in other softwares. Figure 1 shows an overview of the structure of our environment.

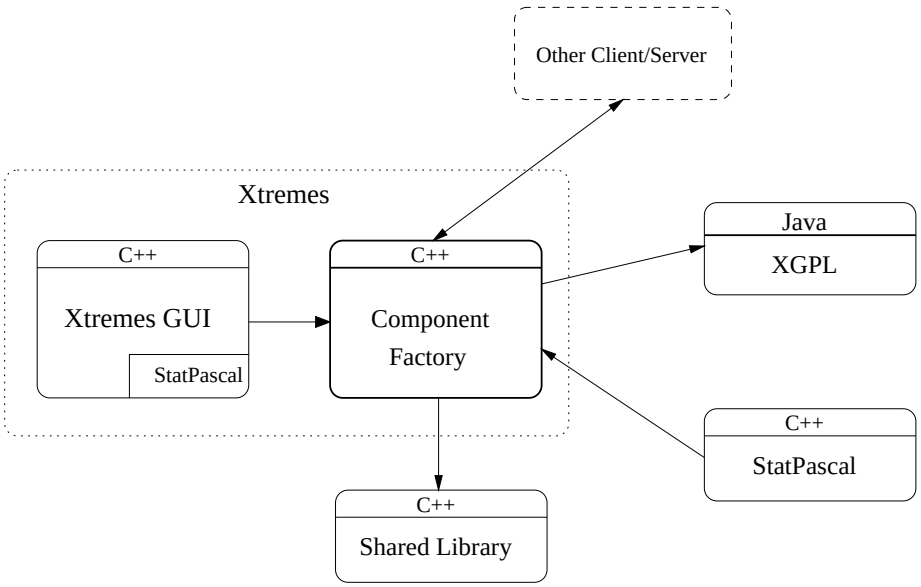

Figure 1: Client/Server structure of component integrating statistical environment.

One recognizes two usages of StatPascal. Firstly, it is included within the

host system Xtremes [11], a menu oriented statistical software. Interactive StatPascal programs can be used to extend the Xtremes system.

Secondly, the Xtremes system exports a component server (consisting of a factory managing prototypes [3]) where StatPascal components can be registered and produced, besides components provided by Xtremes.

It seems important to hide the complexities of CORBA from users of the system. Within the Risktec environment, there is currently only a single interface definition for the components. They basically implement a function $f : S_1 \times \cdots \times S_n \to T_1 \times \cdots \times T_m$ and provide methods to inform a client about the number and data types of the arguments and results. Such a minimal interface serves well for numerical computations. For details, the reader is referred to the project homepages `http://www.statpascal.de` and `http://www.risktec.de`.

The implementation of a StatPascal server component is straightforward. A special program header defining the name of the component and its parameters is required. For example, the Hill estimator is turned into a CORBA server by using the following header:

```
component hillserver;
inports  x: realvector;
outports alpha: realvector;
...
```

The graphical programming language XGPL allows the combination of these components by building a dependence graph and using an interactive graph editor, see [12] and [13] for details. Moreover, a shared library (DLL) provides access to the components by means of a procedural interface, allowing the usage of StatPascal components from systems like R or MS Excel.

5 Conclusion

By combining vectorized operations from statistical programming languages and strong typing from system programming languages, one can create a statistical language with a good runtime performance. Such a language is well suited to implement statistical components which may be combined using menu oriented systems, interactive textual scripting languages or graphical programming environments.

References

[1] Becker, R.A., Chambers, J.M. (1984). *S: An Interactive Environment for Data Analysis and Graphics.* Wadsworth, Monterey, California.

[2] Chambers, J.M. (1998). Computing with Data: Concepts and Challenges. Technical Report, Bell Labs.

[3] Gamma, E., Helm, R., Johnson, R. and Vlissides, J. (1995). *Design Patterns.* Addison-Wesley, Reading, Massachusetts.

[4] Gentleman, R., Ihaka, R. (1997). The R Language. In: Billard, L., Fisher, N.: *Proceedings of the 28th Symposium on the Interface.* The Interface Foundation of North America.

[5] Härdle, W., Klinke, S., Müller, M. (2000). *Xplore Learning Guide.* Springer, Berlin.

[6] Huber, P.J. (1994). Languages for Statistics and Data Analysis. In: Dirschedl, P., Ostermann, R.: *Computational Statistics.* Physica, Heidelberg.

[7] Jensen, K., Wirth, N. (1974). PASCAL - User Manual and Report. Springer.

[8] Nori, K.V., Ammann, U., Jensen, K., Naegeli, H.H., Jacobi, Ch. (1981). Pascal P implementation notes. In: Barron, D.W.: *Pascal – The Language and its Implementation.* Wiley, Chichester.

[9] Object Management Group (1995). The Common Object Request Broker: Architecture and Specification.

[10] Ousterhout, J.K. (1998). Scripting: Higher Level Programming for the 21st Century. IEEE Computer. March 1998.

[11] Reiss, R.-D., Thomas, M. (1997). *Statistical Analysis of Extreme Values.* Birkhäuser, Basel.

[12] Thomas, M., Reiss, R.-D. (2000). Graphical Programming in Statistics: The XGPL prototype. In: Decker, R., Gaul, W. *Classification and Information Processing at the Turn of the Millenium.* Springer, Berlin.

[13] Thomas, M. (1998). *A Statistical Programming Language for Extreme Value Analysis.* PhD-thesis, University of Siegen.