The open source RFortran library for accessing R from Fortran, with applications in environmental modelling

M. Thyer, M. Leonard, D. Kavetski, S. Need, Benjamin Renard

The open source RFortran library for accessing R from Fortran, with applications in environmental modelling

Mark Thyer [1]*, Michael Leonard [2], Dmitri Kavetski [1], Stephen Need [2], and Benjamin Renard [3]

1. School of Engineering
University of Newcastle
Callaghan, New South Wales 2308, Australia

2. School of Civil, Environmental and Mining
University of Adelaide
Adelaide, South Australia 5005, Australia

3. Cemagref Lyon
UR HHLY, Hydrology-Hydraulics
3 bis quai Chauveau
CP220, 69336 Lyon Cedex 09, France

* Corresponding Author:
Mark Thyer
Phone: +61 2 4921 6057
Fax: +61 2 4921 6991
Email: mark.thyer@newcastle.edu.au

Abstract

The open source RFortran library is introduced as a convenient tool for accessing the functionality and packages of the R programming language from Fortran programs. It significantly enhances Fortran programming by providing a set of easy-to-use functions that enable access to R's very rapidly growing statistical, numerical and visualization capabilities, and support a richer and more interactive model development, debugging and analysis setup. RFortran differs from current approaches that require calling Fortran DLLs from R, and instead enables the Fortran program to transfer data to/from R and invoke R-based procedures via the R command interpreter. More generally, RFortran obviates the need to re-organize Fortran code into DLLs callable from R, or to re-write existing R packages in Fortran, or to jointly compile their Fortran code with the R language itself. Code snippets illustrate the basic transfer of data and commands to and from R using RFortran, while two case studies discuss its advantages and limitations in realistic environmental modelling applications. These case studies include the generation of automated and interactive inference diagnostics in hydrological calibration, and the integration of R statistical packages into a Fortran-based numerical quadrature code for joint probability analysis of coastal flooding using numerical hydraulic models. Currently, RFortran uses the Component Object Model (COM) interface for data/command transfer and is supported on the Microsoft Windows operating system and the Intel and Compaq Visual Fortran compilers. Extending its support to other operating systems and compilers is planned for the future. We hope that RFortran expedites method and software development for scientists and engineers with primary programming expertise in Fortran, but who wish to take advantage of R's extensive statistical, mathematical and visualization packages by calling them from their Fortran code. Further information can be found at www.rfortran.org.

1. Background and motivation

1.1. Fortran

The Fortran programming language is well established within the scientific and engineering communities because of its computational speed, mathematical support and memory efficiency. Considerable Fortran-based code has been developed over the last 40 years and it remains widely used in contemporary scientific and engineering software. In particular, many environmental models are written in Fortran, including the Australian Community Climate and Earth-System Simulator (ACCESS) [*Abramowitz et al.*, 2007], the Princeton Ocean Model [*Giunta et al.*, 2007] and MODFLOW-2005 [*McDonald and Harbaugh*, 2003].

Since its now-dated 1977 standard, Fortran has undergone major revisions in 1990 and 2003, extending its parallel computing capabilities, array handling and introducing elements of object-oriented programming. However, Fortran has notoriously limited support for visualization and interoperability with other languages and applications (though Fortran-2003 implements interoperability with C). Often, Fortran users will write intermediate and final results to text files and post-process them, manually or automatically, using other software (e.g., Excel, Matlab). This can be exceedingly tedious and time-consuming, especially in large-scale applications, and may be impractical for real-time interactive visualization and debugging. Other Fortran programmers will develop prototype software in visual languages such as Matlab and IDL and then port this code to Fortran. This duplication may be time-inefficient and error-prone.

Given the proliferation of programming environments, mixed-language paradigms offer appealing opportunities for exploiting algorithm developments in other platforms. For example, the widely used Matlab software supports calling Fortran DLLs, and also makes its computational engines available to Fortran and C codes using Application Programming Interface (API) utilities [*Mathworks*, 2007]. In this paper, we introduce a software solution enabling Fortran programs to access the R software platform, overcoming traditional drawbacks of Fortran in visualization and exploiting common and cutting-edge algorithm packages available in R.

1.2. The R project

R (www.r-project.org) is an open-source programming environment increasingly popular in the statistical and mathematical communities [*Vance*, 2009; *R Development Core Team*, 2010]. It can be operated in interactive command prompt and scripted modes, and has extensive graphic capabilities of suitable standard for scientific reports and publications.

An attractive feature of R is the ability of users to contribute packages implementing their work: a large and rapidly growing number of packages implementing recent advances in statistics, computation and visualization are now available. It is natural for programmers in other languages to wish to exploit these capabilities. In particular, inter-system interfaces allow languages such as Java, Python, Octave and others to be called from the S language (on which R was modelled) and, importantly, vice versa [*Lang*, 2005]. Similarly, the RExcel package [*Baier and Neuwirth*, 2007] allows invoking the R functionality from the widely used Microsoft Excel software, while Rpy (http://rpy.sourceforge.net/) provides a simple and efficient access to R from the Python language. Here, we develop an analogous connectivity to R to enhance the Fortran programming language.

1.3. Current interoperability of R and Fortran

Currently, the interoperability between R and Fortran is limited to R calling Fortran code compiled as dynamic link libraries (DLLs). Indeed, many of R's computationally intensive tasks are implemented using DLLs, typically in Fortran or C. It is also common for researchers to extend R's core functionality with DLL packages of their own [*Lemmon and Schafer*, 2005]. By using DLLs, an R-based program can exploit computationally fast Fortran implementations of numerical models and algorithms, and integrate them within R's appealing scripted visual environment.

In many circumstances, the DLL-based strategy is convenient and efficient, e.g., when the Fortran routines do not require the communication of large amounts of data and variables through the DLL interface, or when the primary expertise of the programmer is in R, or when Fortran represents a comparatively small component of the overall project. However, successfully using DLLs requires familiarity with often-finicky calling and naming conventions [*Lemmon and Schafer*, 2005].

Moreover, Fortran-90 derived types, pointers and deferred-shape arrays are not readily transferrable across mixed-language DLL interfaces. Finally, the very basis of the R-centred DLL-paradigm, which in this case requires a Fortran programmer to significantly alter and re-coordinate their code in order to work with R, may be unduly restrictive in many common situations.

For example, it is common for researchers and practitioners in applied science and engineering to want to quickly evaluate or incorporate algorithms for some specific task, often within a much larger existing code that cannot be readily modified. The code itself may not be amenable to bundling into DLLs, often due to poor legacy design, or because it may consist of a large body of routines spread across multiple libraries, with nontrivial data communication patterns (in particular, using Fortran derived types, pointers and deferred-shape arrays, etc), multiple input and output files, modules written in other languages, user interaction via GUIs, etc.

The DLL approach is especially convenient, and was originally designed, for dedicated software developers distributing code to others, and assumes that sufficient resources are available for code design and development in DLL form. This is not always the case in academic contexts, in particular for postgraduate and postdoctoral researchers.

While a modern scientist and engineer will often spend a major portion of their time programming, it is worth distinguishing between method development versus software development. The former generally aims to create novel methods and models using advanced scientific and engineering insights, whereas the latter generally aims to implement mature approaches in versatile distributable software. In method development, the algorithms are often changing, and so are procedure arguments, interfaces and visualization requirements. Given that many trialled methods may be abandoned, key priorities are computational speed and developmental flexibility, and, importantly, familiarity of the researcher (whose primary expertise is, e.g., in hydrological modelling) with the programming environment.

In many of these situations, treating the Fortran program as a subunit of the R program can be restrictive, inefficient or simply infeasible. Moreover, the required effort and expertise may be discouraging to a practitioner whose primary interest and expertise are in environmental modelling

rather than mixed-language software engineering. Instead, consider the alternative paradigm underlying RExcel and the Fortran APIs in Matlab. A key aspect of these software solutions, from the point of view of an Excel and Fortran programmer, is that they are centred on the software that the latter are familiar with. Given the rapidly growing functionality and popularity of R, it is natural for Fortran users to seek to couple R and Fortran, but in a way that does not require re-organization of the Fortran code and/or switching to R to coordinate the Fortran modules. Note that we do not suggest abandoning sound programming principles developed in software and computer sciences, rather we seek to enable a considerable section of the environmental modelling community to follow these principles to enhance their language of choice. This is the key objective of RFortran.

1.4. RFortran

RFortran is a suite of Fortran-95 modules that allow accessing the full functionality of R, including its statistical and graphical packages, directly from Fortran. It is built on the rcom and statconnDCOM interface also used in *RExcel* [*Baier and Neuwirth*, 2007; *Baier*, 2009]. When using RFortran, the R program becomes a subunit of the Fortran program and R components can be called as they are needed. RFortran provides a set of easy-to-use functions to transfer data between R and Fortran, and execute R commands from Fortran. Using these functions requires minimal modification of the original Fortran source, and the user is only required to know the R syntax for the R functionality of interest. This paper introduces RFortran and illustrates its capabilities using examples ranging from simple code snippets to real environmental modelling applications where RFortran significantly expedited research deliverables. We also outline our implementation of RFortran and discuss advantages and limitations vis-à-vis potential alternatives.

1.5. Outline of the presentation

This paper is organised as follows. Section 2 describes the core functionality of RFortran and illustrates its use with simple code snippets. Section 3 outlines the software implementation of RFortran (readers primarily interested in applications may wish to skip this section). Section 4 summarizes the key capabilities of RFortran. Next, two real-life environmental modelling

applications of RFortran are presented: Section 5 describes the generation of automated and interactive calibration diagnostics for statistical inference in hydrological modelling, while Section 6 outlines the integration of the R copula package into a Fortran program for probabilistic modelling of coastal flooding. The advantages and limitations of RFortran vis-à-vis traditional and other alternatives are discussed in Sections 7-8, while future enhancements are discussed in Section 9, followed by conclusions in Section 10.

2. Using RFortran

2.1. Core functionality

RFortran is operated using six core functions: `Rinit`, `Rclose`, `Rput`, `Rget`, `Reval` and `Rcall`. All RFortran functions return an integer error flag (with non-zero values indicating error conditions).

`ok = Rinit([RGuiVisibleIn])` - initialises RFortran: it loads R, initialises the COM architecture and the error/warning message log (see section 2.3) There are several optional arguments for `Rinit` (not listed here) that provide the user with greater control of the Fortran – R interaction. These include auto-loading of R scripts in a given file path and setting the R working directory. Here, only the argument governing the "visibility" of R will be described (a complete description is provided at http://code.google.com/p/rfortran/wiki/UsingRFortran#Rinit)

The optional logical argument `RGuiVisibleIn` selects between a (default) visible Rgui console (`RGuiVisibleIn=TRUE`) and an invisible R instance (`RGuiVisibleIn=FALSE`). During the debugging stage, the visible Rgui would normally be used, allowing variables transferred to/from R to be viewed in the Rgui console. However, since multiple Fortran executables access the same visible Rgui instance, this can lead to potential conflicts between variables passed to R by different Fortran executables. Consequently, in operational setups, the invisible R mode would be preferred, where each Fortran executable generates its own independent R instance with a separate namespace.

The capability of having multiple visible Rgui's is currently not supported, but may be implemented in future work.

`ok = Rclose()` – closes the RFortran connection and disconnects R from the Fortran program. If RFortran is in visible mode, the Rgui console and all open plot windows will remain open. If RFortran is in "invisible" mode, this step is necessary to ensure that the invisible R instance is closed and its associated resources released (otherwise re-running the Fortran executable will keep creating additional R instances that will use system resources unnecessarily).

`ok = Rput('R_var',F_var,[mv],[status])` - transfers data from Fortran to R by creating or updating an R variable `R_var` to contain the value of Fortran variable `F_var`. While not strictly necessary, using identical variable names in Fortran and R may improve readability and avoid unintended loss/duplication of data. The Fortran variable can be `integer(2)`, `integer(4)`, `real(4)`, `real(8)`, `logical(4)`, `character(*)`, a scalar or array of up to rank 3. Support for complex types and higher array ranks are planned. A literal constant can also be passed.

The optional argument `mv` is used to specify missing values in `R_var` when `F_var` is a vector/array. `mv` is a logical vector/array, with `.TRUE.` values indicating the elements of `R_var` to be set to `NA`.

Note that by default, Rput will overwrite the existing value of `'R_var'` in the R environment. If this is not desirable, the optional argument `status` can be used. Setting `status="old"` will return an error if `R_var` does not exist in the R environment, whereas `status="new"` will return an error and will not over-write `R_var` if it already exists. By default, `status="unknown"` and `Rput` will create or overwrite `R_var` depending on its existence. Since this behaviour parallels the semantics of the Fortran OPEN statement, its usage for error detection, handling and debugging will be familiar to Fortran programmers.

ok = Rget('R_var',F_var,[alloc],[rm]) - copies the value of the R variable `R_var` to the Fortran variable `F_var`. The same types and shapes of arguments can be used as with the `Rput` function. The Fortran variable provided to this function must match the type of data being requested and in the standard case arrays must have the correct size.

If the logical optional argument, `alloc=.TRUE.` and `F_var` is an allocatable array, it is sized to match `R_var`, otherwise an error is reported if the size of `F_var` does not match `R_var` (the default action if `alloc` is not present). If the logical optional argument, `rm=.TRUE.`, the variable `R_var` is removed from the R environment after it has been transferred to Fortran, otherwise it remains defined (default).

Finally, `Rget` can also handle R expressions, provided they result in an R variable that matches the Fortran variable, i.e.,

1. components of objects ('$' symbol), e.g., `Rget('a$statistic',scalar),`

2. sections of arrays ('[]' symbol), e.g., `Rget('x[3:10]',array),`

3. results of functions ('()' symbol), e.g., `Rget('length(x)',scalar),`

4. arithmetic expressions ('+/-*' and other symbols), e.g., `Rget('1+2',scalar),`

5. all of the above, e.g., `Rget('arima(rnorm(50),c(1,0,0))$coef[1]',scalar)`

`Rput` and `Rget` have additional optional arguments supporting missing values, error handling of type mismatches and invalid values (`NA, NaN, Inf,` etc) (see www.rfortran.org for details).

ok = Reval('command_string')- sends the character string "command_string" as command to the R interpreter, e.g., `Reval("x=1")` and `Reval("plot(a,b,xlab='x')")`.

ok = Rcall (vars, cmds)- transfers multiple variables and commands to R. The first argument, `vars` is an integer vector, and can be used to pass multiple variables to R using a series

of Rput function calls. The second argument, `cmds`, is a character vector of commands to be executed in R (usually using the variables passed by `vars`). For example, transferring the Fortran variables, `xdata` and `ydata`, plotting them and then removing them can be undertaken using the following:

```
Rcall(vars=(/Rput("x",xdata),Rput("y",ydata)/),&
      cmds=(/cl("plot(x,y),cl("rm(x,y)")/))
```

The `cl` function is simply a Fortran function to ensure each of the character strings of the character vector `cmds` have a common length and is used to avoid a compiler warning for violating the Fortran 95 standard.

The `Rcall` function is more efficient then using multiple `Rput/Reval` commands to transfer multiple variables and commands to R, because it provides internal error checking of each of the commands and variables transferred to R – see example 4 in section 2.2.4 for an illustration.

2.2. Basic usage of RFortran

This section uses short code snippets to illustrate the basic functionality of RFortran. For brevity and clarity, and to allow a clearer focus on the key aspects of RFortran, some sections of the Fortran code have been omitted, e.g., checking of error codes (a single illustration of this is provided). For all tutorial examples, complete Fortran sources, R scripts, data files and project files for supported compilers are included as part of the RFortran installer available from www.rfortran.org.

2.2.1. Example 1 – Passing data to R and plotting a simple graph

Figure 1 shows the use of RFortran to produce a rainfall map. Line 2 enables access to RFortran routines. Lines 8-12 read average Australian annual rainfall data from a file (available from http://www.bom.gov.au/cgi-bin/climate/cgi_bin_scripts/annual_rnfall.cgi) into a Fortran array. Line 14 initialises the R server. Line 15 transfers the data array to R and uses Rput's optional argument mv to convert negative rainfall values to NA, used here to denote ocean pixels. The `Reval` function on line 16 invokes R to produce the plot shown in Figure 6 (a) (the next examples show

additional formatting options). Lines 17-20 check the error code and report any errors/warnings to the screen (see section 2.3 for further explanation). `Rclose` disconnects R from the Fortran code.

2.2.2. Example 2 – Passing data to R functions and returning results to Fortran

Figure 2 shows how to pass data from Fortran to R, fit a linear regression and return the output to Fortran (based on a similar example in *Maindonald and Braun* [see also 2007]). Here, the explanatory variable is the Southern Oscillation Index (http://www.bom.gov.au/climate/current/soihtm1.shtml), defined as the pressure ratio between Darwin and Tahiti and associated with fluctuations in hydrological variables in the Pacific Basin. The dependent variable is the average rainfall over the entire Australian continent (http://www.bom.gov.au/climate/change/rain02.txt). To save space and emphasize the differences, only changes in the code from the previous example are shown.

Following lines 7-10, which read the values from a data file into Fortran variables, line 11 initialises R, while line 12 transfers them to R. While here these datasets are small and could have been loaded directly into R, in general they could be much larger and/or represent the output of previously invoked Fortran procedures. On line 13, the linear model is fitted and stored in an R object. On lines 14 and 15, components of this object (the fitted slope and intercept parameters) and the residual time series are retrieved using the `Rget` function. Having returned these values to Fortran, they can be used in further computations as shown on line 18.

Note if the regression diagnostics (beyond the scope of this simple illustration) indicate that the assumption of independent Gaussian errors is poor, this procedure can be replaced with more sophisticated R procedures - see *Maindonald and Braun* [see also 2007].

2.2.3. Example 3 – Using Fortran wrappers for a series of RFortran/R commands

To avoid cluttering the Fortran code, repeated sequences of the same `Rput/Rget/Reval` commands can be wrapped in dedicated Fortran procedures. This is illustrated in Figure 3(a), where the `Rimage` Fortran wrapper function is used on line 14 to plot the map of average Australian

rainfall used in Example 1. Line 14 uses two additional arguments: (i) `cmd` supplies additional commands, here, axis labels and the breaks used for the colours of the image plot are demonstrated, and (ii) `mv` (missing value) provides a logical mask indicating the elements of the Fortran array to be denoted as missing values (`NA`) in R (here, missing values represent ocean pixels). The `Rimage` Fortran wrapper is shown in Figure 3(b). In this function, the arguments x, y and z are passed to R (lines 9-11), plotted in lines 12-16 and removed from R in lines 17. The resulting R plot is shown in Figure 6 (c) and represents a considerable improvement over Figure 6 (a).

Note that wrappers such as `Rimage` form part of the extended functionality of RFortran, as distinct from core functionality such as Rput. The file `RFortran_Rplots.f90` contains wrappers for the following R functions: `plot`, `plot.filled.line`, `points`, `lines`, `persp`, `image`, `contour`, `filledcontour`, `colpersp3d`, `legend` and `pairs`. More wrappers will be implemented as RFortran continues growing in the future.

2.2.4. Example 4 – Using R scripts and Rcall for a series of RFortran/R commands

Sequences of R commands can be placed in dedicated scripts and invoked from Fortran using a single `Reval` call, e.g., `ok = Reval('script_name(arg1,arg2)')`. Scripts must first be loaded using `Reval("source('myscript.R')")` or the dedicated RFortran function `LoadRscripts()` specifically designed for loading scripts. Note that the Fortran program calling the R script via `Reval` waits for the script to execute before resuming control.

The approach of calling Rscripts is illustrated in Figure 4, using a similar example as example 3. The R function `plot.australia.image` (defined in the script of the same name) (Figure 4(b)) enhances the `image()` plot with contours, coastlines, capital cities, etc., to produce a high quality plot (Figure 6(d)). The Fortran main program in this example (Figure 4(a)) uses the RFortran function `LoadRscripts` (line 15) to load the R script, (`plot.australia.image.r`), while lines 17-19 transfer the data to R with a series of `Rput` function calls and the script is executed using `Reval` on line 20. The final plot is shown in Figure 6

(d) and illustrates the power and versatility of R, where just a few commands can be used to produce high quality, specialized graphics of publication quality.

Using R scripts with RFortran is flexible and powerful because they are easier to develop, debug, execute and modify than transferring individual R commands using multiple `Reval` commands. For example, producing a high-quality graph usually requires altering scales, legends, formatting symbols and lines, etc, and this is easier to achieve using scripts. As the script can be developed independently of RFortran it will be easier to debug than multiple `Reval` commands and can also be re-used outside RFortran in other R applications. Furthermore, modifying an R script called using RFortran does not require re-compiling the Fortran code (provided the argument list of the script does not change), whereas altering the `Reval` calls within a wrapper does.

The final part of the Fortran main program in Example 4 (Figure 4(a)) demonstrates the use of the `Rcall` function (line 22). This `Rcall` passes the data to R, opens a new window, executes the script and removes the data in one Fortran function call. Using `Rcall` to transfer data and execute R scripts (or any other R command) is the most powerful and flexible way to easily transfer data to, and execute a series of commands in R. This saves having a series of `Rput/Reval` commands and obviates the need to check their return values (ok) for success/failure, since `Rcall` provides internal error checking of each R command executed.

Unless there are Fortran statements interspersed within the sequence of R commands, using R scripts and/or using `Rcall` is generally preferable to using sequences of `Rput/Reval/Rget` calls (such as in the Fortran wrapper, in Example 3).

Although not shown here, the most general use of RFortran is to call R scripts and/or `Rcall` from within Fortran wrappers, which pass Fortran data to R and remove it after the script has completed. Using Fortran wrappers around R scripts has the following advantages: (1) Transfer and removal of data required for the R script is part of the Fortran wrapper, which makes it easier to maintain and re-used in other parts of the Fortan code; and (2) The calling interface is fully Fortran-based and

therefore the programmer may not even be aware that R functionality is invoked within the Fortran procedures being called.

2.2.5. Example 5 – Using RFortran as an interactive debugging environment

RFortran can also be used for enhanced interactive debugging of Fortran programs. For example, intermediate and final results of Fortran functions being developed can be quickly passed to R and plotted for visual inspection and/or compared to analogous functions in R (if they exist). Example 5 in Figure 5 provides a simple illustration of this capability. Suppose a Fortran function to evaluate the Gaussian probability density (lines 22-27) is being developed. The program `Test_GaussianDensity` uses the Fortran function to evaluate the probability density over a range of values (-4,+4) (lines 8-11), then sends the data to R (line 13) and uses the R function `dnorm` to evaluate the probability density over the same range (line 14). On lines 15-17, the Fortran and R results are then plotted for comparison (see Figure 6(e)).

2.3. Error handling and debugging

All RFortran functions return integer error flags, using non-zero values to indicate error conditions. In addition, descriptive error messages are written to a log file and can be retrieved using the function `get_messages` (for Fortran console applications, this message is also automatically echoed to the screen).

When using `Rinit` errors generally indicate a problem communicating with R, usually because `rcom` and/or its dependencies have not been installed properly.

When using `Rput`, errors indicate a problem in copying the data from Fortran to R (see also Section 2.1). It is also straightforward to manually check that the data has been passed correctly to R, by typing the name of `R_var` at the R prompt started by `Rinit`.

When using `Rget`, RFortran flags an error if there are type, dimension or array-size mismatches between `F_var` and `R_var`, or if R values such as `NA`, `NaN` or `Inf` are being passed to Fortran.

When using `Reval`, if the command passed to R has syntax errors, or is syntactically correct but produces a runtime error, the return `ok` value is non-zero and the log file contains the message from the R console as would have been obtained using the R command `geterrmessage()`. The examples in lines 17-20 of Figure 1 illustrate the basics of error handling: the Fortran procedure calling `Reval` tests its return value, accesses the error log, echoes the message without unnecessarily crashing or halting the code execution.

A listing and description of all the error codes (including possible remedies) are given online at http://code.google.com/p/rfortran/wiki/UsingRFortran#Error_Codes.

When using RFortran, native debuggers can be (concurrently) used for each language, e.g., Intel Visual Fortran for the Fortran code, and the R `browser` function to debug R scripts. This does require the Fortran programmer to understand the R debugging environment.

2.4. System requirements and installation

RFortran interacts with R using the Component Object Model (COM) interface for R implemented in the *rcom* package [*Baier*, 2009] and also used in the RExcel software [*Baier and Neuwirth*, 2007]. This requires COM support, implemented in Intel Fortran (v 10 and above) and Compaq Visual Fortran (v 6.6) compilers, both under the Microsoft Windows operating system (XP and above).

While RFortran is essentially a suite of Fortran modules, it does require a separate installation to add the package *rcom* (and its dependencies) to the R program and register the COM server with the Windows registry [*Baier*, 2009]. An automatic installer is available from www.rfortran.org.

3. Structure and organization of RFortran

3.1. The COM architecture

Component Object Model (COM) is a standardized language-neutral mechanism for accessing objects and methods, including entire programming environments, under the Microsoft Windows operating system. It is very widely used in general and scientific software development, in

particular, Excel, Matlab, SigmaPlot and Scilab all expose their functionality using the COM interface.

In the case of RFortran, R is a server and the Fortran program is a client making requests to functions in the server via *rcom*. Each server has a unique programmatic identifier (the *ClsId*) stored in the computer's registry during the installation *rcom*. When RFortran is run, `Rinit` initialises the R server using COM-based utilities and its specific identifier.

RFortran passes all data between the R and Fortran units via the COM interface using variant objects implemented as Fortran-90 derived types. In COM the native type for arrays is SAFEARRAY. As Fortran arrays are incompatible with the SAFEARRAY type they first must be packed into a SAFEARRAY before being transferred to the variant object (see Canaimasoft [2000]). This ensures robust data transfer across the COM interface, but incurs a small computational penalty when packing/unpacking data in/out of SAFEARRAY objects (see section 7.1).

The `RFortran_COMinterface.f90` file contains the Fortran module *rdcom*, which passes/retrieves the Fortran variant object across the COM interface. Since the implementation and behaviour of pointers, strings, arrays and objects in Fortran is different than in the COM interface, *rdcom_f90* requires support routines provided by Intel and Compaq Visual Fortran compilers.

The *rcom* package handles the R-side of the COM interface.

3.2. Components of RFortran

RFortran comprises several components, as shown in Figure 7:

(i) COM interface: system-level components enabling the communication between R and Fortran using the COM facilities of the Fortran compiler and the *rcom* package;

(ii) Core functionality: generic Fortran functions `Rput`, `Rget`, `Reval` and `Rcall` for communicating data and sending commands to R, and `Rinit` and `Rclose` for opening/closing RFortran;

(iii) Extended functionality: Fortran wrappers for standard R tasks such as plotting, etc. It is envisaged that RFortran users and developers may contribute routines analogously to R packages.

(iv) Auxiliary routines (*MUtilsLib* folder) for error-handling, etc (see www.rfortran.org).

The arrows in Figure 7 show the dependencies between the various components of RFortran. Note that RFortran has been designed to "hide" all system-level communication, hence the Fortran user is not required to understand its advanced technicalities.

3.3. Linking with RFortran

To use RFortran, its source code (in the folders with grey names in Figure 7) must be included in the calling application. This can be done by: (i) jointly compiling the source files within the user project, or (ii) linking to a pre-compiled static library of RFortran. A DLL-based connectivity is also possible but not currently provided (see Section 9).

Fortran procedures accessing RFortran must include the Fortran statement `use RFortran` as indicated in Figures 1-4. This enables access to all RFortran routines.

4. Summary of key capabilities of RFortran

The major capabilities of RFortran to enhance Fortran programs using R are summarized below:

1. Interactive debugging environment: RFortran can be used for enhanced interactive debugging of Fortran programs. (e.g. Section 2.2.5)

2. Interactive run-time visualization, e.g., from inside iterative loops (e.g., Section 5);

3. Automated post-processing and visualization (e.g., Section 5);

4. Exploiting R computational toolkits in Fortran code (e.g., Section 6).

The next sections illustrate the use of these capabilities in two realistic applications in environmental modelling, aiming to reduce Fortran development time, automate visualization and debugging of intermediate results, and avoid error-prone recoding of R procedures in Fortran. Following these case studies, alternatives and limitations of RFortran are discussed (Sections 7-8).

5. Case Study 1: Diagnostics for Bayesian inference in hydrology

5.1. Background

The quantification of uncertainty in hydrological models and their predictions is a key challenge in scientific and operational hydrology. The Bayesian Total Error Analysis (BATEA) [e.g., *Kavetski et al.*, 2006] integrates hypothesized error models describing individual sources of uncertainty and provides a systematic methodology to hypothesize, infer and evaluate hypotheses regarding input, structural and output errors, and to make robust predictions [*Thyer et al.*, 2009]. The BATEA probability distributions are high-dimensional and require specialised Markov chain Monte Carlo (MCMC) sampling to accommodate recursive numerical solutions of nonlinear differential-equations underlying continuous-simulation hydrological models [*Kuczera et al.*, 2010].

5.2. Motivation for using Fortran and RFortran

The BATEA developers had considerable prior experience with the inference setup and existing Fortran-95 libraries implementing many of the data-processing and numerical-solution components needed for BATEA. These specialised components, the requirement for high computational efficiency (due to the mathematical structure and computational complexity of the hydrological model) and previous experience of the developers favoured an implementation in Fortran-95.

In general, Bayesian inference generates extensive output and requires monitoring to ensure the user's hypotheses are adequate. Since Fortran has limited graphical capabilities, RFortran was used within the BATEA Fortran code to communicate with R to document and visualize intermediate and final results, as well as to generate and visualize a large number of performance diagnostics. While the visualization could have been implemented by processing output files using software such as Microsoft Excel or Matlab, automated access to R's cutting-edge statistical graphics capabilities, as well as its open-source licensing, were seen as important practical advantages.

5.2.1. Interactive MCMC convergence diagnostics

MCMC diagnostics are necessary to monitor the convergence of the sampler [*Cowles and Carlin*, 1996]. Since MCMC convergence can be slow, both automated and interactive options are desirable to allow the user to make an informed decision when terminating MCMC sampling before formal convergence criteria are satisfied. Given several R packages designed specifically to compute and visualize MCMC diagnostics, it was natural to exploit this existing functionality. This was accomplished directly from the Fortran-based BATEA code using RFortran. An interactive option was also implemented to enable manual early termination of MCMC sampling

5.2.2. Automated Bayesian posterior diagnostics

In order to provide confidence in its results, statistical inference requires analysis of multiple diagnostics to scrutinize the assumptions and hypotheses underlying the model calibration and prediction (e.g., distributional assumptions of data errors, autocorrelation of residuals, etc). Again, the BATEA developers wished to use the several existing R packages for these diagnostic tests.

RFortran was therefore used to pass the data from BATEA to R, with the latter automatically producing and writing all diagnostic plots for a given calibration into a single report file in "portable document format" (pdf). In addition to computational convenience, this simplified and improved the documentation and comparison of calibration results and various diagnostics.

5.3. Outline of BATEA diagnostics

This section outlines the key diagnostics considered for the calibration of the GR4J rainfall-runoff model to the Horton catchment (Australia) [*Thyer et al.*, 2009]. In the language of Bayesian hierarchical inference, BATEA treated rainfall errors as "latent" (unobservable) variables. The diagnostics included distributional checks of the hypothesized multiplicative rainfall error model and the runoff error model derived from rating curve analysis [*Thyer et al.*, 2009].

The following diagnostics were generated by calling R functions from within the Fortran code:

1. Visualization of MCMC samples (Figure 8). The scatter plot matrix generated by R describes the marginal and bivariate joint distributions of all GR4J parameters and yields insights into the correlation structure. For example, Figure 8 shows that parameters x_2 and x_3 are notably correlated.

2. Diagnostics for latent variables and residuals (Figure 9). The adequacy of hypothesized distributions of latent variables (here, lognormal) was inspected using quantile-quantile (QQ) plots and partial autocorrelation functions readily available in R packages. Further diagnostics not shown here include time series of multipliers, relationships between rainfall multipliers and observed rainfall, etc., to uncover any systematic trends and gain insights into data and model accuracy. Similar to the latent variables, the distributional and autocorrelation hypotheses describing the residual errors can be checked using QQ plots and autocorrelation functions. Although not shown, the same R functionality as above was used to produce diagnostic plots for the residuals.

3. Diagnostics for assessing the predictive distribution (Figure 9). A formal assessment of the predictive distribution was carried out using the predictive QQ plot, with departures from the 1:1 line indicating a lack of predictive reliability and the type of deviations yielding useful insights into specific inadequacies of the predictions [e.g., *Thyer et al.*, 2009]. In this study, the deficiencies indicate the need to improve the hypothesized error models.

4. Time series of observed runoff versus its predicted distribution (Figure 10). The comparison was implemented for both calibration and validation periods (the latter uses data not used in calibration) and yields immediate visual insights into the model performance. Using the diagnostics in RFortran, the entire calibration and validation period can be visualized, in addition displaying partial contributions of individual sources of predictive uncertainty estimated using BATEA (Figure 10).

It is not our intention to list all possible diagnostics for statistical inference, rather to illustrate that even a single model application can produce large amounts of output requiring specialized diagnostics, and that post-processing and documentation of this information can be a considerable logistic and computational challenge in its own right.

5.4. Integration of R visualization into Fortran using RFortran

5.4.1. Interactive MCMC convergence diagnostics

The flowchart in Figure 11(a) outlines how RFortran allowed incorporating R-based functionality into BATEA to generate interactive calibration diagnostics. At regular intervals during MCMC sampling, the data for the convergence diagnostics was passed from Fortran to R for screen plotting. The user interaction was achieved as follows. When RFortran is initialised, it opens the R console. During the MCMC iterations, the BATEA software checks whether a convergence flag (`CONFLAG`) is set to yes (1) or no (0). Initially `CONFLAG` is set to 0 by the BATEA software and the user will view the evolution of the convergence diagnostic plotted in R during the MCMC iterations. If the user deems adequate convergence has been achieved, they set `CONFLAG` to 1 at the R prompt and this is transferred using RFortran back to BATEA to terminate the MCMC sampling.

5.4.2. Automated Bayesian posterior diagnostics

This section details how RFortran automated the production and documentation of posterior Bayesian diagnostics in an easy-to-use pdf format. The corresponding flow chart is shown in Figure 11(b). The RFortran library was deployed to exploit R from within Fortran for the following tasks: open a report file (in pdf format), receive the data from Fortran, produce multiple diagnostic plots, write them to a report file and close the pdf. For these tasks, two additional commands from the `RFortran_Rgraphicsdevice` module are used: (i) `Rpdf()`, which is simply a wrapper for R's function to open a pdf, and (ii) `Rclosegraphicsdevice()`, which uses the R command `dev.off()` to close the current graphics device.

5.5. Benefits of using RFortran

5.5.1. Interactive MCMC convergence diagnostics

RFortran enabled calling R from Fortran to implement an easy-to-use interactive and automated assessment of MCMC convergence and the Bayesian inference. This greatly simplified the development and use of the BATEA software and expedited research results, their documentation

and publication. The interactive plots allow identifying and (whenever possible) rectifying problems without waiting for completion of long calibration runs, which can take days or longer for long-period calibration, especially for numerically expensive hydrological models.

An alternative approach would be to write intermediate results at each iteration inside the Fortran-based MCMC do-loop to an output file and then process it in real-time using some third party software (e.g., Matlab or Excel). In comparison to RFortran (or a similar automation), this approach is tedious to set up and quite cumbersome, especially since the diagnostic information is generated in real-time inside a do-loop, hence requiring careful file management and data access by the stand-alone plotting software. In our case, since the computation/plotting of the diagnostics was already implemented in R, all that was needed was to use a few `Rput/Reval` calls to send the data from Fortran to R and execute the diagnostic procedures available via existing packages.

Another alternative would have been to re-organize the BATEA modules into DLL files and coordinate them from the R platform. This would have required passing considerable amounts of data through the DLL interface, which would have been awkward due to extensive use of specialized data structures (derived types) across the Fortran-based BATEA components. Even if it were possible to organize the code so that only data to be plotted by R was to be passed to R (a considerable challenge given that information from inside BATEA MCMC do-loops was to be passed to R), the DLL interface would have to be modified every time an additional data object was to be plotted. This was frequently the case, given that BATEA is a research development where both theoretical and applied aspects are being examined and routinely modified/improved. Conversely, using RFortran meant that (i) the overall code structure was intact and instead the data of interest was transparently passed to R (few lines of code for each transfer), and (ii) only the R functionality of interest had to be accessed. This considerably simplified BATEA code development.

5.5.2. Automated generation of posterior diagnostics

RFortran allowed invoking R directly from the Fortran code to automate the generation of posterior diagnostics. An alternative was to produce output files using Fortran and then independently read the output files and produce the required graphs using another software package (e.g., Matlab and R). Again, RFortran avoided independently calling (or worse, manually operating) third-party scripts, which would have been tedious and time-consuming considering that dozens of distinct calibration settings were being investigated. R also conveniently allowed writing all diagnostic plots into a single pdf file immediately available for inspection and stored, if necessary, for later analysis and comparison. By integrating the calls to R within the BATEA executable, a large numbers of diagnostic plots are generated automatically with minimal user intervention: a BATEA user does not need to run any additional scripts to produce a major set of diagnostics.

Also note that, in the context of this case study, the alternative implementation of BATEA using DLLs would suffer from the same limitations as outlined in the previous section.

Finally, since R is open-source, the data-analysis scripts for BATEA can be shared by all users. Conversely, using commercial software would have restricted them to licensed users only.

6. Case Study 2: Exploiting R copula packages for flood modelling

6.1. Background

Flood risk assessment in coastal estuarine locations is an important aspect of environmental engineering. Since in general flood risk varies spatially over areas of interest, Need et al. (2008) used the total probability integral to estimate the total probability of flooding,

$$\Pr\left(y\left(x\right) > y_{thresh}\right) = \iint \Pr\left(y\left(x,Q,H\right) > y_{thresh}\right) f\left(Q,H\right) \mathrm{d}Q\,\mathrm{d}H \qquad (1)$$

where y_{thresh} is the water level threshold defining a flood event, and the bathymetry function $y(x, Q, H)$ yields the water level y at location x as a function of ocean levels H and freshwater inflows Q.

Eqn (1) computes the expected probability of flooding by integrating over inflows and ocean heights weighted by their joint probability $f(Q, H)$. Since evaluating the bathymetry function in

general requires using numerical hydraulic models, the integral (1) must be approximated using numerical quadrature. In addition, the joint probability of Q and H must reflect correlations between ocean heights and estuarine flows. This could be modeled using copulas [*Favre et al.*, 2004].

6.2. Motivation for using RFortran

Computationally efficient hydraulic models have already been implemented in Fortran [e.g., *Need et al.*, 2008; *Tan et al.*, 2008]. In addition, high-performance numerical quadrature routines for evaluating the double integral (1) were also available in Fortran. However, the characterization of the joint density $f(Q, H)$, where the variables are non-Gaussian and may have varying degrees of correlation, was not available in Fortran, yet was available in several R statistics packages.

Since this study formed part of a larger research project developing along several investigative directions, flexibility and speed of implementation were major priorities. The `copula` package [*Kojadinovic and Yan*, 2010] in R was particularly suitable in this respect because it implemented the general families of elliptical and Archimedian copulas, as well as corresponding estimation and visualization utilities. While the Gumbel-Hougaard copula was chosen initially, it was anticipated that alternative marginal distributions and/or copula types were to be trialed and compared. The ability to use the R package obviated the need to re-develop general copula code, significantly reducing research and development time and allowing flexibility in the selection and parameterization of the copulas.

6.3. Integration of R computation into Fortran using RFortran

Figure 12 shows how RFortran was used to incorporate the R copula package into the numerical quadrature code for the integration of the hydraulic model. First, R was initialised and the copula package loaded via `Reval("library(copula)")`. The double integral (1) was then approximated using suitable discretizations ΔQ and ΔH determined by the Fortran-based quadrature code. The $y()$ term in Eqn (1) was evaluated using the hydraulic model (Fortran-based), while the $f()$ term was evaluated using the `dcopula()` function (R-based). Eq. (1) was implemented repeatedly for a range of water depths and for all reach locations to obtain the probability distribution of water

depth at any point of interest. Contours of these distributions at 10, 20, 50 and 100 year annual recurrence intervals (ARI) are shown in Figure 13(a) for the case where Q and H are independent and in Figure 13(b) where their correlation is modelled using a copula.

6.4. Benefits of using RFortran

The main advantage of using RFortran in this context is the simplicity with which a flexible R package having a recently-developed statistical technique was embedded into a Fortran-based framework implementing carefully optimized numerical quadrature and hydraulic models.

With enough time and resources, the copula model could have been re-implemented in Fortran, or the hydraulic model re-implemented in R, or broken up into DLL modules called from R via the .fortran facility. Yet the major deliverable of this project was improved scientific and engineering analysis of flood risk and a thorough investigation of alternative hydraulic models and probabilistic assumptions underlying the oceanic forcing, rather than re-implementation of well-tested individual components in a new language. Given this project objective, RFortran provided a convenient and efficient mechanism for combining, evaluating and replacing existing individual components already developed by dedicated specialists in their respective fields. The ability to operate within a Fortran-based hydraulic modelling framework that was familiar to the researchers was also nontrivially beneficial in terms of overall productivity and reliability of the analysis.

To highlight the advantages of RFortran in this context, consider the alternative of calling Fortran DLLs with R's .fortran functionality. Firstly, there is the challenge of breaking up the hydraulic model code into DLLs and using the correct naming conventions for their interfaces, writing new R code to call the DLL-based procedures. Sharing the data via R would be finicky due to the need to implement the communication of Fortran data structures derived types across the R-Fortran DLL interface. Secondly, unless the 2D numerical quadrature algorithm was itself re-implemented in R, the R-based copula density would still need evaluation from within the Fortran environment, which is problematic in the absence of RFortran (see Section 8 for further discussion). All these challenges relate to software design issues and are generally outside the primary expertise of hydrologists and

water engineers. Conversely, using RFortran avoided these software challenges in the first place and allowed the researchers to focus on the primary environmental modeling challenge of characterizing the bathymetry of the coastline and its probabilistic oceanic forcing.

7. Comparison to alternatives: Advantages and limitations

7.1. RFortran versus the "R calls Fortran DLLs" approach

A key design feature of RFortran is its "hiding" of all system-level communication, freeing the Fortran user from having to understand the technicalities of mixed-language programming and COM. As a result, accessing R using RFortran only requires knowledge of the specific R commands of interest. In contrast, the "R calls Fortran DLLs" alternative requires, in addition:

(i) knowledge of DLL calling and naming conventions when compiling Fortran code into DLLs. In particular, multiple conventions exist for "pass-by-reference" versus "pass-by-value" arguments, column-major versus row-major array representation, several options for character string handling, as well as several options for procedure name decoration. Moreover, certain Fortran data structures, in particular, derived types, pointers, deferred-shape arrays and overloaded generic procedure names, are difficult to transfer across mixed-language interfaces. Difficult-to-debug compatibility problems may arise between the DLL and the calling application, especially when the interfaces are changing frequently (which is common during research software development, especially in the early stages of algorithm implementation, testing and refinement). Finally, compiling into DLLs may disable inter-procedural compiler optimizations (e.g., inlining and loop fusion) and lead to a loss of computational efficiency;

(ii) knowledge of R to coordinate the Fortran DLLs, passing data and procedures between them. If the data flow and/or computational sequence of the model is modified, both the Fortran DLLs and the R code may need updating (including changes to DLL arguments and their characteristics).

(iii) Re-organising the Fortran code into DLL format. For scientists and engineers with primary interest/expertise in environmental modelling, these purely software-design steps may be time-

consuming and discouraging, especially if the desired use of R is simply to utilize a relatively small number of R functions for visualization and auxiliary numerical processing.

Importantly, RFortran operates entirely from the Fortran environment familiar to our intended audience, and requires no modification of existing codes beyond modifications actually associated with calling R functions (i.e., no re-organization of Fortran code hierarchy, etc). This allows Fortran programmers to continue exploiting their primary programming expertise and gradually build R into their existing and new programs, carefully monitoring the result of each addition. In contrast, using DLLs requires the programmer to fully move to R as the main platform binding Fortran modules and coordinating the communication of any data that must be visible to R or communicated between the Fortran modules. This may not be optimal in many practical contexts.

Conversely, developers comfortable in higher-level languages such as Python, Octave, Matlab and R may prefer using Fortran DLLs from these languages, in particular, due to better interoperability with additional languages and applications. As we move towards dedicated software development, the advantages of higher-level languages such as Python become more pronounced. In addition, using the DLL approach allows passing Fortran procedures to R functions, whereas this cannot be accomplished using RFortran alone (see Section 8 for potential solutions).

Another current limitation of RFortran is its exclusive reliance on the COM technology, limiting it to the Windows operating system. The DLL approach is not subject to this limitation. The Common Object Request Broker Architecture (CORBA) for Linux and other platforms [*Object Management Group*, 2002] is broadly analogous to COM and will be explored in future work. Indeed, the Fortran APIs in Matlab are supported under both Windows and Linux. Another option to enable RFortran to be used on Linux is to replace statconnDCOM and use statconnWS [*Baier*, 2010]. These options will be evaluated in the future to provide Linux support for RFortran.

Since RFortran uses COM to transfer data between R and Fortran, there is a computational cost in packing the Fortran arrays in/out of the safeArrays that is not present in the R-based Fortran DLL approaches. **Figure 14** compares the data transfer times for RFortran and R-based Fortran DLL

approaches for arrays of varying sizes, undertaken on a Intel Core© i7 2.66 GHz desktop PC. For array size below 10^5 elements, the data transfer time is negligible in all cases. For larger array sizes ($> 10^5$ elements), the data transfer using RFortran is 8-10 times slower than calling a Fortran DLL from R. This computational overhead of using RFortran needs to be seen in the context of the application and traded-off against the ease of application of RFortran relative to transforming the Fortran code into DLLs. The absolute time for RFortran transfer of arrays with up to 10^7 elements are below 1 sec. This is far below runtimes typically incurred when using large data sets in actual computations. For example, runtimes of many environmental models are of the order of minutes for typical calibration periods (few years of daily data, corresponding to time series with 10^3-10^4 elements) [e.g. *Tolson and Shoemaker*, 2008]. Therefore, the computational overhead of RFortran array transfer is negligible in comparison with typical computational demands of environmental data analysis and modelling, and will generally represent a minor component of the total runtime.

7.2. Jointly compiling Fortran code with the R language

Since the R language itself is implemented in Fortran and C, it is possible to call R functions by compiling and linking their source jointly with the user's Fortran code. This has a number of advantages and disadvantages. The major advantage is avoiding reliance on COM, making the approach largely independent from the compiler and operating system. It also allows using a single Fortran debugger for both the Fortran and R-based components. Limitations of joint compilation include the requirement to understand the data structures returned by R procedures, in particular, the SEXP C-based R data types. In addition, the executable files produced from joint compilation may be unduly large because they will include all the procedures within the calling hierarchy, whereas RFortran uses the R interpreter built and distributed by the R development team. Finally, joint compilation requires re-building the application every time R is updated, whereas using RFortran only requires updating the path to the R command interpreter and installing the *rcom* package in the latest R version and does not require any re-compilation.

Similarly to the previous section, we anticipate that dedicated software engineers, e.g., comfortable with the mapping of SEXP and Fortran data structures, etc, will work directly with the R language source, whereas Fortran-oriented environmental researchers and practitioners may prefer RFortran.

7.3. Alternative COM implementations

The system-level components of RFortran utilize the *rcom* package to connect with R [*Baier*, 2009]. A key feature of *rcom* is its use of the R command interpreter to invoke R functions; commands sent to R are built up from character strings, often at runtime if their exact arguments are not known a priori. This paradigm is widely used in existing software, e.g. in RExcel (which also uses *rcom*) [*Baier and Neuwirth*, 2007] and in the `engEval()` API in Matlab [*Mathworks*, 2007].

However, the COM mechanism can support a tighter level of integration, for example, creating local, rather than global, contexts for the variables passed to the R functions. Moreover, instead of `Reval("Rfunction(arg1,arg2)")`, the ability to use `Reval("Rfunction",arg1, arg2)` could be implemented. This approach more closely mimics calling native Fortran functions, simplifies the management of the R memory space and avoids potential loss of data if a previously passed variable is accidentally overwritten. This approach is not yet implemented in RFortran, in part because it would require extensive overloading of `Reval` to support different combinations of types and shapes of arguments. In addition, the existing RFortran functionality can also be used to manage the R memory space to avoid within-process memory conflicts, and using multiple R instances prevents memory conflicts from different processes accessing the same R instance.

For object-oriented languages, COM can support even more direct mixed-language interfacing, where R functions are invoked directly from the host language, rather than via the command interpreter. For example, in the Visual Basic (VB) language, one could create an R object using `Set Robj = CreateObject("Robject")`, and then directly manipulate the object in VB using the object's R-based methods as if they were native VB constructs.

For R, this approach is supported by the RDCOMServer/RDCOMClient packages [*Lang*, 2005] and is analogous to COM-based automation of VB-oriented applications such as SigmaPlot. In a sense, it resembles direct compilation of Fortran code and the R language, representing a deeper level of integration and truer interface bi-directionality. However, creating and correctly receiving/ manipulating the objects returned by R functions could be problematic in Fortran due to its current lack of object-orientation. Consequently, this option was deferred to future work, in particular, when the object-oriented features of Fortran-2008 become more widely implemented.

8. Towards a unified bi-directional interface between Fortran and R

A fairly fundamental current limitation of RFortran and existing methods for R-Fortran interaction (e.g., the .fortran facility in R) is the lack of a transparent bi-directional interface between these two languages. For example, while RFortran can be used to supply an R function to a Fortran procedure (e.g., by wrapping the former in a standard Fortran interface), it cannot supply a Fortran function to R (because RFortran only supports calling R from Fortran, not vice versa). This means, for example, that while RFortran enables Fortran-based optimization of an R-based environmental model, it can not do the reverse, i.e., optimize a Fortran-based model using an R-based optimizer. The DLL approach is also limited in this respect: while it can pass a Fortran function to an R procedure (e.g., by wrapping it in an R interface), it cannot carry out the reverse.

Note that the mutual complementarity of the RFortran and DLL approaches permits ad-hoc bi-directional programming, where RFortran enables calling R from Fortran while the R .fortran DLL functionality supports the reverse operation. While operational, this is “clunky” and ad-hoc. A more unified bi-directional interface could, at least in principle, be implemented using RDCOMServer/ RDCOMClient [*Lang*, 2005], though it is not clear how to support R objects in Fortran due to lack of object-orientation support. This enhancement is therefore deferred to future work.

9. Future work

RFortran can be extended to other Microsoft Windows compilers supporting the COM interface, e.g., by building it as a DLL embedding the COM functionality of the Intel and Compaq compilers.

Also, while RFortran is currently limited to Windows due to its reliance on COM, future work will explore using the CORBA interface [*Object Management Group*, 2002] and/or using statconnWS [*Baier*, 2010] to enable RFortran to be used under Linux.

We also anticipate designing Fortran derived type to match R objects such as dataframes, etc. This will allow a more natural data communication between R and Fortran, and may aid in the implementation of the more object-oriented interactions described in Sections 7.3 and 8.

Future development will also include the implementation of analogous approaches for linking Fortran to other popular scientific software such as Scilab (http://www.scilab.org) and Matlab.

10. Conclusions

The RFortran library provides a convenient tool for accessing R-based functionality from a Fortran program, which, given R's extensive and rapidly growing library of statistical, numerical and visualization algorithms of all levels of sophistication, is an appealing capability.

RFortran treats the Fortran program as the main platform and R as a subunit of this program. It provides an alternative to existing approaches for integrating R and Fortran, which require the Fortran programmer to operate from within the R platform by loading and calling Fortran procedures built as DLLs. The RFortran approach is useful for situations where conversion to Fortran DLLs is impractical, e.g., for legacy reasons, or when R is needed to carry out specific tasks, such as plotting or computation, subordinated within Fortran-based procedures. In these contexts, being required to switch to R as the control platform to coordinate Fortran subprograms may be infeasible, or deemed unwarranted, by a scientist or engineer with primary programming expertise in Fortran, and may discourage them from using R altogether.

The core and extended functionalities of RFortran were described using several simple examples. The code snippets demonstrate the use of RFortran functions to transfer data to/from R and evaluate R commands and scripts from Fortran, with particular emphasis on visualization applications.

Furthermore, two real-life environmental modelling applications illustrated the motivation and usage of RFortran for environmental scientists and engineers with primary programming expertise in Fortran. The first study used RFortran to produce interactive and automated R-based calibration diagnostics in a Fortran-based rainfall-runoff model inference code. The second study used RFortran to access an R-based copula statistical package directly from within a Fortran-based numerical quadrature code for probabilistic flood risk analysis in coastal areas. In both studies, RFortran expedited method development, computation of results and publication, by allowing the developers to immediately exploit existing R functionality within familiar Fortran programming environments.

Several alternatives to RFortran are discussed, including DLL-based .fortran capability in R, joint compilation of the R language itself with the user Fortran code, and the use of alternative COM packages. In comparison with the DLL approach, RFortran is more naturally suited to a Fortran programmer, whereas an R programmer would prefer to use the DLL approach. Joint compilation of the R language and Fortran can allow better memory-space sharing and allow calling R functions using Fortran conventions, but requires the programmer to master the R data structures to a greater extent than when using RFortran. Finally, alternative COM implementations may allow creating more comprehensive bi-directional interfaces, but requires object-orientation facilities currently lacking in Fortran. The practicality of these approaches will be explored in future work.

RFortran is released under an open source lesser license (lesser GNU Public license) and is available from www.rfortran.org. This site provides installation and implementation details, further examples and sample code, and allows users and developers to report defects, request enhancements, submit extensions and access updates and fixes to RFortran. We hope that RFortran helps expedite method development for scientists and engineers with primary programming expertise in Fortran, but who wish to take advantage of R's visualization and statistical/mathematical modelling packages by calling them directly from their code.

11. Acknowledgements

We thank the Editor and the 4 anonymous reviewers for their constructive suggestions and criticisms, which significantly improved the RFortran software and this paper.

12. References

Abramowitz, G., Y. Wang, and B. Pak (2007). A user guide for the ACCESS land surface model. CSIRO Marine and Atmospheric Research.

Baier, T. (2009), rcom: R COM Client Interface and internal COM Server. R package version 2.1.1

http://cran.r-project.org/web/packages/rcom/

Baier, T. (2010). R as a Service: statconnWS. http://learnserver2.csd.univie.ac.at/rcomwiki/doku.php?id=statconnws, Accessed on 12th June 2010

Baier, T., and E. Neuwirth (2007), Excel :: COM :: R, Computational Statistics, 22 (1),91-108

Canaimasoft (2000). f90VB User Manual. http://www.canaimasoft.com/f90vb/

Cowles, M., and B. P. Carlin (1996), Markov chain Monte Carlo convergence diagnostics: A comparative review, Journal of the American Statistical Association, 91,883-904

Favre, A. C., S. E. Adlouni, L. Perreault, N. Thiemonge, and B. Bobee (2004), Multivariate hydrological frequency analysis using copulas, Water Resources Research, 40 (1),W01101

Giunta, G., P. Mariani, R. Montella, and A. Riccio (2007), pPOM: A nested, scalable, parallel and Fortran 90 implementation of the Princeton Ocean Model, Environmental Modelling and Software, 22 (1),117-122

Kavetski, D., G. Kuczera, and S. W. Franks (2006), Bayesian analysis of input uncertainty in hydrological modeling: 1. Theory, Water Resources Research, 42 (3),doi:10.1029/2005WR004368

Kojadinovic, I., and J. Yan (2010), Modeling Multivariate Distributions with Continuous Margins Using the copula R Package, Journal of Statistical Software, 34 (9),1-20.http://www.jstatsoft.org/v34/i09/.

Kuczera, G., B. Renard, D. Kavetski, and M. A. Thyer (2010), A limited-memory acceleration strategy for MCMC sampling in hierarchical Bayesian calibration of hydrological models, Water Resources Research,, in press (accepted 12/2/2010)

Lang, D. T. (2005), RDCOMServer: R-DCOM object server. http://www.omegahat.org,

Lemmon, D. R., and J. L. Schafer (2005) Developing Statistical Software in Fortran 95. Springer, New York, USA.

Maindonald, J., and J. Braun (2007) Data Analysis and Graphics Using R - An Example-Based Approach, 2nd ed. Cambridge University Press.

Mathworks (2007). Matlab: The language of scientific computing.

McDonald, M. G., and A. W. Harbaugh (2003), The History of MODFLOW, Groundwater, 41 (2),280-283

Need, S., M. Lambert, and A. Metcalfe (2008) A total probability approach to flood frequency analysis in tidal river reaches. *Proceedings of the World Environmental and Water Resources Congress*, Ahupua'a, Hawaii, ISBN:

Object Management Group, I. (2002), Common object request broker architecture: Core specification. Specification 3.0, Nov 2002.,

R Development Core Team (2010). R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria. http://www.R-project.org.

Tan, K. S., F. H. S. Chiew, and B. Grayson (2008), Stochastic event-based approach to generate concurrent hourly mean sea level pressure and wind sequences for estuarine flood risk assessment, ASCE Journal of Hydrologic Engineering, 13 (6),449-460

Thyer, M., B. Renard, D. Kavetski, G. Kuczera, S. W. Franks, and S. Srikanthan (2009), Critical evaluation of parameter consistency and predictive uncertainty in hydrological modeling: A case study using Bayesian total error analysis, Water Resources Research, 45,W00B14, doi:10.1029/2008WR006825

Tolson, B. A., and C. A. Shoemaker (2008), Efficient prediction uncertainty approximation in the calibration of environmental simulation models, Water Resour. Res, 44, (11)

Vance, A. (2009). Data analysts captivated by R's power. http://www.nytimes.com/2009/01/07/technology/business-computing/07program.html.

List of Figures

Figure 13. Flood depths for various ARIs along the length of the channel shown using a log scale from the downstream boundary for (a) independent driving forces; and (b) driving forces with correlated residuals drawn from a Gumbel copula.

Figure 14. Comparison of array transfer times for RFortran versus Fortran DLL approaches.

```
01 program AustraliaRainfall_SimpleGraph
02   use RFortran                       ! Access RFortan
03   implicit none
04   integer(4) :: i,j,ok               ! Index counters, error flag
05   real(8)    :: x(178,139)           ! Data matrix
06   logical    :: mv(178,139)          ! Missing value matrix
07
08   open(10,file="../../../data/AustraliaRainfall.txt") ! Read file
09   do i = 1,178
10     read(10,*) (x(i,j),j=1,139)
11   end do
12   close(10)
13
14   ok = Rinit()             ! Initialise R
15   ok = Rput("x",x,mv=x<0) ! Transfer data (and missing values where x<0) to R
16   ok = Reval("image(x)")  ! Create a map of the rainfall
17   if (ok/=0) then          ! Error Handling, check ok==0
18     call get_messages(lastmessage=lastmsg) ! If ok/=0 retrieve last message
19     write(*,*) "Rfortran error: "//TRIM(lastmsg)//" check message.log"
                              ! Write message to screen - unnecessary
                              ! for Fortran console applications
                              ! as message log is echoed to screen
20   end if
21   ok = Rclose()            ! Close R
22 end program AustraliaRainfall_SimpleGraph
```

Figure 1. Passing data to R and plotting a simple graph (example 1).

```
! missing lines as per example 1 , lines 1-4
! output gives "Intercept 456.5 mm, Slope 5.76 mm, SSE 548497 mm^2"
05  real(8)     :: yr(102),rain(102),soi(102)  ! Data matrix
06  real(8)     :: par(2),resid(102)           ! Fitted parameters,residuals
! read data here - full code provided at www.rfortran.org
11    ok = Rinit()                                ! Initialise R, no scripts
12    ok = Rput("soi",soi);ok = Rput("rain",rain) ! Transfer data to R
13    ok = Reval("ans<-lm(rain~soi)")             ! Fit a linear model
14    ok = Rget("ans$coefficients",par)           ! Get slope,intercept
15    ok = Rget("ans$residuals",resid)            ! Get slope,intercept
16    ok = Reval("plot(soi,rain,xlab='SOI',ylab='Ave. Annual Rainfall (mm)')")
17    ok = Reval("lines(soi,ans$fitted.values)")
18    write(*,*) "Intercept",par(1),"Slope",par(2),"SSE",sum(resid**2)
! missing lines as per example 1, lines 21-22
```

Figure 2. Passing data to R functions and returning results to Fortran (example 2).

```
! missing lines as per example 1,lines 1-4
05  real(8)    :: x(178,139),lon(178),lat(139) ! Data,Longitude, Latitudes
! read data here - full code provided at www.rfortran.org
11  lon = 112+(/(i,i=0,177)/)*0.25            ! 178 x 0.25 degree increments
12  lat= -44.5+(/(j,j=0,138)/)*0.25           ! 139 x 0.25 degree increments
13  ok = Rinit()                              ! Initialise R
14  ok = Rimage(lon,lat,x,cmd="xlab='Longitude',ylab='Latitude', &
15       & breaks=c(100,200,500,1000,2000,5000),col=topo.colors(5)[5:1]",mv=x<0)
! missing lines as per example 1,lines 21-22
```

(a) Fortran main program

```
01 function Rimage(x,y,z,cmd,mv,status) result(ok)
02  ! Description: Wrapper for image() function, real data x, y and z
03  implicit none
04  integer(4) :: ok
05  real(8), dimension(:), intent(in) :: x,y ! x,y axes spacing
06  real(8), dimension(:,:), intent(in) :: z ! data to be plotted
07  character(len=*), optional :: cmd,status ! additional commands and status
08  logical, dimension(:,:), intent(in), optional :: mv  ! missing values
09  ok = Rput('x.temp',x,status) ! transfer data to R
10  ok = Rput('y.temp',y,status) ! transfer data to R
11  ok = Rput('z.temp',z,mv,status) ! transfer data to R
12  if(present(cmd)) then
13   ok = Reval('image(x.temp,y.temp,z.temp,' // trim(cmd) // ')') ! Plot image
14  else
15   ok = Reval('image(x.temp,y.temp,z.temp)')  ! Plot image
16  end if
17  ok = Reval('rm(x.temp,y.temp,z.temp)')!! clean up temp variables
18 end function
```

(b) Fortran wrapper for image()

Figure 3. Using Fortran wrappers for a series of RFortran/R commands (example 3).

```
! missing lines as per example 3,lines 1-12
15  ok = Rinit();                                          ! Initialise R
16  ok = LoadRscripts(path=trim(RFortran_Path)//"\Rscripts\tutorial_examples",&
              Rscript="plot.australia.image.r");          ! Load R script
17  ok = Rput("lon",lon)                                   ! Transfer data to R
18  ok = Rput("lat",lat)                                   ! Transfer data to R
19  ok = Rput("rain",rain,mv=rain<0)                       ! Transfer data and missing
                                                             values to R
20  ok = Reval("plot.australia.image(lon,lat,rain)") ! Execute script
21  ! Do all in one step using Rcall
22  ok = Rcall(vars=(/Rput("lon",lon),Rput("lat",lat),&
                      Rput("rain",rain,mv=rain<0)/),&! Transfer data
            cmds=(/cl("windows()"),&                     ! Generate plot window
                   cl("plot.australia.image(lon,lat,rain)"),& ! Execute script
                   cl("rm(lon,lat,rain)")/))                  ! Remove Data
! missing lines as per example 1,lines 21-22
```

(a) Fortran main program

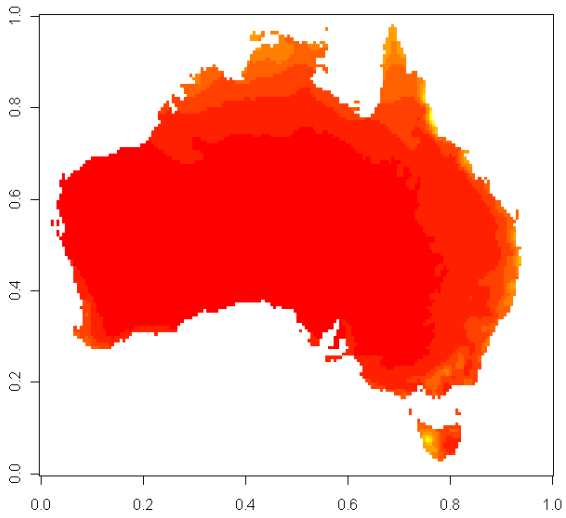
```
01 plot.australia.image<-function(lat,lon,x,…){
02  # Requires package oz to plot coastlines
02  image(lon,lat,x, xlab="Longitude",ylab="Latitude",cex.lab=1.5,cex.axis=1.5,
03        breaks=c(100,200,500,1000,2000,5000),col=topo.colors(5)[5:1])
04  contour(lon,lat,x,levels=c(100,200,500,1000,2000,5000),add=TRUE,labcex = 1)
05  if (require(oz)) oz(add=T,states=F)
06  site.x<-c(115.9,138.6,147.3,145.0,151.0,153.1,130.9)
07  site.y<-c(-31.95,-34.92,-42.87,-37.78,-34.0,-27.48,-12.47)
08  names<-c("Perth","Adelaide","Hobart","Melb.","Sydney","Brisbane","Darwin")
09  text(site.x,site.y,names,cex=1.2)
10 }
```

(b) R script

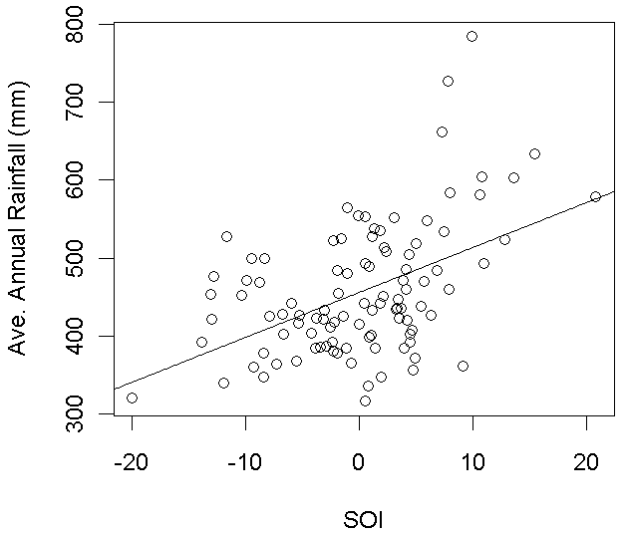
Figure 4. Using R scripts and Rcall for a series of RFortran/R commands (example 4)

```
01 program EvaluteGaussianDensity     ! Evaluate Gaussin Density
02   use RFortran                           ! Access RFortan
03   implicit none
04   integer(4) :: i,ok                     ! index counter, error flag
05   real(8)    :: x(100),density(100),delta  ! Data array
06   real(8), external :: gaussianDensity     ! External Function
07   ! Calculate prob density in Fortran
08   delta=(4-(-4)/(100); x(1)=-4
09   do i = 1,(n-1)
10    density(i)=GaussianDensity(x(i),0.0_8,1.0_8); x(i+1)=x(i)+delta
11   end do
12   ok = Rinit()                                  ! Initialise R
13   ok = Rput("x",x);ok = Rput("probdensity",probdensity) ! Transfer data to R
14   ok = Reval("R_Density=dnorm(x)")              ! Calculate prob density in R
15   ok = Reval("plot(x,probdensity,col='blue')")! Plot Fortran prob density
16   ok = Reval("lines(x,R_Density,col='red')")  ! Plot R prob density
17   ok = Reval("legend(x='topleft',legend=c('Fortran','R'),col=c('blue','red'),
               lty=c(NA,1),pch=c(1,NA))")       ! Add legend
18   ok = Rclose()
19   pause
20 end program EvaluteGaussianDensity
21 ! Gaussian Density Function
22 real(8) function GaussianDensity(x,mu,sigma)
23  implicit none
24  real(8), intent(in) :: x,mu,sigma
25  real(8), parameter :: pi=3.1415926535897932_8
26  GaussianDensity=1.0_8/sqrt(2*PI)*exp(-0.5_8*(x-mu)**2/sigma**2)
27 end function
```

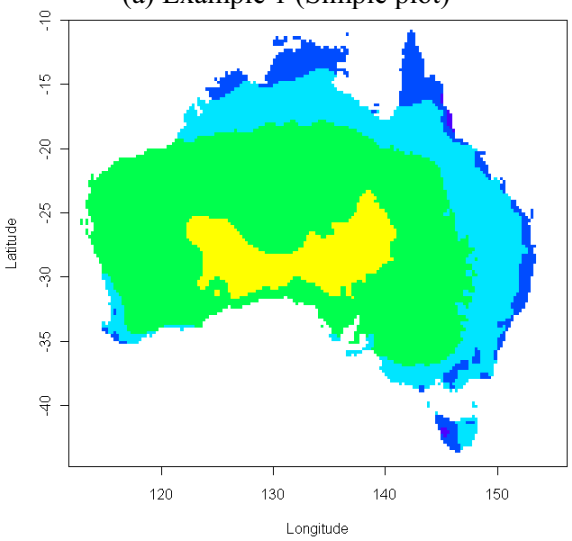
Figure 5. Using RFortran for interactive debugging, evaluating Gaussian density (example 5)



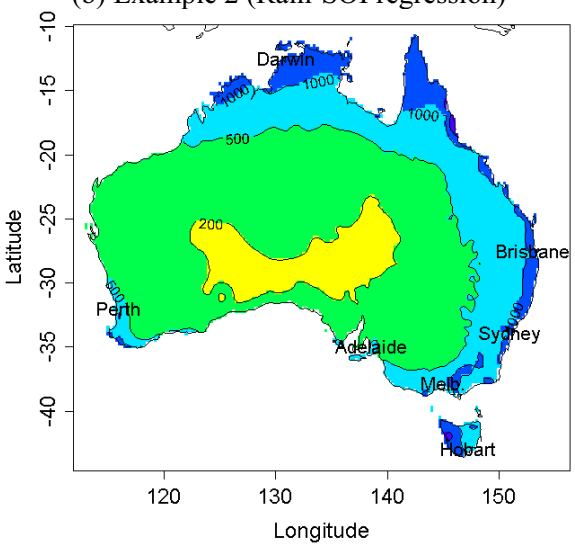
(a) Example 1 (Simple plot)



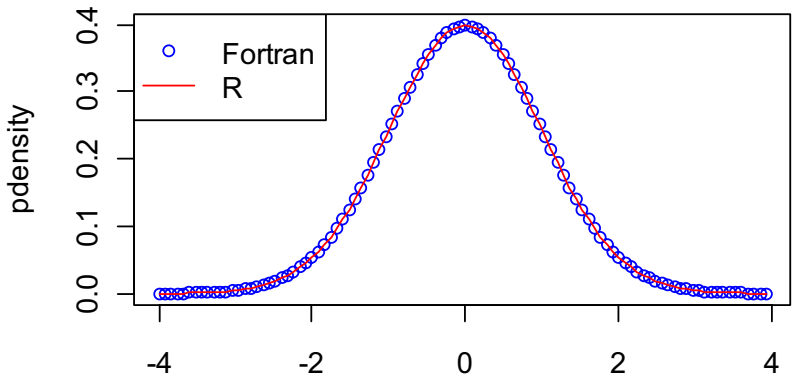
(b) Example 2 (Rain-SOI regression)



(c) Example 3 (Using the wrapper Rimage)



(d) Example 4 (Using an R script)



(e) Example 5 (Evaluating a Gaussian density)

Figure 6. Output graphics from RFortran examples in Section 2.2.

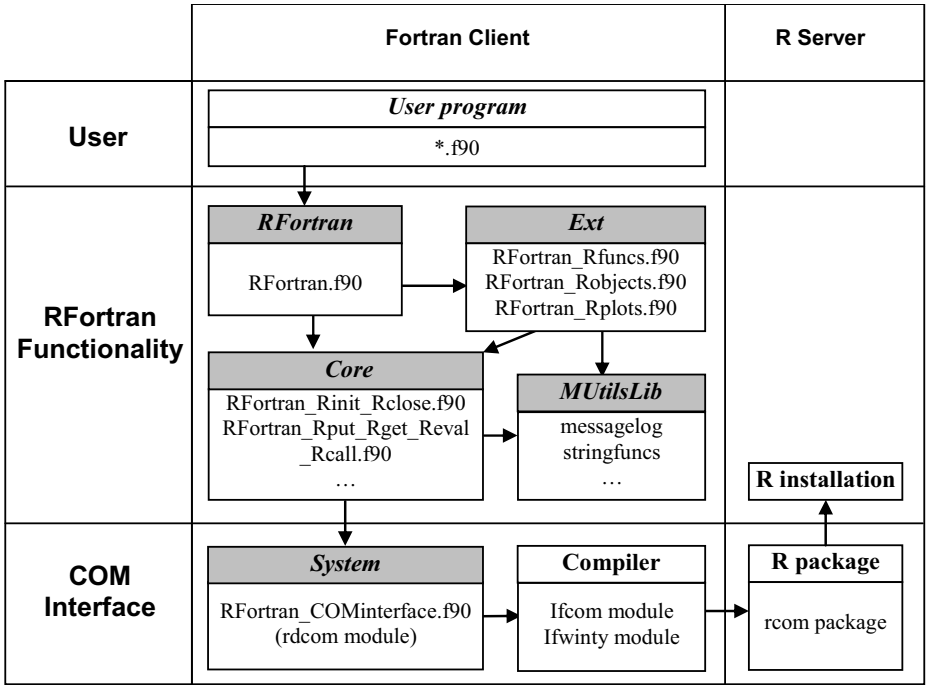


Figure 7. Schematic of the RFortran library. Boxes represent file folders and Fortran modules. Grey headings denote files forming RFortran. "..." denotes auxiliary files and modules. Unless noted otherwise, files and modules have the same name.

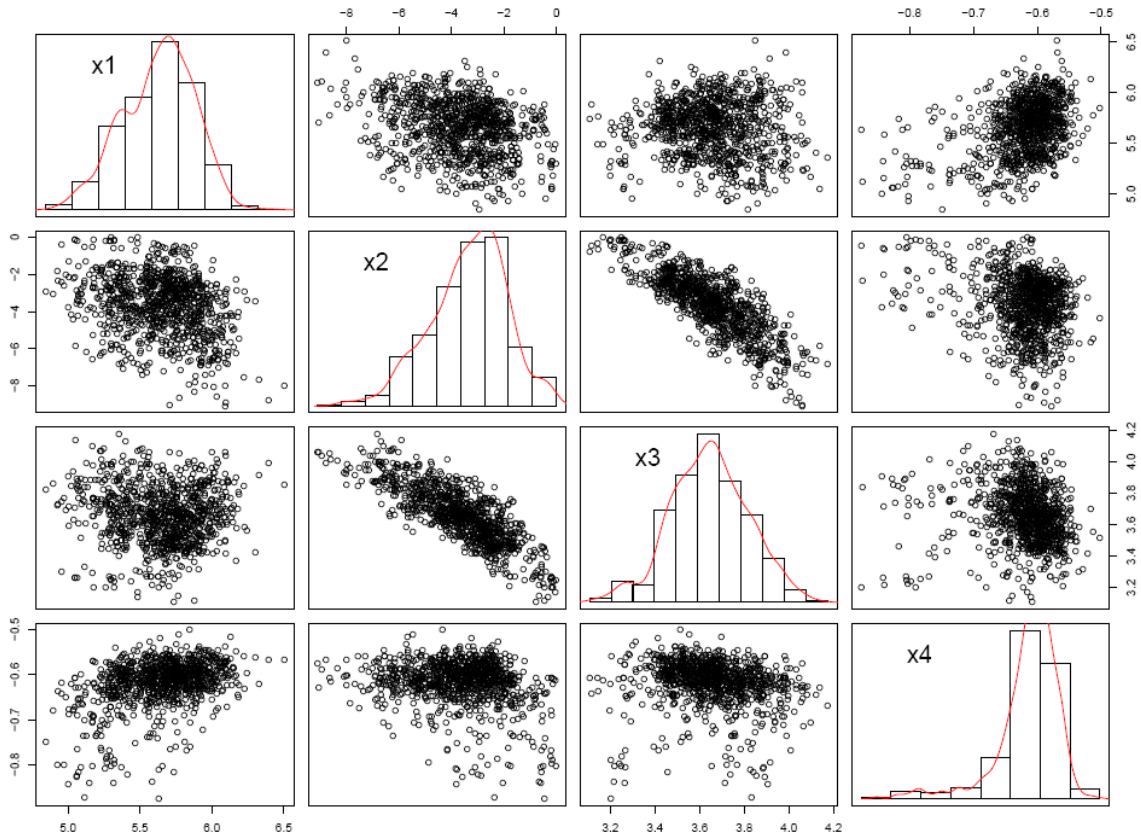
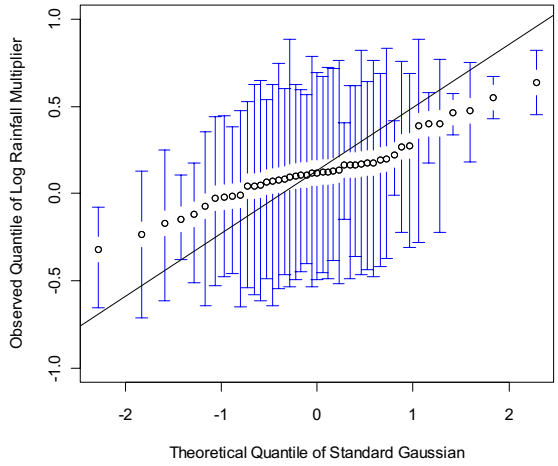
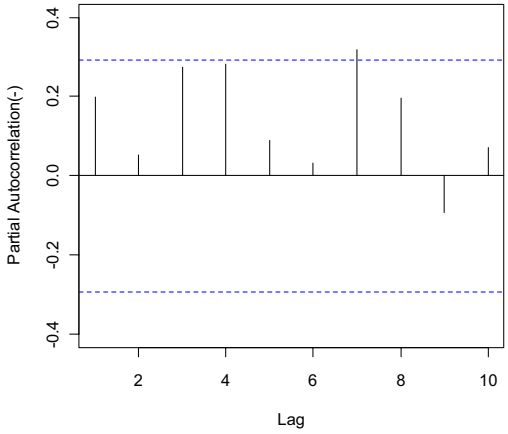


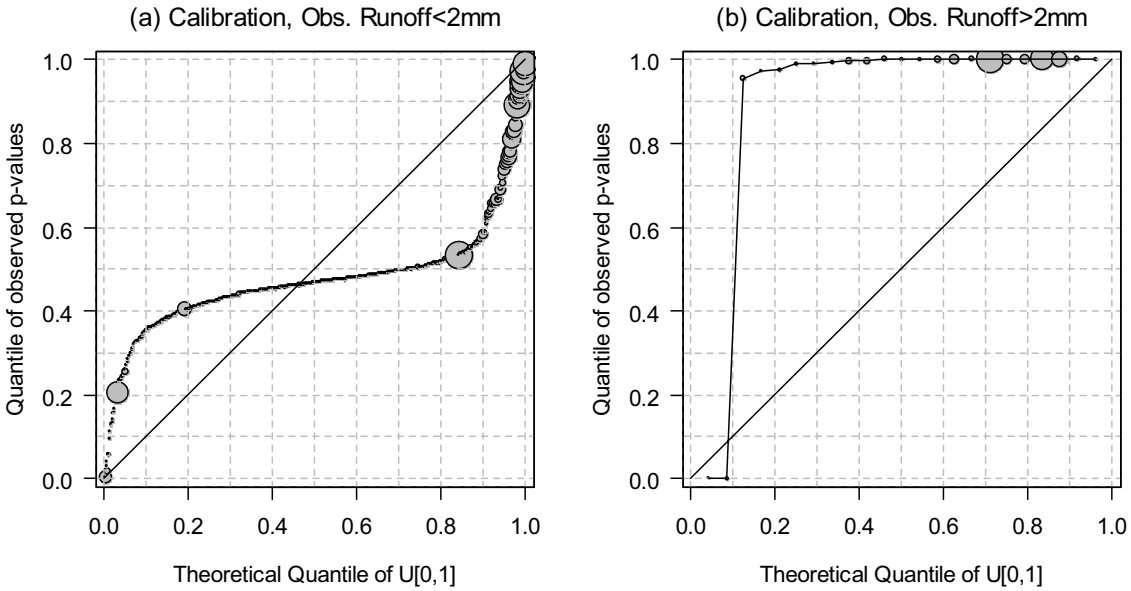
Figure 8. Scatterplot matrix of MCMC samples, produced using RFortran.



(a)

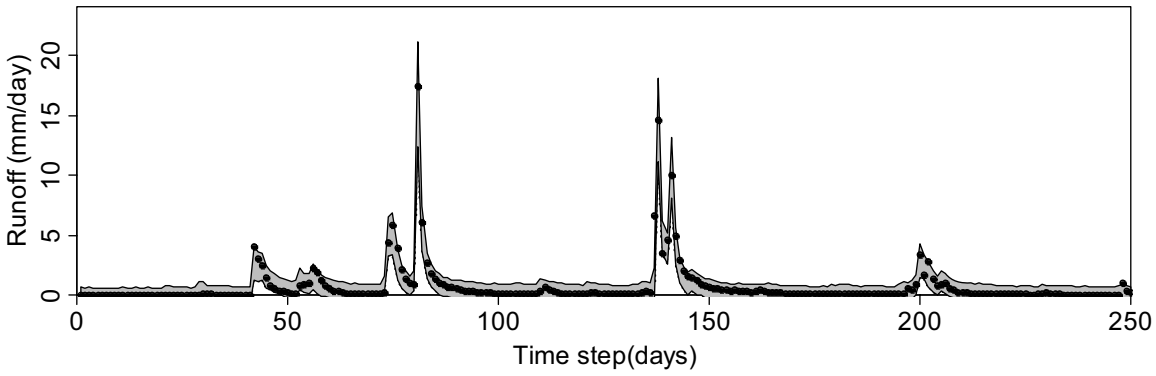


(b)

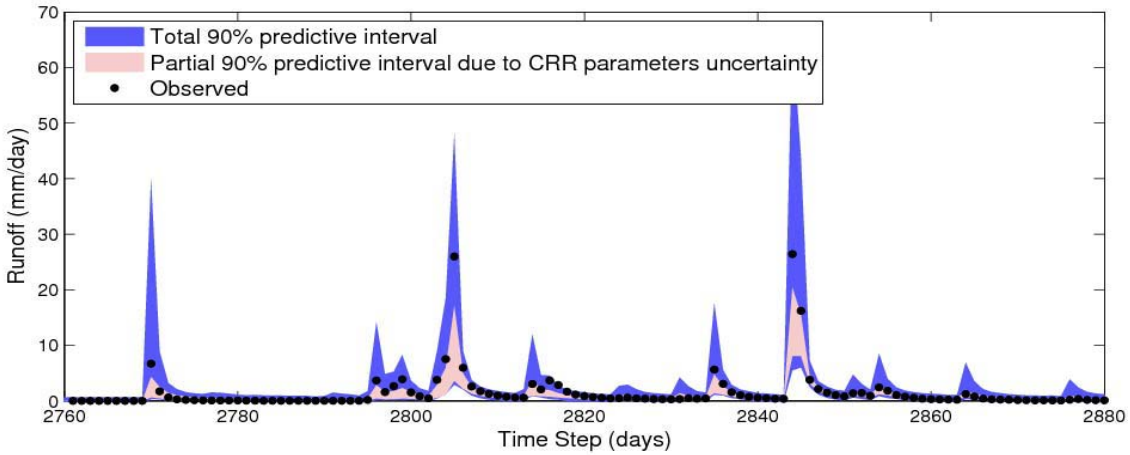


(c)

Figure 9. Representative posterior diagnostics for BATEA calibration, produced using RFortran: (a) QQ plot of estimated rainfall errors, (b) Partial autocorrelation of rainfall errors and (c) Predictive QQ plot for runoff in validation.



(a) Calibration period



(b) Validation period

Figure 10. Representative time series plot of observed and predicted runoff (including uncertainty bounds), produced using RFortran.

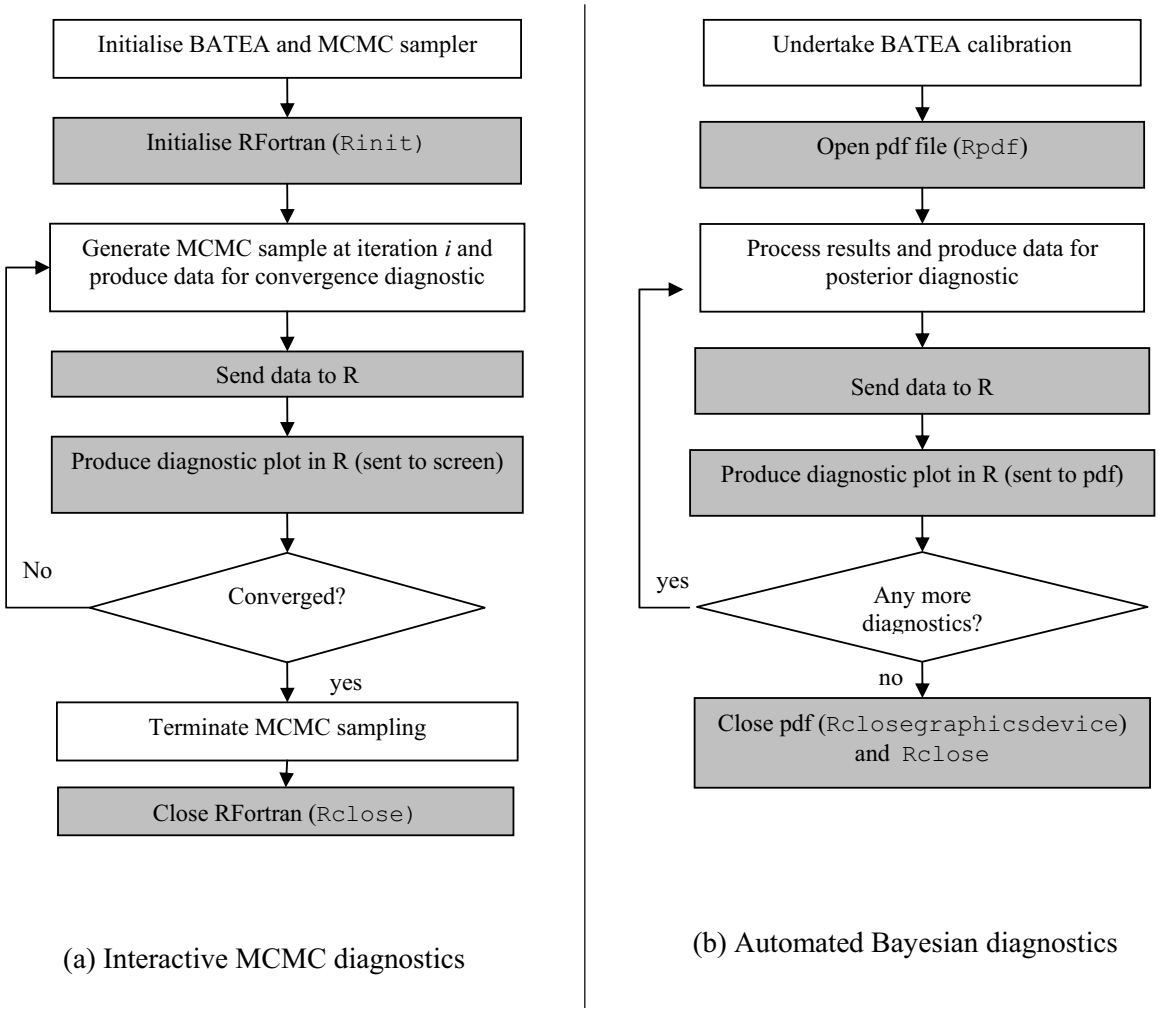


Figure 11. Flowchart of using RFortran to produce R-based MCMC and Bayesian diagnostics within the Fortran-based BATEA code. Grey-shaded boxes indicate steps using RFortran.

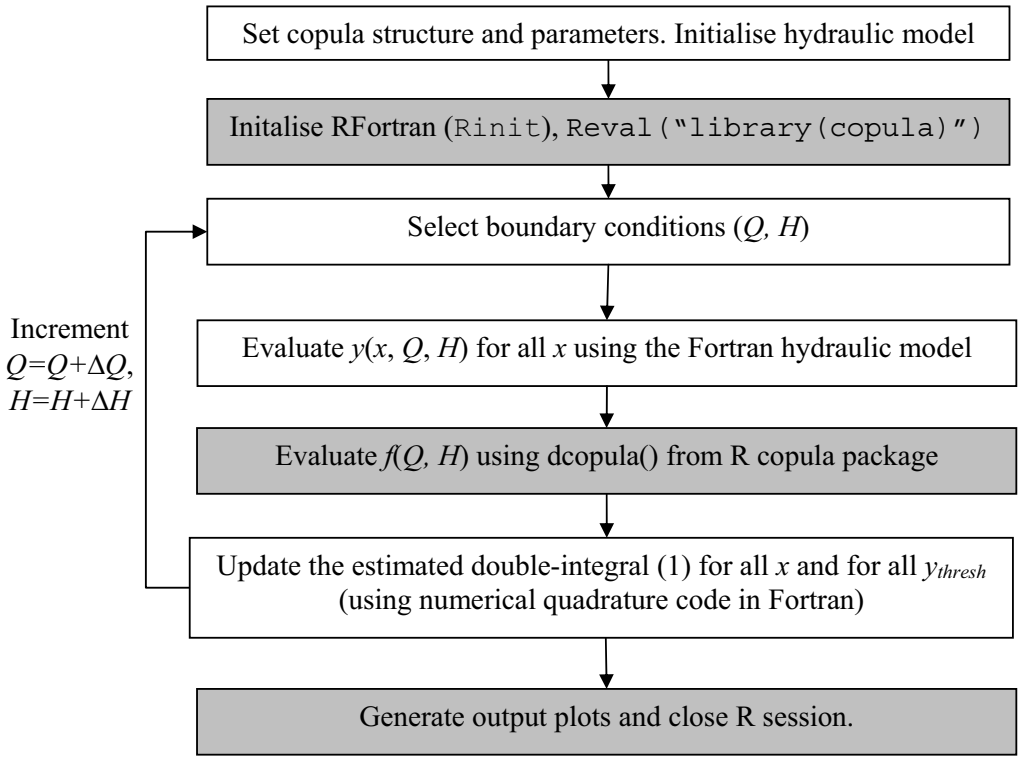


Figure 12. Flowchart of using RFortran to incorporate the R-based copula package into a Fortran-based numerical quadrature code for probabilistic flood risk analysis using hydraulic models. Grey-shaded boxes indicate steps using RFortran.

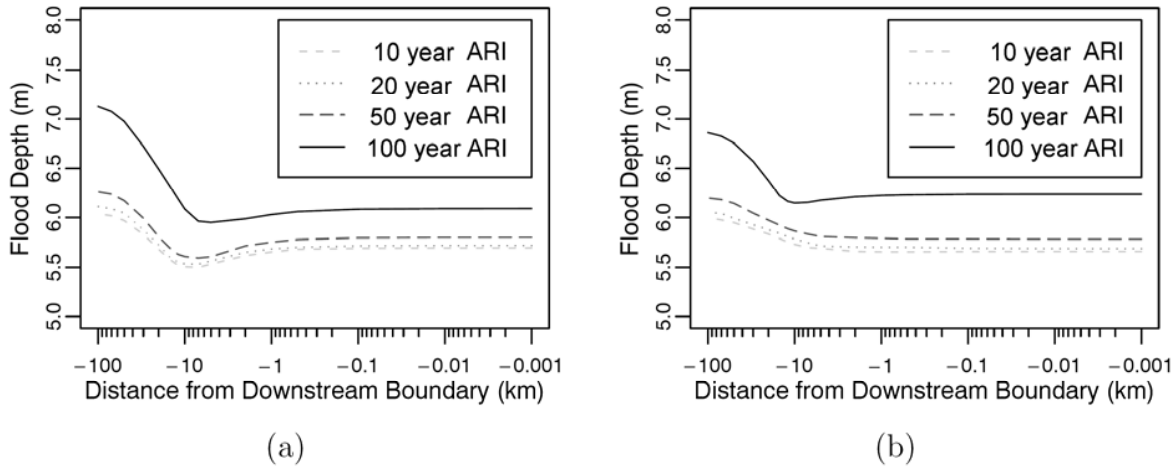


Figure 13. Flood depths for various ARIs along the length of the channel shown using a log scale from the downstream boundary for (a) independent driving forces; and (b) driving forces with correlated residuals drawn from a Gumbel copula.

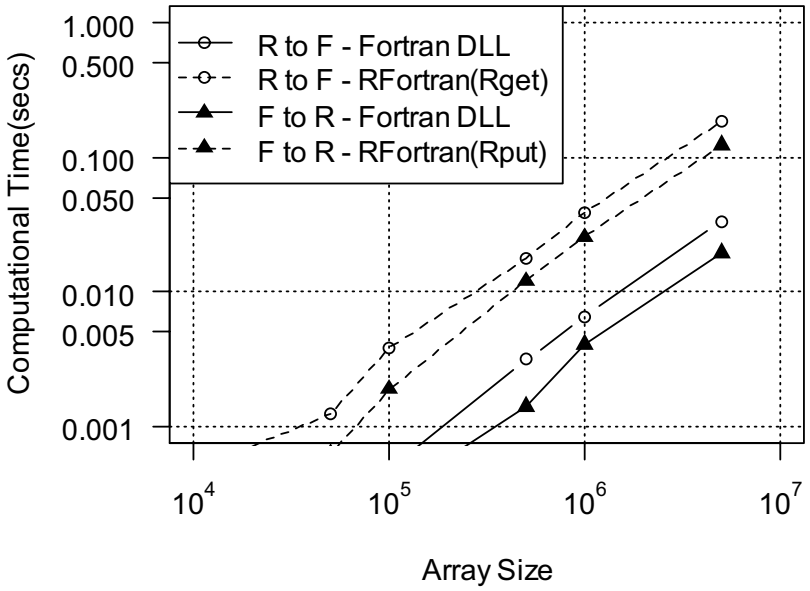


Figure 14. Comparison of array transfer times for RFortran versus Fortran DLL approaches.