AquaFortR: Streamlining Atmospheric Science, Oceanography, Climate, and Water Research with Fortran-accelerated R

Ahmed Homoudi[1]

[1]TUD Dresden University of Technology, Institute of Hydrology and Meteorology

04 August 2025

Table of contents

Preface

Welcome to the enlightening journey through the pages of **"AquaFortR: Streamlining Atmospheric Science, Oceanography, Climate, and Water Research with Fortran-accelerated R"**. It is an educational book aimed, in general, at R programmers who want to increase the performance of their codes using Fortran, particularly for bachelor's, master's, and PhD students and researchers in the fields mentioned above. Typically, simulation and modelling of the environmental processes are accomplished on the grid level in which the investigation region is discretised to numerous grid points in time and space. Consequently, these simulations produce enormous data sets and processing this data extends beyond the current average personal computer capacity. Nevertheless, a few have access to high-performance computing infrastructures. The possibility of speeding up calculations and modelling exists in each PC through compiled programming languages such as Fortran. This solution speeds up computations and can reduce the CO_2 emissions drastically. Fortran is well-suited for numerical and scientific computations due to its array processing capabilities, performance, and efficiency. Combining R with Fortran, data can be smoothly wrangled and visualised. In this book, you will gain invaluable insights into seamlessly speeding up R scripts by harnessing the power of Fortran. You will acquire essential perspectives into speeding up your package using simple Fortran codes. Furthermore, you will accumulate tweaks to accelerate your scripts or packages, and supplementary reading will prove to be both advantageous and highly beneficial for further optimisation and efficiency.

Funding

This work has been funded by the German Research Foundation (DFG) through the project NFDI4Earth (DFG project no. 460036893, https://www.nfdi4earth.de/) within the German National Research Data Infrastructure (NFDI, https://www.nfdi.de/).

Acknowledgement

Appreciation is extended to Dr. Klemens Barfus for providing invaluable Fortran routines to estimate CAPE.

License

Chapter 1

Introduction

This chapter briefly introduces the R programming language and how to install R on different operating systems. Furthermore, we will have a look at the installation of RStudio. Finally, we will learn briefly about Fortran.

The book is designed to cater to individuals with a foundational understanding of programming, irrespective of their familiarity with a specific programming language.

1.1 R

R is a powerful and versatile open-source language and environment for statistical computing and graphics (R Core Team, 2023). R was developed to facilitate data manipulation, exploration, and visualisation, providing various statistical and graphical tools.

The R environment is designed for effective data handling, array and matrix operations, and is a well-developed and simple programming language (R Core Team, 2023). Its syntax is concise and expressive, making it an accessible language for newbies and seasoned programmers. R can perform tasks either by executing scripts or interactively. The latter is advantageous for beginners and during the first stages of script development. The interactive environment is accessible through command lines or various integrated development environments (IDEs), such RStudio.

Many factors play an important role in the popularity of R. For example, the ability to produce graphics with a publication's quality. Simultaneously, the users maintain full control over customising the plots according to their preferences and intricate details. Another significant factor is its ability to be extended to meet the user's demand. The Comprehensive R Archive Network (CRAN) contains an extensive array of libraries and packages to extend R's functionality for various tasks. With over 20447 available packages (March 2024), the CRAN package repository is a testament to R's versatility and adaptability.

Tidyverse (Wickham et al., 2019; Wickham, 2023) is the most popular bundle of R packages for data science. It was developed by Hadley Wickham and his team. The common shared design among tidyverse packages increases the consistency across functions and makes each new function or package a little easier to comprehend and utilise (Wickham et al., 2023).

Many packages have been developed for spatial data analysis to address various challenging tasks. R-spatial provides a rich set of packages for handling spatial or spatiotemporal data. For example, sf (Pebesma, 2018, 2024a; Pebesma & Bivand, 2023) provides simple features access in R, which is a standardized method for encoding spatial vector data. The stars package (Pebesma, 2024b) aims to handle spatiotemporal arrays. Additionally, the terra package (Hijmans, 2024) works with spatial data and has the ability to process large datasets on the disk when loading into memory (RAM) is not feasible.

Furthermore, research produces a large data sets; therefore, it is essential to store them according to the FAIR principles (**F**indability, **A**ccessibility, **I**nteroperability, and **R**euse of digital assets; Wilkinson et al. (2016)). NetCDF and HDF5 are among the most prominent scientific data formats owing to their numerous capabilities. The ncdf4 package (Pierce, 2023) delivers a high-level R interface to data files written using Unidata's netCDF library. Additionally, rhdf5 (Fischer et al., 2023) provides an interface between HDF5 and R.

Regarding the integration of other programming languages, R has diverse interfaces, which are either a fundamental implementation of R or attainable via another R package (Chambers, 2016). The fundamental interfaces are `.Call()`, `.C()`, and `.Fortran()` to C and Fortran. The development of the Rcpp package (Eddelbuettel et al., 2024) has revolutionised seamless access to C++. Python is also accessible using the reticulate package (Ushey et al., 2024). Finally, the Java interface was granted using the rJava package (Urbanek, 2024).

In conclusion, R is a valuable tool in the Earth System, as it can effectively tackle multifarious scientific tasks and address numerous outstanding research inquiries.

1.2 Fortran

Fortran, short for **For**mula **Tran**slation, is one of the oldest high-level programming languages. It was first developed in the 1950s, and is nonetheless widely used in scientific and engineering applications. Key features of Fortran include its ability to efficiently handle arrays and matrices, making it well-suited for numerical computations. Additionally, Fortran has a simple and straightforward syntax that makes it easy to learn and use, because it is possible to write mathematical formulas almost as they written in mathematical texts (Metcalf et al., 2018).

Fortran has been revised multiple times, with the most recent iteration being Fortran 2023. Another important feature of Fortran is its support for parallel programming, enabling developers to take advantage of multicore processors and high-performance computing architectures.

There are frequently numerous good reasons to integrate different programming languages to achieve tasks. Interoperability with C programming language is a feature that was introduced with Fortran 95 (Metcalf et al., 2018). Given that C is widely used for system-level programming, many of the other languages include support for C. Therefore, the C Application Programming Interface (API) can also be used to connect two non-C languages (Chirila & Lohmann, 2014). For instance, in atmospheric modelling, Fortran is used for its high performance and capcity to handle large data sets, while C is utilised for its efficiency and control over memory usage.

Noteworthy, developing software using Fortran necessitates utilisation of its primitive procedures and developing from scratch. This is because Fortran is not similar to scripting languages (i.e. R) that requires a special environment (Masuda, 2020). However, Fortran is privileged with its persistent backward compatibility, resulting in the usability of countless (legacy) codes written decades ago.

Since R was designed to streamline data analysis and statistics (Wickham, 2015) and Fortran is renowned for its high performance, it makes seance to integrate the two languages. It should be noted that both programming languages present arrays in column-major order, which makes it easier to bridge without causing confusion. Additionally, R is developed using Fortran, C, and R programming languages, and it features `.Call` and `.External()` functions that allows users to utilise compiled code from other R packages.

Despite the development of newer programming languages, Fortran remains a popular choice for many scientists and engineers due to its reliability, efficiency, and ability to handle large amounts of data.

1.3 Installation

R

Installation of R differs according to the operating system:

- The webpage provides information on installing R according to the Linux distribution (https://cran.r-project.org/bin/linux/).
- For Windows, the executable can be downloaded from (https://cran.r-project.org/bin/windows/base/). Additionally, previous releases of R for Windows exit at https://cran.r-project.org/bin/windows/base/old/.
- For macOS, various releases and versions are accessible at https://cran.r-project.

org/bin/macosx/.

RStudio

As mentioned earlier, RStudio is an IDE for R. Although it is the most prominent, other IDEs exist, such as Jupyter Notebook and Visual Studio Code. In this book, RStudio will be the main IDE. To install Rstudio, visit the posit to download the suitable installers (https://posit.co/download/rstudio-desktop/).

Fortran

Fortran doesn't require an explicit installation, unlike interpreted languages such as R. The source code would be translated to the machine language using the Fortran compiler, and then it can be executed. Therefore, it is important to make sure that a Fortran compiler, i.e., the GNU Fortran (https://gcc.gnu.org/fortran/) (gfortran) compiler, exists in the working machine.

- In the majority of Linux distributions, the GCC compilers, including gfortran, come pre-installed.
- For Windows, installing rtools should ensure the existence of gfortran
- For macOS, binaries for gfortran are available at (https://gcc.gnu.org/wiki/GFortranBinaries). Furthermore, more information is available on R for macOS at https://cran.r-project.org/bin/macosx/tools/.

Chapter 2

Accelerate R Scripts with Fortran

In this chapter, we will compare the efficiency of running three computationally demanding examples in pure R script versus using another R script with the core computations performed in Fortran.

2.1 2D Coss-Correlation

In signal processing, cross-correlation measures similarity between two signals as a function of the displacement of one relative to the other (Wang, 2019). It can deliver information about the time lag between the two signals. 2D cross-correlation is often applied in computer vision for visual tracking. For example, it is used in template matching, feature detection, and motion tracking. 2D cross-correlation also plays an important role in convolutional networks and machine learning.

In atmospheric science, oceanography, climate, and water research, 2D cross-correlation can be applied in various ways. For example, it can be used to estimate ocean surface currents (Warren et al., 2016), cloud tracking using satellite imagery (Seelig et al., 2021), and Particle Image Velocimetry (PIV) in fluid dynamics applications (Willert & Gharib, 1991).

The 2D cross-correlation of an array $F_{(M,N)}$, and array $G_{(P,Q)}$, can be given as the array $CC_{(M+P-1,N+Q-1)}$ as shown in Equation 2.1.

$$CC_{(s,t)} = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} F_{(m,n)} G_{(m-s,n-t)} \tag{2.1}$$

where s varies between $-(P-1)$ and $(M-1)$ while t varies between $-(Q-1)$ and $(N-1)$. F and $G \in R$.

Now, let us define the `xcorr2D_r` function as shown in Listing 2.1. The function receives

two matrices or arrays `a` & `b` and return the full cross-correlation plane `cc`.

Moving forward, we can define the `xcorr2d_f` subroutine in Fortran as shown in Listing 2.2. Subroutines are generally the approach for integrating Fortran in R. Function in Fortran return a single value with no option of altering the input arguments, while subroutines have the ability to perform complex tasks while altering input arguments. This proofs to be helpful e.g., in solving equations system.

Another imperative point is to define the dimension of the arrays when passing them to Fortran (i.e. explicit-shape arrays). To illustrate, `m, n, p, q, k, l` are the dimension of input arrays `a`and `b`, and the out array `cc`.

Since Fortran is a compiled language, we need to save the subroutine in `xcorr2D.f90` file and compile it using: `R CMD SHLIB xcorr2D.f90`.

> **Note**
>
> Please use the terminal tab in Rstudio or open a new terminal using `Alt+Shift+R`

As mentioned earlier, we need to pass the dimension of the arrays to Fortran. Therefore, it would logical to write a wrapping function for Fortran subroutine that provides other input arguments.

In the wrapper function (Listing 2.3), we initially require loading the shared object (`.so` or `.dll`), which is the compiled Fortran subroutine, as `dyn.load("path/to/xcorr2D.so")`. Furthermore, it is important to prepare other input variables for Fortran such as the dimensions of the input and output arrays. Imperatively, data types should be approached carefully. Before calling `.Fortran()`, all storage mode of the variables in R was converted to the appropriate type using either `as.double()` or `as.integer()`. If the wrong type is passed, it can result in a hard-to-catch error or unexpected results[1].

> **Important**
>
> On Windows, `R CMD SHLIB` produces dynamic-link library (dll) files. Please adjust the library extension in R functions according to your OS.

Now, we can use an example to compare the performance of the two functions. In order to do so, `microbenchmark` package (Mersmann, 2024) and `ggplot2` (Wickham, 2016; Wickham et al., 2024) are required.

The obtained benchmarking data allows (`mbm`) for a quantitative comparison of the computational efficiency between the two methods. By printing "mbm" in the console (`print(mbm)`) it is evident that Fortran outperforms the R implementation of 2D

[1] Writing R Extensions, 5.2 Interface functions .C and .Fortran

cross-correlation by a factor of ~10. The significance of leveraging Fortran becomes evident in Figure 2.1.

```
library(microbenchmark)
library(ggplot2)

set.seed(72)
# Assume a
a <- structure(runif(64), dim = c(8L, 8L))
# Assume b
b <- structure(runif(64), dim = c(8L, 8L))
mbm <- microbenchmark(
  xcorr2D_r0 = xcorr2D_r0(a, b),
  xcorr2D_f0 = xcorr2D_f0(a, b)
)

autoplot(mbm) +
  stat_summary(
    fun = "median",
    geom = "crossbar",
    width = 0.6,
    colour = "red"
  )
```

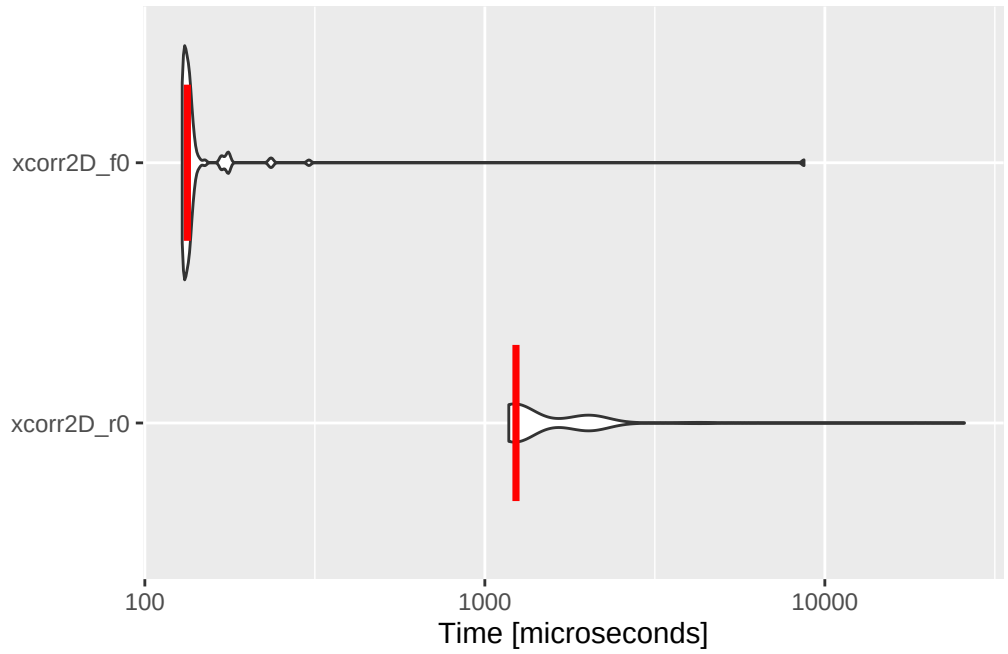

Figure 2.1: Performance comparison of 2D Cross-correlation in R and Fortran. Median is shown as red vertical line.

2.2 2D Convolution

Convolution and cross-correlation are both operations applied to two dimensional data (e.g., matrix). Cross-correlation involves sliding a kernel (filter) across a matrix, while convolution involves sliding a flipped kernel across an matrix (Draelos, 2019). Most spatial data in earth science are is discretised resulting in large data sets. Sometimes, these data sets include noise which can obscure meaningful patterns and relationships. One of the prominent methods to remove this nose while preserving important features and structures is the Gaussian smoothing filter. Gaussian smoothing is often achieved by convolution where F is the original data, and G is the kernel representing the 2D Gaussian coefficients.

The 2D convolution of an array $F_{(M,N)}$, and array $G_{(P,Q)}$, can be given as the array $Conv_{(M+P-1,N+Q-1)}$. hv means that G is flipped.

$$Conv_{(s,t)} = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} F_{(m,n)} G^{hv}_{(m-s,n-t)} \tag{2.2}$$

where s varies between $-(P-1)$ and $(M-1)$ while t varies between $-(Q-1)$ and $(N-1)$. F and $G \in R$.

Indeed, it is possible to flip the second array and utilise the functions from Section 2.1. Nevertheless, our focus is on the comprehensive workflow. Listing 2.4 presents the implementation of convolution in R, whereas Listing 2.5 demonstrates the Fortran version.

> **Note**
>
> A Gaussian smoothing filter can be applied to an array `a` using `b` as the Gaussian kernel or the 2D Gaussian coefficients. However, the convolution and cross-correlation can be optimised using the Fast Fourier Transform (FFT). See Chapter 4.

The gfortran compiler is also capable of creating shared libraries. It allows for easy addition of other flags, such as enabling the generation of the run-time check (`-fcheck=all`). The code below shows two options for compiling `conv2D.f90` by R or the gfortran compiler.

```
# R
R CMD SHLIB conv2D.f90

# gfortran on Unix-like
gfortran -shared conv2D.f90 -o conv2D.so

# gfortran on Windows
gfortran -shared conv2D.f90 -o conv2D.dll
```

In R wrapper function, `.C64()` from dotCall64 package (Gerber et al., 2018; Gerber & Mösinger, 2023) will be used instead of `.Fortran()`. According to Gerber et al. (2018), `.C64()` transcends other foreign function interfaces in many aspects:

- It supports long vectors.
- The `SIGNATURE` argument ensures that the interfaced R objects are of the specified types
- The `INTENT` argument helps avoid unnecessary copies of R objects between languages.

In Listing 2.6, the basic input arguments, such as the dimensions of input and output arrays, are prepared. Afterwards, the `SIGNATURE` is defined as six integers and three doubles corresponding to the required types in the subroutine. `INTENT` will ensure that only the `conv` argument is copied between R and Fortran. This is particularly important when processing large data set, where coping the subroutine arguments extends beyond the available memory (RAM).

```
library(microbenchmark)
library(ggplot2)


set.seed(72)
# Assume a
a <- structure(runif(64), dim = c(8L, 8L))
# Assume b
b <- structure(runif(64), dim = c(8L, 8L))
mbm <- microbenchmark(
  conv2D_r0 = conv2D_r0(a, b),
  conv2D_f0 = conv2D_f0(a, b)
)

autoplot(mbm) +
  stat_summary(
    fun = "mean",
    geom = "crossbar",
    width = 0.6,
    colour = "red"
  )
```

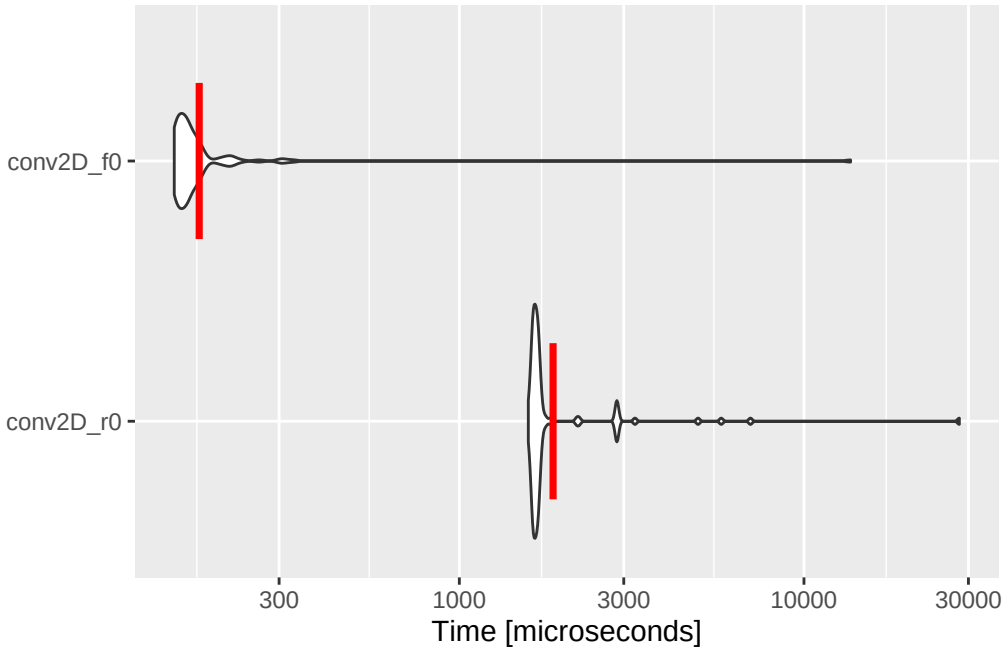

Figure 2.2: Performance comparison of 2D Convolution in R and Fortran. Median is shown as red vertical line.

Similar to cross-correlation calculation, the Fortran implementation of convolution outperforms the R one by a factor of ~10. Performing convolution in large data set using R and Fortran is beneficial since it reduce the required computational resources.

> Question
>
> After learning about `.Fortran()` and `.C64()`, you can use one of the two examples above and compare the performance of the two interfaces using `microbenchmark()`. Which function is faster?

2.3 Convective Available Potential Energy (CAPE)

According to the Glossary of Meteorology, CAPE is "the potential energy of an air parcel due to positive buoyancy, which is a useful tool for forecasting, parameterising, and estimating the potential updraft strength of convective clouds." CAPE is calculated as follows (Stull, 2016):

$$CAPE = R_d \sum_{p_{LFC}}^{p_{EL}} (T_p - T_v).ln(\frac{p_{bottom}}{p_{top}}) \tag{2.3}$$

where R_d is the gas constant for dry air, T_p is the parcel temperature, T_e is the environment temperature, p is pressure, LFC is the Level of Free Convection, and EL is the Equilibrium Level.

In a warming climate, CAPE is expected to increase (Chen et al., 2020), which can result in an elevated risk of thunderstorms. It is crucial for humanity to quantify the future risk for proper preparation and mitigation. Typically, thunderstorms are investigated with convective-permitting modelling (CPM) where the horizontal resolution is less than 4km. CPM simulations produce vast amount of data sets, and CAPE estimation at a specific gridpoint and time is an integration along the vertical profile (Equation 2.3).

Given the rapid advancements in computing power, it is anticipated that CPM is expected to be performed at finer horizontal and vertical resolution, thereby increasing the complexity of the CAPE estimation. It is essential that the enhancement of computing power is accompanied by responsible management and resource allocation.

Because the CAPE calculation scripts are highly complex and lengthy, the necessary codes are only available in the supplementary materials. Additionally, to test the two implementations of CAPE, the AquaFortR package was installed to utilise the example data. See Listing 2.7.

> **!** Important
>
> Foremost, the Fortran subroutine need to be complied as shown in previous sections. The path to the shared library cape_f.so in cape_f.R file should be adapted to the correct path.

As mentioned, integration between R and Fortran should be accomplished through subroutines. Nevertheless, some calculations are complex, and using functions or other subroutines is inevitable. In cape.f90, a module containing all the required utilities was written, and then the main subroutine was included. The approach ensures `cape_f` has access to the module and is simultaneously available to R.

Exploring Figure 2.3, it is evident that the implementation of Fortran is faster than R by a factor of ~28, proofing that integrating Fortran in R is vital for performance and beneficial for the environment.

```
library(microbenchmark)
library(ggplot2)

mbm <- microbenchmark(
  cape_r = cape_r0(t_parcel, dwpt_parcel, mr_parcel,
    Pressure, Temperature, MixingRatio,
    vtc = TRUE
  ),
  cape_f = cape_f0(t_parcel, dwpt_parcel, mr_parcel,
    Pressure, Temperature, MixingRatio,
```

```
    vtc = TRUE
  )
)

autoplot(mbm) +
  stat_summary(
    fun = "mean",
    geom = "crossbar",
    width = 0.6,
    colour = "red"
  )
```

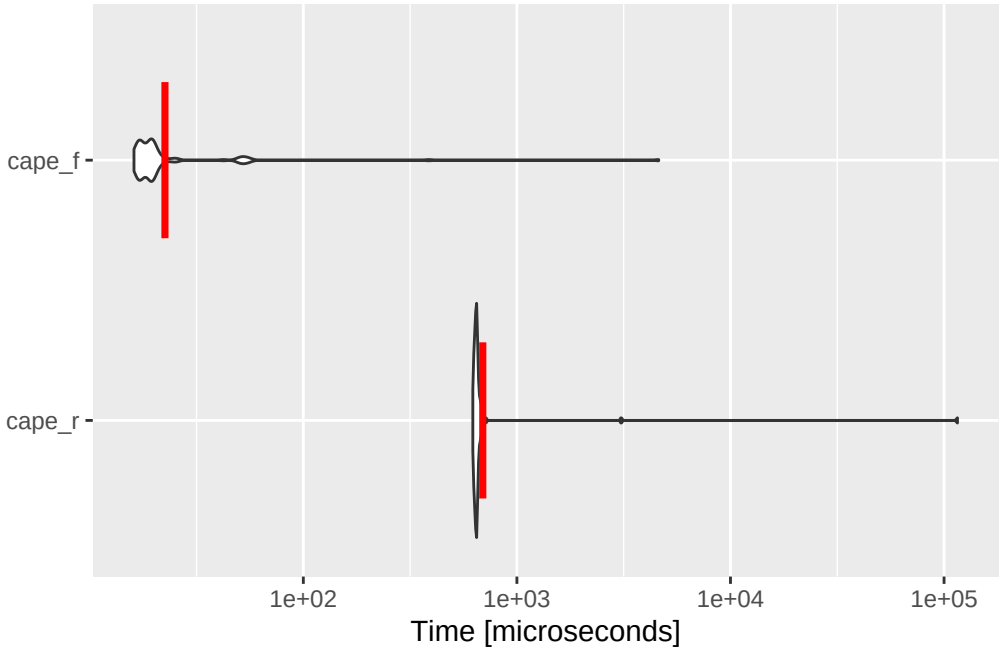

Figure 2.3: Performance comparison of CAPE in R and Fortran

Listing 2.1 Cross-correlation in R

```
xcorr2D_r0 <- function(a, b) {
  # the full CC matrix
  cc_row <- nrow(a) + nrow(b) - 1
  cc_col <- ncol(a) + ncol(b) - 1
  cc <- matrix(1:c(cc_row * cc_col),
    byrow = FALSE, ncol = cc_col
  )

  # obtain possible shifts
  min_row_shift <- -(nrow(b) - 1)
  max_row_shift <- (nrow(a) - 1)
  min_col_shift <- -(ncol(b) - 1)
  max_col_shift <- (ncol(a) - 1)

  # Padded matrix
  rows_padded <- abs(min_row_shift) +
    nrow(a) + abs(max_row_shift)
  cols_padded <- abs(min_col_shift) +
    ncol(a) + abs(max_col_shift)
  # a
  padded_a <- matrix(0,
    nrow = rows_padded,
    ncol = cols_padded
  )
  padded_a[
    (abs(min_row_shift) + 1):(abs(min_row_shift) + nrow(a)),
    (abs(min_col_shift) + 1):(abs(min_col_shift) + ncol(a))
  ] <- a

  for (icol in 1:cc_col) {
    for (irow in 1:cc_row) {
      icc <- irow + ((icol - 1) * cc_row)
      cols <- (icol):(icol + ncol(b) - 1)
      rows <- (irow):(irow + nrow(b) - 1)
      # b
      padded_b <- array(0,
        dim = c(rows_padded, cols_padded)
      )
      padded_b[rows, cols] <- b

      cc[irow, icol] <- sum(padded_a * padded_b)
    }
  }

  return(cc)
}
```

Listing 2.2 Cross-correlation in Fortran

```
subroutine xcorr2d_f(m, n, p, q, k, l, a, b, cc)
   implicit none
   integer                                  :: m, n, p, q, k, l
   double precision, dimension(m, n)        :: a
   double precision, dimension(p, q)        :: b
   double precision, dimension(k, l)        :: cc
   !     dummy vars
   integer                                 :: min_row_shift, min_col_shift
   integer                                 :: max_row_shift, max_col_shift
   integer                                 :: rows_padded, cols_padded
   integer                                 :: icol, irow, icc, icol2, irow2
   real, allocatable, dimension(:, :)      :: padded_a, padded_b

   !     obtain possible shfits
   min_row_shift = -1*(p - 1)
   max_row_shift = m - 1
   min_col_shift = -1*(q - 1)
   max_col_shift = n - 1

   !   Padded arrray
   rows_padded = abs(min_row_shift) + m + abs(max_row_shift)
   cols_padded = abs(min_col_shift) + n + abs(max_col_shift)
   !    A
   allocate (padded_a(rows_padded, cols_padded))
   padded_a = 0.0
   padded_a((abs(min_row_shift) + 1):(abs(min_row_shift) + m), &
            (abs(min_col_shift) + 1):(abs(min_col_shift) + n)) = a

   !    B
   allocate (padded_b(rows_padded, cols_padded))
   padded_b = 0.0
   do icol = 1, l
      do irow = 1, k
         icc = irow + ((icol - 1)*k)
         icol2 = icol + q - 1
         irow2 = irow + p - 1
         padded_b(irow:irow2, icol:icol2) = b
         cc(irow, icol) = sum(padded_a*padded_b)
         padded_b = 0.0
      end do
   end do
end subroutine xcorr2d_f
```

Listing 2.3 Cross-correlation wrapping function

```
xcorr2D_f0 <- function(a, b) {
  # Please adjust the path to your setup.
  dyn.load("AquaFortR_Codes/xcorr2D.so")

  # the full CC matrix
  cc_row <- nrow(a) + nrow(b) - 1
  cc_col <- ncol(a) + ncol(b) - 1
  cc <- matrix(1:c(cc_row * cc_col), byrow = FALSE, ncol = cc_col)

  cc<- .Fortran("xcorr2d_f",
    m = as.integer(dim(a)[1]),
    n = as.integer(dim(a)[2]),
    p = as.integer(dim(b)[1]),
    q = as.integer(dim(b)[2]),
    k = as.integer(cc_row),
    l = as.integer(cc_row),
    a = as.double(a),
    b = as.double(b),
    cc = as.double(cc)
  )$cc

  return(cc)
}
```

Listing 2.4 Convolution in R

```
conv2D_r0 <- function(a, b) {
  # the full convolution matrix
  conv_row <- nrow(a) + nrow(b) - 1
  conv_col <- ncol(a) + ncol(b) - 1
  conv <- matrix(1:c(conv_row * conv_col), byrow = FALSE, ncol = conv_col)

  # obtain possible shifts
  min_row_shift <- -(nrow(b) - 1)
  max_row_shift <- (nrow(a) - 1)
  min_col_shift <- -(ncol(b) - 1)
  max_col_shift <- (ncol(a) - 1)

  # Padded matrix
  rows_padded <- abs(min_row_shift) + nrow(a) + abs(max_row_shift)
  cols_padded <- abs(min_col_shift) + ncol(a) + abs(max_col_shift)
  # a
  padded_a <- matrix(0, nrow = rows_padded, ncol = cols_padded)
  padded_a[
    (abs(min_row_shift) + 1):(abs(min_row_shift) + nrow(a)),
    (abs(min_col_shift) + 1):(abs(min_col_shift) + ncol(a))
  ] <- a

  for (icol in 1:conv_col) {
    for (irow in 1:conv_row) {
      iconv <- irow + ((icol - 1) * conv_row)
      cols <- (icol):(icol + ncol(b) - 1)
      rows <- (irow):(irow + nrow(b) - 1)
      # b
      padded_b <- array(0, dim = c(rows_padded, cols_padded))
      # flip the kernel i.e. b
      padded_b[rows, cols] <- b[nrow(b):1, ncol(b):1]

      conv[irow, icol] <- sum(padded_a * padded_b)
    }
  }

  return(conv)
}
```

Listing 2.5 Convolution in Fortran

```
subroutine conv2d_f(m, n, p, q, k, l, a, b, conv)
   implicit none
   integer                                  :: m, n, p, q, k, l, i, j
   double precision, dimension(m, n)        :: a
   double precision, dimension(p, q)        :: b
   double precision, dimension(k, l)        :: conv
   !     dummy vars
   integer                                 :: min_row_shift, min_col_shift
   integer                                 :: max_row_shift, max_col_shift
   integer                                 :: rows_padded, cols_padded
   integer                                 :: icol, irow, iconv, icol2, irow2
   real, allocatable, dimension(:, :)      :: padded_a, padded_b

   !     obtain possible shfits
   min_row_shift = -1*(p - 1)
   max_row_shift = m - 1
   min_col_shift = -1*(q - 1)
   max_col_shift = n - 1


   !   Padded arrray
   rows_padded = abs(min_row_shift) + m + abs(max_row_shift)
   cols_padded = abs(min_col_shift) + n + abs(max_col_shift)
   !    A
   allocate (padded_a(rows_padded, cols_padded))
   padded_a = 0.0
   padded_a((abs(min_row_shift) + 1):(abs(min_row_shift) + m), &
            (abs(min_col_shift) + 1):(abs(min_col_shift) + n)) = a

   !    B
   allocate (padded_b(rows_padded, cols_padded))
   padded_b = 0.0
   do icol = 1, l
      do irow = 1, k
         iconv = irow + ((icol - 1)*k)
         icol2 = icol + q - 1
         irow2 = irow + p - 1
         padded_b(irow:irow2, icol:icol2) = b(p:1:-1,q:1:-1)
         conv(irow, icol) = sum(padded_a*padded_b)
         padded_b = 0.0
      end do
   end do
end subroutine conv2d_f
```

Listing 2.6 Convolution wrapping function

```
conv2D_f0 <- function(a, b) {
  require(dotCall64)
  dyn.load("AquaFortR_Codes/conv2D.so")

  m <- nrow(a)
  n <- ncol(b)

  p <- nrow(b)
  q <- ncol(b)
  # the full convolution matrix
  conv_row <- m + p - 1
  conv_col <- n + q - 1
  conv <- matrix(0,
    ncol = conv_col,
    nrow = conv_row
  )

  conv <- .C64("conv2d_f",
    SIGNATURE = c(
      rep("integer", 6),
      rep("double", 3)),
    INTENT = c(rep("r",8), "rw"),
    m, n, p, q,
    k = conv_row,
    l = conv_col,
    a = a, b = b,
    conv = conv
  )$conv

  return(conv)
}
```

Listing 2.7 CAPE implementation in R and Fortran

```
if (!require(AquaFortR)) {
  remotes::install_github("AHomoudi/AquaFortR", subdir = "RPackage")
}

library(AquaFortR)
data("radiosonde")

Temperature <- radiosonde$temp + 273.15 # K
Dewpoint <- radiosonde$dpt + 273.15 # K
Pressure <- radiosonde$pressure # hPa
# Mixing ratio
MixingRatio <- mixing_ratio_from_dewpoint(Dewpoint, Pressure)
t_parcel <- Temperature[1]
dwpt_parcel <- Dewpoint[1]
mr_parcel <- MixingRatio[1]

source("AquaFortR_Codes/cape_r.R")
source("AquaFortR_Codes/cape_f.R")
```

Chapter 3

Accelerate R Packages with Fortran

The chapter focuses on wrapping the routines developed in the previous chapter in the R package.

3.1 Introduction

A package in R is a bundle of R code, data, and documentation designed to perform a specific task or a set of tasks (Wickham & Bryan, 2023). Packages are the fundamental units of reproducible R code. Packaging has many benefits:

- It produces packages that are easily downloaded and used.
- It forces a tidy code and work process.
- It enables the re-usage of code from and for other projects.
- Files rapidly multiply when using foreign languages such as C, C++, or Fortran; thus, packaging makes it easier to maintain and manage dependencies and ports.

Many tools are available to facilitate developing R packages, such as `devtools`, `usethis`, and `testthat`. Additionally, RStudio operates as an invaluable companion. To start the development of an R package within RStudio, navigate to File > New Project > New Directory > R Package, and then proceed to create the project. Alternatively, you can use the `devtools::create("/path/to/package/location/")` or `usethis::create_package("/path/to/package/location/")` functions.

By employing `usethis::use_c()`, you can easily incorporate the necessary infrastructure to utilise compiled code. It's vital to ensure that your package is properly licensed. Various license templates are available through `usethis`, including the Creative Commons (e.g., CC BY 4.0), which can be implemented via `usethis::use_ccby_license()`.

Figure 3.1 shows a skeleton of a typical R package called `foo`. R code is placed in the `R/` directory, while the compiled code resides in the `src/` directory. R provides a standard-

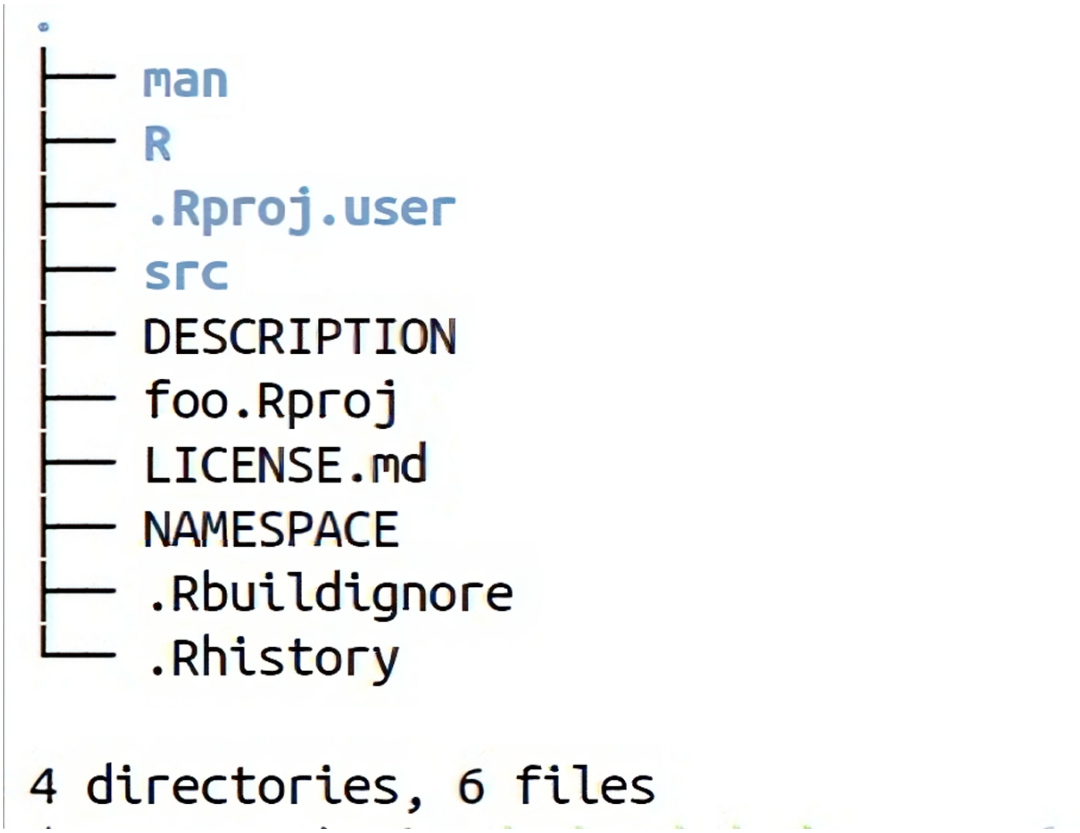

Figure 3.1: R package structure

ised method of documenting packages. The `DESCRIPTION` is generated with the package template and contains essential documentation, including the package's title, what the package does, the author, and dependencies. Additionally, the `SystemRequirements` is important in case the package depends on libraries or tools external to R, e.g., FFTW or GNU make.

The R documentation files `.Rd` are stored within the `man/` folder. When employing `devtools` for package development, the `.Rd` files are automatically modified when the specially formatted "roxgen comments" above R source codes are modified (Wickham & Bryan, 2023). Thus, it is important to run `devtools::document()` whenever a function is completed. The `NAMESPACE` file is a fundamental component of a package. It delivers a context for looking up the value of an object associated with a name (Wickham & Bryan, 2023). Typically, it is modified automatically by `roygen2` and contains routines such as `export` to make the package functions visible, `import` to import all objects from another package's namespace or `importForm` to import selected objects, and `useDynLib()` for packages with compiled code to register routines from DLL.

The second edition of the R Packages book by Hadley Wickham and Jennifer Bryan, particularly the chapter "The Whole Game", offers essential insights and presents a comprehensive overview of creating an R package.

3.2 Developing AquaFortR

In this section, we will wrap the routines developed in Chapter 2 within an R package. For simplicity, we will focus on implementing 2D cross-correlation here. In the package ecosystem, it is advantageous to avoid the `.Fortran` interface. The interface is mainly tailored for Fortran 77, predates any support for 'modern' Fortran, and carries a significant overhead. Therefore, we will use `.Call` the modern interface to C/C++.

The development consists mainly of three stages:

- Writing the Fortran subroutine
- Crafting C functions to interact between R and Fortran
- And finally, creating an R function to call the C function.

3.2.1 Fortran Subroutine

The best approach to organising Fortran code with an R package is to use modules, storing each module in individual files. The intrinsic module `iso_c_binding` ensures that precisely the same variable type and kind is used in C and Fortran.

Listing 3.1 shows the structure of the Fortran file `AquaFortRmodulef.f90`. Fortran files can multiply rapidly, and thus, managing module dependencies can be a tedious task. Numbering the files and writing the routines sequentially, such as `AquaFortRmodulesf001.f90` to `AquaFortRmodulesf00n.f90`, can serve as a practical workaround.

Listing 3.1 Fortran file structure

```
module AquaFortRmodule
   use, intrinsic :: iso_c_binding
   implicit none
contains
   ! place subroutines and functions
   ! ...
   ! ...
   ! ...
end module AquaFortRmodule
```

Listing 3.2 shows the subroutine `xcorr2d_f`. The differences in comparison to the script version in Listing 2.2 are:

- The kind of the variable is defined according to `iso_c_binding`. Integers are of the `c_int` kind, while double precision variables are defined as real with the `c_double` kind.
- Typically, variables are passed to Fortran (C) by reference (value). The `value` attribute allows passing by value to Fortran.

- Using `bind()` facilitates the interoperability of Fortran procedures with C. The binding label is the name by which the C processer recognises the Fortran procedure. The `F77_` prefixes in C add a trailing underscore.

Listing 3.2 Cross-correlation in Fortran in R package

```
subroutine xcorr2d_f(m, n, p, q, k, l, a, b, cc) bind(C, name="xcorr2d_f_")
  implicit none
  integer(kind=c_int), intent(in), value                :: m, n, p, q, k, l
  real(kind=c_double), intent(in), dimension(m, n)      :: a
  real(kind=c_double), intent(in), dimension(p, q)      :: b
  real(kind=c_double), intent(out), dimension(k, l)     :: cc
  !     dummy vars
  integer(kind=c_int)        :: min_row_shift, min_col_shift
  integer(kind=c_int)        :: max_row_shift, max_col_shift
  integer(kind=c_int)        :: rows_padded, cols_padded
  integer(kind=c_int)        :: icol, irow, icc, icol2, irow2
  real(kind=c_double), allocatable, dimension(:, :)     :: padded_a, padded_b

  ! The rest of the subroutine is similar to Listing 2.2
  ! or it can be found in src/AquaFortRmodulef.f90
end subroutine xcorr2d_f
```

3.2.2 C Functions

Now, we write C functions that communicate between Fortran and R. The R API has many entry points for the C code. To use the public stable API, the header files in Listing 3.3 should be included in the `AquaFortRmodulec.c`. Additionally, `stdlib.h` is included for `NULL`. It is recommended to use `R_NO_REMAP` so all API functions have the prefix `R_` or `Rf_`.

Listing 3.3 C headers

```
#define R_NO_REMAP
#include <R.h>
#include <Rinternals.h> // Access to "public" internal API
#include <stdlib.h> // for NULL
```

`F77_NAME()` is used to declare the Fortran subroutine in C. Note that variables with value attribute should not be pointers, as shown in Listing 3.4.

Listing 3.4 Declaring Fortran routine in C

```
void F77_NAME(xcorr2d_f)(int m, int n, int p, int q, int k, int l,
                         double *a, double *b, double *cc);
```

In Listing 3.5, the `c_xcorr2d_f` function communicates between R and C, and invoke the declared Fortran subroutine in C. Everything that moves between R and C should be `SEXP`. Additional variables required by the Fortran subroutine, such as matrix dimensions, are

created within the C code. The `F77_CALL()` macro is to call the Fortran routine, declared by `F77_NAME()`, from C. The `PROTECT` and `UNPROTECT` macros are required to save the result, i.e., `ret`, from being destroyed during R's garbage collection. Variables of type `double` need to be altered to `real` before being passed to Fortran.

Listing 3.5 Calling Fortran subroutine in C

```
extern SEXP c_xcorr2d_f(SEXP a, SEXP b)
{
    int m = Rf_nrows(a);
    int n = Rf_ncols(a);
    //
    int p = Rf_nrows(b);
    int q = Rf_ncols(b);
    //
    int k = m + p - 1;
    int l = n + q - 1;

    SEXP ret;
    PROTECT(ret = Rf_allocMatrix(REALSXP, k, l));
    F77_CALL(xcorr2d_f)
    (m, n, p, q, k, l, REAL(a), REAL(b), REAL(ret));
    UNPROTECT(1);
    return (ret);
}
```

Registration of native routines for compiled code is achieved in two stages: first, creating an array describing individual routines using `R_CallMethodDef`; then, actually registering the routines with R using `R_registerRoutines`, as shown in Listing 3.6. Registration's benefits include a faster way to find the address of the entry point and a run-time check concerning the number and type of arguments. The `R_useDynamicSymbols` routine instructs the `.Call` function to find only registered routines, even when no package is provided, preventing unnecessary delay due to searching.

For further information, please refer to Writing R Extension: 5.4 Registering native routines.

> **!** Important
>
> All the C code in Section 3.2.2 is stored within the `AquaFortRmodulec.c` file.

3.2.3 R Function

After finishing the Fortran subroutine and C functions, we can write the R function, as shown in Listing 3.7. The first few lines contain the roxgen2 documentation for the function. We use the `.Call` function to invoke the C function `c_xcorr2d_f` that will invoke a Fortran subroutine `xcorr2d_f`.

Listing 3.6 Registration of native routines

```
static const R_CallMethodDef CallEntries[] = {
    {"c_xcorr2d_f", (DL_FUNC)&c_xcorr2d_f, 2},
    {"c_conv2d_f", (DL_FUNC)&c_conv2d_f, 2},
    {"c_cape_f", (DL_FUNC)&c_cape_f, 7},
    {NULL, NULL, 0}};

void R_init_AquaFortR(DllInfo *dll)
{
    R_registerRoutines(dll, NULL, CallEntries, NULL, NULL);
    R_useDynamicSymbols(dll, FALSE);
}
```

It is crucial to add the following line to the package-level documentation (i.e.`AquaFortR-package.R` file) to load the DLL and define them in the package's namespace.

```
#' @useDynLib AquaFortR, .registration=TRUE
```

This is the end of the general workflow for developing an R package with Fortran code using the `.Call` interface. In the `src/` directory, Fortran modules reside in .f90 files, while C routines are found in .c files. Some packages store their C code in an init.c file. All R code should be kept in the `R/` directory.

3.2.4 System Dependencies

Occasionally, you develop a package that needs system-dependent configuration or libraries before installation, such as FFTW. An executable is then executed by `R CMD INSTALL` before any other action is performed to ensure the successful installation of the package. R allows `/configure` scripts to check for these dependencies. The `/configure` script can be written manually or using Autoconf.

Aaron Jacobs wrote about "An Autoconf Primer for R Package Authors." It is a vital tutorial on developing a `/configure` script for an R package. Furthermore, detailed information concerning configuration and cleanup is available in Writing R extensions, 1.2 Configure-and-cleanup.

In case the configure script requires auxiliary files, it is advised that they should be shipped with the R package in the tools directory. Numerous macros are available at the GNU Autoconf Archive.

Listing 3.7 Calling C in R

```
#' @title 2D cross-correlation using Fortran.
#'
#' @description  Calculates the 2D cross-correlation
#'  of two matrices `a` and `b` using compiled Fortran subroutine.
#'
#' @param a A matrix (2D array) of values.
#' @param b A matrix (2D array) of values.
#' @return A matrix representing the 2D cross-correlation of
#' the input matrices.
#' @export
#' @examples
#' a <- matrix(c(1, 2, 3, 4), ncol = 2)
#' b <- matrix(c(5, 6, 7, 8), ncol = 2)
#' xcorr2D_f(a, b)
#' @author Ahmed Homoudi
#' @export
xcorr2D_f <- function(a, b) {
  stopifnot(length(dim(a)) == 2 | length(dim(b)) == 2)
  result <- .Call(
    c_xcorr2d_f,
    a,
    b
  )
  return(result)
}
```

Chapter 4

Conclusions and Optimization Insights

AquaFortR is an educational project for students and researchers in Atmospheric Science, Oceanography, Climate, and Water Research. It aims to demonstrate that simple Fortran scripts can meet the demand for accelerating R, especially considering that most data sets in these fields consist of large discretised arrays representing earth system processes.

Fortran is one of the fastest programming languages, and multidimensional arrays are a core part of it. Fortran subprogram calls are based on call by reference. In simpler terms, they directly modify the variables in memory, and no additional space is allocated, saving a lot of memory when dealing with immense arrays.

When integrating Fortran in R scripts, the old `.Fortran` interface should be avoided. Instead, the `dotCall64::.C64` interface should be utilised. It supports long vectors and type 64-bit integers and provides a mechanism to avoid excessive argument copying. Another option to integrate Fortran in R is packaging and employing `.Call`, the modern C/C++ interface. Packaging is advantageous since it delivers numerous benefits like code tidiness and reusability.

Integrating Fortran in R provides access to the Open Multi-Processing (OpenMP), a standardised API for writing shared-memory multi-process applications (i.e. all processors share memory and data). The R package Romp, by Drew Schmidt, presents introductory OpenMP implementation with R for C, C++, F77, and Fortran 2003.

Nowadays, multicore CPUs are easily accessible. With their proliferation, harnessing the capability of parallelism through OpenMP is a practical reality. For example, the performance of CAPE estimation can easily be improved by passing the atmospheric profiles from R to Fortran (i.e. array[latitude, longitude, level, time]) and distributing the calculation among cores, simultaneously exploiting the power of Fortran and parallelism. Moreover, the convolution of precipitation data can be sped up by passing the data to

Fortran and sharing the calculation along the time axis among cores.

Cross-correlation and convolution can be optimised by utilising the Fast Fourier Transform (FTT). The computational efficiency originates from the fact that FFT reduces the computation from O[N^2] operations to O[$N\log_2 N$] operations. It is possible to use the FFTW C subroutine library to compute the discrete Fourier transform (DFT) in one or more dimensions, either in Fortran or C. Furthermore, Fortran can directly utilise linear algebra libraries such as BLAS, LAPACK, and LINPACK.

R is a versatile, growing, and expanding language and environment for statistical computing and graphics. It excels in wrangling data and generating publication-quality visualisations (e.g., `ggplot2`), making it a standout choice. While it is primarily focused on flexibility and functionality rather than performance, integrating Fortran compiled codes can render substantial speed enhancements.

Chapter 5

Further Reading

- Fortran and R – Speed Things Up by Steve Pittard (https://www.r-bloggers.com/2014/04/fortran-and-r-speed-things-up/)
- The Need for Speed Part 1: Building an R Package with Fortran by Avraham Adler (https://www.r-bloggers.com/2018/12/the-need-for-speed-part-1-building-an-r-package-with-fortran-or-c/)
- The Need for Speed Part 2: C++ vs. Fortran vs. C by Avraham Adler (https://www.avrahamadler.com/2018/12/23/the-need-for-speed-part-2-c-vs-fortran-vs-c/)
- The R Manuals edited by the R Development Core Team (https://cran.r-project.org/manuals.html)
- Writing R Extensions by R Core Team (https://cran.r-project.org/doc/manuals/r-release/R-exts.html#Writing-R-Extensions)
- R internals by Hadley Wickham (https://github.com/hadley/r-internals)
- How to write your own R package and publish it on CRAN by Cosima Meyer & Dennis Hammerschmidt (https://www.mzes.uni-mannheim.de/socialsciencedatalab/article/r-package/#section1)
- Advanced R by Hadley Wickham (http://adv-r.had.co.nz/)
- Modern Fortran Tutorial by Yutaka Masuda (https://masuday.github.io/fortran_tutorial/index.html)
- Extend R with Fortran by Yutaka Masuda (https://masuday.github.io/fortran_tutorial/r.html)
- Fortran 90 Tutorial by Stanford University (http://web.stanford.edu/class/me200c/tutorial_90/)
- Fortran Libraries by Fortran Wiki (https://fortranwiki.org/fortran/show/Libraries)
- Fortran Best Practices by Fortran Community (https://fortran-lang.org/en/learn/best_practices/#fortran-best-practices)
- Fortran 90 Reference Card by Michael Goerz. (https://web.pa.msu.edu/people/duxbury/courses/phy480/fortran90_refcard.pdf)

- Hands-On Programming with R by Garrett Grolemund (https://rstudio-education.github.io/hopr/)
- r-spatial by Edzer Pebesma, Marius Appel, and Daniel Nüst (https://r-spatial.org/)
- Spatial Data Science: With Applications in R by Edzer Pebesma and Roger Bivand (https://r-spatial.org/book/)
- R for Data Science by Hadley Wickham, Mine Çetinkaya-Rundel, and Garrett Grolemund (https://r4ds.hadley.nz/)
- Introduction to Environmental Data Science by Jerry Davis (https://bookdown.org/igisc/EnvDataSci/)

References

Chambers, F., John M. CRC Press Boca Raton. (2016). *Extending r*. CRC Press, Taylor & Francis Group. https://www.routledge.com/Extending-R/Chambers/p/book/9781498775717

Chen, J., Dai, A., Zhang, Y., & Rasmussen, K. L. (2020). Changes in convective available potential energy and convective inhibition under global warming. *Journal of Climate*, *33*(6), 2025–2050. https://doi.org/10.1175/JCLI-D-19-0461.1

Chirila, D. B., & Lohmann, G. (2014). *Introduction to modern fortran for the earth system sciences*. Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-37009-0

Draelos, R. (2019). Convolution vs. Cross-correlation. In *Glass Box*. https://glassboxmedicine.com/2019/07/26/convolution-vs-cross-correlation/

Eddelbuettel, D., Francois, R., Allaire, J., Ushey, K., Kou, Q., Russell, N., Ucar, I., Bates, D., & Chambers, J. (2024). *Rcpp: Seamless r and c++ integration*. https://www.rcpp.org

Fischer, B., Smith, M., & Pau, G. (2023). *rhdf5: R interface to HDF5*. https://doi.org/10.18129/B9.bioc.rhdf5

Gerber, F., Moesinger, K., & Furrer, R. (2018). dotCall64: An R package providing an efficient interface to compiled C, C++, and Fortran code supporting long vectors. *SoftwareX*, *7*, 217–221. https://doi.org/10.1016/j.softx.2018.06.002

Gerber, F., & Mösinger, K. (2023). *dotCall64: Enhanced foreign function interface supporting long vectors*. https://git.math.uzh.ch/reinhard.furrer/dotCall64

Hijmans, R. J. (2024). *Terra: Spatial data analysis*. https://rspatial.org/

Masuda, Y. (2020). *Modern fortran tutorial*. Introduction. https://masuday.github.io/fortran_tutorial/introduction.html

Mersmann, O. (2024). *Microbenchmark: Accurate timing functions*. https://github.com/joshuaulrich/microbenchmark/

Metcalf, M., Reid, J., & Cohen, M. (2018). *Modern Fortran Explained: Incorporating Fortran 2018*. Oxford University Press. https://doi.org/10.1093/oso/9780198811893.001.0001

Pebesma, E. (2018). Simple Features for R: Standardized Support for Spatial Vector Data. *The R Journal*, *10*(1), 439–446. https://doi.org/10.32614/RJ-2018-009

Pebesma, E. (2024a). *Sf: Simple features for r*. https://r-spatial.github.io/sf/

Pebesma, E. (2024b). *Stars: Spatiotemporal arrays, raster and vector data cubes*. https://r-spatial.github.io/stars/

Pebesma, E., & Bivand, R. (2023). *Spatial Data Science: With applications in R*. Chapman and Hall/CRC. https://doi.org/10.1201/9780429459016

Pierce, D. (2023). *ncdf4: Interface to unidata netCDF (version 4 or earlier) format data files*. https://cirrus.ucsd.edu/~pierce/ncdf/

R Core Team. (2023). *R: A language and environment for statistical computing*. R Foundation for Statistical Computing. https://www.R-project.org/

Seelig, T., Deneke, H., Quaas, J., & Tesche, M. (2021). Life Cycle of Shallow Marine Cumulus Clouds From Geostationary Satellite Observations. *Journal of Geophysical Research: Atmospheres*, *126*(22). https://doi.org/10.1029/2021JD035577

Stull, R. (2016). *Practical meteorology: An algebra-based survey of atmospheric science*. AVP International, University of British Columbia. https://www.eoas.ubc.ca/books/Practical_Meteorology/

Urbanek, S. (2024). *rJava: Low-level r to java interface*. http://www.rforge.net/rJava/

Ushey, K., Allaire, J., & Tang, Y. (2024). *Reticulate: Interface to python*. https://rstudio.github.io/reticulate/

Wang, C. (2019). *Kernel learning for visual perception*. https://doi.org/10.32657/10220/47835

Warren, M. A., Quartly, G. D., Shutler, J. D., Miller, P. I., & Yoshikawa, Y. (2016). Estimation of ocean surface currents from maximum cross correlation applied to GOCI geostationary satellite remote sensing data over the tsushima (korea) straits. *Journal of Geophysical Research: Oceans*, *121*(9), 6993–7009. https://doi.org/https://doi.org/10.1002/2016JC011814

Wickham, H. (2015). *Advanced r*. CRC Press. https://www.oreilly.com/library/view/advanced-r/9781466586963/

Wickham, H. (2016). *ggplot2: Elegant graphics for data analysis*. Springer-Verlag New York. https://ggplot2.tidyverse.org

Wickham, H. (2023). *Tidyverse: Easily install and load the tidyverse*. https://tidyverse.tidyverse.org

Wickham, H., Averick, M., Bryan, J., Chang, W., McGowan, L. D., François, R., Grolemund, G., Hayes, A., Henry, L., Hester, J., Kuhn, M., Pedersen, T. L., Miller, E., Bache, S. M., Müller, K., Ooms, J., Robinson, D., Seidel, D. P., Spinu, V., … Yutani, H. (2019). Welcome to the tidyverse. *Journal of Open Source Software*, *4*(43), 1686. https://doi.org/10.21105/joss.01686

Wickham, H., & Bryan, J. (2023). *R packages: Organize, test, document, and share your code*. O'Reilly.

Wickham, H., Çetinkaya-Rundel, M., & Grolemund, G. (2023). *R for data science import, tidy, transform, visualize, and model data* (2nd edition). O'Reilly Media.

Wickham, H., Chang, W., Henry, L., Pedersen, T. L., Takahashi, K., Wilke, C., Woo,

K., Yutani, H., Dunnington, D., & van den Brand, T. (2024). *ggplot2: Create elegant data visualisations using the grammar of graphics*. https://ggplot2.tidyverse.org

Wilkinson, M. D., Dumontier, M., Aalbersberg, Ij. J., Appleton, G., Axton, M., Baak, A., Blomberg, N., Boiten, J.-W., da Silva Santos, L. B., Bourne, P. E., Bouwman, J., Brookes, A. J., Clark, T., Crosas, M., Dillo, I., Dumon, O., Edmunds, S., Evelo, C. T., Finkers, R., … Mons, B. (2016). The FAIR Guiding Principles for scientific data management and stewardship. *Scientific Data*, *3*(1), 160018. https://doi.org/10.1038/sdata.2016.18

Willert, C. E., & Gharib, M. (1991). Digital particle image velocimetry. *Experiments in Fluids*, *10*(4), 181–193. https://doi.org/10.1007/BF00190388