# R package development of the shrinkISO method

02 January, 2022

Jipke H. van Lanen, 5489342

Supervised by: Renaud Tissier, Renee de Menezes and Adrien Melquiond

# Abstract

By varying the location of splice sites, a single gene can produce different transcripts. This alternative splicing leads to the production of different protein variants. Genomic differences between cells can lead to differential exon usage (DEU): a difference in the distribution of exon counts from the same gene. The shrinkISO method was developed to find DEU using an empirical Bayesian approach, mixed model methodology and integrated nested Laplacian approximation at the gene-level. The aim of this project was to develop an R package for this method. Prior to the development of the package, the method was used in the DEU analysis of chromosome 22. Results of this analysis were used in the development of the package manual. The package improves the replicability as well as the usability of the method. It will form a starting point for a detailed paper on the shrinkISO method.

# Layman's Summary

Our genome is still a mysterious puzzle. Scientists have been working on solving this puzzle piece by piece for many years. Common understanding is that a single gene contains a DNA code that can be translated to just a single protein, but that is not entirely true. A gene does not translate to just one protein, but genes can split and stitch their code together in different ways. As a result, a single gene can translate to multiple, slightly different proteins, possibly with different functions. The parts of the gene that are split and stitched are called *exons*, and when this splitting and stitching happens in different ways it is called *alternative splicing*.

Alternative splicing is necessary to create the complexity of many living creatures. It makes sure that there are enough possibilities, so that our heart cells and our brain cells can do different things. When a gene uses a different way of alternative splicing in different groups of cells, this gene has *differential exon usage*.

We can try to estimate differential exon usage by comparing two groups of different cells, using a method called *shrinkISO*. This method consists of a set of functions with statistical calculations. In order to make it easier to use this method, we will bundle the functions together in a package.

In this project we show the development of the package. Because the focus of the project was on the development, it includes an extensive method in which we explain how package development works. It further covers a small analysis, to get familiar with the method. Based on the results of this analysis, we chose a set of genes to use as an example in the package manual. The manual gives a good idea of the content of the package and how to use it. The package is an improvement of the method, because it is much easier to use and to share.

# Introduction

A single gene can produce different mRNAs by variations of the splicing sites. This leads to the production of different protein variants, which may have different cellular functions. Different combinations of exons produce mRNA isoforms that diversify the transcriptome. This tightly regulated biological process is called *alternative splicing*.[1,2] Genomic differences or genetic mutations can lead to *differential exon usage* or *DEU*: a difference in the distribution of exon counts from the same gene between two groups.[1–3] Annotation of mRNA sequences can be used to map genome wide expression levels. By analyzing the expression on exon-level for each gene, two groups can be compared for DEU.[4] However, there are not many models that perform this analysis.

The shrinkISO method was developed to find DEU from exon count data. It is an empirical Bayesian approach method utilizing mixed effect models as well as integrated nested Laplacian approximation (INLA[5]) in order to identify the presence or absence of DEU at the gene-level in a pairwise comparison.

The main goal of this project was to develop an R package for the shrinkISO method, to include the functions that were written and create a user manual. Prior to the package development of shrinkISO, the method was applied to analyse chromosome 22. This analysis was used as basis for the example in the user manual.

The project can not be described as a research project per se. That is reflected in the structure of this report. In this project, a total of three packages were created. This report is written, leading with these three packages.

Similar to a regular report, which would be written after a research internship, this report contains a method section, results and a conclusion/discussion. However, this report is slightly different in the sense that these subjects will be discussed in the light of one of the three different packages.

In the method section, a package that was called `HiDiHelper` will be featured. This package was created in order to practice *“creating a package”*, and therefore will function as *method*. This part is not written in the typical past tense, as a method section usually is, rather in a present (somewhat imperative) tense, addressing the reader as *you*. This way the method could seemly function as a manual, if need be.*

In the result section, the focus will be on shrinkISO. This chapter will cover the most important parts of the internship. It includes a short introduction of the method, a

confined analysis of chromosome 22, and a detailed explanation of the package itself. The vignette is included as part of the results, with additional comments.

After a conclusion, the discussion will address some ideas for improvements. It will also reflect on what was learned by featuring a small package that was created with the knowledge and experience that was gained during the internship.

---

* This was a topic after one of the presentations. As only the slides were available at the time, this might work as an additional guide.

# Method

## Creation of the HiDiHelper package

Creating a package starts with the desire to create order in a chaotic collection of functions that belong together. Putting functions together in a package provides for clean and tranquil filing systems and more reproducible research methods.

In this chapter, the different steps that were taken in order to create a package are explained, based on a package that was created in the first months of the internship. This package, `HiDiHelper`, contains functions that make it easier to handle high dimensional data, such as sequence data.

The following steps are based on Hilary Parker's "Writing an R package from scratch"[6]. Her directives were followed for all packages created during this internship.

### Set up

Preparatory to creating your own package, you will need to install two existing packages. Package devtools[7] provides `R` functions that simplify common tasks when building an R package and roxygen2[8] provides for automated documentation. Create a parent directory in which you want to build your package, and set this as the current working directory in R. With the command `create_package("HiDiHelper", roxygen = TRUE)`*, you create the fundamentals of a package.

The fundamental elements of the package include a `DESCRIPTION`**, a `NAMESPACE` and two folders, `/R/` and `/man/`. The `NAMESPACE` is a confusing part of package building, but roxygen2 makes this easier. Roxygen2 will write it for you, as long as you use the right syntax. Important for that is the `DESCRIPTION`. In the `DESCRIPTION` (shown below) you write the specifics of your package, like the package name and title, the creator(s), a short description, etc. If your package depends on functions from other packages, you can use the `Import:` command. When a user installs your package, the packages listed in `Imports:` will also be installed, if not already present. It is advised to explicitly use `package::function()` when you refer to functions from other packages within your

functions. Under `Suggest:` you can list packages that could be beneficial to your package, but are not absolutely required. These packages will *not* be automatically installed.

```
Package: HiDiHelper
Title: High-Dimensional Data Helper Functions
Version: 0.0.0.9000
Authors@R:
    person(given = "Renee",
           family = "de Menezes",
           role = c("aut", "cre")),
    person(given = "Jipke",
           family = "van Lanen",
           role = c("aut", "cre"))
Description: These helper functions are useful for high-dimensional data analysis.
    They are very general, and are independent of the type of analysis run.
License: `use_mit_license()`, `use_gpl3_license()` or friends to pick a license
Encoding: UTF-8
LazyData: true
Roxygen: list(markdown = TRUE)
RoxygenNote: 7.1.1
Import:
    multtest
Suggests:
    gplot,
    knitr,
    rmarkdown
VignetteBuilder: knitr
```

DESCRIPTION

## Functions & Help files

The functions that are to be included in the package, you save in the `/R/` folder as `function.R` function files. For roxygen2 it works best if you store each function in a separate file. Hidden functions, which do not require a help file (a.k.a. functions you only use internally and therefore do not need a help file), can be saved in a single file.

Roxygen2 makes use of special comments (`#'`). It will ignore regular comments (`#`), so you can keep using those for your own convenience. As an example, the function file of `HiDiHelper::mysplit()` is shown below. Start your function file with the title of the function, an empty (but commented) line and a short description. For other information in the function file, roxygen2 uses 'tags' (`@tag`) at the beginning of a line (but after the special comment). The most common tags in function files are `@param`, `@return` and `@examples`. `@param` is used to describe the input of the function. The tag is followed by the name of the argument, and then the description. (`#' @param` **`argument_name`** Description.) `@return` is used to give a description of the output, and with `@examples` you can show users how to run the function. The tag `@export` is used to make the function publicly available and usable outside of your package. (By default, roxygen2 does not export anything.) A function file of HiDiHelper function `mysplit()` is shown below as an example.

```r
#' My Split
#'
#' This function can be used to apply \code{strsplit} to each entry of a character vector
#' and select always the same entry of the result.
#' It was written to be applied with \code{sapply}.
#' @param xi The element to be selected (from 1 to the length of the character vector, typically).
#' @param char.vector The vector containing the strings to be split.
#' @param split.by The string to be used to split each entry in \code{char.vector}.
#' @param result.sel The element after splitting to be selected.
#' @return A vector: Unlike \code{strsplit}, this function returns the split string as a vector, not a list.
#' @export
#' @examples
#' sapply(1:nrow(data.dat), mysplit, data.dat$abc, split.by = "_", result.sel = 2)
#'
mysplit <- function(xi, char.vector, split.by, result.sel, fixed=FALSE){
  char.vector <- as.character(char.vector)
  mychar <- char.vector[xi]
  # If we want to have all bits of the split as a matrix/array, we can use
  # split.result <- strsplit(mychar,split=split.by)[[result.sel]]
  if(!fixed) {split.result <- strsplit(mychar,split=split.by)[[1]][result.sel]
  split.result} else {
    split.result <- strsplit(mychar,split=split.by,fixed=fixed)[[1]][result.sel]
    split.result
  }
}
```

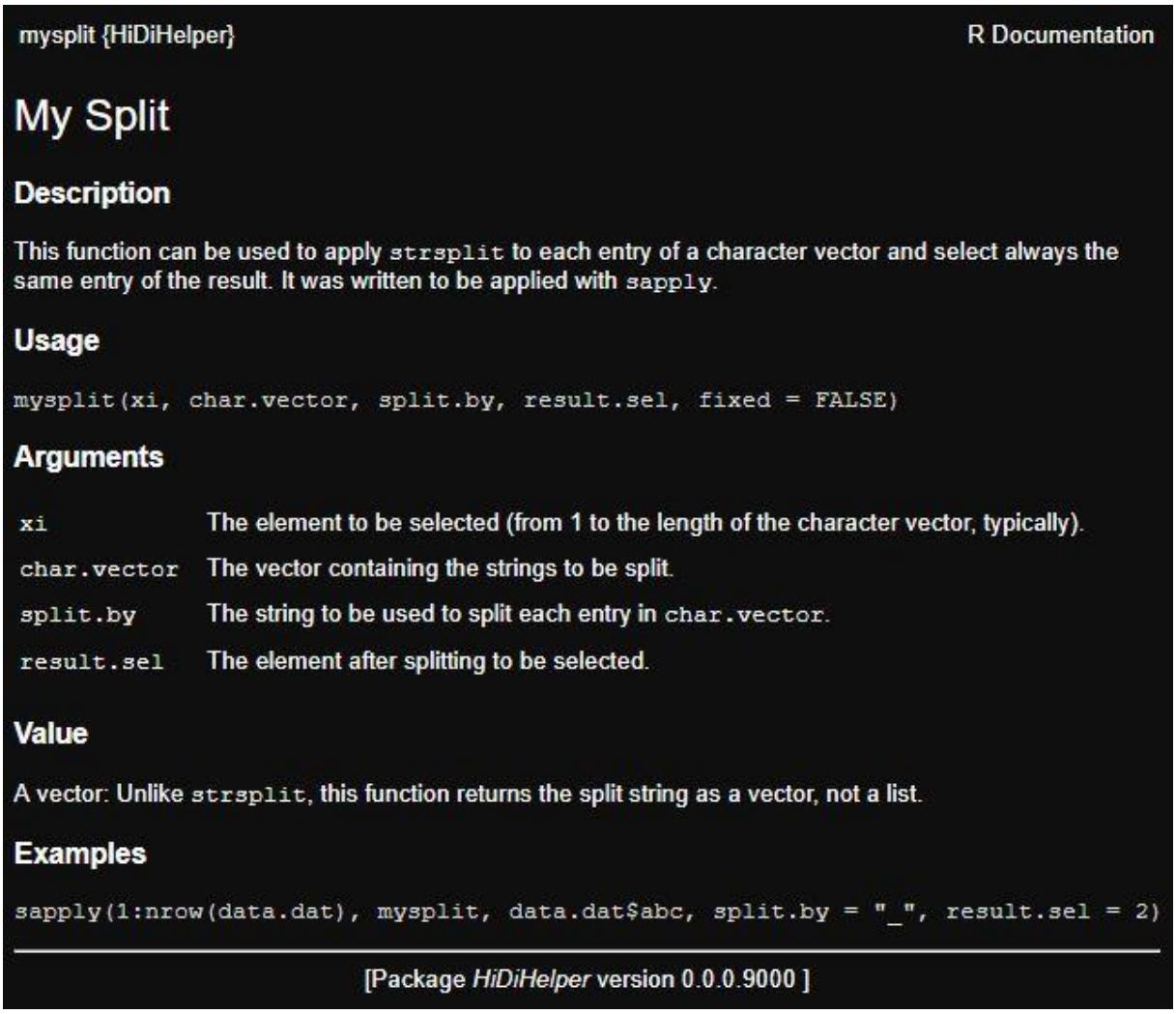
Function file `mysplit.R`

To create and update the help files, you run `devtools::document()`. (Make sure your working directory is set to **inside** the package). Roxygen2 will then automatically create (or update) `function.Rd` help files in the `/man/` folder. As an example, the `.Rd` file of `mysplit()` is shown below. The tags are now converted to the correct format for a help file.

```r
% Generated by roxygen2: do not edit by hand
% Please edit documentation in R/mysplit.R
\name{mysplit}
\alias{mysplit}
\title{My Split}
\usage{
mysplit(xi, char.vector, split.by, result.sel, fixed = FALSE)
}
\arguments{
\item{xi}{The element to be selected (from 1 to the length of the character vector, typically).}

\item{char.vector}{The vector containing the strings to be split.}

\item{split.by}{The string to be used to split each entry in \code{char.vector}.}

\item{result.sel}{The element after splitting to be selected.}
}
\value{
A vector: Unlike \code{strsplit}, this function returns the split string as a vector, not a list.
}
\description{
This function can be used to apply \code{strsplit} to each entry of a character vector
and select always the same entry of the result.
It was written to be applied with \code{sapply}.
}
\examples{
sapply(1:nrow(data.dat), mysplit, data.dat$abc, split.by = "_", result.sel = 2)
}
```

Rd file `mysplit.Rd`

The help file that will emerge from the previous examples is shown below. When using `?mysplit()` a document like this one will show up in R.



mysplit {HiDiHelper} R Documentation

# My Split

## Description

This function can be used to apply `strsplit` to each entry of a character vector and select always the same entry of the result. It was written to be applied with `sapply`.

## Usage

```
mysplit(xi, char.vector, split.by, result.sel, fixed = FALSE)
```

## Arguments

| | |
|---|---|
| `xi` | The element to be selected (from 1 to the length of the character vector, typically). |
| `char.vector` | The vector containing the strings to be split. |
| `split.by` | The string to be used to split each entry in `char.vector`. |
| `result.sel` | The element after splitting to be selected. |

## Value

A vector: Unlike `strsplit`, this function returns the split string as a vector, not a list.

## Examples

```
sapply(1:nrow(data.dat), mysplit, data.dat$abc, split.by = "_", result.sel = 2)
```

[Package *HiDiHelper* version 0.0.0.9000 ]

Help with `?mysplit()`

### Add more guidance

You can make your package more user friendly by adding some more guidance in the form of a `README.md` file. A `README` will show up on the main page of for example GitHub or BitBucket. You can use this to write a short description of your package and the installation procedure.

Finally, you can add a detailed manual to your package. In R this is called a *vignette*. A vignette can include various examples of your functions, an exemplar analysis or a step-by-step roadmap. The easiest way to write a vignette is with R Markdown[9] (and knitr[10]). The vignette is best included in the package as both an `.Rmd` and an `.html`/`.pdf` file in its own `/vignettes/` folder. It is possible to add (toy) data to your package that is used by (the examples in) the vignette. These data you generally add to a separate `/data/` folder.

---

* In this step we advise you to use `create_package()`, opposed to just `create()` as Parker's guidelines suggest. The function `create()` does not cooperate well with the current version of roxygen2.

** *"In fact, it [the `DESCRIPTION`] is the defining feature of a package (RStudio and devtools consider any directory containing DESCRIPTION to be a package)."* [11]

# Results

## ShrinkISO method and package

The main goal of this internship was to put a collection of functions from a method that was previously called shrinkISO together in a package. This method was designed for the analysis of differential exon usage (DEU) from exon count data. It is an empirical Bayesian approach method utilizing mixed effect models as well as integrated nested Laplacian approximation (INLA[5]) in order to identify the presence or absence of DEU at the gene-level in a pairwise comparison. The analysis relies on three steps: Calculation of normalization factors (yellow), estimation of the full and the null model, and the computation of posterior probabilities, Bayes factors and multiple testing correction. In the next paragraphs, these steps will be explained in more detail.

In this chapter, the shrinkISO method is used on a small analysis of chromosome 22 to get an idea of method, and the use of the functions. This experience is used to develop the shrinkISO package, which is featured in the second part of this chapter.

## DEU analysis of chromosome 22

### Introduction

To get familiar with the shrinkISO method, a small analysis of chromosome 22 was performed. In this analysis, exon-level count data from two cancer types were compared for DEU. Group one contained ten samples of breast cancer (BRCA) counts and group two held ten colon adenocarinoma (COAD) samples.

Count data for this analysis came from the cancer genome atlas (TCGA) database. The database is used for its large sample sizes, which allows to check the replicability of the results. This is important for proving the method reliable. However, biological outcomes are of less interest. Doing this analysis as practice gives an idea of the use and convenience of the functions and the existing scripts, and the user friendliness of the method as is.

### Method

For the extraction and organization of the data, the R tool `recount2`[12] and scripts from Robin Schutter and Terry Chan were used. A disjoint exon-level data count file was extracted from the TCGA data base with the scripts `data_extraction.Rmd` and `get_count_info.R`.

The data consisted of three different data sets: Sample data, Annotation data, and Count data. Because of the large size of the data, this method made use of indices to identify the different groups. Indices were extracted from the sample data and saved separately.

With the Shell script `join_counts.sh`, annotation data and count data were linked. A hiccup presented itself while running Shell scripts. The scripts were written in Linux; using them in Windows gave an error. As it appeared, the difference in end-of-line-character of the different operating systems was the cause of trouble. This problem was solved by re-extracting part of the data and running the scripts via WSL-Ubuntu. Another part of the data was made fit by changing end-of-line-characters using Notepad++.

The samples of cancer types BRCA and COAD were extracted from the count data with a python script `filter.py`. The same was done for the annotation data with the `get_count_info.R` script. Still, the filtered count data file was to large to load into R at once. Therefore, chunks were made by dividing each sample over separate files with `split.py`.

Normalization was performed with the `TMM_norm.R` and `annotation.R` scripts. These scripts included functions from the shrinkISO package that were later rewritten in order to make the package more applicable on broader data. This TMM-based normalization makes use of the `calcNormFac()` function from the `EdgeR` package. The scripts resulted in a vector with TMM factors, a vector with the library sizes, and a list of offsets for each sample.

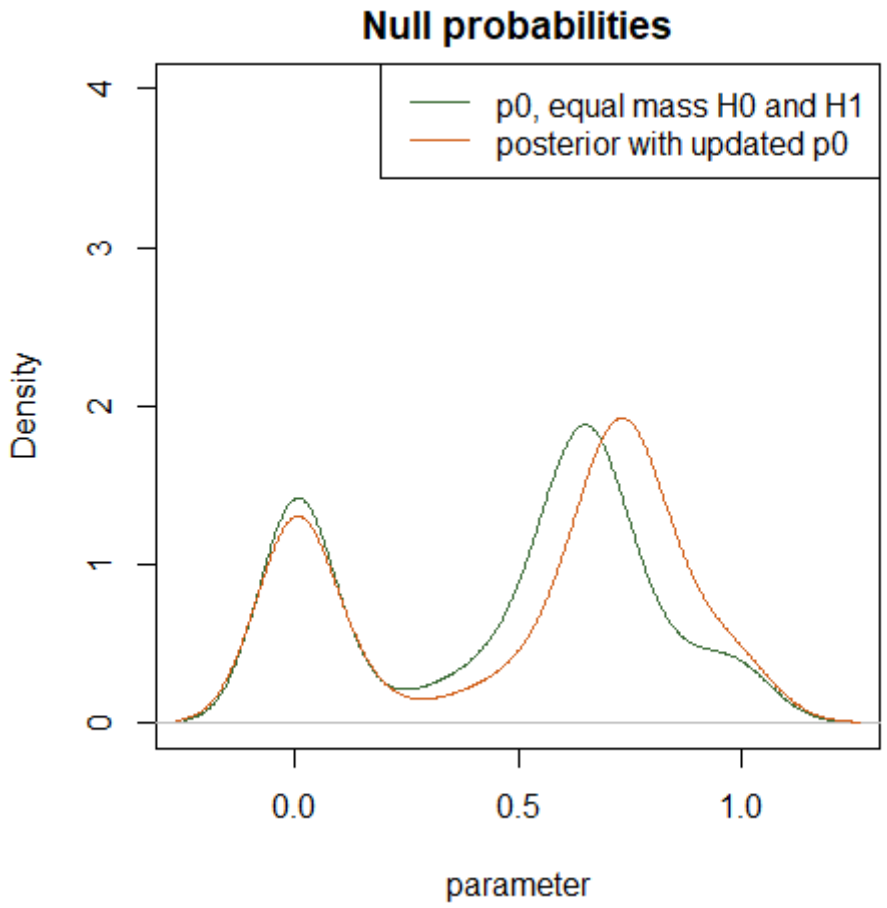
To restrict computational power needed for the estimation of the models, data was split by chromosome with `split.sh`. From here on only chromosome 22 remained in the analysis.

The analysis was ran by three scripts. The first script `split.R` randomly picked ten samples from each cancer type using the indices. The script `run_all.R` ran these samples, together with the offsets, sample data and annotation data through `INLA`, twice. Both the full and the null model were estimated in this script. Running chromosome 22 (on four cores) took three hours.

From the two models, the `lFDR.R` script computed the *a priori* probability, the local FDR, the posterior, and the Bayes factors for each gene. This script automatically computed three objects: `res25` with an fdr.cut of 0.1 and a threshold of 1/25, `res100` with the same cut, but a threshold of 1/100, and `resall` with an fdr.cut of 1.0. Results from the analysis were saved in `.RData` files.

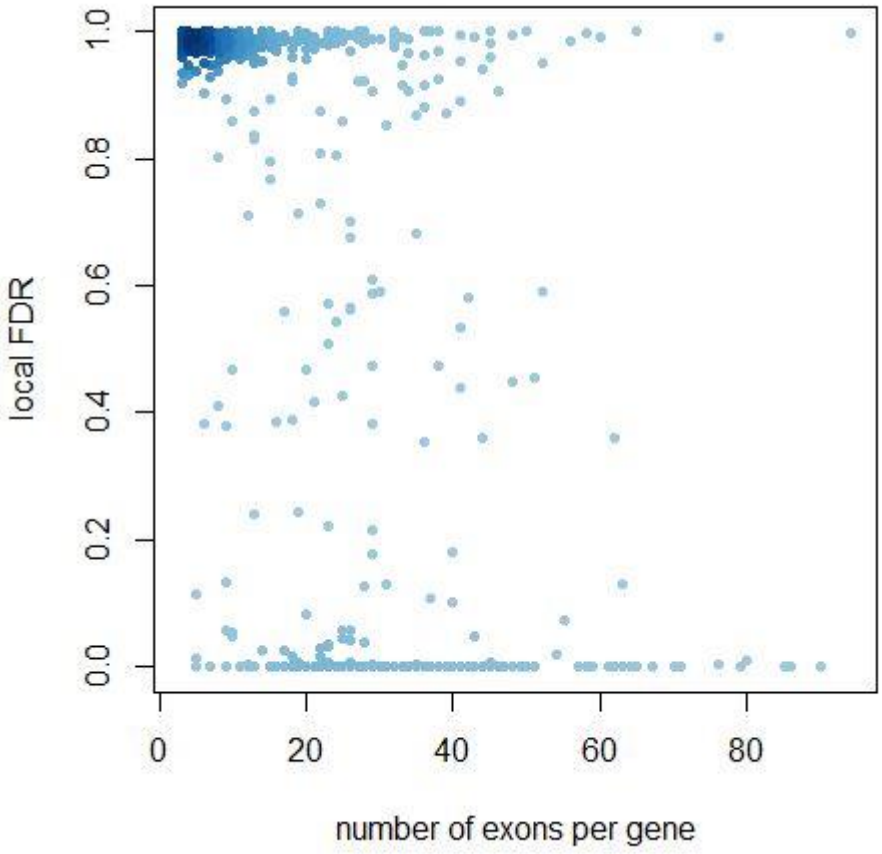
Results

To get familiar with the plotting functions, the structure of the results, and further processing steps, the results of the chromosome 22 analysis are shown in various plots. Some of the plot functions will be included in the package, others are from R base.



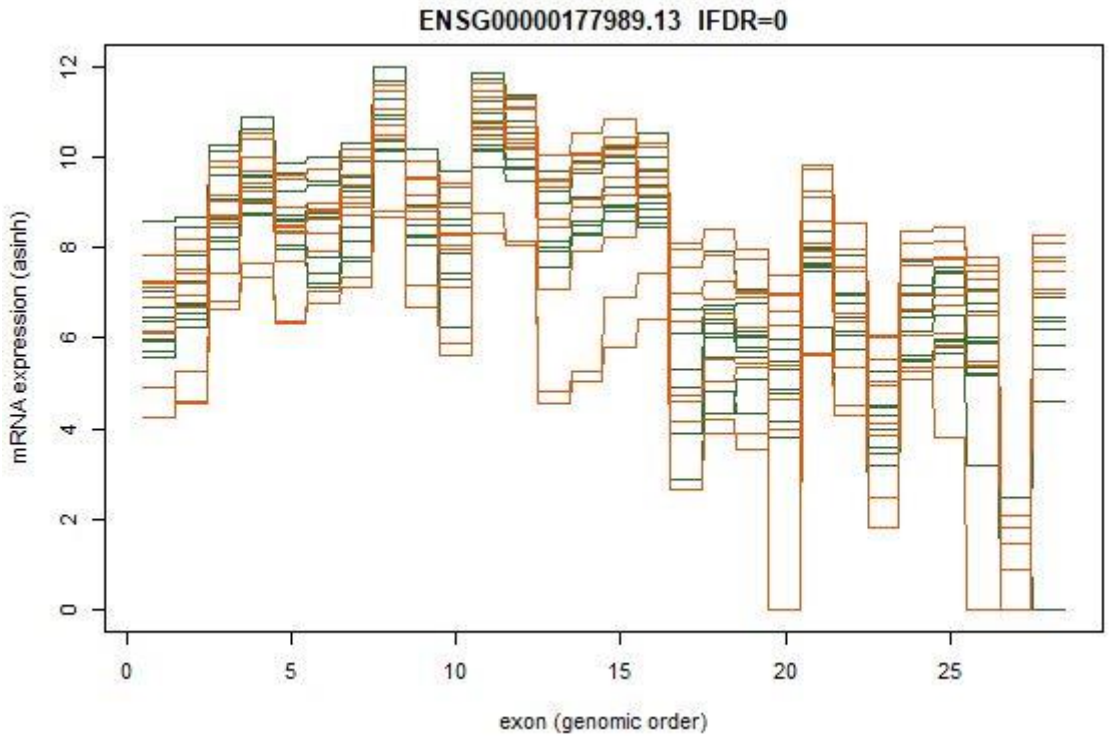

Prior vs. Posterior

This plot of the prior and posterior is drawn with the R base plot function and the `density()` function. It shows the occurrence (density) of the initial and updated probabilities from `resall`.




Significance of genes

In this plot the number of exons on a gene against the local False Discovery Rate is shown. The coloring is done by p0 density. It demonstrates whether the significance of a gene depends on the number of exons this gene contains. Genes with a higher number of exons seem to have a significant outcome more often. Of the 641 genes in chromosome 22, 132 genes were found significant.




Significant gene in a step plot

The plot above shows one of the genes that was found significant. The x-axis represents a gene by plotting the exons in genomic order along the axis. Each line represents the TMM corrected and arc-hyperbolic-sine transformed exon count of a sample. In the first half of this gene, BRCA samples show more counts, while COAD samples have higher counts from exon 17 and on. This `step.plot()` function was specifically designed to visualize these type of results, and will be included in the package.

### Conclusion

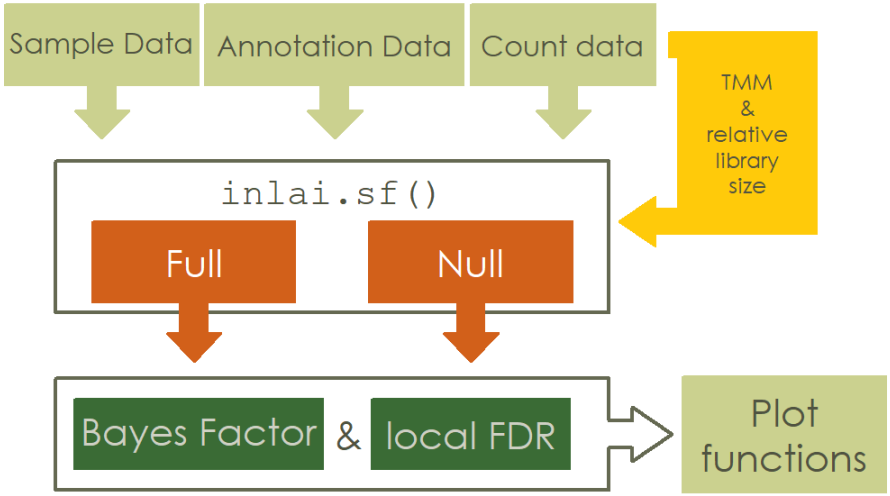
Even though the method stands, there is some room for improvement. Especially when it comes to the usability and the aim on broader use. The existing scripts tend to make this a niche method, only applicable to a certain type of data. Developing a package will contribute to a more replicable method, and will improve user friendliness.

## ShrinkISO package development

Once familiar with the possibilities of R coding and package development, and more comfortable around the shrinkISO method, it is time to combine these aspects.

### Overview

As a first step towards creating the package, a visual representation of the method was composed. It was important to understand the exact course of events; origin of certain data, change in data structure during certain processes, input, output and goals of functions etc. In order to better comprehend these things, (a predecessor of) the following overview was contrived.




This simple but veracious figure gives a good idea of the method. In fact, the overview gave such a strong grasp of the whole concept, that was decided to include it in the package manual. In order to do so, the figure needed to be developed using R. Soon the realization came that this was more work than it looks like; R lacks a simple option to draw overviews like these. Trying to make this as easy as possible, functions were written for the different building blocks of the overview figure. Quickly, this became a project on its own. (More on this project in the discussion.)

## The package

The package was created according to the method as described above. Help files were written with roxygen2 for every user-accessible function. (Internal functions can be found in `hidden_functions.R`.) Dependencies on three other packages were added: EdgeR[13] for the TMM normalization function `calcNormFac()` inside `calcOffs()`, ShrinkBayes[14] for calling the `INLA()` function during the model estimations, and snowfall[15] for parallel computation of the model. A `README` was written to include a short description. Installation details will need to be completed once the package becomes publicly available.

The majority of the last months was spend on the shrinkISO vignette. The vignette has become a step-by-step guide for using the method in this package, based on toy data. To have an exemplary analysis with data that do not consume too much space, a semi-random selection of 40 genes was made. This selection was based on the analysis of chromosome 22. Several functions were written in order to randomly select 20 genes that were considered significant, and combine these with 20 randomly selected genes that

were found non-significant. Sample, annotation and count data were filtered on solely these genes. These were saved in the `shrinkISO/data/` folder as `pdata` (sample/patient data, 1 KB), `Gencode` (original name of the annotation data file when extracted with recount2, 8 KB), and `exon_counts` (count data, 46 KB).

The original vignette can be found in `shrinkISO/vignettes/` as `.Rmd` and `.html` file. It serves as user manual for the method and consists of four chapters: an introduction, an explanation of the input data, the analysis itself and the visualization of results. The vignette is added as part of this report in the appendix, with some additional comments on the development and background information in orange .

# Discussion

The shrinkISO package is an improvement upon the initial method in that it is more widely applicable, and more user friendly and accessible. The hurdles that were encountered during the first analysis made this clear. Using a method from personalized scripts was time consuming. The package is an improvement, not only in that it is easier to share, it is also more reliable. It makes for better reproducibility of results. It is open to users with less experience and just a lot faster to master for the ones with experience.

This project, including the vignette, will be a starting point for a detailed paper on the shrinkISO package. A genome wide analysis covering BRCA vs COAD comparison with different sample sizes is still in progress, as well as an analysis of BRCA sub types. Using the package, it will be easier to do a replicability analysis and check the robustness of the method under different conditions.

Still, there is some room for improvements on the package. A niftier formulation of some of the functions would contribute to user friendliness. For example, the `inlai.sf()` function still needs to run twice with practically the same settings. However, while keeping the original function available for users with very specific desires, another overarching function could be written to run both the full and the null model in one go. The function `calcOffs()` might need some altering when it comes to digesting very large data sets. As the analysis of chromosome 22 needed to be divided in chunks in order to go through the normalization procedure, `calcOffs()` was written to do this all at once. There was no time left to delve into this and investigate, but it might still require a closer look. Furthermore, a suggestion was made to take the length of the exons and position of introns into account when drawing a step.plot. Implementing this might ask for some puzzling, but the data are all there. Currently only the order of the exons is taken into account, but combining genomic locations of the exons (start and end position, present in `data.ann`) and count data could give a new and surprising perspective.

Other plans for the future concern the visual representation of the method that was shown in the the shrinkISO package overview section. Beginning a new project, sometimes it is difficult to see the whole picture. Encountering the project in a visual manner can help to get a hold of a project. Many people prefer a visual explanation over text. However, R has no easy way of drawing these overviews. To make things easier (primarily for this project), functions to draw the several different shapes to create these overviews were written. But

since this project might not be the only R project benefiting from a clear overview, an additional package was created. The package *arrows* is not yet finished, but will be available on GitHub as soon as it is.

An interesting next move could be to bring this package to the web with shiny[16]. Drawing figures using the shiny user interface might be more approachable for people that are less experienced in R. As this figure will be turned to code that can be copied, they will have somewhere to start from and build upon. Bioinformatics is a fast growing field. Yet, not all scientists are comfortable using R, and for some it might even seem scary. Web tools like shiny could help make it more approachable.

# Acknowledgements

I would like to thank all members of de Menezes' group, for their time and support. A special thanks to *Terry Chan*, for being patient, extremely helpful and, above all, kind when I struggled starting up this project. I would like to thank a past member of the group, *Robin Schutter*. A share of the work that was done during this internship, was build on work to which he contributed during his internship. And finally, my supervisors, *Renee de Menezes, Renaud Tissier and Adrien Melquiond*. Thank you for your patience, understanding, and over all positive attitude.

# References & links

1. Nilsen, T., Graveley, B. Expansion of the eukaryotic proteome by alternative splicing. Nature 463, 457–463 (2010).
2. Park, E., et al., The Expanding Landscape of Alternative Splicing Variation in Human Populations. AJHG 102.1, 11-26 (2018).
3. Yeo, G., Holste, D., Kreiman, G. et al. Variation in alternative splicing across human tissues. Genome Biol 5, R74 (2004).
4. Anders S., Reyes A., Huber W., Detecting differential usage of exons from RNA-seq data. Genome Res. 2008-17, (2012)
5. R-INLA Project
6. Parker, H., Writing an R package from scratch. (2014)
7. Wickham, H., Bryan J., et al. Devtools.
8. Wickham, H. et al. Roxygen2
9. Dervieux, C, et al. R Markdown
10. Xie, Y., et al. Knitr
11. quote: Bryan, J., Wickham, H., R Packages, chapter 8. Package metadata.
12. Collado-Torres, L. et al. recount2
13. Chen, Y., et al. edgeR
14. van de Wiel, M. A., ShrinkBayes
15. Knaus, J., Snowfall
16. Chang, W. et al. Shiny, shiny.rstudio.com