The R Journal

Volume 16/3, September 2024

A peer-reviewed, open-access publication of the
R Foundation for Statistical Computing

## Contents

Prospective authors will find detailed and up-to-date submission instructions on the Journal's homepage.



The R Journal is indexed/abstracted by EBSCO, DOAJ, Thomson Reuters.

# Editorial

*by Mark P.J. van der Loo*

On behalf of the editorial board, I am pleased to present Volume 16 Issue 3 of the R Journal.

## In this issue

News from the R Foundation and Bioconductor are included in this issue.

This issue features 10 contributed research articles the majority of which relate to R packages on a diverse range of topics. All packages are available on CRAN. Supplementary material with fully reproducible code is available for download from the Journal website. Topics covered in this issue are the following.

### Time series

- boiwsa: An R Package for Seasonal Adjustment of Weekly Data
- SNSeg: An R Package for Time Series Segmentation via Self-Normalization

### Modeling and Inference

- SIHR: Statistical Inference in High-Dimensional Linear and Logistic Regression Models
- SLCARE: An R Package for Semiparametric Latent Class Analysis of Recurrent Events
- fmeffects: An R Package for Forward Marginal Effects

### Geospatial

- GeoAdjust: Adjusting for Positional Uncertainty in Geostatistial Analysis of DHS Data
- Kernel Heaping - Kernel Density Estimation from Regional Aggregates via Measurement Error Model

### Applications

- Validating and Extracting Information from National Identification Numbers in R: The Case of Finland and Sweden
- GSSTDA: Implementation in an R Package of the Progression of Disease with Survival Analysis (PAD-S) that Integrates Information on Genes Linked to Survival in the Mapper Filter Function
- PubChemR: An R Package for Accessing Chemical Data from PubChem

*Mark P.J. van der Loo*
*Statistics Netherlands and Leiden University*

https://journal.r-project.org
r-journal@r-project.org

# Validating and Extracting Information from National Identification Numbers in R: The Case of Finland and Sweden

*by Pyry Kantanen, Erik Bülow, Aleksi Lahtinen, Måns Magnusson, Jussi Paananen, and Leo Lahti*

**Abstract** National identification numbers (NIN) and similar identification code systems are widely used for uniquely identifying individuals and organizations in Finland, Sweden, and many other countries. To increase the general understanding of such techniques of identification, openly available methods and tools for NIN analysis and validation are needed. The hetu and sweidnumbr R packages provide functions for extracting embedded information, checking the validity, and generating random but valid numbers in the context of Finnish and Swedish NINs and other identification codes. In this article, we demonstrate these functions from both packages and provide theoretical context and motivation on the importance of the subject matter. Our work contributes to the growing toolkit of standardized methods for computational social science research, epidemiology, demographic studies, and other register-based inquiries.

## 1 Introduction

Technical systems for identifying people, organizations, places and other objects are an important but often overlooked aspect of governance and management tools in modern societies (Dodge and Kitchin, 2005). Universal and persistent identification numbering systems for natural persons are vital for facilitating research activities that combine data from different sources, for example in the fields of epidemiology, population studies and social research (Gissler and Haukka, 2004). Outside the field of academic research, universal identifiers for natural persons enable work in a multi-disciplinary and multi-agency context due to greater *administrative fluency* and *bureaucratic effectiveness* (Alastalo and Helén, 2022). This may be useful, for example, in tackling wicked social problems that require cooperation from professionals from multiple fields, such as social workers, psychologists, police, and health care professionals.

The **hetu** and **sweidnumbr** R packages provide open tools for handling and extracting data from identification codes for natural persons and juridical persons in the national contexts of Finland and Sweden. These tools can also be used to handle Finnish and Swedish Business ID codes. Prior R packages with similar scope include **numbersBR** for Brazilian identity numbers for individuals, vehicles, and organizations (Freitas, 2018), **cprr** for Danish "Det Centrale Personregister" (CPR) numbers (Anhøj, 2019), and **generator** for generating various types of Personally Identifiable Information (PII), such as fake e-mail addresses, names and the United States Social Security Numbers (Hendricks, 2015).

Identification code generators and validators are not a novel concept. In the case of Finland, the handling of identification codes can be seen as a common entry-level task for new computer science students to familiarize themselves with regular expressions, handling dates, string subsetting and similar concepts, and examples in various programming languages can easily be found online. However, in research there is a need for standardized, well documented and reproducible methods. These requirements are the main reason for developing R packages for handling Finnish and Swedish identification codes. The packages ensure reproducibility and transparency while also offering user manuals, semantic versioning and documentation of changes between versions (as outlined by Wickham and Bryan, 2024).

## 2 Features of identification number systems

Reliably keeping track of individuals, organizations, objects and flows in a given territory has long been seen as an important feature of modern governance (Dodge and Kitchin, 2005). Foucault (2009, 115-120) observed a historical pattern where practices first implemented in disciplinary institutions, such as prisons, military units and schools have spread to influence whole societies. We can see the results of this development today when alphanumeric identifiers originally assigned to prisoners, soldiers and students have been transformed into nationwide identification code systems around the world. Differences in legal, political and historical frameworks in different countries have affected how these systems are implemented in practice, causing heterogeneity for example in identification system designs across Europe (Otjacques et al., 2007).

This heterogeneity and linguistic differences seem to contribute to variance in the terminology

used when referring to identification code systems. *Unique identifier* (UID) is an umbrella term that can be used to refer to unique identifiers for all sorts of things, from books (ISBN), chemicals (CAS) and legal entities (LEI) to anything imaginable (see Dodge and Kitchin, 2005). In this paper, we are mainly interested in unique identifiers for natural persons and juridical persons.

Names such as *personal identification code* (Dodge and Kitchin, 2005), *personal identity number* (Alastalo and Helén, 2022) and *single identification number* (SIN) (Otjacques et al., 2007) are used in literature as generic terms. Personal identification codes can sometimes be confused with *personal identification numbers* (PINs) that refer to numeric or alphanumeric passcodes used for authentication, for example, to withdraw cash or open a locked mobile phone. On the other hand names such as *personal identity code* (PIC) (Digital and Population Data Services Agency, 2022a; Sund, 2012), name number (Watson, 2010) and personal number (Statistics Sweden, 2016) are used in official translations to refer to national implementations of NINs; in the mentioned cases, Finnish, Icelandic and Swedish NINs, respectively. Due to *function creep* (see Brensinger and Eyal, 2021; Alastalo and Helén, 2022) or *control creep* (see Dodge and Kitchin, 2005), historically sector-specific identifiers may also be used as a *de facto* NIN. This is the case with the US *social security number* (SSN) (Brensinger and Eyal, 2021) and Finnish employee pension card numbers and social security codes (Alastalo and Helén, 2022).

For clarity's sake, we will be using the generic term *national identification number* (NIN) to refer to all identification number systems and their country-specific implementations for natural persons, in this case, the Finnish *personal identity code* and the Swedish *personal number*. For organizations, we will use the generic term *organization identifier* when discussing Finnish *business IDs* (BID) and Swedish *organizational identity numbers* (OIN) / *Swedish organizational numbers* (SON).

All identification systems should strive to be both *unique* and *self-same* over time. Self-sameness refers to a degree of immutability that allows organizations to identify and reidentify a person over time. A combination of attributes such as name, occupation and address would probably form a unique identifier even in relatively large crowds, but such attributes might not stay the same over time. (Brensinger and Eyal, 2021).

According to Alterman (2003), a distinction can be made between *biocentric data* and *indexical data*. The former is biometric data connected to the individual's physical features whereas the latter has no distinguishable relation to the individual, physiologically, psychologically, or otherwise. An example of biocentric data could be a fingerprint or an iris scan and an example of indexical data could be a randomly assigned number from which nothing can be deduced. [1]

For several reasons, many identification numbers are not just random strings. The American SSN originally contained information about the person's birth year and where the number was first registered (Brensinger and Eyal, 2021, 32) whereas Nordic countries' NINs often contain (or used to contain) information about the individual, usually birth date and sex (Watson, 2010; Salste, 2021). One reason for this was to make the code easier to remember (Alastalo and Helén, 2022). Even when sex and birth date are not biocentric data in the sense as Alterman (2003) defined it, including them takes Nordic NINs further away from being pure indexical data, thus making them arguably more sensitive to handle. Table 1 provides a summary of the introduction of NINs in the Nordic countries as well as information which they contain.

**Table 1:** Nordic NINs: year introduced and embedded information.

| country | NIN name | introduced | characters (n) | birth date | sex | birth place |
|---|---|---|---|---|---|---|
| Sweden | personnummer | 1947 | 11 | yes | yes | yes |
| Iceland | kennitala | 1950 | 10 | yes | no | no |
| Norway | fødselsnummer | 1964 | 11 | yes | yes | no |
| Denmark | CPR-nummer | 1968 | 11 | yes | yes | no |
| Finland | henkilötunnus | 1968 | 11 | yes | yes | no |

In the Nordic countries, comprehensive national identification number systems were developed and implemented from the 1940s to the 1960s (Watson, 2010). In Sweden, the personal identity number (PIN) was introduced in 1947 and it consisted of both date of birth and an additional three-digit birth number. In 1967 a check digit was added finalizing the design of Swedish PIN (Åke Johansson, 2003;

[1] Brensinger and Eyal (2021) discuss the concept of *dividuals*, manufactured objects that represent the living individual: address, fingerprints, name and so on. These dividuals need to go through the process of disembedding, standardization and re-embedding to be useful. Disembedding means data gathering (e.g. taking a fingerprint sample), standardization means making the disembedded transcription into a standardized digital sample that can be easily compared with other similar samples and re-embedding means linking these standardized records back to their actual flesh-and-blood counterparts. Without a way to re-embed a huge and well-standardized archive of fingerprints back to the population, it is essentially useless. This is also a reason why biometric samples such as iris scans or fingerprints can never replace primary keys in databases.

Statistics Sweden, 2016). The Finnish personal identity code has its roots in specialized employment pension number introduced in 1962, which was then gradually expanded to cover the whole population in the form of social security number. In Finland personal identity code was introduced as specialized employment pension number in 1962, which was gradually expanded to cover the whole population in the form of social security number. The Finnish PIN was most likely inspired by early iteration of the Swedish NIN (Alastalo and Helén, 2022). The design has proven to be resilient and with some minor tweaks, it continues to be used, with the modern iteration being called a *personal identity code* [2] (Salste, 2021).

Like Finland, other Nordic countries took inspiration from Sweden as well (Krogness, 2011). Table 2 illustrates the structural similarity of NINs in different Nordic countries. Some national variation does exist. In Norway and Denmark, individual numbers are used to denote the century in the birth date in addition to differentiating individuals from one another (Furseth and Ljones, 2015; CPR-kontoret, 2008). Since 2007 the Danish CPR numbers have dropped the check digit in favour of having an extra individual number to expand the pool of available unique numbers from roughly 500 per day to 4000-6000 per day (CPR-kontoret, 2008; Jerlach, 2009). All Nordic NIN designs prioritize a shorter length of 10-11 characters and use a combination of 2-digit years and a century marker or certain individual number ranges to denote the full year. For example, "52-" from the Finnish number and "99551" from the Norwegian number translate to 1952 and 1899, respectively. The more unambiguous Swedish 12-digit variant is used exclusively in automatic data processing systems (ADB, from Swedish *automatisk databehandling*) and not in day-to-day interactions.

**Table 2:** Examples of national identification numbers and their composition in five Nordic countries. DD: day, MM: month, YY: year, C: century marker, N: individual number / serial number, Q: check digit or a control character.

| country | NIN name | NIN example | NIN structure |
|---|---|---|---|
| Sweden | personnummer | 610321-3499 | YYMMDDCNNNQ |
| Iceland | kennitala | 121212-1239 | DDMMYYNNQC |
| Norway | fødselsnummer | 01129955131 | DDMMYYNNNQQ |
| Denmark | CPR-nummer | 300280-1178 | DDMMYY-NNNN |
| Finland | henkilötunnus | 131052-308T | DDMMYYCNNNQ |
| Sweden | personnummer (ADB) | 196103213499 | YYYYMMDDNNNQ |

In Finland, the expansion of sector-specific social security numbers and employment pension numbers to universal NINs in 1969 has contributed to widespread secondary use of different data sources [3] in public administration, education, and research and development. In Sweden, the NIN is currently used extensively in all parts of society, not only for taxation. It is used in education, for military service, in health care and by financial institutions and insurance companies. The role of the Swedish NIN has also made it central to register-based research (Statistics Sweden, 2016). It could be argued that the most important feature of NIN systems is the interoperability it enables between different sectors of society (Alastalo and Helén, 2022).

## 3 Working with national identification numbers in R

The method of validating and extracting information from identification numbers is manually doable and simple in principle but in practice becomes unfeasible with datasets larger than a few dozen observations. The **hetu** and **sweidnumbr** packages provide easy-to-use tools for programmatic handling of Finnish and Swedish personal identity codes and Business ID codes [4]. As shown in Table 3, both packages share several core functions and function names.

Both packages utilize R's efficient vectorized operations, generating and validating over 5 million personal identity codes or Business Identity Codes in less than 10 minutes on a regular laptop [5]. This can meet the practical upper limit set by the current population of Finland (5.5 million people) (Official Statistics of Finland (OSF), 2022) and Sweden (10.5 million people) (Statistiska centralbyrån, 2022), providing adequate headroom for the handling of relatively large registry datasets containing information on people currently alive and deceased.

[2] In Finnish: henkilötunnus, or *hetu* for short, hence the name of the package
[3] Secondary data: Data that has not been collected primarily for a specific research question
[4] In Finnish: Yritys- ja Yhteisötunnus, or Y-tunnus for short, In Swedish: Organisationsnummer
[5] Tested on a 2015 Macbook Pro with Intel i5-5257U @ 2.70GHz

**Table 3:** Exported functions that are shared between both 'sweidnumbr' and 'hetu'. Function alias in parentheses.

| sweidnumbr | hetu | Description |
|---|---|---|
| rpin | rpin (rhetu) | Generate a vector of random NINs |
| pin_age | pin_age (hetu_age) | Calculate age from NIN |
| luhn_algo | hetu_control_char | Calculate check digit / control character from NIN |
| pin_ctrl | pin_ctrl (hetu_ctrl) | Check NIN validity |
| pin_date (pin_to_date) | pin_date (hetu_date) | Extract Birth date from NIN |
| pin_sex | pin_sex (hetu_sex) | Extract Sex from NIN |
| oin_ctrl | bid_ctrl | Check OIN/BID validity |
| roin | rbid | Generate a vector of random OINs/BIDs |

## 4 The hetu package

Printing a data frame containing extracted information in a structured form can be done as follows:

```
library(hetu)
x <- c("010101A0101", "111111-111C", "290201A010M")
hetu(x)
```

The `hetu()` function is the workhorse of the **hetu** package. Without additional parameters, it prints out a data frame with all information that can be extracted from Finnish NINs as well as a single column that indicates if the NIN is valid as a whole or if it has any problems that make it invalid. For demonstration purposes the 3rd NIN listed below has an invalid date part; the 29th of February would only be a valid date if the year was a leap year, which we know that 2001 is not. The NIN would be correct if the year was changed from 2001 to 2000.

```
         hetu    sex p.num ctrl.char       date day month year century
1 010101A0101 Female   010         1 2001-01-01   1     1 2001       A
2 111111-111C   Male   111         C 1911-11-11  11    11 1911       -
3 290201A010M Female   010         M       <NA>  29     2 2001       A
  valid.pin
1      TRUE
2      TRUE
3     FALSE
```

The full birth year is constructed by reading the 2-digit year information and the century marker; in the case of the first-row example, "01" and "A". "A" means that the person is born in the 2000s, "-" means the 1900s and "+" means the 1800s. A binary sex classification can be constructed simply by calculating if the personal number (`p.num` column) is an odd or an even number: Even numbers (for example "010") denote a female and odd numbers (for example "111") denote a male. The final character of the NIN is the control character, which is determined by dividing the concatenated integers (for example 290201010) by 31 and using the remainder as a key to retrieve a value from a list that includes numbers between 0 and 9 and English alphabets (without letters that might be mixed with numbers: I, G, O, Q and Z) (Digital and Population Data Services Agency, 2022a; Salste, 2021).

In 2023 there was a reform of the personal identity code seperators. In addition to letter A, letters B, C, D, E and F also signify that the person was born in 2000s. For 1900s letters Y, X, W, V and U were added. These new seperators were added to ensure that there are enough personal identity codes. The change also made the seperator a distinguishing element of the NIN. (Digital and Population Data Services Agency, 2022b)

The generic way of outputting information found on individual columns is to use the standard `hetu()` function with extract-parameter.

```
hetu("010101A0101", extract = "sex")
```

```
[1] "Female"
```

```
hetu("010101A0101", extract = "date")
```

```
[1] "2001-01-01"
```

All column names printed out by the `hetu()` function are valid extract parameters. Most commonly used columns have wrapper functions that are identical in output:

```
pin_sex("010101A0101")
```

```
[1] "Female"
```

```
pin_date("010101A0101")
```

```
[1] "2001-01-01"
```

With the help of imported functions from the **lubridate** package (Grolemund and Wickham, 2011), we can calculate ages from NINs not only in years and days but also in months and weeks by using the `pin_age()` function. By default, the age is calculated in years at the current date and time but this end date can also be manually set by using the date parameter.

```
pin_age("010101A0101", date = "2004-02-01", timespan = "months")
```

```
[1] 37
```

All NINs passed through the `hetu()` function are checked with 10 different tests to determine their validity. All tests need to be passed for a NIN to be valid. The results from different tests are summarized in the valid.pin column of the `hetu()` function output data frame. The user can print individual test results with the `hetu_diagnostic()` function for debugging purposes.

```
hetu_diagnostic("290201A010M")
```

```
         hetu is.temp valid.p.num valid.ctrl.char correct.ctrl.char valid.date
1 290201A010M   FALSE        TRUE            TRUE             FALSE      FALSE
  valid.day valid.month valid.year valid.length valid.century
1      TRUE        TRUE       TRUE         TRUE          TRUE
```

When data is inputted manually without validity checks, input errors can creep in. The control character in Finnish personal identity codes combined with validity checks in the `hetu()` function can help to catch the most obvious errors. In the example above we can see that the date is incorrect, but also the control character is incorrect [6]. We can simply try three different dates to see if the input error is in the day, month or year part, assuming that the personal number and control character parts were inputted correctly. In this manufactured example the error was in the year part, resulting in the rare leap day date being the correct one.

```
example_vector <- c("290201A010M", "280201A010M", "290301A010M", "290200A010M")
columns <- c("valid.p.num", "valid.ctrl.char", "correct.ctrl.char", "valid.date")
hetu_diagnostic(example_vector, extract = columns)
```

```
         hetu valid.p.num valid.ctrl.char correct.ctrl.char valid.date
1 290201A010M        TRUE            TRUE             FALSE      FALSE
2 280201A010M        TRUE            TRUE             FALSE       TRUE
3 290301A010M        TRUE            TRUE             FALSE       TRUE
4 290200A010M        TRUE            TRUE              TRUE       TRUE
```

The **hetu** package can generate a large number of personal identity codes with the `rpin()` function. The date range of the generated identity codes can be changed with parameters, but it has a hardcoded lower limit at the year 1860 and an upper limit at the current date. It has been theorized that the oldest individuals that received a personal identity code in the 1960s were born the in 1850s or 1860s. Personal identity codes are never assigned beforehand and therefore it is impossible to have valid personal identity codes that have a future date. (Salste, 2021)

The function can also be used to generate so-called temporary personal identity codes. Temporary identity codes are never used as a persistent and unique identifier for a single individual but as a

[6] By validity we mean that the control character itself is an allowed character. By correctness, we mean that the inputted control character matches the calculated control character

placeholder in institutions such as hospitals when a person does not have a Finnish NIN or the NIN is not known. They can be identified by having a personal number (p.num column in `hetu()` function output or NNN as in Table 2) in the range of 900-999.

Below is an example of generating 4 temporary Finnish NINs and checking their validity with the `pin_ctrl()` function. Since all NINs are temporary, they do not pass the check validity checks meant for normal pins if they are not explicitly allowed. A vector with no valid NINs returns a single NA.

```
set.seed(125)
x <- rpin(n = 4, p.male = 0.25, p.temp = 1.0)
x

[1] "201215-940S" "080854-929H" "241258-9669" "090405A980X"

pin_ctrl(x)

[1] NA

pin_ctrl(x, allow.temp = TRUE)

[1] TRUE TRUE TRUE TRUE
```

As mentioned earlier, our package also supports similarly generating and checking the validity of Finnish organization identifiers, or Finnish Business ID (BID) numbers. Despite the name, BIDs are used not only for companies and businesses but also for other types of organizations and other juridical persons. Unlike personal identity codes, BIDs do not contain any information about the company. BIDs consist of a random string of 7 numbers followed by a dash and 1 check digit, a number between 0 and 9.

In addition, we have added support for the less known and less widely used numbering scheme for natural persons, Finnish Unique Identification (FINUID) numbers.[7] FINUID numbers consist of 8 random numbers and 1 control character that is calculated in the same way as in Finnish NINs. FINUID numbers are similar to BID numbers in the sense that they do not contain any biocentric data on the individual or the corporation [8], but unlike BID numbers that are ubiquitous in corporate documents and public databases, FINUID numbers are mainly used by government authorities in internal IT systems.

```
bid_ctrl(c("0000000-0", "0000001-9"))

[1] TRUE TRUE

satu_ctrl("10000001N")

[1] TRUE
```

The **hetu** package contains some functions that are not shared with the **sweidnumbr** package, the most notable being the `hetu()` function. These functions are listed and described in Table 4.

**Table 4:** Functions that are unique to the 'hetu' package and have no equivalent in the 'sweidnumbr' package. Function alias in parentheses.

| Function (alias) | Description |
|---|---|
| hetu | Finnish personal identification number extraction |
| pin_diagnostic (hetu_diagnostic) | Diagnostics Tool for HETU |
| satu_control_char | FINUID Number Control Character Calculator |
| satu_ctrl | Check FINUID Number validity |

Version 1.1.0 of the **hetu** has been released, which addresses feedback on the earlier version. This new version implements summary and plot methods for the data frames produced by `hetu_diagnostic()`. Using the summary methods prints a neat diagnostic of the data frame.

[7] FINUID in Finnish: *sähköinen asiointitunniste* (SATU)

[8] Aside from sole trader/business name companies that are closely related to the individual entrepreneur, the term "biocentric" is badly suited when talking about corporations and other juridical persons. However, it could be argued that if the BID number contained information such as the company form, the place of registration or the date of registration it could be seen as analogous to biocentric information contained in NINs.

```
diagnostics <- hetu_diagnostic(example_vector)
summary(diagnostics)

Diagnostics for 4 hetu objects:
Number of valid hetu objects: 1
Number of valid and non-temporary* hetu objects: 1
Number of invalid hetu objects: 3
Number of invalid and non-temporary* hetu objects: 3

 * non-temporary: p.num in range [002-899]
```

### 4.1 The sweidnumbr package

The **sweidnumbr** R package has similar functionality as the **hetu** package, but for Swedish NINs and with a slightly different syntax. At the time of writing, the package has been downloaded roughly 30 000 times from CRAN [9]. The example NINs below are taken from the example published by the Swedish Tax Authority (The Swedish Tax Agency, 2007).

```
library(sweidnumbr)
example_pin <- c("640823-3234", "6408233234", "19640823-3230")
example_pin <- as.pin(example_pin)
example_pin

[1] "196408233234" "196408233234" "196408233230"
Personal identity number(s)
```

Unlike the **hetu** package, the **sweidnumbr** takes advantage of a custom S3 class structure. Therefore the first step is to convert strings with different Swedish NIN formats or numeric variables into a `pin` vector using the `as.pin()` function. The `as.pin()` function formats all inputted numbers to a so-called ADB-format [10] with 12 digits and no century marker, which results in less ambiguity and no need to change the century marker from "-" to "+" when a person turns 100 years old. The `pin` vector is an S3 object and can be checked by using the `is.pin()` function.

```
is.pin(example_pin)

[1] TRUE
```

This function only checks that the vector is a `pin` object, but not if the actual NINs are valid. To check the Swedish NIN using the control numbers, or check digits, we simply use the `pin_ctrl()` function.

```
pin_ctrl(example_pin)

[1]  TRUE  TRUE FALSE
```

Just as in the **hetu** package we can extract information from the Swedish NIN with specialized functions. We can now use `pin_birthplace()`, `pin_sex()`, and `pin_age()` to extract information on county of birth (for NINs assigned before 1990), sex, and age.

```
pin_sex(example_pin)

[1] Male Male Male
Levels: Male

pin_birthplace(example_pin)

[1] Gotlands län Gotlands län Gotlands län
28 Levels: Stockholm stad Stockholms län Uppsala län ... Born after 31 december 1989
```

[9] Source: CRANlogs API, data retrieved at 2022-03-22.

[10] ADB: Short from the Swedish term *automatisk databehandling*, meaning *automatic data processing* (ADP) in English

```
pin_age(example_pin)

[1] 60 60 60

pin_age(example_pin, date = "2000-01-01")

[1] 35 35 35
```

As with the **hetu** R package, we can also generate, or simulate, NINs with the `rpin()` function. Shared functions exist also for Swedish organization identifiers, or Swedish organizational numbers (SON), in the form of `as.oin()`, `is.oin()`, and `oin_ctrl()` functions. Unlike the Finnish BID, the `oin` number contains information on the type of organization of a given SON, which can be determined by using the `oin_group()` function.

```
example_oin <- c("556000-4615", "232100-0156", "802002-4280")
oin_group(example_oin)

[1] Aktiebolag
[2] Stat, landsting, kommuner, församlingar
[3] Ideella föreningar och stiftelser
3 Levels: Aktiebolag ... Stat, landsting, kommuner, församlingar
```

Similar to the `rbid()` function from the **hetu** package, we can generate new SONs using the `roin()` function from the **sweidnumbr** package.

```
set.seed(125)
roin(3)

[1] "776264-6144" "274657-0148" "827230-7631"
Organizational identity number(s)
```

Due to the national characteristics of Swedish numbering schemes for natural and juridical persons some functions are unique to the **sweidnumbr** package. These functions are listed in Table 5.

**Table 5:** Functions that are unique to the 'sweidnumbr' package and have no equivalent in the 'hetu' package.

| Function | Description |
|---|---|
| as.oin | Parse organizational identity numbers |
| as.pin | Parse personal identity numbers to ADP format |
| format_pin | Formatting pin |
| is.oin | Test if a character vector contains correct 'oin' |
| is.pin | Parse personal identity numbers to ADP format |
| oin_group | Calculate organization group from 'oin' |
| pin_birthplace | Calculate the birthplace of 'pin' |
| pin_coordn | Check if 'pin' is a coordination number |

## 5 Discussion

The **hetu** and **sweidnumbr** R packages provide free and open-source methods for validating and extracting data from a large number of Finnish and Swedish national identity numbers (NIN). While the packages' target audience most likely mainly consists of Finnish and Swedish users and people with a particular interest in NIN systems around the world, the packages make a generic contribution to developing methodologies related to NIN handling in R, and more generally for *structured data* in the field of computational humanities (see Mäkelä et al., 2020), epidemiology and demographic studies (see Gissler and Haukka, 2004). A possible direction for future developments could be to create more generic class structures or even a completely new R package that could recognize and handle NIN systems from several different countries around the world.

The origins of the **hetu** package can be traced to the early 2010s when one curious individual wanted to analyze a large number of Finnish NINs that were leaked to the internet by an anonymous hacker, to identify the source of the leak. The legality and morality of handling such datasets containing personal information was and is in a grey area at best. As developers of these packages, we cannot condone such activities, even if they are conducted out of curiosity and not of ill intentions, but we acknowledge that we cannot prevent our users from doing that either. Both **hetu** and **sweidnumbr** packages are free software with permissive licenses and pre-emptively limiting their use to only "good, not evil" causes would be problematic as well. [11]

We have acknowledged beforehand that random NINs generated with the **hetu** and **sweidnumbr** packages could, theoretically, be used for purposes such as synthetic identity fraud. ^[see (Brensinger and Eyal, 2021, 32 for a short description of synthetic fraud related to American SSNs) On the other hand it is important to note that such NINs could also be created by hand as information on valid NINs is readily available e.g. on the Finnish Digital and Population Data Services Agency and Swedish Tax Authority websites (Digital and Population Data Services Agency, 2022a; The Swedish Tax Agency, 2007). Our package can be useful for many, and it does not make fraudulent activities significantly easier for malevolent individuals, which is essential in judging the pros and cons of releasing this software to the public.

Similar data breaches have made people warier about digital services. Privacy concerns can push Finland, Sweden and other Nordic countries towards redesigning their national identification numbers to omit some or all of the embedded personal information sometime in the future. For example, in Finland there has been a project run by the Finnish Ministry of Finance to redesign the Finnish NIN structure (Valtiovarainministeriö, 2022). However the project was pushed back in 2023 due to parliamentary term coming to end (Valtiovarainministeriö, 2023). At the moment there is no new information on the state of the project. We will continue to monitor for such policy changes and make changes to the packages if necessary.

As mentioned earlier, both packages are published under a permissive BSD 2-clause license. We encourage our users to give feedback on the packages and their materials, report bugs or any legislative or policy changes related to NIN system implementations, study the source code and submit improvements to our public code repositories [12] or fork the code to better suit their needs.

## 6 Acknowledgements

We are grateful to all contributors, in particular Juuso Parkkinen and Joona Lehtomäki for their support in the initial package development. This work is part of rOpenGov [13] and contributes to the FIN-CLARIAH research infrastructure for computational humanities. LL, PK and AL were supported by the Research Council of Finland: decision 358720 (FIN-CLARIAH research infrastructure) and decision 352604 (Strategic Research Council, YOUNG Despair Research Consortium).

## References

M. Alastalo and I. Helén. A code for care and control: The pin as an operator of interoperability in the nordic welfare state. *History of the Human Sciences*, 35(1):242–265, 2022. URL https://doi.org/10.1177/09526951211017731. [p4, 5, 6]

A. Alterman. "A piece of yourself": Ethical issues in biometric identification. *Ethics and information technology*, 5(3):139–150, 2003. ISSN 1388-1957. [p5]

J. Anhøj. cprr: Functions for Working with Danish CPR Numbers, 2019. URL https://CRAN.R-project.org/package=cprr. R package version 0.2.0. [p4]

J. Brensinger and G. Eyal. The Sociology of Personal Identification. *Sociological Theory*, 2021. URL https://doi.org/10.1177/07352751211055771. OnlineFirst. [p5, 12]

CPR-kontoret. Personnummeret i CPR-systemet, 2008. URL https://cpr.dk/media/12066/personnummeret-i-cpr.pdf. Accessed: 22.4.2022. [p6]

Digital and Population Data Services Agency. The personal identity code, 2022a. URL https://dvv.fi/en/personal-identity-code. Accessed: 2022-01-17. [p5, 7, 12]

[11] For example, JSON has a license that states that "The Software shall be used for Good, not Evil". Defining what is good and evil is at least in part up to everyone's personal judgment, making the license clause ambiguous.

[12] https://github.com/rOpenGov/hetu, https://github.com/rOpenGov/sweidnumbr

[13] https://ropengov.org

Digital and Population Data Services Agency. Reform of the separators in the personal identity code, 2022b. URL https://dvv.fi/en/reform-of-personal-identity-code. Accessed: 2025-01-08. [p7]

M. Dodge and R. Kitchin. Codes of life: identification codes and the machine-readable world. *Environment and Planning D: Society and Space*, 23:851–881, 2005. [p4, 5]

M. Foucault. *Security, territory, population: lectures at the Collège de France, 1977-1978*. Palgrave Macmillan, New York, 2009. Editors: Michel Senellart, François Ewald, Alessandro Fontana, Arnold I. Davidson. [p4]

W. Freitas. numbersBR: Validate, Compare and Format Identification Numbers from Brazil, 2018. URL https://CRAN.R-project.org/package=numbersBR. R package version 0.0.2. [p4]

J. Furseth and O. Ljones. 50-årsjubilant med behov for oppgradering. *Samfunnsspeilet*, 2015(1), 2015. URL https://www.ssb.no/befolkning/artikler-og-publikasjoner/50-arsjubilant-med-behov-for-oppgradering. [p6]

M. Gissler and J. Haukka. Finnish health and social welfare registers in epidemiological research. *Norsk Epidemiologi*, 14(1):113–120, 2004. [p4, 11]

G. Grolemund and H. Wickham. Dates and times made easy with lubridate. *Journal of Statistical Software*, 40(3):1–25, 2011. URL https://www.jstatsoft.org/v40/i03/. [p8]

P. Hendricks. generator: Generate data containing fake personally identifiable information, 2015. URL https://CRAN.R-project.org/package=generator. R package version 0.1.0. [p4]

T. Jerlach. Udviklingen på CPR-området i de seneste 20-25 år frem til 2009, April 2009. URL https://cpr.dk/media/12060/udviklingen-paa-cpr-omraadet-frem-til-2009.pdf. [p6]

K. J. Krogness. Numbered individuals, digital traditions, and individual rights: civil status registration in Denmark 1645 to 2010. *Ritsumeikan Law Review*, 28:87–126, 2011. [p6]

E. Mäkelä, K. Lagus, L. Lahti, T. Säily, M. Tolonen, M. Hämäläinen, S. Kaislaniemi, and T. Nevalainen. Wrangling with non-standard data, 2020. [p11]

Official Statistics of Finland (OSF). Preliminary population statistics [online publication], March 2022. URL https://www.stat.fi/en/publication/cktih2lwgb3db0b531gwi04h8. Accessed: 22.4.2022. [p6]

B. Otjacques, P. Hitzelberger, and F. Feltz. Interoperability of E-Government Information Systems: Issues of Identification and Data Sharing. *Journal of Management Information Systems*, 23(4):29–51, 2007. URL https://doi.org/10.2753/MIS0742-1222230403. [p4, 5]

T. Salste. Henkilötunnus – ihmisten koodaaja, 2021. URL https://www.tuomas.salste.net/doc/tunnus/henkilotunnus.html. Accessed: 2021-12-13. [p5, 6, 7, 8]

Statistics Sweden. Personal identity number, 2016. [p5, 6]

Statistiska centralbyrån. SCB statistikdatabasen. [Elektronisk resurs] : Statistical database, 2022. URL https://www.scb.se/hitta-statistik/statistik-efter-amne/befolkning/befolkningens-sammansattning/befolkningsstatistik/pong/tabell-och-diagram/manadsstatistik--riket/befolkningsstatistik-2022/. Accessed: 22.4.2022. [p6]

R. Sund. Quality of the Finnish Hospital Discharge Register: A systematic review. *Scandinavian journal of Public Health*, 40:505–15, 8 2012. doi: 10.1177/1403494812456637. [p5]

The Swedish Tax Agency. Personnummer: Skv 704 ed. 8, 2007. [p10, 12]

Valtiovarainministeriö. Redesign of the personal identity code system lays the foundation for development of digital services, 2022. URL https://vm.fi/en/-/redesign-of-the-personal-identity-code-system-lays-the-foundation-for-development-of-digital-services. Accessed: 2025-01-08. [p12]

Valtiovarainministeriö. Legislative proposals on digital identity and redesigning the system of personal identity codes will not be considered during this parliamentary session, 2023. URL https://valtioneuvosto.fi/-/10623/lakiesityksia-digitaalisesta-henkilollisyydesta-ja-henkilotunnuksen-uudistamisesta-ei-ehdita-kasitella-talla-istuntokaudella?languageId=en_US. Acessed: 2025-01-08. [p12]

I. Watson. A short history of national identification numbering in Iceland. *Bifröst Journal of Social Science / Tímarit um félagsvísindi*, 1:51–89, 2010. ISSN 1670-7796. [p5]

H. Wickham and J. Bryan. R packages (2e), 2024. URL https://r-pkgs.org/introduction.html. Accessed: 202X-DD-MM. [p4]

Åke Johansson. Från bläckpenna till datorhjärna. *Deklarationen 100 år och andra tillbakablickar*, 2003. [p5]

*Pyry Kantanen*
*Department of Computing, University of Turku*
*Department of Computing, PO Box 20014 University of Turku, Finland*
*ORCiD: 0000-0003-2853-2765*
pyry.kantanen@utu.fi

*Erik Bülow*
*Department of Orthopaedics, Institute of Clinical Sciences, Sahlgrenska Academy at University of Gothenburg*
*Department of Orthopaedics, Institute of Clinical Sciences, Sahlgrenska Academy at University of Gothenburg, Sweden*
*ORCiD: 0000-0002-9973-456X*
erik.bulow@gu.se

*Aleksi Lahtinen*
*Department of Computing, University of Turku*
*Department of Computing, PO Box 20014 University of Turku, Finland*
*ORCiD: 0009-0009-9640-5187*
aleksi.l.lahtinen@utu.fi

*Måns Magnusson*
*Department of Statistics Uppsala University Sweden*
*Department of Statistics Uppsala University*
*ORCiD: 0000-0002-0296-2719*
mans.magnusson@statistik.uu.se

*Jussi Paananen*
*Institute of Biomedicine University of Eastern Finland*
*Institute of Biomedicine University of Eastern Finland, Finland*
*ORCiD: 0000-0001-5100-4907*
jussi.paananen@uef.fi

*Leo Lahti*
*Department of Computing, University of Turku*
*Department of Computing, PO Box 20014 University of Turku, Finland*
*ORCiD: 0000-0001-5537-637X*
leo.lahti@utu.fi

# GeoAdjust: Adjusting for Positional Uncertainty in Geostatistial Analysis of DHS Data

*by Umut Altay, John Paige, Andrea Riebler, and Geir-Arne Fuglstad*

**Abstract** The R-package GeoAdjust adjusts for positional uncertainty in GPS coordinates and performs fast empirical Bayesian geostatistical inference for household survey data from the Demographic and Health Surveys (DHS) Program. DHS household survey data is important for tracking demographic and health indicators, but is published with intentional positional error to preserve the privacy of the household respondents. Such jittering has recently been shown to deteriorate geostatistical inference and prediction, and GeoAdjust is the first software package that corrects for jittering in geostatistical models containing both spatial random effects and raster- and distance-based covariates. The package provides inference for model parameters and predictions at unobserved locations, and supports Gaussian, binomial and Poisson likelihoods with identity link, logit link, and log link functions, respectively. GeoAdjust provides functions that make model and prior specification intuitive and flexible for the user, as well as routines for plotting and output analysis.

## 1 Introduction

The Demographic and Health Surveys (DHS) Program[1] implements household surveys to collect and disseminate nationally representative data about health, population, HIV and nutrition in low- and middle-income countries. The DHS Program started in 1984, and has been implemented in overlapping 5-year phases. So far more than 400 surveys have been conducted in over 90 countries. Standard DHS surveys usually include between 5 000 and 30 000 households. GPS coordinates of household centres are provided to allow for spatial analyses of the collected demographic and health data, but the DHS has added intentional positional errors into the published GPS coordinates to protect the privacy of the survey respondents (Burgert et al., 2013).

The random displacement procedure, or *jittering* scheme, is publicly known (Burgert et al., 2013), but traditional geostatistical analyses assume that the locations are known exactly and ignore the jittering. However, we have recently demonstrated that ignoring the positional error in DHS data may lead to attenuated estimates of the covariate effect sizes and reduced predictive performance (Altay et al., 2022b).

While common practice is to ignore jittering, some approaches have been proposed to account for it. With respect to the error induced in spatial covariates, Warren et al. (2016) proposed regression calibration for distance-based covariates, and Perez-Heydrich et al. (2013, 2016) proposed using a 5 km moving window (or buffer zone) for raster-based covariates. However, these approaches do not address the attenuation arising in the covariate effect sizes when replacing the true covariate with a proxy. With respect to the error induced in the spatial effect, Fanshawe and Diggle (2011) proposed a Bayesian approach in the limited setting of no covariates and Gaussian observation model. Wilson and Wakefield (2021) proposed a more complex approach using INLA-within-MCMC (Rue et al., 2009; Gómez-Rubio and Rue, 2018), which could handle the error induced in both the spatial random effect and in spatial covariates, but computation time is too extensive for routine use of the approach. None of the mentioned papers provide an R package for easy application of the methods.

The datasets from DHS are semi-public and one must apply for access. A step-by-step explanation of the application procedure can be found at `https://dhsprogram.com/data/new-user-registration.cfm`. The application requires a brief project description explaining why the data set is needed and how it will be used. Permission is typically granted within a few days.

With the R package **GeoAdjust**, we address the need for fast, flexible and user-friendly software to estimate geostatistical models for DHS data subject to positional uncertainty. **GeoAdjust** corrects for the positional uncertainty by adjusting for jittering both in the spatial random effect and the spatial covariates, and achieves fast inference by combining the computational efficiency of the stochastic partial differential equations (SPDE) approach (Lindgren et al., 2011) with the autodifferentiation features of Template Model Builder (TMB) (Kristensen et al., 2016). We use the R-package **fmesher** (Lindgren, 2023) for computing the mesh and discretization matrices for the SPDE approach, and the R-package **TMB** for easy use of the TMB methodology. **GeoAdjust** is available on CRAN (R Core Team, 2022) and can be installed with the command `install.packages("GeoAdjust")`. While there are

[1] `https://dhsprogram.com`

other R packages such as **SUMMER** (Li et al., 2020, 2022) that can perform spatial or spatio-temporal areal analysis of DHS data, none can account for jittering in geostatistical analysis.

## 2 Geostatistical inference under jittering

We describe a country of interest as a spatial domain $\mathcal{D} \subset \mathbb{R}^2$, and we assume that $C$ small groups of households, called *clusters*, are observed within the country. For clusters $c = 1, \ldots, C$, we denote the true location by $\boldsymbol{s}_c^* \in \mathcal{D}$, and we denote the observed (jittered) location, provided by DHS surveys, by $\boldsymbol{s}_c \in \mathcal{D}$. Additionally, each cluster has a known classification as urban (U) or rural (R). The urban/rural designation, $\text{Urb}[c] \in \{\text{U}, \text{R}\}$, is important since DHS surveys use different jittering mechanisms in urban and rural clusters. Urban clusters are jittered up to 2 km, and rural clusters are jittered up to 5 km with probability 0.99 and jittered up to 10 km with probability 0.01 (Burgert et al., 2013). The angle and jittering distance are sampled from uniform distributions, but the boundaries of either the first or the second administrative level are respected. We denote these *known* jittering distributions by $\pi_{\text{Urb}[c]}(\boldsymbol{s}_c|\boldsymbol{s}_c^*)$, $c = 1, \ldots, C$.

An observation $y_c$ is made at each cluster $c$, and the responses $y_1, \ldots, y_C$ and the observed locations $\boldsymbol{s}_1, \ldots, \boldsymbol{s}_C$ are modelled jointly as

$$y_c \mid \mu_c, \boldsymbol{\phi} \sim \pi(y_c \mid \mu_c, \boldsymbol{\phi}), \quad \boldsymbol{s}_c|\boldsymbol{s}_c^* \sim \pi_{\text{Urb}[c]}(\boldsymbol{s}_c|\boldsymbol{s}_c^*), \quad c = 1, \ldots, C, \tag{1}$$

where $\pi(y_c \mid \mu_c, \boldsymbol{\phi})$ denotes the likelihood of $y_c$ given the mean $\mu_c$ and the vector of likelihood parameters $\boldsymbol{\phi}$. The intuition is that the observed response $y_c$ and the observed location $\boldsymbol{s}_c$ are independent random variables, which differ from the mean $\mu_c$ and the true location $\boldsymbol{s}_c^*$, respectively. The mean is linked to a linear predictor $\eta_c$ through a link function $g$ as

$$g(\mu_c) = \eta_c.$$

The package implements the identity link in the case of a Gaussian likelihood, the log-link for Poisson likelihood, and the logit-link for the binomial likelihood.

We model latent spatial variation in a traditional way as

$$\eta(\boldsymbol{s}^*) = \boldsymbol{x}(\boldsymbol{s}^*)^{\text{T}}\boldsymbol{\beta} + u(\boldsymbol{s}^*), \quad \boldsymbol{s}^* \in \mathcal{D},$$

where $\boldsymbol{x}(\cdot)$ is a vector of $p$ spatial covariates, $\boldsymbol{\beta}$ is a vector of $p$ coefficients, and $u(\cdot)$ is a Matérn Gaussian random field (GRF). The GRF $u(\cdot)$ is controlled by the three parameters: smoothness $\nu$, which is fixed to 1, spatial range $\rho_{\text{S}}$, and marginal variance $\sigma_{\text{S}}^2$. The key difference from a standard geostatistical model is that the mean

$$\mu_c = g^{-1}(\eta_c) = g^{-1}(\eta(\boldsymbol{s}_c^*))$$

depends on the *unknown* true location $\boldsymbol{s}_c^*$. This means that we do not know from which pixel to extract covariates and we do not know at which location to evaluate the GRF.

We choose a uniform prior $\boldsymbol{s}_c^* \sim \mathcal{U}(\mathcal{D})$ for the true cluster location, implying that all $\boldsymbol{s}_c^*$ compatible with $\boldsymbol{s}_c$ are equally likely *a priori*, $c = 1, \ldots, C$. In the case of Nigeria, which is the country used as an example in later sections, compatible refers to all potential true cluster locations lying in the same second administrative area (admin2) as the observed location and within the maximum jittering distance. More complicated priors that take population density or urban/rural status into account are possible, but such rasters would have to be estimated and could be biased and uncertain. Further, $\boldsymbol{\beta} \sim \mathcal{N}_p(\mathbf{0}, V\mathbf{I}_p)$, where $V$ is a fixed variance, and $\rho_{\text{S}}$ and $\sigma_{\text{S}}^2$ are assigned penalized complexity (PC) priors with $\text{P}(\rho_{\text{S}} > \rho_0) = 0.50$ and $\text{P}(\sigma_{\text{S}} > 1) = 0.05$ (Fuglstad et al., 2019). We recommend choosing the median range $\rho_0$ as 10% of the diameter of $\mathcal{D}$ to be able to capture the spatial variability at moderate distances.

For inference, **GeoAdjust** treats the unknown true locations as nuisance parameters and integrates them out,

$$\begin{aligned}\pi(y_c, \boldsymbol{s}_c|\eta(\cdot)) &= \int_{\mathcal{D}} \pi(y_c, \boldsymbol{s}_c|\eta(\cdot), \boldsymbol{s}_c^*)\pi(\boldsymbol{s}_c^*)\, \text{d}\boldsymbol{s}_c^* \\ &= \int_{\mathcal{D}} \pi(y_c|\eta(\boldsymbol{s}_c^*))\pi_{\text{Urb}[c]}(\boldsymbol{s}_c|\boldsymbol{s}_c^*)\pi(\boldsymbol{s}_c^*)\, \text{d}\boldsymbol{s}_c^*. \end{aligned} \tag{2}$$

This means that the likelihood of $y_c$ is considered to be a mixture distribution over all true locations $\boldsymbol{s}_c^*$ compatible with the observed location $\boldsymbol{s}_c$, where the weighting is informed by the prior and the known jittering mechanism. In the implementation, the integral is computed numerically using a integration scheme constructed with rings of integration points centered around the DHS provided

cluster location $s_c$. Hence, Equation (2) is approximated by a finite mixture over potential true locations. Computational efficiency is achieved by combining the SPDE approach (Lindgren et al., 2011) to describe $u(\cdot)$ and TMB which allows for fast and flexible autodifferentiation. For details on the the integration scheme and the inference scheme, we refer to Altay et al. (2022a,b).

## 3 Package structure and functionality

**GeoAdjust** handles the technical steps of the method described in the previous section in order to make the adjustment for jittering widely accessible. Figure 1 illustrates the structure of **GeoAdjust**, and how various data inputs are processed through the package workflow. The main functionality of the package is described below, and is broken down into the three main steps of the workflow illustrated in Figure 1: input preparation, estimation, and prediction.

### 3.1 Step 1: Input preparation

Before estimation and prediction, **GeoAdjust** requires a set of triangular basis functions forming a 'mesh' that is necessary for the SPDE approach, and also a separate data structure containing relevant information about the input datasets and the jittering. **GeoAdjust** facilitates the preparation of these inputs via the functions `meshCountry` and `prepareInput`. In **GeoAdjust**, the GRF $u(\cdot)$ is approximated using the so-called SPDE approach. This requires the construction of a constrained refined Delaunay triangulation (CRDT), a mesh over the country of interest. The approximated spatial field can then be projected from the mesh nodes to the cluster centers via projector matrices (Lindgren et al., 2011). The function `meshCountry` creates a triangular mesh based on the national borders. It has five arguments: `max.edge` is a vector of two values, where its first and second elements represent the largest allowed triangle edge lengths for the inner and outer mesh, respectively, and `offset` stands for the extension distance outside the country borders. A negative value is interpreted as a relative extension, e.g., $-0.08$ means an extension of 8%. The argument `admin0` is an **sf** (simple features) object of class `MULTIPOLYGON` containing the geometry of the national borders of the country, `cutoff` is the minimum allowed distance of the vertices to each other (Lindgren, 2023), and `target_crs` describes the coordinate reference system (CRS) that the function operates within. The CRS string is set by the user based on where on earth the user wishes to do their modeling, since different projections are intended for use in different parts of the world. See Step 1 in the Nigeria example for a demonstration.

The integration in Equation (2) is performed numerically. To calculate the integrals, we need a set of integration points around the associated jittered survey cluster centers. **GeoAdjust** specifies the cluster center itself as the first integration point and builds either 5 or 10 rings around it, depending on whether it is located in an urban or a rural stratum, respectively. Each ring contains a set of 15 angularly equidistant 'primary' integration points as illustrated in the left panel of Figure 2. The first 5 rings are called the "inner rings". An additional 5 rings are constructed for the rural cluster centers, and are called the "outer rings". The primary integration points are assigned equal weight *a priori* within any single ring, where the weight for a given ring is determined by the jittering distribution. Weights of individual points are adjusted, however, for relevant subnational boundaries. If an observed cluster location is closer to the nearest relevant subnational border than the maximum jittering distance, a set of secondary integration points are constructed, each with an associated primary integration point, and the weight of each primary integration point is distributed among the associated secondary integration points. Zero weight is assigned to any secondary integration points that are across the border, and weights are then reaggregated to the primary integration points to create the final integration weights of the primary integration points. Figure 2 shows an example set of primary and secondary integration points and the corresponding integration weights for a single cluster from the Kenya 2014 DHS household survey. The supplementary materials of (Altay et al., 2022a) provides a detailed mathematical explanation of the procedure.

The function `prepareInput` creates the set of integration points and weights with respect to the urban/rural strata, and constructs the urban and rural design matrices by extracting the covariate values at each integration point. Internally, `prepareInput` function conducts various distance based calculations and comparisons, all measured in kilometers. Therefore, the measurement unit of the `target_crs` **must** also be in kilometers. The output of `prepareInput` is a list containing the strata-wise design matrices and response vectors, together with the sparse matrix components of the SPDE model, and strata-wise projector matrices. The argument `likelihood` can be `0` (Gaussian), `1` (Binomial), and `2` (Poisson). For the Gaussian and Poisson likelihoods, the argument `response` contains a list containing a vector `ys` containing observed values, and, for the Binomial likelihood, the argument `response` contains a list with a vector `ns` with the number of trials and a vector `ys` with the number of successes. See Step 1 in the Nigeria example for a demonstration.

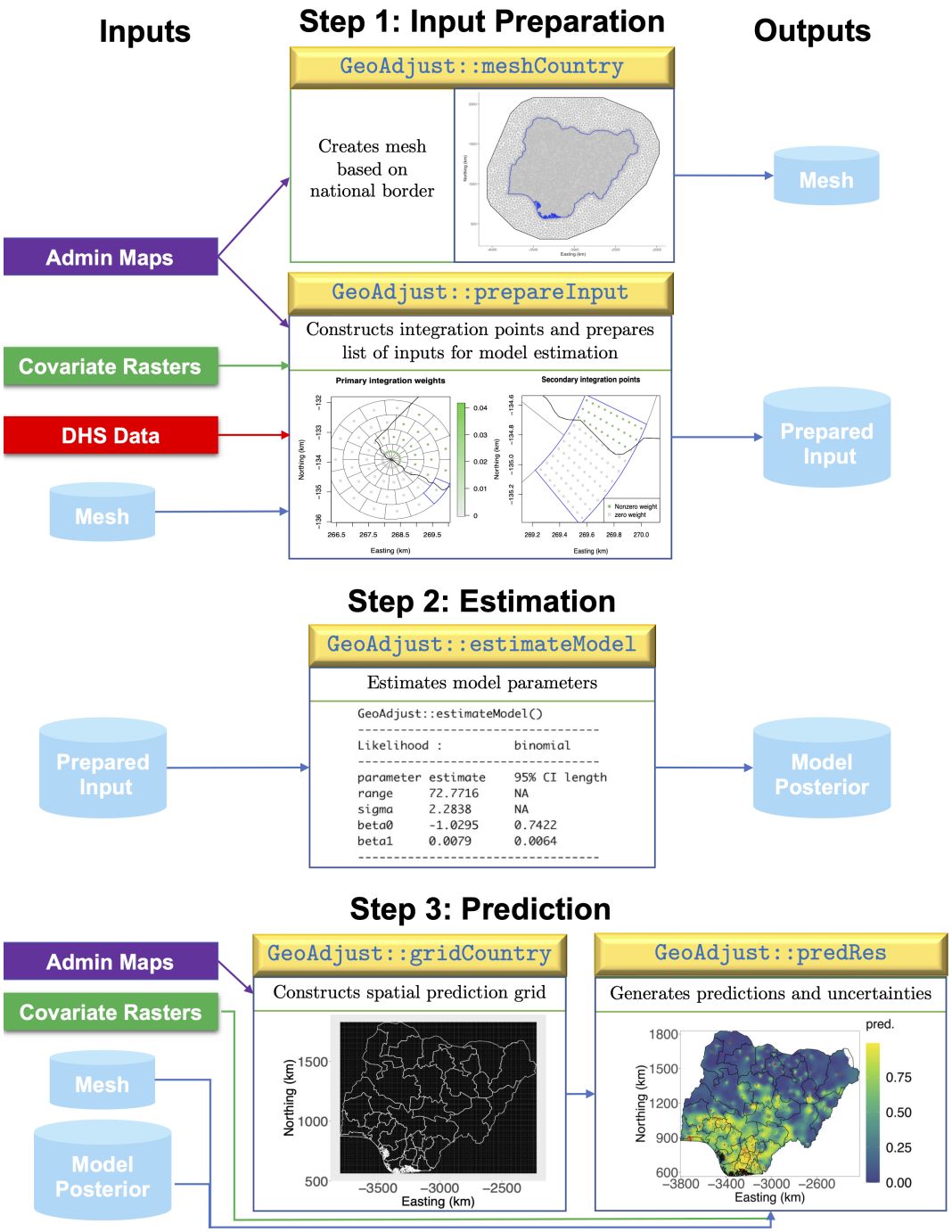


**Figure 1:** A visual representation of GeoAdjust workflow.

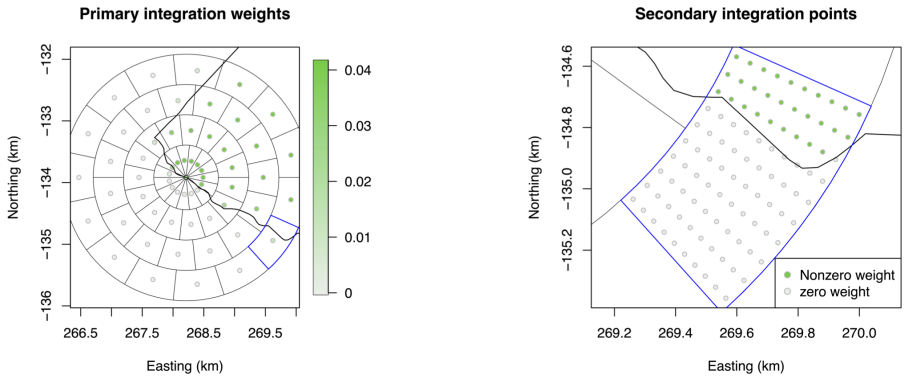


**Figure 2:** Illustration of primary (left) and secondary (right) integration weights for one cluster from Kenya 2014 DHS household survey.

### 3.2 Step 2: Estimation

The list returned by the `prepareInput` function described in the previous section contains elements that will be processed by TMB in `estimateModel`. The `estimateModel` function is a wrapper function built around C++ code implementing our model in TMB, allowing the user to estimate model parameters and to use TMB's autodifferentiation features without needing to know or program in C++. The main argument of `estimateModel` is a list called `data`, referring to the input list that has been created by `prepareInput` function. The function also allows different prior choices for the model components, via its argument called `priors`.

The `priors` argument allows the user to specify the parameters of the Gaussian prior for covariate effect sizes, and of the penalized complexity (PC) priors for the spatial range. These values can be passed into the function as a list of two elements, namely, `beta` and `range`. The element `beta` needs to be a vector of length two. The first and the second elements of the vector `beta` are the mean and the standard deviation of the Gaussian priors that are assigned for the intercept and the covariate effect sizes. The element `range` refers to the *a priori* median range. Further, the PC priors for marginal variance and measurement variance are passed as `Uspatial`, `alphaSpatial`, `UNugget`, and `alphaNug`. `USpatial` is the upper `alphaSpatial` percentile of the marginal standard deviation, and `UNugget` and `alphaNug` are the hyperparameters for the PC-prior on the nugget variance. The hyperparameters `UNugget` and `alphaNug` pass into the function as 1 and 0.05, by default, but they are only used in the calculations when the likelihood is Gaussian. See Step 2 in the Nigeria example for a demonstration.

Parameter estimation and model fitting via `estimateModel` integrates out the unknown true coordinates by computing the contribution of each integration point to the joint negative log-likelihood. Internally, once the TMB function `MakeADFun` constructs the core model object (Kristensen, 2022), `estimateModel` inputs the objective function and its gradient into the optimization routine, `optim`. Afterwards, `estimateModel` extracts the estimated model parameters from the optimized core model object, and draws `n.sims` posterior samples. This includes samples of the intercept, each covariate effect, and the spatial random effect coefficients for each mesh node as well. The samples for the intercept and the covariate effect sizes are then used for constructing the 95% credible interval lengths as the measure of uncertainty corresponding to the estimated parameters.

The function `estimateModel` returns a list of four elements. The list contains a data frame of the estimated model parameters, together with the optimized core model object, a matrix containing the `n.sims` posterior samples, and information about the type of the likelihood. The core model object and the posterior draws can then be passed to the function `predRes` to generate predictions at a set of prediction locations. The object returned by `estimateModel` can be printed in a tidy way using `print`. See Step 2 in the Nigeria example for a demonstration.

### 3.3 Step 3: Prediction

Once the model parameters are estimated, the model can be used for predicting the model outcomes at a new set of locations. The function `gridCountry` in **GeoAdjust** helps with the construction of a set of prediction points. The function creates a `SpatRaster` of the desired resolution within the bounding box of the national level shape file, extracts the coordinates of the cell centers and returns them as an `sf` class `POINT` object together with the raster, as the elements of a list.

The `gridCountry` function has three arguments. The first argument, `admin0`, should be set to an `sf` object of class `MULTIPOLYGON` containing the national borders. The second argument, `res`, indicates

the desired resolution of the grid in kilometers, and the last argument is `target_crs`. Internally, `gridCountry` first creates a `SpatRaster` within the bounding box of the `admin0` `MULTIPOLYGON`, with the chosen resolution. Afterwards, it extracts the coordinates of the cell centroids and converts them into an `sf` class `POINT` object. The function returns the `sf` `POINT` object and the `SpatRaster` within a list. See Step 3 in the Nigeria example for a demonstration.

The `sf` `POINT` object goes into the function `predRes` as the prediction locations. Obtaining predictions at a new set of locations with the function `predRes` requires the optimized core model object, drawn samples of the parameters and the random effect coefficients, triangular mesh, a list of covariate rasters, coordinates of the prediction locations and an argument called `flag` to be passed as inputs. The argument `flag` is used for passing the likelihood type into the function. The integers 0, 1 and 2 indicate the Gaussian, binomial and Poisson likelihoods, respectively, and the function deploys the corresponding link function as outlined before. The package allows the use of any number of covariates, as long as they are `SpatRaster` objects. The covariates are passed into the function within a single list. The function will extract the values from each one of them at the prediction locations and form a design matrix. The coordinates of the prediction locations has to be an `sf` `POINT` object.

Internally, `predRes` combines the sampled covariate effect sizes and the random effect coefficients with the design matrix and forms one model per sample, `n.sims` models in total. Each model predicts outcomes across the set of prediction locations. Finally, the function calculates the mean, median, standard deviation, and the upper and lower bounds of 95% credible intervals of predictions for each prediction point. These results are returned in a matrix with a number of rows equal to the number of prediction points, and 5 columns. See the Step 3 in the Nigeria example for a demonstration.

The prediction raster will be used by the function `plotPred`, which internally utilizes `geom_raster` from `ggplot2`, to plot the predictions and the corresponding uncertainty across the country, as demonstrated in Step 3 of the Nigeria example.

# 4 Example: Spatial analysis of completion of secondary education in Nigeria

## 4.1 Problem description

This example considers spatial analysis of the completion of secondary education among women aged 20–49 years. As demonstrated in Altay et al. (2022b), this is a case where not accounting for jittering would substantially change the results. The data source is the 2018 DHS survey in Nigeria (NDHS2018) (National Population Commission - NPC and ICF, 2019), where there are $C = 1380$ clusters with valid GPS coordinates inside Nigeria. In these clusters, 15 490 out of 33 193 women aged 20–49 completed secondary education. We demonstrate how to conduct the geostatistical analysis using **GeoAdjust**.

We assume a binomial likelihood for the model described in the method section, and assume

$$\begin{aligned} y_c|r_c, n_c &\sim \text{Binomial}(n_c, r_c), \quad \boldsymbol{s}_c|\boldsymbol{s}_c^* \sim \pi_{\text{Urb}[c]}(\boldsymbol{s}_c|\boldsymbol{s}_c^*), \\ r_c &= r(\boldsymbol{s}_c^*) = \text{logit}^{-1}(\eta(\boldsymbol{s}_c^*)), \end{aligned} \tag{3}$$

where $y_c$ is the number of women who completed secondary education, $n_c$ is the number of women interviewed, and $r_c$ denotes the risk in cluster $c$, for $c = 1, \ldots, C$. The spatially varying risk $r(\cdot) = \text{logit}^{-1}(\eta(\cdot))$ is modelled through the linear predictor

$$\eta(\boldsymbol{s}^*) = \beta_0 + x(\boldsymbol{s}^*)\beta_1 + u(\boldsymbol{s}^*), \quad \boldsymbol{s}^* \in \mathcal{D},$$

where $\beta_0$ is the intercept, $x(\cdot)$ is the spatially varying population density, $\beta_1$ is the coefficient of population density, and $u(\cdot)$ is the Matérn GRF with known smoothness $\nu = 1$, and unknown range $\rho_S$ and marginal variance $\sigma_S^2$.

There are 774 admin2 areas, which are called local government areas, and, in Nigeria, DHS's jittering mechanism does not move the GPS coordinates of a cluster outside its original admin2 area. The goal of the spatial analysis is to produce estimates of model parameters, and to map spatial variation in completion of secondary association with associated uncertainties.

## 4.2 Step 0: Data preprocessing

The population density raster file ('Nga_ppp_v2c_2015.tif') can be downloaded from WorldPop (World Pop, 2022). Further, we need a description of the national (admin0) borders, and the admin2 borders. Shape files of the administrative levels for different countries can be obtained The Database of Global

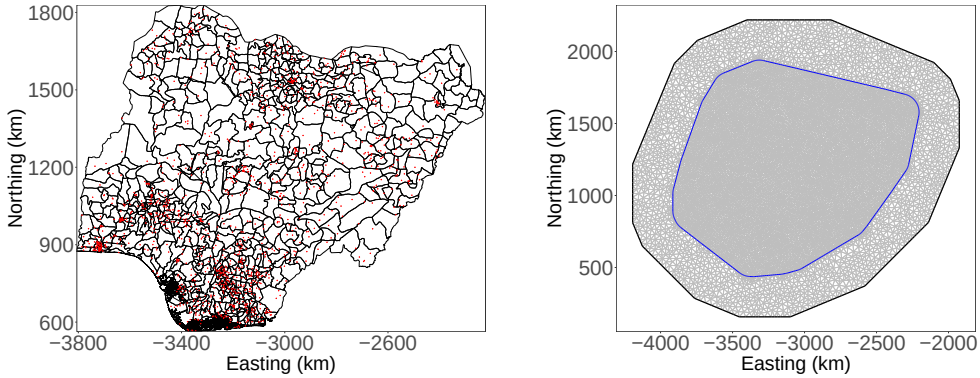


**Figure 3:** Nigeria subnational level map (left) and the triangular mesh (right). The red points represent the jittered cluster centers.

Administrative Areas (GADM)[2]. Appendix A shows how to load the shape files, and we assume that admin0 level and admin2 level are stored in the data objects `admin0` and `admin2`, respectively.

The DHS surveys consists of individual level responses together with geographic information about the associated cluster. The surveys consists of many questions, and extracting the desired 0/1 response require preprocessing steps that are specific to a given response. The R code for pre-processing the DHS data, administrative borders shape files and the covariate rasters is given in Appendix A. To ease reading, we assume that the preprocessing steps in Appendix A have been completed and that the required variables have been stored in a data frame object named `nigeria.data`. This data object is used in the following sections and contains:

- `clusterID`: The identification number that is assigned by DHS to each household cluster center
- `long`: Longitude coordinate of the corresponding cluster center.
- `lat`: Latitude coordinate of the corresponding cluster center.
- `ys`: Number of 20–49 years old women who reported completing their secondary education in the cluster.
- `ns`: Total number of 20–49 years old women survey participants in the cluster.
- `urbanRuralDHS`: Urbanization strata of the cluster.

Further, `pointsKM` contains the household cluster center coordinates.

### 4.3 Step 1: Input preparation

In the analysis we use the local coordinate system UTM 37 with km as the length unit. Since the shape files use a longitude/latitude coordinate reference system, they must be transformed into the local coordinate system. We construct the triangular mesh using the function `meshCountry`. We use an offset of 8% for the external mesh, maximum edge length of 25 km in the internal mesh, and maximum edge length 50 km in the external mesh, and do not include boundary points closer than 4 km in the mesh.

```
# Set target geometry
target_crs = "+units=km +proj=utm +zone=37 +ellps=clrk80
              +towgs84=-160,-6,-302,0,0,0,0 +no_defs"

# transform admin0 borders into target_crs:
admin0_trnsfrmd = sf::st_transform(admin0, target_crs)

# construct the mesh
mesh.s = meshCountry(admin0= admin0_trnsfrmd,
                     max.edge = c(25, 50),
                     offset = -.08, cutoff=4,
                     target_crs = target_crs)
```

Figure 3 shows the admin2 borders together with the resulting triangular mesh. We use the function `prepareInput` to collect all data required for estimation and to precompute integration points and integration weights necessary for the method described in the method section.

[2] https://gadm.org/data.html

```
# read the covariate raster
library(terra)
r = terra::rast("Nga_ppp_v2c_2015.tif")

inputData = prepareInput(response=list(ys=nigeria.data$ys,ns=nigeria.data$ns),
                                  locObs = pointsKM,
                                  likelihood = 1,
                                  urban = nigeria.data$urbanRuralDHS,
                                  mesh.s = mesh.s,
                                  adminMap = admin2,
                                  covariateData = list(r),
                                  target_crs = target_crs)
```

Since the likelihood is binomial, we set the argument `response` to a list containing the number of trials `ns` and a list of the number of successes `ys` for the clusters. Here, `ns` is $(n_1, \ldots, n_{1380})^{\mathrm{T}}$ and `ys` is $(y_1, \ldots, y_{1380})^{\mathrm{T}}$. We pass the covariate `SpatRaster` objects within a list, through the argument `list(terra::rast(r = r))`. **GeoAdjust** allows modelling the data with either one of Gaussian, binomial or Poisson likelihoods. Accordingly, the likelihood type needs to be passed into the function via the argument `likelihood`, by setting it to either 0, 1 or 2, respectively. We set the binomial likelihood with `likelihood = 1`.

## 5 Step 2: Estimation

We estimate the model using the function `estimateModel`.

```
# estimating the parameters
est = estimateModel(
      data = inputData,
      priors = list(beta = c(0,1), range = 114),
                    USpatial = 1, alphaSpatial = 0.05,
                    UNugget = 1, alphaNug = 0.05,
      n.sims = 1000)
```

Here we set priors $\beta_0, \beta_1 \overset{\text{iid}}{\sim} \mathcal{N}(0,1)$ using `beta = c(0,1)`. We choose the prior on marginal variance $\sigma_S^2$ such that $\mathrm{P}(\sigma_S > 1) = 0.05$ through `USpatial = 1` and `alphaSpatial = 0.05`. We use `n.sims = 1000` draws from the estimated posteriors. The median of the prior on range $\rho_S$ is set to `range = 114` km. The function `estimateModel` returns a list of four elements: `res`, `obj`, `draws` and `likelihood`.

```
# the output of estimateModel() function:
names(est)
[1] "res"        "obj"        "draws"      "likelihood"

print(est)

GeoAdjust::estimateModel()
---------------------------------
Likelihood :          binomial
---------------------------------
parameter estimate    95% CI length
range     69.7677     NA
sigma     2.1462      NA
intercept -1.2998     0.6578
beta1     0.0069      0.0044
---------------------------------
```

The elements `obj` and `draws` are used for prediction in the next section. The element `likelihood` indicates the likelihood type (`0` is Gaussian, `1` is binomial, and `2` is Poisson) that is used in the model construction, and `res` contains the estimated model parameters and the lengths of 95% credible intervals. The credible interval lengths are calculated as the difference between the 97.5% and 2.5% percentiles. The result object `res` does not contain `CI_Length` values for the range and the marginal variance, as the inference is empirical Bayesian where these parameters are estimated to fixed values.

## 6 Step 3: Prediction

We grid the country using the function `gridCountry` with the admin0 boundaries and a resolution of 5 km.

```
# raster and the prediction coordinates:
predComponents = gridCountry(admin0 = admin0,
                             res = 5,
                             target_crs = target_crs)

names(predComponents)
[1] "loc.pred" "predRast"

# the sf multipoint object containing the prediction locations
loc.pred = predComponents[["loc.pred"]]

> print(loc.pred)
Simple feature collection with 80201 features and 0 fields
Geometry type: POINT
Dimension:     XY
Bounding box:  xmin: -3803.253 ymin: 565.4467
               xmax: -2223.253 ymax: 1825.447
Projected CRS: +units=km +proj=utm +zone=37 +ellps=clrk80 +towgs84=-160,-6,-302,
0,0,0,0 +no_defs
First 10 features:
                     geometry
1  POINT (-3803.253 1825.447)
2  POINT (-3798.253 1825.447)
3  POINT (-3793.253 1825.447)
4  POINT (-3788.253 1825.447)
5  POINT (-3783.253 1825.447)
6  POINT (-3778.253 1825.447)
7  POINT (-3773.253 1825.447)
8  POINT (-3768.253 1825.447)
9  POINT (-3763.253 1825.447)
10 POINT (-3758.253 1825.447)

predRast = predComponents[["predRast"]]

> print(predRast)
class       : SpatRaster
dimensions  : 253, 317, 1  (nrow, ncol, nlyr)
resolution  : 5, 5  (x, y)
extent      : -3805.753, -2220.753, 562.9467, 1827.947
              (xmin, xmax, ymin, ymax)
coord. ref. : +proj=utm +zone=37 +ellps=clrk80 +towgs84=-160,-6,-302,0,0,0,
0 +units=km +no_defs
```

The output shows that the grid cell centroids are 5 km apart and the dimension of the grid is $253 \times 317$. This includes locations that are outside the admin0 boundaries, which will be masked when plotting the predictions.

We use the function `predRes` with `flag = 1` to indicate the Binomial likelihood, and that the inverse of the logit needs to be applied to the linear predictor. The argument `covariateData` contains a list of one element which is the population density raster. Additionally, we input the objects `obj` and `draws` from the function `estimateModel`.

```
predictions = predRes(obj = est[["obj"]] , predCoords = loc.pred,
                      draws = est[["draws"]],
                      covariateData = list(r),
                      mesh.s = mesh.s, flag = 1)

head(predictions)
          mean    median         sd       lower     upper
[1,] 0.2155746 0.2143489 0.02934076 0.165192003 0.2764130
```

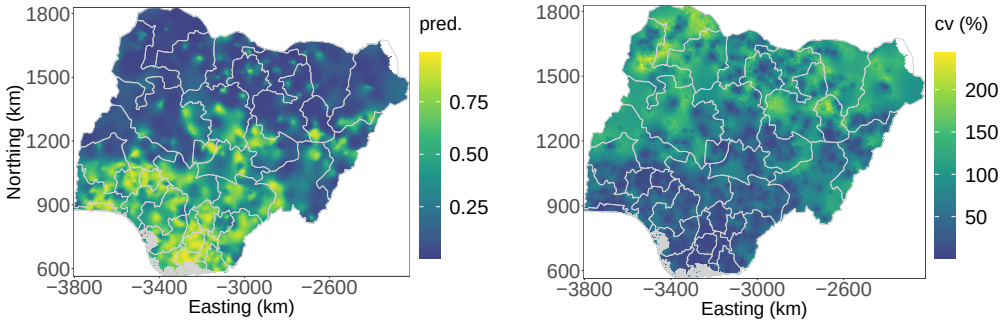


**Figure 4:** Predicted risk (left) and the CVs (right). The red points indicate the example survey cluster centers.

```
[2,] 0.3471947 0.2220279 0.33308573 0.001691273 0.9860286
[3,] 0.3442247 0.2269676 0.32558640 0.002292870 0.9839695
[4,] 0.3447684 0.2402942 0.32414517 0.001921544 0.9817542
[5,] 0.3379962 0.2405484 0.31493574 0.002677005 0.9755113
[6,] 0.3329591 0.2317648 0.30754117 0.003479042 0.9637670

dim(predictions)
[1] 80201     5
```

The result object contains the desired quantities for each grid cell in Nigeria. We use the function `plotPred` to plot the predictions and the corresponding uncertainty accross the studied country. The uncertainty is quantified as the coefficient of variation (CV), which is calculated as $\frac{\sigma}{\mu} \times 100$, where $\sigma$ and $\mu$ is the standard deviation and mean, respectively, of the predictive distribution.

```
admin1 = st_read("gadm40_NGA_shp/gadm40_NGA_1.shp")

plotPred(pred = predictions,
         predRaster = predRast,
         admin0 = admin0,
         admin1 = admin1,
         admin2 = admin2,
         rmPoly = 160,
         target_crs = target_crs)
```

Here we provide `predRaster`, which is the locations and geography for prediction, and the predicted values `pred`. The argument `admin0` is used to mask values outside Nigeria, the argument `admin1` is used to plot the first administrative level (admin1) borders, and the argument `admin2` together with `rmPoly = 160` is used to remove the admin2 area corresponding to the lake, which is not a real admin2 area, from plotting. The arguments `rmPoly` and `admin2` should be set to `NULL` if all admin2 areas should be plotted. Figure 4 shows the resulting predictions and CVs. The function returns a list containing two `ggplot` objects, representing the plots for the predictions and uncertainty across the country of interest.

## 7 Summary

**GeoAdjust** allows fast and easy geostatistical analysis of DHS household survey data while accounting for jittering. The user can take advantage of a novel complex method (Altay et al., 2022a,b) and control settings without being exposed to complex code. The user also has access to convenient plotting functions, and the backend uses **sf** and **terra** to handle spatial data with information on coordinate systems and rasters. **GeoAdjust** is the only package that addresses the positional uncertainty in DHS data, and has the potential to be extended to combine areal and point referenced data from different areas involving the both positional uncertainty and geomasking.

## References


U. Altay, J. Paige, A. Riebler, and G.-A. Fuglstad. Fast geostatistical inference under positional uncertainty: Analysing DHS household survey data. *arXiv preprint arXiv:2202.11035*, 2022a. [p17, 24]

U. Altay, J. Paige, A. Riebler, and G.-A. Fuglstad. Jittering impacts raster- and distance-based geostatistical analyses of DHS data. *arXiv preprint arXiv:2202.07442v1*, 2022b. [p15, 17, 20, 24]

C. R. Burgert, J. Colston, T. Roy, and B. Zachary. Geographic displacement procedure and georeferenced datarelease policy for the Demographic and Health Surveys. https://dhsprogram.com/pubs/pdf/SAR7/SAR7.pdf, 2013. DHS Spatial Analysis Reports No. 7. [p15, 16]

T. Fanshawe and P. Diggle. Spatial prediction in the presence of positional error. *Environmetrics*, 22(2): 109–122, 2011. [p15]

G.-A. Fuglstad, D. Simpson, F. Lindgren, and H. Rue. Constructing priors that penalize the complexity of Gaussian random fields. *Journal of the American Statistical Association*, 114:445–452, 2019. [p16]

V. Gómez-Rubio and H. Rue. Markov chain Monte Carlo with the integrated nested Laplace approximation. *Statistics and Computing*, 28(5):1033–1051, 2018. [p15]

K. Kristensen. *The comprehensive TMB documentation*, 2022. https://kaskr.github.io/adcomp/_book/Introduction.html. [p19]

K. Kristensen, A. Nielsen, C. W. Berg, H. Skaug, and B. M. Bell. TMB: Automatic differentiation and Laplace approximation. *Journal of Statistical Software*, 70(5):1–21, 2016. doi: 10.18637/jss.v070.i05. [p15]

Z. R. Li, B. D. Martin, T. Q. Dong, G.-A. Fuglstad, J. Paige, A. Riebler, S. Clark, and J. Wakefield. Space-time smoothing of demographic and health indicators using the r package summer. *arXiv preprint arXiv:2007.05117*, 2020. [p16]

Z. R. Li, B. D. Martin, Y. Hsiao, J. Godwin, J. Paige, P. Gao, J. Wakefield, S. J. Clark, G.-A. Fuglstad, and A. Riebler. *SUMMER: Small-Area-Estimation Unit/Area Models and Methods for Estimation in R*, 2022. URL https://CRAN.R-project.org/package=SUMMER. R package version 1.3.0. [p16]

F. Lindgren. *fmesher: Triangle Meshes and Related Geometry Tools*, 2023. URL https://CRAN.R-project.org/package=fmesher. R package version 0.1.2. [p15, 17]

F. Lindgren, H. Rue, and J. Lindström. An explicit link between Gaussian fields and Gaussian Markov random fields: the stochastic differential equation approach (with discussion). *Journal of the Royal Statistical Society, Series B*, 73:423–498, 2011. [p15, 17]

National Population Commission - NPC and ICF. Nigeria Demographic and Health Survey 2018 - final report. http://dhsprogram.com/pubs/pdf/FR359/FR359.pdf, 2019. [p20]

C. Perez-Heydrich, J. Warren, C. Burgert, and M. Emch. Guidelines on the use of DHS GPS data. *ICF International, Calverton, Maryland*, 2013. https://dhsprogram.com/pubs/pdf/SAR8/SAR8.pdf/, last accessed on 2023-03-20. [p15]

C. Perez-Heydrich, J. L. Warren, C. R. Burgert, and M. E. Emch. Influence of Demographic and Health Survey point displacements on raster-based analyses. *Spatial Demography*, 4(2):135–153, 2016. [p15]

R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2022. URL https://www.R-project.org/. [p15]

H. Rue, S. Martino, and N. Chopin. Approximate Bayesian inference for latent Gaussian models by using integrated nested Laplace approximations. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 71(2):319–392, 2009. [p15]

J. L. Warren, C. Perez-Heydrich, C. R. Burgert, and M. E. Emch. Influence of demographic and health survey point displacements on distance-based analyses. *Spatial Demography*, 4(2):155–173, 2016. [p15]

K. Wilson and J. Wakefield. Estimation of health and demographic indicators with incomplete geographic information. *Spatial and Spatio-temporal Epidemiology*, 37:100421, 2021. [p15]

World Pop. *Open Spatial Demographic Data and Research*, 2022. https://hub.worldpop.org/doi/10.5258/SOTON/WP00648. [p20]

*Umut Altay*
*Department of Mathematical Sciences, Norwegian University of Science and Technology*
*Trondheim, Norway*
umut.altay@ntnu.no

*John Paige*
*Department of Mathematical Sciences, Norwegian University of Science and Technology*
*Trondheim, Norway*
john.paige@ntnu.no

*Andrea Riebler*
*Department of Mathematical Sciences, Norwegian University of Science and Technology*
*Trondheim, Norway*
andrea.riebler@ntnu.no

*Geir-Arne Fuglstad*
*Department of Mathematical Sciences, Norwegian University of Science and Technology*
*Trondheim, Norway*
geir-arne.fuglstad@ntnu.no

# SIHR: Statistical Inference in High-Dimensional Linear and Logistic Regression Models

*by Prabrisha Rakshit, Zhenyu Wang, Tony Cai, and Zijian Guo*

**Abstract** We introduce the R package SIHR for statistical inference in high-dimensional generalized linear models with continuous and binary outcomes. The package provides functionalities for constructing confidence intervals and performing hypothesis tests for low-dimensional objectives in both one-sample and two-sample regression settings. We illustrate the usage of SIHR through simulated examples and present real data applications to demonstrate the package's performance and practicality.

## 1 Introduction

In many applications, it is common to encounter regression problems where the number of covariates $p$ exceeds the sample size $n$. Much progress has been made in point estimation and support recovery in high-dimensional generalized linear models (GLMs), as evidenced by works such as Bühlmann and van de Geer (2011); Negahban et al. (2009); Huang and Zhang (2012); Tibshirani (1996); Fan and Li (2011); Zhang (2010); Sun and Zhang (2012); Belloni et al. (2011); Meinshausen and Yu (2009). In addition to estimation, van de Geer et al. (2014); Javanmard and Montanari (2014); Zhang and Zhang (2014) have proposed methods to correct the bias of penalized regression estimators and construct confidence intervals (CIs) for individual regression coefficients of the high-dimensional linear model. This debiased approach has sparked a rapidly growing research area focused on CI construction and hypothesis testing for low-dimensional objectives in high-dimensional GLMs.

The current paper presents the R package **SIHR**, which constructs confidence intervals and conducts hypothesis testing for various transformations of high-dimensional regression parameters for both continuous and binary outcomes. We consider the high-dimensional GLMs: for $1 \leq i \leq n$,

$$\mathbb{E}(y_i \mid X_{i\cdot}) = f(X_{i\cdot}^{\intercal}\beta), \quad \text{with } f(z) = \begin{cases} z & \text{for linear model;} \\ \exp(z)/[1+\exp(z)] & \text{for logistic model;} \end{cases} \tag{1}$$

where $y_i \in \mathbb{R}$ and $X_{i\cdot} \in \mathbb{R}^p$ denote respectively the outcome and the measured covariates of the $i$-th observation and $\beta \in \mathbb{R}^p$ denotes the high-dimensional regression vector. Throughout the paper, define $\Sigma = \mathbb{E}X_{i\cdot}X_{i\cdot}^{\intercal}$ and assume $\beta$ to be a sparse vector with its sparsity level denoted as $\|\beta\|_0$. In addition to the one-sample setting, we examine the statistical inference methods for the following two-sample regression models,

$$\mathbb{E}(y_i^{(k)} \mid X_{i\cdot}^{(k)}) = f(X_{i\cdot}^{(k)\intercal}\beta^{(k)}) \quad \text{with } k = 1, 2 \text{ and } 1 \leq i \leq n_k, \tag{2}$$

where $y_i^{(k)} \in \mathbb{R}$ and $X_{i\cdot}^{(k)} \in \mathbb{R}^p$ denote respectively the outcome and the measured covariates in the $k$-th sample, $f(\cdot)$ is the pre-specified link function defined as in (1), and $\beta^{(k)} \in \mathbb{R}^p$ denotes the high-dimensional regression vector in $k$-th sample.

The R package **SIHR** consists of five main functions `LF()`, `QF()`, `CATE()`, `InnProd()`, and `Dist()` implementing the statistical inferences for five different quantities correspondingly.

1. `LF()`, abbreviated for linear functional, implements the inference approach for $x_{\text{new}}^{\intercal}\beta$ in Cai et al. (2021a,b), with $x_{\text{new}} \in \mathbb{R}^p$ denoting a loading vector. With $x_{\text{new}} = e_j$ as a special case, `LF()` infers about the regression coefficient $\beta_j$ (van de Geer et al., 2014; Javanmard and Montanari, 2014; Zhang and Zhang, 2014, e.g.). When $x_{\text{new}}$ denotes a future observation's covariates, `LF()` makes inference for the conditional mean of the outcome for the individual. See the usage of `LF()` in the section Linear functional.
2. `QF()`, abbreviated for quadratic functional, makes inference for $\beta_G^{\intercal}A\beta_G$, following the proposal in Guo et al. (2019, 2021b); Cai and Guo (2020). $\beta_G$ is the subvector of $\beta$ with indices restricted to the pre-specified index set $G \in \{1, \ldots, p\}$ and $A \in \mathbb{R}^{|G|\times|G|}$, with $|G|$ denoting cardinality of G, is either a pre-specified submatrix or the unknown $\Sigma_{G,G}$. $\beta_G^{\intercal}A\beta_G$ can be viewed as a total measure of effects of all the variables in the group G. See the section Quadratic functional for the usage.

3. `CATE()`, abbreviated for conditional average treatment effect, is to make inference for $f(x_{\text{new}}^{\mathsf{T}}\beta^{(2)}) - f(x_{\text{new}}^{\mathsf{T}}\beta^{(1)})$, see Cai et al. (2021a) for detailed discussion. This difference measures the discrepancy between conditional means, closely related to the conditional average treatment effect for the new observation with covariates $x_{\text{new}}$. We demonstrate its usage in the section Conditional average treatment effect.
4. `InnProd()`, abbreviated for inner product, implements the statistical inference for $\beta_{\text{G}}^{(1)\mathsf{T}} A \beta_{\text{G}}^{(2)}$ with $A \in R^{|G|\times|G|}$, which was proposed in Guo et al. (2019); Ma et al. (2022). The inner product measures the similarity between the high-dimensional vectors $\beta^{(1)}$ and $\beta^{(2)}$, which is useful in capturing the genetic relatedness in the GWAS applications (Guo et al., 2019; Ma et al., 2022). The usage is detailed in the section Inner product.
5. `Dist()`, short-handed for distance, makes inference for the weighted distance $\gamma_{\text{G}}^{\mathsf{T}} A \gamma_{\text{G}}$ with $\gamma = \beta^{(2)} - \beta^{(1)}$. The distance measure is useful in comparing different high-dimensional regression vectors and constructing a generalizable model in the multisource learning problem Guo et al. (2023). See the section Distance for its usage.

There are a few other R packages for high-dimensional inference. The packages **hdi** and **SSLasso** (available at `http://web.stanford.edu/~montanar/sslasso/code.html`) implement the coordinate debiased Lasso estimators proposed in van de Geer et al. (2014) and Javanmard and Montanari (2014), respectively. These functions provide debiased estimators of $\beta$ along with their standard error estimators. These existing packages enable confidence interval construction and hypothesis testing for linear transformations of $\beta$, but not the quadratic form or inner products implemented in `QF()`, `InnProd()`, and `Dist()`. Even for the linear transformation, their implementation requires debiasing $p$ regression parameters. In contrast, our R package **SIHR** is computationally more efficient as it directly performs a single debiasing for the pre-specified linear transformation.

The **DoubleML** package focuses on estimating low-dimensional parameters of interest, such as causal or treatment effect parameters, in the presence of high-dimensional nuisance parameters that can be estimated using machine learning methods, while our package aims to estimate arbitrary linear and weighted quadratic combinations of the coefficient vector in high-dimensional regression. The selective inference is implemented by the R package **selectiveInference**. They focus on parameters based on the selected model, while we focus on fixed parameters independent of the selected models.

In the remainder of this paper, we review the inference methods in Section Methodological background and introduce the main functions of the package in Section Usage of the package, accompanied by illustrative examples. Then, a comparative analysis is conducted in Section Comparative analysis. Finally, we demonstrate the application of our proposed methods to real data in Section Real data applications.

## 2 Methodological background

We briefly review the penalized maximum likelihood estimator of $\beta$ in the high-dimensional GLM (1), defined as:

$$\widehat{\beta} = \arg\min_{\beta\in\mathbb{R}^p} \ell(\beta) + \lambda \sum_{j=2}^{p} \frac{\|X_{\cdot j}\|_2}{\sqrt{n}} |\beta_j|, \tag{3}$$

with $X_{\cdot j}$ denoting the $j$-th column of $X$, and

$$\ell(\beta) = \begin{cases} \frac{1}{n}\sum_{i=1}\left(y_i - X_{i\cdot}^{\mathsf{T}}\beta\right)^2 & \text{for linear model} \\ -\frac{1}{n}\sum_{i=1}^{n} y_i \log\left[\frac{f(X_{i\cdot}^{\mathsf{T}}\beta)}{1-f(X_{i\cdot}^{\mathsf{T}}\beta)}\right] - \frac{1}{n}\sum_{i=1}^{n}\log\left(1-f(X_{i\cdot}^{\mathsf{T}}\beta)\right) & \text{for GLM with binary outcome.} \end{cases} \tag{4}$$

To facilitate the methodological discussion, we take the first column of $X$ set as the constant 1 and hence does not include a penalty on $\beta_1$ in the above equation (3). In the penalized regression (3), we do not penalize the intercept coefficient $\beta_1$ and the tuning parameter $\lambda \asymp \sqrt{\log p/n}$ is chosen by cross-validation. The penalized estimators have been shown to achieve the optimal convergence rates and satisfy desirable variable selection properties (Meinshausen and Bühlmann, 2006; Bickel et al., 2009; Zhao and Yu, 2006; Wainwright, 2009). However, these estimators are not ready for statistical inference due to the non-negligible estimation bias induced by the penalty term (van de Geer et al., 2014; Javanmard and Montanari, 2014; Zhang and Zhang, 2014).

In section Linear functional for GLM, we propose a unified inference method for $x_{\text{new}}^{\mathsf{T}}\beta$ under linear and logistic outcome models. We also discuss inferences for quadratic functionals $\beta_{\text{G}}^{\mathsf{T}} A \beta_{\text{G}}$ and $\beta_{\text{G}}^{\mathsf{T}}\Sigma_{\text{G,G}}\beta_{\text{G}}$ in section Quadratic functional for GLM. In the case of the two-sample high-dimensional regression model (2), we develop the inference method for conditional treatment effect $\Delta(x_{\text{new}}) =$

$f(x_{\text{new}}^\intercal \beta^{(2)}) - f(x_{\text{new}}^\intercal \beta^{(1)})$ in section Conditional average treatment effects; we consider inference for $\beta_G^{(1)\intercal} A \beta_G^{(2)}$ and $\beta_G^{(1)\intercal} \Sigma_{G,G} \beta_G^{(2)}$ in section Inner product of regression vectors and $\gamma_G^\intercal A \gamma_G$ and $\gamma_G^\intercal \Sigma_{G,G} \gamma_G$ with $\gamma = \beta^{(2)} - \beta^{(1)}$ in section Distance of regression vectors.

### 2.1 Linear functional for linear model

To illustrate the main idea, we start with the linear functional for the linear model, which will be extended to a unified version in the section Linear functional for GLM. For the linear model in (1), we define $\epsilon_i = y_i - X_{i\cdot}^\intercal \beta$ and rewrite the model as $y_i = X_{i\cdot}^\intercal \beta + \epsilon_i$ for $1 \le i \le n$.

Given the vector $x_{\text{new}} \in \mathbb{R}^p$, a natural idea for the point estimator is to use the plug-in estimator $x_{\text{new}}^\intercal \widehat{\beta}$ with the initial estimator $\widehat{\beta}$ defined in (3). However, the bias $x_{\text{new}}^\intercal (\widehat{\beta} - \beta)$ is not negligible. The work Cai et al. (2021a) proposed the bias-corrected estimator as,

$$\widehat{x_{\text{new}}^\intercal \beta} = x_{\text{new}}^\intercal \widehat{\beta} + \widehat{u}^\intercal \frac{1}{n} \sum_{i=1}^n X_{i\cdot} \left( y_i - X_{i\cdot}^\intercal \widehat{\beta} \right), \tag{5}$$

where the second term on the right hand side in (5) is the estimate of negative bias $-x_{\text{new}}^\intercal (\widehat{\beta} - \beta)$, and the projection direction $\widehat{u}$ is defined as

$$\widehat{u} = \arg\min_{u \in \mathbb{R}^p} u^\intercal \widehat{\Sigma} u \quad \text{subject to:} \quad \|\widehat{\Sigma} u - x_{\text{new}}\|_\infty \le \|x_{\text{new}}\|_2 \mu_0 \tag{6}$$

$$\left| x_{\text{new}}^\intercal \widehat{\Sigma} u - \|x_{\text{new}}\|_2^2 \right| \le \|x_{\text{new}}\|_2^2 \mu_0, \tag{7}$$

where $\widehat{\Sigma} = \frac{1}{n} \sum_{i=1}^n X_{i\cdot} X_{i\cdot}^\intercal$ and $\mu_0 \asymp \sqrt{\log p / n}$. The bias-corrected estimator $\widehat{x_{\text{new}}^\intercal \beta}$ satisfies the following error decomposition,

$$\widehat{x_{\text{new}}^\intercal \beta} - x_{\text{new}}^\intercal \beta = \underbrace{\widehat{u}^\intercal \frac{1}{n} \sum_{i=1}^n X_{i\cdot}^\intercal \epsilon_i}_{\text{asymp. normal}} + \underbrace{\left( \widehat{\Sigma} \widehat{u} - x_{\text{new}} \right)^\intercal (\beta - \widehat{\beta})}_{\text{remaining bias}}. \tag{8}$$

The constrained optimization problem in (6) and (7) is designed to minimize the error on the right-hand side of the above equation: the first constraint in (6) controls the "remaining bias" term in the above equation while the objective function in (6) is used to minimize the variance of the "asymp. normal" term. Importantly, the second constraint in (7) ensures the standard error of the "asymp. normal" term always dominates the "remaining bias" term. Based on the asymptotic normality, we construct the CI for $x_{\text{new}}^\intercal \beta$ as

$$\text{CI} = \left( \widehat{x_{\text{new}}^\intercal \beta} - z_{\alpha/2} \sqrt{\widehat{\text{V}}}, \quad \widehat{x_{\text{new}}^\intercal \beta} + z_{\alpha/2} \sqrt{\widehat{\text{V}}} \right) \quad \text{with } \widehat{\text{V}} = \frac{\widehat{\sigma}^2}{n} \widehat{u}^\intercal \widehat{\Sigma} \widehat{u},$$

where $\widehat{\sigma}^2 = \frac{1}{n} \sum_{i=1}^n (y_i - X_{i\cdot}^\intercal \widehat{\beta})^2$, and $z_{\alpha/2}$ denotes the upper $\alpha/2$ quantile for the standard normal distribution.

**Remark 1** *It has been shown in Cai et al. (2021a) that the remaining bias term in (8) becomes negligible in comparison to the variance of the asymptotic normal term when the sample size is relatively large. However, for applications with a given sample size, we may also enlarge the standard error by a certain factor (e.g., 1.1) to accommodate the bias component in (8).*

### 2.2 Linear functional for GLM

In this subsection, we generalize the inference method specifically for the linear model in Linear functional for linear model to GLM in (1). Given the initial estimator $\widehat{\beta}$ defined in (3), the key step is to estimate the bias $x_{\text{new}}^\intercal (\widehat{\beta} - \beta)$. We can propose a generalized version of the bias-corrected estimator for $x_{\text{new}}^\intercal \beta$ as

$$\widehat{x_{\text{new}}^\intercal \beta} = x_{\text{new}}^\intercal \widehat{\beta} + \widehat{u}^\intercal \frac{1}{n} \sum_{i=1}^n \omega(X_{i\cdot}^\intercal \widehat{\beta}) \left( y_i - f(X_{i\cdot}^\intercal \widehat{\beta}) \right) X_{i\cdot}, \tag{9}$$

where the projection direction $\widehat{u}$ is defined in the following (10) and $\omega : \mathbb{R} \to \mathbb{R}$ denotes a weight function specified in the following Table 1 associated with different link functions.

In Table 1, we consider different GLM models and present the link function $f(\cdot)$, its derivative $f'(\cdot)$, and the corresponding weight function $\omega(\cdot)$. Note that there are two ways of specifying the weights $w(z)$ for logistic regression, where the linearization weighting was proposed in Guo et al.

| Model | Outcome Type | $f(z)$ | $f'(z)$ | $\omega(z)$ | Weighting |
|---|---|---|---|---|---|
| linear | Continuous | z | 1 | 1 | |
| logistic | Binary | $\frac{e^z}{1+e^z}$ | $\frac{e^z}{(1+e^z)^2}$ | $\frac{(1+e^z)^2}{e^z}$ | Linearization |
| logistic_alter | Binary | $\frac{e^z}{1+e^z}$ | $\frac{e^z}{(1+e^z)^2}$ | 1 | Link-specific |

**Table 1:** Definitions of the functions $\omega$ and $f$ for different GLMs.

(2021b) for logistic regression while the link-specific weighting function was proposed in Cai et al. (2021b) for general link function $f(\cdot)$. The projection direction $\widehat{u} \in \mathbb{R}^p$ in (9) is constructed as follows:

$$\begin{aligned} \widehat{u} = \arg\min_{u\in\mathbb{R}^p} u^\intercal \left[\frac{1}{n}\sum_{i=1}^n \omega(X_{i\cdot}^\intercal\widehat{\beta}) f'(X_{i\cdot}^\intercal\widehat{\beta}) X_{i\cdot} X_{i\cdot}^\intercal\right] u \quad \text{subject to:} \\ \left\| \frac{1}{n}\sum_{i=1}^n \omega(X_{i\cdot}^\intercal\widehat{\beta}) f'(X_{i\cdot}^\intercal\widehat{\beta}) X_{i\cdot} X_{i\cdot}^\intercal u - x_{\text{new}} \right\|_\infty \le \|x_{\text{new}}\|_2 \mu_0 \\ \left| x_{\text{new}}^\intercal \frac{1}{n}\sum_{i=1}^n \omega(X_{i\cdot}^\intercal\widehat{\beta}) f'(X_{i\cdot}^\intercal\widehat{\beta}) X_{i\cdot} X_{i\cdot}^\intercal u - \|x_{\text{new}}\|_2^2 \right| \le \|x_{\text{new}}\|_2^2 \mu_0. \end{aligned} \tag{10}$$

It has been established that $\widehat{x_{\text{new}}^\intercal\beta}$ in (9) is asymptotically unbiased and normal for the linear model (Cai et al., 2021a), the logistic model (Guo et al., 2021a; Cai et al., 2021b). The variance of $\widehat{x_{\text{new}}^\intercal\beta}$ can be estimated by $\widehat{\mathrm{V}}$, defined as

$$\widehat{\mathrm{V}} = \widehat{u}^\intercal \left[\frac{1}{n^2}\sum_{i=1}^n \left(\omega(X_{i\cdot}^\intercal\widehat{\beta})\right)^2 \widehat{\sigma}_i^2 X_{i\cdot} X_{i\cdot}^\intercal\right] \widehat{u} \quad \text{with :} \tag{11}$$

$$\widehat{\sigma}_i^2 = \begin{cases} \frac{1}{n}\sum_{j=1}^n \left(y_j - X_{j\cdot}^\intercal\widehat{\beta}\right)^2, & \text{for linear model} \\ f(X_{i\cdot}^\intercal\widehat{\beta})(1 - f(X_{i\cdot}^\intercal\widehat{\beta})), & \text{for logistic regression with } f(z) = \exp(z)/[1+\exp(z)] \end{cases}. \tag{12}$$

Based on the asymptotic normality, the CI for $x_{\text{new}}^\intercal\beta$ is:

$$\mathrm{CI} = \left(\widehat{x_{\text{new}}^\intercal\beta} - z_{\alpha/2}\sqrt{\widehat{\mathrm{V}}}, \quad \widehat{x_{\text{new}}^\intercal\beta} + z_{\alpha/2}\sqrt{\widehat{\mathrm{V}}}\right).$$

Subsequently, for the binary outcome case, we estimate the case probability $\mathbb{P}(y_i = 1 \mid X_{i\cdot} = x_{\text{new}})$ by $f(\widehat{x_{\text{new}}^\intercal\beta})$ and construct the CI for $f(x_{\text{new}}^\intercal\beta)$, with $f(z) = \exp(z)/[1+\exp(z)]$, as:

$$\mathrm{CI} = \left(f\left(\widehat{x_{\text{new}}^\intercal\beta} - z_{\alpha/2}\sqrt{\widehat{\mathrm{V}}}\right), f\left(\widehat{x_{\text{new}}^\intercal\beta} + z_{\alpha/2}\sqrt{\widehat{\mathrm{V}}}\right)\right).$$

### 2.3 Quadratic functional for GLM

We now move our focus to inference for the quadratic functional $\mathrm{Q}_A = \beta_{\mathrm{G}}^\intercal A \beta_{\mathrm{G}}$, where $G \subset \{1, \ldots, p\}$ and $A \in \mathbb{R}^{|G|\times|G|}$ denotes a pre-specified matrix of interest. Without loss of generality, we set $G = \{1, 2, \cdots, |G|\}$. With the initial estimator $\widehat{\beta}$ defined in (3), the plug-in estimator $\widehat{\beta}_{\mathrm{G}}^\intercal A \widehat{\beta}_{\mathrm{G}}$ has the following estimation error,

$$\widehat{\beta}_{\mathrm{G}}^\intercal A \widehat{\beta}_{\mathrm{G}} - \beta_{\mathrm{G}}^\intercal A \beta_{\mathrm{G}} = 2\widehat{\beta}_{\mathrm{G}}^\intercal A(\widehat{\beta}_{\mathrm{G}} - \beta_{\mathrm{G}}) - (\widehat{\beta}_{\mathrm{G}} - \beta_{\mathrm{G}})^\intercal A(\widehat{\beta}_{\mathrm{G}} - \beta_{\mathrm{G}}).$$

The last term in the above decomposition $(\widehat{\beta}_{\mathrm{G}} - \beta_{\mathrm{G}})^\intercal A(\widehat{\beta}_{\mathrm{G}} - \beta_{\mathrm{G}})$ is the higher-order approximation error under regular conditions; thus the bias of $\widehat{\beta}_{\mathrm{G}}^\intercal A \widehat{\beta}_{\mathrm{G}}$ mainly comes from the term $2\widehat{\beta}_{\mathrm{G}}^\intercal A(\widehat{\beta}_{\mathrm{G}} - \beta_{\mathrm{G}})$, which can be expressed as $2\, x_{\text{new}}^\intercal(\widehat{\beta} - \beta)$ with $x_{\text{new}} = (\widehat{\beta}_{\mathrm{G}}^\intercal A,\ \mathbf{0})^\intercal$. Hence the term can be estimated directly by applying the linear functional approach in section Linear functional for GLM. Utilizing this idea, Guo et al. (2021b, 2019) proposed the following estimator of $\mathrm{Q}_A$,

$$\widehat{\mathrm{Q}}_A = \widehat{\beta}_{\mathrm{G}}^\intercal A \widehat{\beta}_{\mathrm{G}} + 2\,\widehat{u}_A^\intercal \left[\frac{1}{n}\sum_{i=1}^n \omega(X_{i\cdot}^\intercal\widehat{\beta}) \left(y_i - f(X_{i\cdot}^\intercal\widehat{\beta})\right) X_{i\cdot}\right], \tag{13}$$

where $\widehat{u}_A$ is the projection direction defined in (10) with $x_{\text{new}} = (\widehat{\beta}_{\mathrm{G}}^\intercal A,\ \mathbf{0}^\intercal)^\intercal$. Since $\mathrm{Q}_A$ is non-negative if $A$ is positive semi-definite, we truncate $\widehat{\mathrm{Q}}_A$ at 0 and define $\widehat{\mathrm{Q}}_A = \max\left(\widehat{\mathrm{Q}}_A,\ 0\right)$. We further estimate

the variance of the $\widehat{\mathrm{Q}}_A$ by

$$\widehat{\mathrm{V}}_A(\tau) = 4\widehat{u}_A^{\intercal}\left[\frac{1}{n^2}\sum_{i=1}^{n}\omega^2(X_{i\cdot}^{\intercal}\widehat{\beta})\widehat{\sigma}_i^2 X_{i\cdot}X_{i\cdot}^{\intercal}\right]\widehat{u}_A + \frac{\tau}{n}, \tag{14}$$

where $\widehat{\sigma}_i^2$ is defined in (12) and the term $\tau/n$ with $\tau > 0$ (default value $\tau = 1$) is introduced as an upper bound for the term $(\widehat{\beta}_{\mathrm{G}} - \beta_{\mathrm{G}})^{\intercal}A(\widehat{\beta}_{\mathrm{G}} - \beta_{\mathrm{G}})$. Then given a fixed value of $\tau$, we construct the CI for $\mathrm{Q}_A$ as $\mathrm{CI}(\tau) = \left(\max\left(\widehat{\mathrm{Q}}_A - z_{\alpha/2}\sqrt{\widehat{\mathrm{V}}_A(\tau)},\ 0\right),\ \widehat{\mathrm{Q}}_A + z_{\alpha/2}\sqrt{\widehat{\mathrm{V}}_A(\tau)}\right)$.

Now we turn to the estimation of $\mathrm{Q}_\Sigma = \beta_{\mathrm{G}}^{\intercal}\Sigma_{\mathrm{G,G}}\beta_{\mathrm{G}}$ where the matrix $\Sigma_{\mathrm{G,G}}$ is unknown and estimated by $\widehat{\Sigma}_{\mathrm{G,G}} = \frac{1}{n}\sum_{i=1}^{n}X_{i\mathrm{G}}X_{i\mathrm{G}}^{\intercal}$. Decompose the error of the plug-in estimator $\widehat{\beta}_{\mathrm{G}}^{\intercal}\widehat{\Sigma}_{\mathrm{G,G}}\widehat{\beta}$:

$$\widehat{\beta}_{\mathrm{G}}^{\intercal}\widehat{\Sigma}_{\mathrm{G,G}}\widehat{\beta} - \beta_{\mathrm{G}}\Sigma_{\mathrm{G,G}}\beta_{\mathrm{G}} = 2\,\widehat{\beta}_{\mathrm{G}}^{\intercal}\widehat{\Sigma}_{\mathrm{G,G}}(\widehat{\beta}_{\mathrm{G}} - \beta_{\mathrm{G}}) + \beta_{\mathrm{G}}^{\intercal}(\widehat{\Sigma}_{\mathrm{G,G}} - \Sigma_{\mathrm{G,G}})\beta_{\mathrm{G}} - (\widehat{\beta}_{\mathrm{G}} - \beta_{\mathrm{G}})^{\intercal}\widehat{\Sigma}_{\mathrm{G,G}}(\widehat{\beta}_{\mathrm{G}} - \beta_{\mathrm{G}}).$$

The first term $\widehat{\beta}_{\mathrm{G}}^{\intercal}\widehat{\Sigma}_{\mathrm{G,G}}(\widehat{\beta}_{\mathrm{G}} - \beta_{\mathrm{G}})$ is estimated by applying linear functional approach in Linear functional for GLM with $x_{\mathrm{new}} = (\widehat{\beta}_{\mathrm{G}}^{\intercal}\widehat{\Sigma}_{\mathrm{G,G}},\ \mathbf{0})^{\intercal}$; the second term $\beta_{\mathrm{G}}^{\intercal}(\widehat{\Sigma}_{\mathrm{G,G}} - \Sigma_{\mathrm{G,G}})\beta_{\mathrm{G}}$ can be controlled asymptotically by central limit theorem; and the last term $(\widehat{\beta}_{\mathrm{G}} - \beta_{\mathrm{G}})^{\intercal}\widehat{\Sigma}_{\mathrm{G,G}}(\widehat{\beta}_{\mathrm{G}} - \beta_{\mathrm{G}})$ is negligible due to high-order bias. Guo et al. (2021b) proposed the following estimator of $\mathrm{Q}_\Sigma$

$$\widehat{\mathrm{Q}}_\Sigma = \widehat{\beta}_{\mathrm{G}}^{\intercal}\widehat{\Sigma}_{\mathrm{G,G}}\widehat{\beta}_{\mathrm{G}} + 2\,\widehat{u}_\Sigma^{\intercal}\left[\frac{1}{n}\sum_{i=1}^{n}\omega(X_{i\cdot}^{\intercal}\widehat{\beta})\left(y_i - f(X_{i\cdot}^{\intercal}\widehat{\beta})\right)X_{i\cdot}\right],$$

where $\widehat{u}_\Sigma$ is the projection direction constructed in (10) with $x_{\mathrm{new}} = (\widehat{\beta}_{\mathrm{G}}^{\intercal}\widehat{\Sigma}_{\mathrm{G,G}},\ \mathbf{0})^{\intercal}$. We introduce the estimator $\widehat{\mathrm{Q}}_\Sigma = \max(\widehat{\mathrm{Q}}_\Sigma,\ 0)$ and estimate its variance as

$$\widehat{\mathrm{V}}_\Sigma(\tau) = 4\widehat{u}_\Sigma^{\intercal}\left[\frac{1}{n^2}\sum_{i=1}^{n}\omega^2(X_{i\cdot}^{\intercal}\widehat{\beta})\widehat{\sigma}_i^2 X_{i\cdot}X_{i\cdot}^{\intercal}\right]\widehat{u}_\Sigma + \frac{1}{n^2}\sum_{i=1}^{n}\left(\widehat{\beta}_{\mathrm{G}}^{\intercal}X_{i,\mathrm{G}}X_{i,\mathrm{G}}^{\intercal}\widehat{\beta}_{\mathrm{G}} - \widehat{\beta}_{\mathrm{G}}^{\intercal}\widehat{\Sigma}_{\mathrm{G,G}}\widehat{\beta}_{\mathrm{G}}\right)^2 + \frac{\tau}{n}, \tag{15}$$

where $\widehat{\sigma}_i^2$ is defined in (12) and the term $\tau/n$ with $\tau > 0$ is introduced as an upper bound for the term $(\widehat{\beta}_{\mathrm{G}} - \beta_{\mathrm{G}})^{\intercal}\widehat{\Sigma}_{\mathrm{G,G}}(\widehat{\beta}_{\mathrm{G}} - \beta_{\mathrm{G}})$. Then, for a fixed value of $\tau$, we can construct the CI for $\mathrm{Q}_\Sigma$ as

$$\mathrm{CI}(\tau) = \left(\max\left(\widehat{\mathrm{Q}}_\Sigma - z_{\alpha/2}\sqrt{\widehat{\mathrm{V}}_\Sigma(\tau)},\ 0\right),\ \widehat{\mathrm{Q}}_\Sigma + z_{\alpha/2}\sqrt{\widehat{\mathrm{V}}_\Sigma(\tau)}\right). \tag{16}$$

### 2.4 Conditional average treatment effects

The inference methods developed for one sample can be generalized to make inferences for conditional average treatment effects (CATE). From a causality viewpoint, we consider the data set $\{(X_{i\cdot}, y_i, D_i)\}$ for $i = 1, \ldots, n$, where $D_i \in \{1, 2\}$ indicates the treatment assigned to the $i$-th observation. For a new observation with covariates $X_{i\cdot} = x_{\mathrm{new}}$, we define CATE as $\Delta(x_{\mathrm{new}}) = \mathbb{E}(y_i|X_{i\cdot}, D_i = 2) - \mathbb{E}(y_i|X_{i\cdot}, D_i = 1)$.

We group observations $\{i : D_i = k\}$ into the $k$-th data sample $\{(X_{i\cdot}^{(k)}, y_i^{(k)}\}$ for $k = 1, 2$, where $1 \le i \le n_k$ and $n_1 + n_2 = n$. Subsequently, we rewrite $\mathbb{E}(y_i|X_{i\cdot}, D_i = k)$ as $\mathbb{E}[y_i^{(k)}|X_i^{(k)} = x_{\mathrm{new}}]$ for $k = 1, 2$. Using the GLM model outlined in (2), the CATE can be formulated as

$$\Delta(x_{\mathrm{new}}) = \mathbb{E}[y_i^{(2)}|X_i^{(2)} = x_{\mathrm{new}}] - \mathbb{E}[y_i^{(1)}|X_i^{(1)} = x_{\mathrm{new}}] = f(x_{\mathrm{new}}^{\intercal}\beta^{(2)}) - f(x_{\mathrm{new}}^{\intercal}\beta^{(1)}).$$

Following (9), we construct the bias-corrected point estimators of $\widehat{x_{\mathrm{new}}^{\intercal}\beta^{(1)}}$ and $\widehat{x_{\mathrm{new}}^{\intercal}\beta^{(2)}}$, together with their corresponding variances $\widehat{\mathrm{V}}^{(1)}$ and $\widehat{\mathrm{V}}^{(2)}$ as (11). For the first sample $(X_i^{(1)}, y_i^{(1)})$, where $1 \le i \le n_1$, we use the methods described in equations (9) and (11) to compute the bias-corrected point estimator $\widehat{x_{\mathrm{new}}^{\intercal}\beta^{(1)}}$ and the variance estimator $\widehat{\mathrm{V}}^{(1)}$, respectively. Similarly, for the second sample $(X_i^{(2)}, y_i^{(2)})$, where $1 \le i \le n_2$, we apply the same procedures to derive the point estimator $\widehat{x_{\mathrm{new}}^{\intercal}\beta^{(2)}}$ and the variance estimator $\widehat{\mathrm{V}}^{(2)}$.

The paper Cai et al. (2021a) proposed to estimate $\Delta(x_{\mathrm{new}})$ by $\widehat{\Delta}(x_{\mathrm{new}})$ as follows,

$$\widehat{\Delta}(x_{\mathrm{new}}) = f(\widehat{x_{\mathrm{new}}^{\intercal}\beta^{(2)}}) - f(\widehat{x_{\mathrm{new}}^{\intercal}\beta^{(1)}}).$$

Its variance can be estimated with delta method by:

$$\widehat{\mathrm{V}}_\Delta = \left(f'(\widehat{x_{\text{new}}^\intercal \beta^{(1)}})\right)^2 \widehat{\mathrm{V}}^{(1)} + \left(f'(\widehat{x_{\text{new}}^\intercal \beta^{(2)}})\right)^2 \widehat{\mathrm{V}}^{(2)}.$$

Then we construct the CI for $\Delta(x_{\text{new}})$ as

$$\mathrm{CI} = \left(\widehat{\Delta}(x_{\text{new}}) - z_{\alpha/2}\sqrt{\widehat{\mathrm{V}}_\Delta}, \widehat{\Delta}(x_{\text{new}}) + z_{\alpha/2}\sqrt{\widehat{\mathrm{V}}_\Delta}\right).$$

### 2.5 Inner product of regression vectors

The paper Guo et al. (2019); Ma et al. (2022) have investigated the CI construction for $\beta_{\mathrm{G}}^{(1)\intercal} A \beta_{\mathrm{G}}^{(2)}$, provided with a pre-specified submatrix $A \in \mathbb{R}^{|\mathrm{G}|\times|\mathrm{G}|}$ and the set of indices $\mathrm{G} \subset \{1, \ldots, p\}$. With $\widehat{\beta}^{(1)}$ and $\widehat{\beta}^{(2)}$ denoting the initial estimators fitted on first and second data sample via (3), respectively, the plug-in estimator $\widehat{\beta}_{\mathrm{G}}^{(1)\intercal} A \widehat{\beta}_{\mathrm{G}}^{(2)}$ admits the following bias,

$$\widehat{\beta}_{\mathrm{G}}^{(1)\intercal} A \widehat{\beta}_{\mathrm{G}}^{(2)} - \beta_{\mathrm{G}}^{(1)\intercal} A \beta_{\mathrm{G}}^{(2)} = \widehat{\beta}_{\mathrm{G}}^{(2)\intercal} A \left(\widehat{\beta}_{\mathrm{G}}^{(1)} - \beta_{\mathrm{G}}^{(1)}\right) + \widehat{\beta}_{\mathrm{G}}^{(1)\intercal} A \left(\widehat{\beta}_{\mathrm{G}}^{(2)} - \beta_{\mathrm{G}}^{(2)}\right) - \left(\widehat{\beta}_{\mathrm{G}}^{(1)} - \beta_{\mathrm{G}}^{(1)}\right)^\intercal A \left(\widehat{\beta}_{\mathrm{G}}^{(2)} - \beta_{\mathrm{G}}^{(2)}\right).$$

The key step is to estimate the components $\widehat{\beta}_{\mathrm{G}}^{(2)\intercal} A \left(\widehat{\beta}_{\mathrm{G}}^{(1)} - \beta_{\mathrm{G}}^{(1)}\right)$ and $\widehat{\beta}_{\mathrm{G}}^{(1)\intercal} A \left(\widehat{\beta}_{\mathrm{G}}^{(2)} - \beta_{\mathrm{G}}^{(2)}\right)$, since the last term $(\widehat{\beta}_{\mathrm{G}}^{(1)} - \beta_{\mathrm{G}}^{(1)})^\intercal A (\widehat{\beta}_{\mathrm{G}}^{(2)} - \beta_{\mathrm{G}}^{(2)})$ is negligible due to high-order bias. We propose the following bias-corrected estimator for $\beta_{\mathrm{G}}^{(1)\intercal} A \beta_{\mathrm{G}}^{(2)}$

$$\begin{aligned}\widehat{\beta_{\mathrm{G}}^{(1)\intercal} A \beta_{\mathrm{G}}^{(2)}} = \widehat{\beta}_{\mathrm{G}}^{(1)\intercal} A \widehat{\beta}_{\mathrm{G}}^{(2)} &+ \widehat{u}_1^\intercal \frac{1}{n_1} \sum_{i=1}^{n_1} \omega(X_{i\cdot}^{(1)\intercal} \widehat{\beta}^{(1)}) \left(y_i^{(1)} - f(X_{i\cdot}^{(1)\intercal} \widehat{\beta}^{(1)})\right) X_{i\cdot}^{(1)} \\ &+ \widehat{u}_2^\intercal \frac{1}{n_2} \sum_{i=1}^{n_2} \omega(X_{i\cdot}^{(2)\intercal} \widehat{\beta}^{(2)}) \left(y_i^{(2)} - f(X_{i\cdot}^{(2)\intercal} \widehat{\beta}^{(2)})\right) X_{i\cdot}^{(2)}.\end{aligned} \tag{17}$$

Here $\widehat{u}_1$ represents the projection direction computed in (10), using the first sample data and $x_{\text{new}} = (\widehat{\beta}_{\mathrm{G}}^{(2)\intercal} A,\ \mathbf{0})^\intercal$. Similarly, $\widehat{u}_2$ is the projection direction derived from the second sample data, using $x_{\text{new}} = (\widehat{\beta}_{\mathrm{G}}^{(2)\intercal} A,\ \mathbf{0})^\intercal$. The corresponding variance of $\widehat{\beta_{\mathrm{G}}^{(1)\intercal} A \beta_{\mathrm{G}}^{(2)}}$, when $A$ is a known positive definite matrix, is estimated as

$$\widehat{\mathrm{V}}_A(\tau) = \widehat{\mathrm{V}}^{(1)} + \widehat{\mathrm{V}}^{(2)} + \frac{\tau}{\min(n_1, n_2)},$$

where $\widehat{V}^{(k)}$ is computed as in (11) for the $k$-th regression model ($k = 1, 2$) and the term $\tau / \min(n_1, n_2)$ with $\tau > 0$ is introduced as an upper bound for the term $(\widehat{\beta}_{\mathrm{G}}^{(1)} - \beta_{\mathrm{G}}^{(1)})^\intercal A (\widehat{\beta}_{\mathrm{G}}^{(2)} - \beta_{\mathrm{G}}^{(2)})$.

We also consider the case of unknown $A = \Sigma_{\mathrm{G,G}}$. As a natural generalization, the quantity $\beta_{\mathrm{G}}^{(1)\intercal} \Sigma_{\mathrm{G,G}} \beta_{\mathrm{G}}^{(2)}$ is well defined if the two regression models in (2) share the design covariance matrix $\Sigma = \mathbb{E} X_{i\cdot}^{(1)} X_{i\cdot}^{(1)\intercal} = \mathbb{E} X_{i\cdot}^{(2)} X_{i\cdot}^{(2)\intercal}$. We follow the above procedures by replacing $A$ with $\widehat{\Sigma}_{\mathrm{G,G}} = \frac{1}{n_1+n_2} \sum_{i=1}^{n_1+n_2} X_{i,\mathrm{G}} X_{i,\mathrm{G}}^\intercal$ where $X$ is the row-combined matrix of $X^{(1)}$ and $X^{(2)}$. The variance of $\widehat{\beta_{\mathrm{G}}^{(1)\intercal} \Sigma_{\mathrm{G,G}} \beta_{\mathrm{G}}^{(2)}}$ is now estimated as

$$\widehat{\mathrm{V}}_\Sigma(\tau) = \widehat{\mathrm{V}}^{(1)} + \widehat{\mathrm{V}}^{(2)} + \frac{1}{(n_1+n_2)^2} \sum_{i=1}^{n_1+n_2} \left(\widehat{\beta}_{\mathrm{G}}^{(1)\intercal} X_{i,\mathrm{G}} X_{i,\mathrm{G}}^\intercal \widehat{\beta}_{\mathrm{G}}^{(2)} - \widehat{\beta}_{\mathrm{G}}^{(1)\intercal} \widehat{\Sigma}_{\mathrm{G,G}} \widehat{\beta}_{\mathrm{G}}^{(2)}\right)^2 + \frac{\tau}{\min(n_1, n_2)}.$$

We then construct the CI for $\beta_{\mathrm{G}}^{(1)\intercal} A \beta_{\mathrm{G}}^{(2)}$ as

$$\mathrm{CI}(\tau) = \begin{cases} \left(\widehat{\beta_{\mathrm{G}}^{(1)\intercal} A \beta_{\mathrm{G}}^{(2)}} - z_{\alpha/2} \widehat{\mathrm{V}}_A(\tau),\ \widehat{\beta_{\mathrm{G}}^{(1)\intercal} A \beta_{\mathrm{G}}^{(2)}} + z_{\alpha/2} \widehat{\mathrm{V}}_A(\tau)\right) & \text{if } A \text{ is specified} \\ \left(\widehat{\beta_{\mathrm{G}}^{(1)\intercal} \Sigma_{\mathrm{G,G}} \beta_{\mathrm{G}}^{(2)}} - z_{\alpha/2} \widehat{\mathrm{V}}_\Sigma(\tau),\ \widehat{\beta_{\mathrm{G}}^{(1)\intercal} \Sigma_{\mathrm{G,G}} \beta_{\mathrm{G}}^{(2)}} + z_{\alpha/2} \widehat{\mathrm{V}}_\Sigma(\tau)\right) & A = \Sigma_{\mathrm{G,G}} \text{ is unknown.} \end{cases}$$

### 2.6 Distance of regression vectors

We denote $\gamma = \beta^{(2)} - \beta^{(1)}$ and its initial estimator $\widehat{\gamma} = \widehat{\beta}^{(2)} - \widehat{\beta}^{(1)}$. The quantity of interest is the distance between two regression vectors $\gamma_{\mathrm{G}}^{\intercal} A \gamma_{\mathrm{G}}$, given a pre-specified submatrix $A \in \mathbb{R}^{|\mathrm{G}|\times|\mathrm{G}|}$ and the set of indices $\mathrm{G} \in \{1, \ldots, p\}$. The bias of the plug-in estimator $\widehat{\gamma}_{\mathrm{G}}^{\intercal} A \widehat{\gamma}_{\mathrm{G}}$ is:

$$\widehat{\gamma}_{\mathrm{G}}^{\intercal} A \widehat{\gamma}_{\mathrm{G}} - \gamma_{\mathrm{G}}^{\intercal} A \gamma_{\mathrm{G}} = 2\, \widehat{\gamma}_{\mathrm{G}}^{\intercal} A \left(\widehat{\beta}_{\mathrm{G}}^{(2)} - \beta_{\mathrm{G}}^{(2)}\right) - 2\, \widehat{\gamma}_{\mathrm{G}}^{\intercal} A \left(\widehat{\beta}_{\mathrm{G}}^{(1)} - \beta_{\mathrm{G}}^{(1)}\right) - (\widehat{\gamma}_{\mathrm{G}} - \gamma_{\mathrm{G}})^{\intercal} A\, (\widehat{\gamma}_{\mathrm{G}} - \gamma_{\mathrm{G}})\,.$$

The key step is to estimate the error components $\widehat{\gamma}_{\mathrm{G}}^{\intercal} A \left(\widehat{\beta}_{\mathrm{G}}^{(1)} - \beta_{\mathrm{G}}^{(1)}\right)$ and $\widehat{\gamma}_{\mathrm{G}}^{\intercal} A \left(\widehat{\beta}_{\mathrm{G}}^{(2)} - \beta_{\mathrm{G}}^{(2)}\right)$ in the above decomposition. We apply linear functional techniques twice here, and propose the bias-corrected estimator:

$$\begin{aligned}\widehat{\gamma_{\mathrm{G}}^{\intercal} A \gamma_{\mathrm{G}}} = \widehat{\gamma}_{\mathrm{G}}^{\intercal} A \widehat{\gamma}_{\mathrm{G}} &- 2\, \widehat{u}_1^{\intercal} \frac{1}{n_1} \sum_{i=1}^{n_1} \omega(X_{i\cdot}^{(1)\intercal} \widehat{\beta}^{(1)}) \left(y_i^{(1)} - f(X_{i\cdot}^{(1)\intercal} \widehat{\beta}^{(1)})\right) X_{i\cdot}^{(1)} \\ &+ 2\, \widehat{u}_2^{\intercal} \frac{1}{n_2} \sum_{i=1}^{n_2} \omega(X_{i\cdot}^{(2)\intercal} \widehat{\beta}^{(2)}) \left(y_i^{(2)} - f(X_{i\cdot}^{(2)\intercal} \widehat{\beta}^{(2)})\right) X_{i\cdot}^{(2)},\end{aligned} \tag{18}$$

where $\widehat{u}_1$ and $\widehat{u}_2$ are the projection directions defined in (10) with $x_{\mathrm{new}} = \left(\widehat{\gamma}_{\mathrm{G}}^{\intercal} A, \mathbf{0}\right)^{\intercal}$ but on two different sample data respectively. The second term on right-hand-side of (18) is to estimate $-2\, x_{\mathrm{new}}^{\intercal}(\widehat{\beta}_{\mathrm{G}}^{(1)} - \beta_{\mathrm{G}}^{(1)})$ and the third term on right-hand-side of (18) is to estimate $-2\, x_{\mathrm{new}}^{\intercal}(\widehat{\beta}_{\mathrm{G}}^{(2)} - \beta_{\mathrm{G}}^{(2)})$.

To maintain non-negativity of distance, we define $\widehat{\gamma_{\mathrm{G}}^{\intercal} A \gamma_{\mathrm{G}}} = \max\left\{\widehat{\gamma_{\mathrm{G}}^{\intercal} A \gamma_{\mathrm{G}}},\ 0\right\}$ and estimate its corresponding asymptotic variance as

$$\widehat{\mathrm{V}}_A(\tau) = 4\, \widehat{\mathrm{V}}^{(1)} + 4\, \widehat{\mathrm{V}}^{(2)} + \frac{\tau}{\min(n_1, n_2)},$$

where $\widehat{\mathrm{V}}^{(k)}$ is computed as in (11) for the $k$-th regression model ($k = 1, 2$) and the term $\tau / \min(n_1, n_2)$ with $\tau > 0$ is introduced as an upper bound for the term $(\widehat{\gamma}_{\mathrm{G}} - \gamma_{\mathrm{G}})^{\intercal} A (\widehat{\gamma}_{\mathrm{G}} - \gamma_{\mathrm{G}})$. With asymptotic normality, we construct the CI for $\gamma_{\mathrm{G}}^{\intercal} A \gamma_{\mathrm{G}}$ as

$$\mathrm{CI}(\tau) = \left(\max\left(\widehat{\gamma_{\mathrm{G}}^{\intercal} A \gamma_{\mathrm{G}}} - z_{\alpha/2} \sqrt{\widehat{\mathrm{V}}_A(\tau)},\ 0\right),\ \widehat{\gamma_{\mathrm{G}}^{\intercal} A \gamma_{\mathrm{G}}} + z_{\alpha/2} \sqrt{\widehat{\mathrm{V}}_A(\tau)}\right).$$

We further consider the unknown matrix $A = \Sigma_{\mathrm{G,G}}$ and construct the point estimator $\widehat{\gamma_{\mathrm{G}}^{\intercal} \Sigma_{\mathrm{G,G}} \gamma_{\mathrm{G}}}$ in a similar way as outlined in (18). In this case, the submatrix $A$ is substituted with $\widehat{\Sigma}_{\mathrm{G,G}}$, where $\widehat{\Sigma}_{\mathrm{G,G}} = \frac{1}{n_1+n_2} \sum_{i=1}^{n_1+n_2} X_{i,\mathrm{G}} X_{i,\mathrm{G}}^{\intercal}$ with $X$ as the row-combined matrix of $X^{(1)}$ and $X^{(2)}$. Its corresponding asymptotic variance is

$$\widehat{\mathrm{V}}_\Sigma(\tau) = 4\, \widehat{\mathrm{V}}^{(1)} + 4\, \widehat{\mathrm{V}}^{(2)} + \frac{1}{(n_1+n_2)^2} \sum_{i=1}^{n_1+n_2} \left(\widehat{\gamma}_{\mathrm{G}}^{\intercal} X_{i,\mathrm{G}} X_{i,\mathrm{G}}^{\intercal} \widehat{\gamma}_{\mathrm{G}} - \widehat{\gamma}_{\mathrm{G}}^{\intercal} \widehat{\Sigma}_{\mathrm{G,G}} \widehat{\gamma}_{\mathrm{G}}\right)^2 + \frac{\tau}{\min(n_1, n_2)}.$$

Next we present the CI for $\gamma_{\mathrm{G}}^{\intercal} \Sigma_{\mathrm{G,G}} \gamma_{\mathrm{G}}$.

$$\mathrm{CI}(\tau) = \left(\max\left(\widehat{\gamma_{\mathrm{G}}^{\intercal} \Sigma_{\mathrm{G,G}} \gamma_{\mathrm{G}}} - z_{\alpha/2} \sqrt{\widehat{\mathrm{V}}_\Sigma(\tau)},\ 0\right),\ \widehat{\gamma_{\mathrm{G}}^{\intercal} \Sigma_{\mathrm{G,G}} \gamma_{\mathrm{G}}} + z_{\alpha/2} \sqrt{\widehat{\mathrm{V}}_\Sigma(\tau)}\right).$$

## 3 Usage of the package

The **SIHR** package contains a set of functions for conducting inference for various transformations of high-dimensional regression vectors, such as linear and quadratic functions. We summarize the functions and their corresponding objectives in the following Table 2.

### 3.1 Linear functional

The function `LF()`, shorthanded for Linear Functional, performs inference for $x_{\mathrm{new}}^{\intercal} \beta$, under the high-dimensional model (1), where $x_{\mathrm{new}}$ is a given vector. A typical `LF()` code snippet looks like:

| Function | Inference Objective | Description |
|---|---|---|
| LF() | $x_{\text{new}}^{\mathsf{T}}\beta$ | Generates a LF object which includes the bias-corrected estimator of $x_{\text{new}}^{\mathsf{T}}\beta$ in high-dimensional GLM and the corresponding standard error, which are further used to construct CI and conduct hypothesis testing related to $x_{\text{new}}^{\mathsf{T}}\beta$. |
| QF() | $\beta_{\mathrm{G}}^{\mathsf{T}} A \beta_{\mathrm{G}}$ | Generates a QF object which includes the bias-corrected estimator of $\beta_{\mathrm{G}}^{\mathsf{T}} A \beta_{\mathrm{G}}$ in high-dimensional GLM, for $A \in \mathbb{R}^{\lvert\mathrm{G}\rvert\times\lvert\mathrm{G}\rvert}$ and index set $\mathrm{G} \in \{1,\ldots,p\}$, and computes the corresponding standard error, which are further used to construct CI and conduct hypothesis testing related to $\beta_{\mathrm{G}}^{\mathsf{T}} A \beta_{\mathrm{G}}$. |
| CATE() | $f(x_{\text{new}}^{\mathsf{T}}\beta^{(2)}) - f(x_{\text{new}}^{\mathsf{T}}\beta^{(1)})$ | Generates a CATE object which includes the bias-corrected estimator of $f(x_{\text{new}}^{\mathsf{T}}\beta^{(2)}) - f(x_{\text{new}}^{\mathsf{T}}\beta^{(1)})$ in high-dimensional GLMs and the corresponding standard error, which are further used to construct CI and conduct hypothesis testing related to $f(x_{\text{new}}^{\mathsf{T}}\beta^{(2)}) - f(x_{\text{new}}^{\mathsf{T}}\beta^{(1)})$. |
| InnProd() | $\beta_{\mathrm{G}}^{(1)\mathsf{T}} A \beta_{\mathrm{G}}^{(2)}$ | Generates an InnProd object which includes the bias-corrected estimator of $\beta_{\mathrm{G}}^{(1)\mathsf{T}} A \beta_{\mathrm{G}}^{(2)}$ in high-dimensional GLMs, for $A \in \mathbb{R}^{\lvert\mathrm{G}\rvert\times\lvert\mathrm{G}\rvert}$ and index set $\mathrm{G} \in \{1,\ldots,p\}$, and computes the corresponding standard error, which are further used to construct CI and conduct hypothesis testing related to $\beta_{\mathrm{G}}^{(1)\mathsf{T}} A \beta_{\mathrm{G}}^{(2)}$. |
| Dist() | $\gamma_{\mathrm{G}}^{\mathsf{T}} A \gamma_{\mathrm{G}}$ with $\gamma = \beta^{(2)} - \beta^{(1)}$ | Generates a Dist object which includes the bias-corrected estimator of $\gamma_{\mathrm{G}}^{\mathsf{T}} A \gamma_{\mathrm{G}}$ in high-dimensional GLMs, for $A \in \mathbb{R}^{\lvert\mathrm{G}\rvert\times\lvert\mathrm{G}\rvert}$ and index set $\mathrm{G} \in \{1,\ldots,p\}$, and the corresponding standard error, which are further used to construct CI and conduct hypothesis testing related to $\gamma_{\mathrm{G}}^{\mathsf{T}} A \gamma_{\mathrm{G}}$. |
| ci() | — | Input object (`LF/ QF/ CATE/ InnProd/ Dist`), returns CI. |
| summary() | — | Input object (`LF/ QF/ CATE/ InnProd/ Dist`), computes and returns a list of summary statistics, including plug-in estimator, bias-corrected estimator together with associated standard error and p-value. |

**Table 2:** Functions of **SIHR**, which perform statistical inference for low-dimensional objectives in high-dimensional GLM for continuous and binary outcomes.

```
LF(X, y, loading.mat, model = c("linear", "logistic", "logistic_alter"),
   intercept = TRUE, intercept.loading = FALSE, beta.init = NULL, lambda = NULL,
   mu = NULL, prob.filter = 0.05, rescale = 1.1, verbose = FALSE)
```

In the following we provide descriptions of the various arguments of the `LF` function :

- `X` is the design matrix of dimension $n \times p$ and `y` is the response vector of length $n$.
- `loading.mat` is the matrix of loading vectors where each column corresponds to a new future observation $x_{\text{new}}$. It is designed to allow for taking multiple $x_{\text{new}}$'s as input, thereby saving the computational time of constructing the initial estimator multiple times.
- `model` (default = `"linear"`) specifies which high-dimensional regression model to be fitted, the choices being `c("linear", "logistic", "logistic_alter")`, where "linear" corresponds to the linear model and "logistic" and "logistic_alter" correspond to the logistic regression; see Table 1.
- `intercept` (default = `TRUE`) is a logical argument that specifies whether an intercept term should be fitted while computing the initial estimator in (3).
- `intercept.loading` (default = `FALSE`) is a logical argument that specifies whether the intercept term should be included for defining the objective $x_{\text{new}}^{\mathsf{T}}\beta$. Specifically, setting `intercept.loading = TRUE` prepend a column of 1's to the matrix `loading.mat`.
- `beta.init` (default = `NULL`) allows the user to supply the initial estimator $\widehat{\beta}$ of the regression vector. If `beta.init` is left as `NULL`, the initial estimator $\widehat{\beta}$ in (3) is computed using function `cv.glmnet` in **glmnet**.
- `lambda` (default = `NULL`) denotes the scaled tuning parameter $\lambda$ used for computing the initial estimator $\widehat{\beta}$ in (3) which can either be pre-specified or can be set to `NULL` whence `LF` uses the function `cv.glmnet` in **glmnet** to compute the tuning parameter.

- `mu` (default = `NULL`) denotes the tuning parameter $\mu_0$ in (10). When `mu` is set as `NULL`, it is computed as the smallest $\mu_0$ such that (10) has a finite solution.
- `prob.filter` (default = `0.05`) is specific to `model = "logistic"`. From Table 1, observe that `model = "logistic"` sets the weight for $i$-th individual as $\frac{1}{\mathbb{P}(y_i=1|X_{i\cdot})\cdot(1-\mathbb{P}(y_i=1|X_{i\cdot}))}$ which can blow up if the estimated probabilities $\mathbb{P}(y_i \mid X_{i\cdot})$ are very close to 0 or 1. We discard those samples for which the estimated probability lies outside $[$`prob.filter`$, 1 -$ `prob.filter`$]$ before proceeding with the algorithm.
- `rescale` (default = `1.1`) denotes the factor used to enlarge the standard error to account for the finite sample bias, as pointed out in Remark 1.
- `verbose` (default = `FALSE`) is a logical argument that specifies whether intermediate message(s) should be printed, the projection direction be returned.

**Remark 2** *The structure of the* `loading.mat` *is designed so that each column corresponds to a future observation $x_{\text{new}}$. This matrix structure optimizes computational efficiency by allowing the debiasing algorithm to process multiple linear functionals simultaneously; that is, when* `loading.mat` *contains multiple columns, the* `LF()` *function only requires computing the initial estimator $\widehat{\beta}$ in (3) once. Specifically, when* `loading.mat` *is set as the identity matrix of dimension $p$, where $p$ represents the number of covariates, the* `LF()` *function conducts inference for all $p$ individual regression coefficients concurrently.*

Next, we provide an example to illustrate the usage of `LF()` in the linear regression model.

**Example 1.** For $1 \le i \le n$ with $n = 100$, the covariates $X_{i\cdot}$ are independently generated from the multivariate normal distribution with mean $\mu = 0_p$ and covariance $\Sigma = \mathbf{I}_p$ with $p = 120$, where $\mathbf{I}_p$ is an identity matrix of dimension $p$. The regression vector $\beta \in \mathbb{R}^p$ is generated as $\beta_1 = 0.5, \beta_2 = 1$ and $\beta_j = 0$ if $3 \le j \le p$. The outcome is generated as $y_i = X_{i\cdot}^\intercal \beta + \epsilon_i$ with independently generated standard normal $\epsilon_i$.

```
n <- 100; p <- 120
mu <- rep(0,p); Cov <- diag(p)
beta <- rep(0,p); beta[c(1,2)] <- c(0.5, 1)
X <- MASS::mvrnorm(n, mu, Cov)
y <- X %*% beta + rnorm(n)
```

We now generate two observations $x_{\text{new}}^{(1)}, x_{\text{new}}^{(2)}$ and apply the `LF()` function to construct the point estimators of $x_{\text{new}}^{(1)\intercal}\beta$ and $x_{\text{new}}^{(2)\intercal}\beta$, together with their standard error estimates.

```
loading1 <- c(1, 1, rep(0, p-2))
loading2 <- c(-0.5, -1, rep(0, p-2))
loading.mat <- cbind(loading1, loading2)
Est <- LF(X, y, loading.mat, model = 'linear')
```

Having fitted the model, we have two following functions `ci()` and `summary()`. We first report the 95% CIs for $x_{\text{new}}^{(1)\intercal}\beta$ and $x_{\text{new}}^{(2)\intercal}\beta$, where the true values $x_{\text{new}}^{(1)\intercal}\beta = 1.5$ and $x_{\text{new}}^{(2)\intercal}\beta = -1.25$ are contained in the corresponding CIs.

```
ci(Est)
#>  loading     lower      upper
#>1       1  1.167873  1.8753934
#>2       2 -1.544138 -0.7995375
```

Then, we apply the `summary()` function to return a list of the summary statistics, including the plugin estimator, bias-corrected estimator, the standard error for the bias-corrected estimator and the p-value corresponding to the hypothesis testing $H_0 : x_{\text{new}}^\intercal\beta = 0$ vs $H_1 : x_{\text{new}}^\intercal\beta \neq 0$. It is observed that the bias-corrected estimators are closer to the true values compared to the plug-in estimators.

```
summary(Est)
#>Call:
#>Inference for Linear Functional
#>
#>Estimators:
#> loading est.plugin est.debias Std. Error z value  Pr(>|z|)
#>       1      1.268      1.522     0.1805   8.430 0.000e+00 ***
#>       2     -1.033     -1.172     0.1900  -6.169 6.868e-10 ***
```

### 3.2 Quadratic functional

For a given index set $G \subset \{1, \ldots, p\}$, the function `QF()`, abbreviated for Quadratic Functional, conducts inference for $\beta_G^T A \beta_G$ if $A \in \mathbb{R}^{|G| \times |G|}$ is the submatrix pre-specified or $\beta_G^T \Sigma_{G,G} \beta_G$ under the high-dimensional regression model (1). The function `QF()` can be called with the following arguments.

```
QF(X, y, G, A = NULL, model = c("linear", "logistic", "logistic_alter"),
   intercept = TRUE, beta.init = NULL, split = TRUE, lambda = NULL, mu = NULL,
   prob.filter = 0.05, rescale = 1.1, tau = c(0.25, 0.5, 1), verbose = FALSE)
```

In the function `QF()`, the parameters `X`, `y`, `model`, `intercept`, `beta.init`, `lambda`, `mu`, `prob.filter`, `rescale` maintain the same definitions as in the `LF()` function. In the following, we primarily focus on elaborating the additional arguments introduced for the `QF()` function.

- $G \subset \{1, \ldots, p\}$ is the set of indices of interest.
- `A` is the matrix in the quadratic form, of dimension $|G| \times |G|$. If `A` is specified, it will conduct inference for $\beta_G^T A \beta_G$; otherwise, if left `NULL`, it will turn to $\beta_G^T \Sigma_{G,G} \beta_G$.
- `split` (default = `TRUE`) indicates whether we conduct sample splitting. When `split=FALSE`, the initial estimator of regression coefficients in (3) is computed using one half of the sample while the remaining half is used for bias correction in (13). When `split=TRUE`, the full data is used for computing both the initial estimator and conducting the bias correction.
- `tau.vec` (default = `c(0.25,0.5,1)`) allows the user to supply a vector of possible values for $\tau$ used in (14) and (15).

In the following, we illustrate the usage of `QF()` in the linear regression model.

**Example 2.** For $1 \leq i \leq n$, with $n = 200$, the covariates $X_{i\cdot}$ are generated from multivariate normal distribution with mean $\mu = 0_p$ and covariance $\Sigma \in \mathbb{R}^{p \times p}$, with p = 150, where $\Sigma_{j,k} = 0.5^{|j-k|}$ for $1 \leq j, k \leq p$. The regression coefficients $\beta$ is constructed as $\beta_j = 0.2$ for $25 \leq j \leq 50$ and $\beta_j = 0$ otherwise. We generate the outcome following the model $y_i = X_{i\cdot}^T \beta + \epsilon_i$ with $\epsilon_i$ generated as the standard normal.

```
n <- 200; p <- 150
mu <- rep(0,p)
Cov <- matrix(0, p, p)
for(j in 1:p) for(k in 1:p) Cov[j,k] <- 0.5^(abs(j-k))
beta <- rep(0, p); beta[25:50] <- 0.2
X <- MASS::mvrnorm(n, mu, Cov)
y <- X%*%beta + rnorm(n)
```

We apply the `QF()` function to obtain the point estimator of $\beta_G^T \Sigma_{G,G} \beta_G$ with $G = \{40, \ldots, 60\}$ along with the standard error estimator.

```
test.set <- c(40:60)
Est <- QF(X, y, G = test.set, A = NULL, model = "linear", split = FALSE)
```

We run the function `ci()` that outputs the CIs for $Q_\Sigma$ corresponding to different values of $\tau$. With the default $\tau = c(0.25, 0.5, 1)$, we obtain three different CIs for $\beta_G^T \Sigma_{G,G} \beta_G$; see (16). Note that the true value $\beta_G^T \Sigma_{G,G} \beta_G = 1.16$ belongs to all of these constructed CIs.

```
ci(Est)
#>   tau     lower    upper
#>1 0.25 0.8118792 1.466422
#>2 0.50 0.8046235 1.473677
#>3 1.00 0.7905648 1.487736
```

Subsequently, we employ the `summary()` function to yield the bias-corrected and plug-in estimators, alongside the standard errors for the debiased estimator across different values of $\tau$. Additionally, it provides the p-values for the hypothesis testing $H_0 : Q_\Sigma = 0$ versus $H_1 : Q_\Sigma > 0$.

```
summary(Est)
#> Call:
#> Inference for Quadratic Functional
#>
#>  tau est.plugin est.debias Std. Error z value  Pr(>|z|)
#> 0.25      0.904      1.139     0.1670   6.822 8.969e-12 ***
#> 0.50      0.904      1.139     0.1707   6.674 2.486e-11 ***
#> 1.00      0.904      1.139     0.1779   6.405 1.504e-10 ***
```

Similarly to the `LF()` case, our proposed bias-corrected estimator effectively corrects the plugin estimator's bias, where the true value is 1.16.

### 3.3 Conditional average treatment effect

The function `CATE()`, shorthanded for Conditional Average Treatment Effect, conducts inference for $\Delta(x_{\text{new}}) = f(x_{\text{new}}^\top \beta^{(2)}) - f(x_{\text{new}}^\top \beta^{(1)})$ under the high-dimensional regression model (2). This function can be implemented as follows:

```
CATE(X1, y1, X2, y2, loading.mat, model = c("linear", "logistic", "logistic_alter"),
   intercept = TRUE, intercept.loading = FALSE, beta.init1 = NULL, beta.init2 = NULL,
   lambda = NULL, mu = NULL, prob.filter = 0.05, rescale = 1.1, verbose = FALSE)
```

The majority of the arguments remain consistent with those of the `LF()` function. We will highlight the new parameters specific to the `CATE()` function.

- `X1` and `y1` respectively denote the design matrix and the response vector for the first sample of data, while `X2` and `y2` denote those for the second sample of data.
- `beta.init1` (default = `NULL`) is the initial estimator in (3) for the first sample, while `beta.init2` (default = `NULL`) is for the second sample. If left as `NULL`, they are computed using `cv.glmnet` in **glmnet**.
- `lambda` (default = `NULL`) represents the common tuning parameter $\lambda$ for computing the initial estimators `beta.init1` and `beta.init2`. If left as `NULL`, `cv.glmnet` in **glmnet** is employed for its computation, done separately for each sample.
- `mu` (default = `NULL`) represents the common tuning parameter $\mu_0$ in (10) for computing the projection directions for the two samples. When unspecified and left as `NULL`, it is computed as the smallest $\mu_0$ such that (10) has a finite solution, done separately for each sample.

We consider the logistic regression case to illustrate `CATE()` with the argument `model = "logistic_alter"`.

**Example 3.** In the first group of data, the covariates $X_{i\cdot}^{(1)}$, for $1 \le i \le n_1$ with $n_1 = 100$, follow multivariate normal distribution with $\mu = 0_p$ and covariance $\Sigma^{(2)} = \mathbf{I}_p$; in the second group of data, the covariates $X_{i\cdot}^{(2)}$, for $1 \le i \le n_2$ with $n_2 = 180$, follow multivariate normal distribution with $\mu = 0_p$ and covariance $\Sigma^{(2)} \in \mathbb{R}^{p\times p}$ with $p = 120$ and $\Sigma_{j,k}^{(2)} = 0.5^{|j-k|}$ for $1 \le j,k \le p$. We generate the binary outcomes following the model $y_i^{(k)} \sim \text{Bernoulli}(f(X_{i\cdot}^{(k)\top}\beta^{(k)}))$ with $f(z) = \exp(z)/[1+\exp(z)]$ for $k = 1, 2$. See the following code for details of $\beta^{(1)}$ and $\beta^{(2)}$.

```
n1 <- 100; n2 <- 180; p <- 120
mu1 <- mu2 <- rep(0,p)
Cov1 <- diag(p)
Cov2 <- matrix(0, p, p)
for(j in 1:p) for(k in 1:p) Cov2[j,k] <- 0.5^(abs(j-k))
beta1 <- rep(0, p); beta1[c(1,2)] <- c(0.5, 0.5)
beta2 <- rep(0, p); beta2[c(1,2)] <- c(1.8, 1.8)
X1 <- MASS::mvrnorm(n1, mu1, Cov1); val1 <- X1%*%beta1
X2 <- MASS::mvrnorm(n2, mu2, Cov2); val2 <- X2%*%beta2
y1 <- rbinom(n1, 1, exp(val1)/(1+exp(val1)))
y2 <- rbinom(n2, 1, exp(val2)/(1+exp(val2)))
```

We then employ the function `CATE()` to obtain point estimator of $\Delta(x_{\text{new}})$ and the associated standard error estimator. By setting `model = "logistic_alter"`, we set the weight $w(.) = 1$ in (9). See Table 1.

```
loading.mat <- c(1, 1, rep(0, p-2))
Est <- CATE(X1, y1, X2, y2,loading.mat, model = "logistic_alter")
```

Having fitted the model, it allows for method `ci()` and `summary()` as `LF()` does. We mainly demonstrate the `ci()` function and first construct confidence interval for $x_{\text{new}}^\top(\beta^{(2)} - \beta^{(1)})$ and observe that 95% CI covers the true value 2.6.

```
ci(Est)
#>  loading    lower    upper
#>1       1 1.614269 4.514703
```

If we further specify the argument `probability` as `TRUE` for the logistic regression, `ci()` yields the CI for $f(x_{\text{new}}^\top \beta^{(2)}) - f(x_{\text{new}}^\top \beta^{(1)})$ whose true value is 0.2423.

```
ci(Est, probability = TRUE)
#>  loading     lower     upper
#>1       1 0.1531872 0.5086421
```

### 3.4 Inner product

The function `InnProd()`, shorthanded for Inner Product, conducts inference for $\beta_{\mathrm{G}}^{(1)\mathsf{T}} A \beta_{\mathrm{G}}^{(2)}$ with $A \in \mathbb{R}^{|\mathrm{G}|\times|\mathrm{G}|}$ where G denotes the prespecified index set. When the matrix $A$ is not specified, the default inference target becomes $\beta_{\mathrm{G}}^{(1)\mathsf{T}} \Sigma_{\mathrm{G,G}} \beta_{\mathrm{G}}^{(2)}$.

```
InnProd(X1, y1, X2, y2, G, A = NULL, model = c("linear", "logistic", "logistic_alter"),
   intercept = TRUE, beta.init1 = NULL, beta.init2 = NULL, split = TRUE, lambda = NULL,
   mu = NULL, prob.filter = 0.05, rescale = 1.1, tau = c(0.25,0.5,1), verbose = FALSE)
```

The arguments of `InnProd()` are similarly defined as for the function `CATE()`, and we mainly highlight the new arguments in the following.

- `G` is the pre-specified index set, a subset of $\{1, \cdots, p\}$.
- `A` is the matrix in the inner product form. If the matrix `A` is specified, it will conduct inference for $\beta_{\mathrm{G}}^{(1)\mathsf{T}} A \beta_{\mathrm{G}}^{(2)}$; otherwise, it will turn to $\beta_{\mathrm{G}}^{(1)\mathsf{T}} \Sigma_{\mathrm{G,G}} \beta_{\mathrm{G}}^{(2)}$ where $\Sigma$ is the common design covariance matrix corresponding to the two samples.

In the following code, we demonstrate the use of `InnProd()` in the linear regression.

**Example 4.** In the first group of data, the covariates $X_{i\cdot}^{(1)}$, for $1 \le i \le n_1$ with $n_1 = 200$, follow multivariate normal distribution with $\mu = 0_p$ and covariance $\Sigma^{(1)} = \mathbf{I}_p$; in the second group of data, the covariates $X_{i\cdot}^{(2)}$, for $1 \le i \le n_2$ with $n_2 = 260$, follow multivariate normal distribution with $\mu = 0_p$ and covariance $\Sigma^{(2)} \in \mathbb{R}^{p\times p}$ with $p = 120$ and $\Sigma_{j,k}^{(2)} = 0.5^{|j-k|}$ for $1 \le j, k \le p$. We generate following the model $y_i^{(k)} = X_{i\cdot}^{(k)\mathsf{T}} \beta^{(k)} + \epsilon_i^{(k)}$ with standard normal error $\epsilon_i^{(k)}$ for $k = 1, 2$.

```
n1 <- 200; n2 <- 260; p <- 120
mu1 <- mu2 <- rep(0,p)
Cov1 <- diag(p)
Cov2 <- matrix(0, p, p)
for(j in 1:p) for(k in 1:p) Cov2[j,k] <- 0.5^(abs(j-k))
beta1 <- rep(0, p); beta1[1:10] <- 0.5
beta2 <- rep(0, p); beta2[3:12] <- 0.4
X1 <- MASS::mvrnorm(n1, mu1, Cov1)
X2 <- MASS::mvrnorm(n2, mu2, Cov2)
y1 <- X1%*%beta1 + rnorm(n1)
y2 <- X2%*%beta2 + rnorm(n2)
```

After preparing the data, we utilize the `InnProd` function to build a debiased estimator and its associated standard error for $\beta_{\mathrm{G}}^{(1)\mathsf{T}} \beta_{\mathrm{G}}^{(2)}$ with $A = \mathbf{I}_{|\mathrm{G}|}$ and $G = \{1, 2, \ldots, 20\}$.

```
test.set <- c(1:20)
A <- diag(length(test.set))
Est <- InnProd(X1, y1, X2, y2, G = test.set, A, model = "linear")
```

Having fitted the model, it allows for method `ci()` and `summary()` as `QF()` does. Note that the true value $\beta_{\mathrm{G}}^{(1)\mathsf{T}} \beta_{\mathrm{G}}^{(2)} = 1.6$ is included in the following CIs with default values $\tau \in \{0.25, 0.5, 1\}$.

```
ci(Est)
#>    tau     lower    upper
#> 1 0.25 0.7432061 2.490451
#> 2 0.50 0.7128181 2.520839
#> 3 1.00 0.6520422 2.581615
```

### 3.5 Distance

The function `Dist()`, shorthanded for Distance, is designed to perform inference for the quadratic form $\gamma_{\mathrm{G}}^{\mathsf{T}} A \gamma_{\mathrm{G}}$, where $\gamma = \beta^{(2)} - \beta^{(1)}$ and the index set $\mathrm{G} \subset \{1, \ldots, p\}$. The matrix $A$ can either be a pre-specified submatrix in $\mathbb{R}^{|\mathrm{G}|\times|\mathrm{G}|}$ or the covariance matrix $\Sigma_{\mathrm{G,G}}$ within the context of high-dimensional

regression models.

```
Dist(X1, y1, X2, y2, G, A = NULL, model = c("linear", "logistic", "logistic_alter"),
   intercept = TRUE, beta.init1 = NULL, beta.init2 = NULL, split = TRUE, lambda = NULL,
   mu = NULL, prob.filter = 0.05, rescale = 1.1, tau = c(0.25,0.50,1), verbose = FALSE)
```

The arguments of `Dist()` are similarly defined as for the function `CATE()`, and we mainly highlight the new arguments in the following.

- `G` is the pre-specified index set, a subset of $\{1, \cdots, p\}$.
- `A` is the matrix in the inner product form. If the matrix `A` is specified, it will conduct inference for $\gamma_G^\intercal A \gamma_G$; otherwise, it will turn to $\gamma_G^\intercal \Sigma_{G,G} \gamma_G$ where $\Sigma$ is the common design covariance matrix corresponding to the two samples.

In the following example, we demonstrate the application of the `Dist()` function within a linear regression context.

**Example 5.** For the first group of data, the covariates $X_{i\cdot}^{(1)}$, for each $1 \leq i \leq n_1$ where $n_1 = 220$, are drawn from a multivariate normal distribution with mean $\mu = 0_p$ and covariance $\Sigma^{(1)} = \mathbf{I}_p$. In the second group of data, the covariates $X_{i\cdot}^{(2)}$, for each $1 \leq i \leq n_2$ with $n_2 = 180$, also follow multivariate normal distribution with mean $\mu = 0_p$ and covariance $\Sigma^{(2)} \in \mathbb{R}^{p \times p}$ with $p = 100$ and $\Sigma_{j,k}^{(2)} = 0.5^{|j-k|}$ for $1 \leq j, k \leq p$. The regression coefficients $\beta^{(1)}$ and $\beta^{(2)}$ are generated in the following code. Outcomes for both groups are then generated according to the model $y_i^{(k)} = X_{i\cdot}^{(k)\intercal} \beta^{(k)} + \epsilon_i^{(k)}$ where $\epsilon_i^{(k)}$ is a standard normal error for $k = 1, 2$.

```
n1 <- 220; n2 <- 180; p <- 100
mu <- rep(0,p); Cov <- diag(p)
beta1 <- rep(0, p); beta1[1:2] <- c(0.5, 1)
beta2 <- rep(0, p); beta2[1:10] <- c(0.3, 1.5, rep(0.08, 8))
X1 <- MASS::mvrnorm(n1, mu, Cov)
X2 <- MASS::mvrnorm(n2, mu, Cov)
y1 <- X1%*%beta1 + rnorm(n1)
y2 <- X2%*%beta2 + rnorm(n2)
```

Next we employ the `Dist()` function to construct a debiased estimator for $\gamma_G^\intercal \Sigma_{G,G} \gamma_G$ with $G = \{1, \ldots, 10\}$, alongside the corresponding estimated standard error.

```
test.set <- c(1:10)
Est <- Dist(X1, y1, X2, y2, G = test.set, A = NULL, model = "linear", split = FALSE)
```

Having fitted the model, it allows for methods `ci()` and `summary()` as `QF()` does. Here, the true value is $\gamma_G^\intercal \Sigma_{G,G} \gamma_G = 0.3412$. Similar to the previous instances, we note that the bias-corrected estimator effectively corrects the bias of the plugin estimator. Depending on the $\tau$ values, we obtain various CIs, all of which encompass the true value. It is important to mention that in case of negative lower boundaries, they will be truncated at 0 for $\tau = 0.5$ and $\tau = 1$.

```
ci(Est)
#>   tau    lower     upper
#>1 0.25 0.028202 0.6831165
#>2 0.50 0.000000 0.7196383
#>3 1.00 0.000000 0.7926819

summary(Est)
#> Call:
#> Inference for Distance
#>
#>  tau est.plugin est.debias Std. Error z value Pr(>|z|)
#> 0.25     0.4265     0.3557     0.1671   2.129  0.03327 *
#> 0.50     0.4265     0.3557     0.1857   1.915  0.05547 .
#> 1.00     0.4265     0.3557     0.2230   1.595  0.11070
```

## 4 Comparative analysis

In this section, we perform a comparative analysis of **SIHR** compared to existing methods in numerical simulations. Initially, we compare our approach to traditional plug-in Lasso estimators, implemented

by **glmnet**, to demonstrate the effectiveness of bias correction. Subsequently, we will compare our method against other inference techniques implemented by **hdi** and **SSLasso** (available at `http://web.stanford.edu/~montanar/sslasso/code.html`). We aim to demonstrate that our proposed method ensures a unified guarantee of coverage across a wider range of settings while also significantly enhancing computational efficiency.

Throughout this section, we evaluate the performance of our `LF()` estimator using synthetic data, generated as follows. For $1 \le i \le n$,

$$X_{i\cdot} \overset{iid}{\sim} \mathcal{N}(0_p, \mathbf{I}_p), \;\; y_i \sim \begin{cases} \mathcal{N}(X_{i\cdot}^\intercal \beta, 1), & \text{for linear model;} \\ \text{Bernoulli}(X_{i\cdot}^\intercal \beta), & \text{for logistic model} \end{cases}, \;\; \text{where } \beta = (0.5, 0.75, 0.25, 0_{p-3}).$$

The sample size $n$ varies across $\{200, 400\}$ with the number of covariates fixed as $p = 300$. The loading is set as $x_{\text{new}} = (1, 0.75, 0.5, 0_{p-3})^\intercal$, making our inference target $x_{\text{new}}^\intercal \beta = 1.1875$. All results summarized here are based on 500 simulation rounds.

### 4.1 Effectiveness of bias correction

We compare the bias-corrected estimator $\widehat{x_{\text{new}}^\intercal \beta}$, as defined in (9), against the plug-in Lasso estimator $x_{\text{new}}^\intercal \widehat{\beta}$, where $\widehat{\beta}$ represents the Lasso estimator in (3) implemented by **glmnet** with a tuning parameter selected through the package's built-in cross-validation.

Figure 1 reports the performance for logistic regression with $n = 400$ and displays histograms of 500 point estimates for the plug-in Lasso and the debiased estimates outputted by **SIHR**. The target $x_{\text{new}}^\intercal \beta = 1.1875$ is highlighted with red vertical lines. It is evident that the plug-in Lasso estimators exhibit significant bias and are not suitable for inference, whereas our bias-corrected estimators effectively correct this bias.

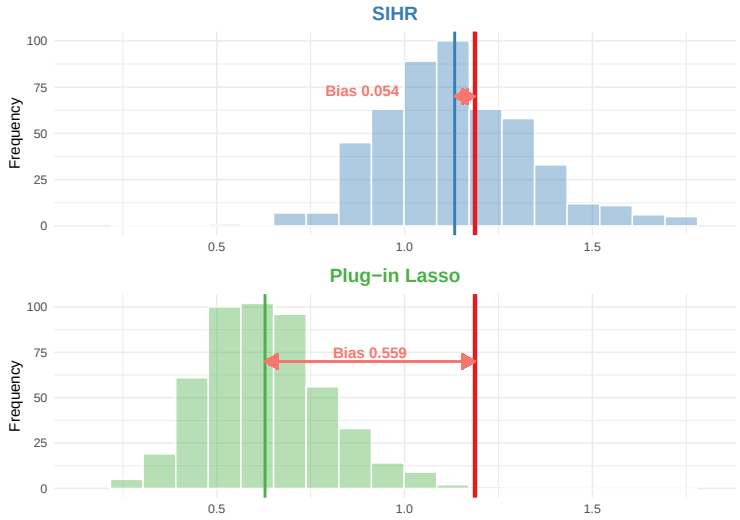


**Figure 1:** Comparison of the debiased estimates output by **SIHR** and plug-in Lasso estimates for $x_{\text{new}}^\intercal \beta$ in the logistic model with $n = 400$. The upper panel shows the bias-corrected point estimates derived using our package **SIHR**, while the lower panel features the plug-in point estimates from the **glmnet** package. Red vertical lines indicate the target value $x_{\text{new}}^\intercal \beta = 1.1875$. Biases between this target and the empirical means of the estimates are highlighted for each method.

### 4.2 Comparison with other inference methods

We compare the performance of the **SIHR** package with existing softwares, including R packages **hdi** and **SSLasso**. The performance metrics include the empirical coverage, averaged length of confidence intervals, and averaged computational time (in seconds). All metrics are reported as the average of 500 simulation rounds. The confidence intervals based on **hdi** and **SSLasso** are defined as follows:

$$\text{CI}_\alpha(x_{\text{new}}) = \left( x_{\text{new}}^\intercal \widetilde{\beta} - z_{\alpha/2} (x_{\text{new}}^\intercal \widetilde{V} x_{\text{new}})^{1/2}, \;\; x_{\text{new}}^\intercal \widetilde{\beta} + z_{\alpha/2} (x_{\text{new}}^\intercal \widetilde{V} x_{\text{new}})^{1/2} \right),$$

where $\widetilde{\beta}$ denotes the debiased estimator and $\widetilde{V}$ denotes the estimated covariate matrix of $\widetilde{\beta}$, outputted by **hdi** and **SSLasso**. Note that **SSLasso** only provides inference for linear regression models.

**Coverage and Length** As shown in Table 3, the CIs based on **SIHR** achieve desired coverage across various scenarios, and the lengths decrease with larger sample sizes. In contrast, the coverage of CIs from **hdi** and **SSLasso** may be slightly undercovered, especially when $n = 200$.

**Computation Efficiency** We examine the computational efficiency of these methods and report the average computation time in the "Time" column, measured in seconds. The **SIHR** package demonstrates notable computational efficiency, with an average processing time of under 10 seconds. In comparison, using the algorithm in **hdi** with parameters $n = 400$ and $p = 300$ requires approximately 8 minutes. This significant difference stems from the fact that the **hdi** algorithm is not tailored for inferring linear functionals and separately implementing $p$ bias correction steps, one for each regression coefficient, while **SIHR** only implements a single bias correction step.

| | | Linear | | | Logistic | | |
|---|---|---|---|---|---|---|---|
| $n$ | Method | Cov | Len | Time | Cov | Len | Time |
| 200 | **SIHR** | 0.94 | 0.47 | 3 | 0.94 | 0.94 | 3 |
| | hdi | 0.90 | 0.38 | 211 | 0.91 | 0.80 | 213 |
| | SSLasso | 0.83 | 0.34 | 17 | - | - | - |
| 400 | **SIHR** | 0.96 | 0.34 | 12 | 0.96 | 0.74 | 14 |
| | hdi | 0.94 | 0.27 | 475 | 0.89 | 0.56 | 489 |
| | SSLasso | 0.94 | 0.29 | 38 | - | - | - |

**Table 3:** Comparison of methods implemented by packages **SIHR**, **hdi**, **SSLasso** in both linear and logistic models with $n \in \{200, 400\}$. The columns labeled "Cov" and "Len" denote the empirical coverage and the length of the confidence intervals over 500 simulation runs, respectively; "Time" indicates the average computation time in seconds. The columns titled "Linear" and "Logistic" refer to the regression model applied. According to the experimental design, a valid inference method should achieve a coverage rate of approximately 0.95.

## 5 Real data applications

### 5.1 Motif regression

We showcase the application of the `LF()` function in motif regression analysis, which investigates the impact of motif matching scores on gene expression levels, as discussed in the literature (Beer and Tavazoie, 2004; Conlon et al., 2003; Das et al., 2004; Yuan et al., 2007). Motifs are specific DNA sequences bound to transcription factors, playing crucial roles in controlling transcription activities, such as gene expressions (Yuan et al., 2007). The matching score of a motif measures its prevalence, reflecting how prominently a motif appears in the upstream regions of genes. These matching scores are recognized for their effectiveness in predicting gene expression levels. Our goal is to quantitatively assess the association between these matching scores and gene expression, elucidating the underlying biological mechanisms. In this analysis, we work with a dataset that includes the expression levels of $n = 2587$ genes, where matching scores of $p = 666$ motifs are observed on each gene. The structure of the data is organized as follows: for $1 \leq i \leq 2587$,

- $y_i$ : the expression level of gene $i$;
- $X_{i,j}$ : the matching score of the $j$-th motif on gene $i$, for $1 \leq j \leq 666$.

Below, we display several observations of the response variable along with the first four covariates out of a total of 666.

```
colnames(X) <- paste0("X",1:ncol(X))
head(cbind(y, X[,1:4]))
#>               y        X1       X2       X3       X4
#>   YAL002W  0.51 1.1595129 1.573024 1.239862 1.144537
#>   YAL003W -3.06 1.9581497 1.928997 1.228753 1.118513
#>   YAL007C -1.86 1.3047351 1.617691 1.299527 1.126370
#>   YAL025C -1.54 0.8057353 1.487356 1.395147 1.003005
#>   YAL034C  1.00 0.8886961 1.860788 1.569881 1.316531
#>   YAL035W -2.05 1.3377646 1.152577 1.532653 1.012072
```

We seek to investigate the relationships between the matching scores of individual motifs ($X_{\cdot,j}$ for $1 \leq j \leq 666$) and gene expression levels ($y$). Our objective is to access the significance of these associations. For this purpose, the `LF()` function from the package **SIHR** is utilized to compute 95% confidence intervals for the 666 regression coefficients.

```
p <- ncol(X)
loading.mat <- diag(p)
Est <- LF(X, y, loading.mat, model='linear')
ci(Est)
```

We then summarize and visualize the resulting 666 confidence intervals in Figure 2. The results reveal that 25 of these intervals, highlighted in blue, lie entirely above zero, indicating a positive association between the matching scores of these specific motifs and gene expression levels. Conversely, 23 intervals, marked in green, fall completely below zero, suggesting a negative influence of these motifs on gene expression levels. Overall, these results demonstrate that 48 motifs out of the total have a statistically significant influence on gene expression, offering valuable insights into the regulatory mechanisms involved.

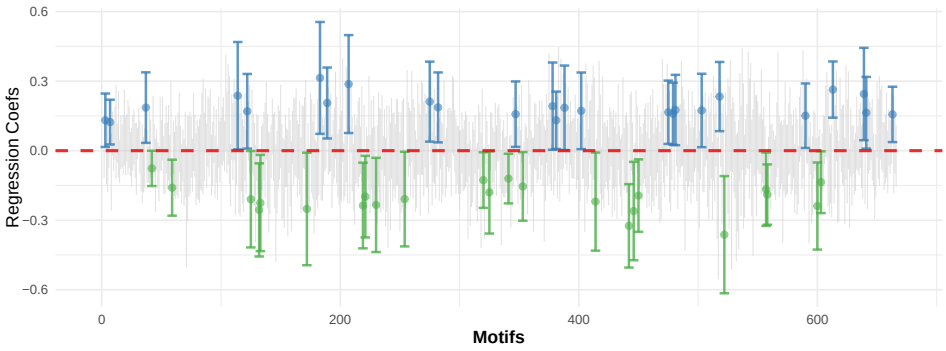


**Figure 2:** Motif: Constructed CIs for the 666 regression coefficients. Motifs represented by blue CIs indicate a significant positive association with gene expression levels, whereas those with green CIs demonstrate significant negative associations.

### 5.2 Fasting glucose level data

We have illustrated the application of the `LF()` function in a linear regression context. We now demonstrate its use on another real application in a logistic regression setting. In this study, we examine the impact of polymorphic genetic markers on glucose levels in a stock mice population. The data, accessible at https://wp.cs.ucl.ac.uk/outbredmice/heterogeneous-stock-mice/, uses fasting glucose levels, dichotomized at 11.1 mmol/L, as the response variable —an important indicator for type-2 diabetes. Specifically, glucose levels below 11.1 mmol/L are considered normal and labeled as $y_i = 0$, while levels above 11.1 mmol/L are classified as high, indicating pre-diabetic or diabetic conditions, and labeled as $y_i = 1$.

The dataset initially comprises $10,346$ polymorphic genetic markers for a sample size $n = 1,269$. Given the large number of markers and the significant correlation among some of them, we implement a selection criterion to ensure the maximum absolute correlation among the markers does not exceed 0.75. After filtering, we narrow down to a subset of $2,341$ genetic markers. Additionally, we include "gender" and "age" as baseline covariates. To prepare for the analysis, both the genetic markers and baseline covariates are standardized. To sum up, the data structure is organized as follows: for $i = 1, \ldots, 1269$:

- $y_i$ : binary indicator of whether the fasting glucose level is above 11.1 mmol/L
- $X_{i,j}$ : genetic marker $j$ for mouse $i$ with $j = 1, \ldots, 2341$
- $X_{i,2342}$ : gender of mouse $i$
- $X_{i,2343}$ : age of mouse $i$

Below, we display several observations of the response variable along with the first four covariates out of a total of 2343.

```
head(cbind(y, X[,1:4]))
#>              y rs3674785_G rs13475705_A rs13475706_G rs3684358_C
#>   A048005080 1    0.184158   -0.5697056    0.6063887  -0.3444252
#>   A048006555 0   -1.258413   -0.5697056   -0.9113771   1.1272098
#>   A048007096 0    0.184158    1.5137423   -0.9113771  -0.3444252
#>   A048010273 1   -1.258413   -0.5697056   -0.9113771   1.1272098
#>   A048010371 0    0.184158   -0.5697056    0.6063887  -0.3444252
#>   A048011287 0    0.184158    1.5137423   -0.9113771  -0.3444252
```

Given the real dataset, we aim to investigate the association of each polymorphic marker ($X_{\cdot,j}$ for $1 \leq j \leq 2341$) with fasting glucose levels ($y$) and determine the statistical significance of each association. We employ the function `LF()` configured with `model = "logistic"` to compute confidence intervals for the initial 2341 regression coefficients, which correspond to all polymorphic markers, as demonstrated in the following code:

```
p <- ncol(X)
loading.mat <- diag(p)[,-c(2342,2343)]
Est <- LF(X, y, loading.mat, model='logistic')
ci(Est)
```

We then visualize the obtained 2341 confidence intervals in Figure 3. It reveals that 13 genetic markers display CIs exclusively above 0 (marked in blue), signifying a significant positive correlation with fasting glucose levels. Conversely, 16 markers exhibit CIs entirely below 0 (marked in green), denoting a significant negative correlation with fasting glucose levels. These results showcase that 29 genetic markers out of the total have a statistically significant impact on glucose levels.

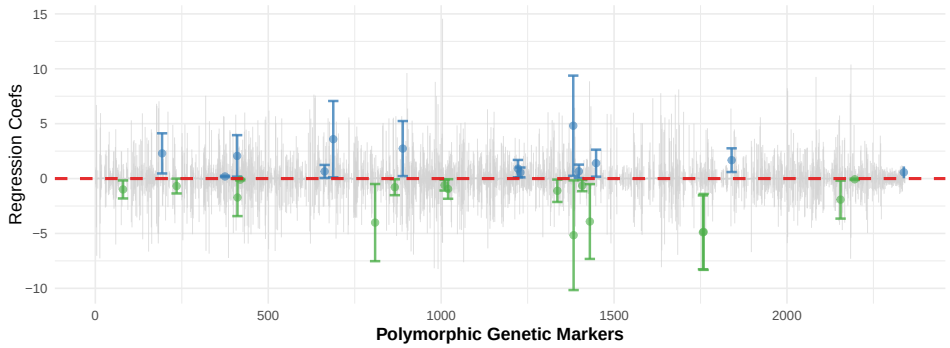


**Figure 3:** Glucose: Constructed CIs for the first 2341 regression coefficients. Genetic markers represented by blue CIs indicate a significant positive association with the fasting glucose level, whereas those with green CIs demonstrate significant negative associations.

## 6 Conclusion

There has been significant recent progress in debiasing inference methods for high-dimensional GLMs. This paper highlights the application of advanced debiasing techniques in high-dimensional GLMs using the R package **SIHR**. The package provides tools for estimating bias-corrected point estimators and constructing CIs for various low-dimensional objectives in both one- and two-sample regression settings. Through extensive simulations and real-data analyses, we demonstrate the practicality and versatility of the package across diverse fields of study, making it an essential addition to the literature.

## 7 Acknowledgement

Prabrisha Rakshit and Zhenyu Wang contributed equally to this work and are considered co-first authors. Dr. Tony Cai's research was supported in part by NSF grant DMS-2015259 and NIH grants R01-GM129781 and R01-GM123056. Dr. Zijian Guo's research was supported in part by NSF grants DMS-1811857 and DMS-2015373 and NIH grants R01-GM140463 and R01-LM013614. Dr. Zijian Guo is grateful to Dr. Lukas Meier for sharing the motif regression data used in this paper.

## References


M. A. Beer and S. Tavazoie. Predicting gene expression from sequence. *Cell*, 117(2):185–198, 2004. [p41]

A. Belloni, V. Chernozhukov, and L. Wang. Square-root lasso : pivotal recovery of sparse signals via conic programming. *Biometrika*, 98(4):791–806, 2011. [p27]

P. J. Bickel, Y. Ritov, and A. B. Tsybakov. Simultaneous analysis of lasso and dantzig selector. *The Annals of statistics*, 37(4):1705–1732, 2009. [p28]

P. Bühlmann and S. van de Geer. *Statistics for high-dimensional data: methods, theory and applications*. Springer Science & Business Media, 2011. [p27]

T. Cai, T. Tony Cai, and Z. Guo. Optimal statistical inference for individualized treatment effects in high-dimensional models. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 83 (4):669–719, 2021a. [p27, 28, 29, 30, 31]

T. T. Cai and Z. Guo. Semisupervised inference for explained variance in high dimensional linear regression and its applications. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 82(2):391–419, 2020. [p27]

T. T. Cai, Z. Guo, and R. Ma. Statistical inference for high-dimensional generalized linear models with binary outcomes. *Journal of the American Statistical Association*, pages 1–14, 2021b. [p27, 30]

E. M. Conlon, X. S. Liu, J. D. Lieb, and J. S. Liu. Integrating regulatory motif discovery and genome-wide expression analysis. *Proceedings of the National Academy of Sciences*, 100(6):3339–3344, 2003. [p41]

D. Das, N. Banerjee, and M. Q. Zhang. Interacting models of cooperative gene regulation. *Proceedings of the National Academy of Sciences*, 101(46):16234–16239, 2004. [p41]

J. Fan and R. Li. Variable selection via nonconcave penalized likelihood and its oracle properties. *Journal of the American Statistical Association*, 96:1348–1360, 2011. [p27]

Z. Guo, W. Wang, T. T. Cai, and H. Li. Optimal estimation of genetic relatedness in high-dimensional linear models. *Journal of the American Statistical Association*, 114:358–369, 2019. [p27, 28, 30, 32]

Z. Guo, P. Rakshit, D. S. Herman, and J. Chen. Inference for the case probability in high-dimensional logistic regression. *The Journal of Machine Learning Research*, 22(1):11480–11533, 2021a. [p30]

Z. Guo, C. Renaux, P. Bühlmann, and T. Cai. Group inference in high dimensions with applications to hierarchical testing. *Electronic Journal of Statistics*, 15(2):6633–6676, 2021b. [p27, 29, 30, 31]

Z. Guo, X. Li, L. Han, and T. Cai. Robust inference for federated meta-learning. *arXiv preprint arXiv:2301.00718*, 2023. [p28]

J. Huang and C.-H. Zhang. Estimation and selection via absolute penalized convex minimization and its multistage adaptive applications. *Journal of Machine Learning Research*, 13(Jun):1839–1864, 2012. [p27]

A. Javanmard and A. Montanari. Confidence intervals and hypothesis testing for high-dimensional regression. *The Journal of Machine Learning Research*, 15(1):2869–2909, 2014. [p27, 28]

R. Ma, Z. Guo, T. T. Cai, and H. Li. Statistical inference for genetic relatedness based on high-dimensional logistic regression. *arXiv preprint arXiv:2202.10007*, 2022. [p28, 32]

N. Meinshausen and P. Bühlmann. High-dimensional graphs and variable selection with the lasso. *The annals of statistics*, 34(3):1436–1462, 2006. [p28]

N. Meinshausen and B. Yu. Lasso-type recovery of sparse representations for high-dimensional data. *Annals of Statistics*, 37(1):246–270, 2009. [p27]

S. Negahban, B. Yu, M. J. Wainwright, and P. K. Ravikumar. A unified framework for high-dimensional analysis of $m$-estimators with decomposable regularizers. In *Advances in Neural Information Processing Systems*, pages 1348–1356, 2009. [p27]

T. Sun and C.-H. Zhang. Scaled sparse linear regression. *Biometrika*, 99(4):879–898, 2012. [p27]

R. Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 58(1):267–288, 1996. [p27]

S. van de Geer, P. Bühlmann, Y. Ritov, and R. Dezeure. On asymptotically optimal confidence regions and tests for high-dimensional models. *The Annals of Statistics*, 42:1166–1202, 2014. [p27, 28]

M. J. Wainwright. Sharp thresholds for high-dimensional and noisy sparsity recovery using $\ell_1$-constrained quadratic programming (lasso). *IEEE transactions on information theory*, 55(5):2183–2202, 2009. [p28]

Y. Yuan, L. Guo, L. Shen, and J. S. Liu. Predicting gene expression from sequence: a reexamination. *PLoS computational biology*, 3(11):e243, 2007. [p41]

C.-H. Zhang. Nearly unbiased variable selection under minimax concave penalty. *Annals of Statistics*, 38(2):894–942, 2010. [p27]

C.-H. Zhang and S. S. Zhang. Confidence intervals for low dimensional parameters in high dimensional linear models. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 76(1):217–242, 2014. [p27, 28]

P. Zhao and B. Yu. On model selection consistency of lasso. *The Journal of Machine Learning Research*, 7: 2541–2563, 2006. [p28]

*Prabrisha Rakshit*
*Rutgers, The State University of New Jersey*
*USA*
prabrisha.rakshit@rutgers.edu

*Zhenyu Wang*
*Rutgers, The State University of New Jersey*
*USA*
zw425@stat.rutgers.edu

*Tony Cai*
*University of Pennsylvania*
*USA*
tcai@wharton.upenn.edu

*Zijian Guo*
*Rutgers, The State University of New Jersey*
*USA*
zijguo@stat.rutgers.edu

# SNSeg: An R Package for Time Series Segmentation via Self-Normalization

*by Shubo Sun, Zifeng Zhao, Feiyu Jiang, and Xiaofeng Shao*

**Abstract** T

## 1 Introduction

Time series segmentation, also known as change-point estimation in time series, has become increasingly popular in various fields such as statistics, bioinformatics, climate science, economics, finance, signal processing, epidemiology, among many others. As a result, numerous methods have been proposed to address different types of change-point estimation problems under various settings. This in turn leads to the development of many **R** packages for their implementation.

Here, we list some commonly used and influential packages for change-point analysis in the **R** programming language. The package **strucchange** (Zeileis et al., 2002) employs algorithms proposed by Zeileis et al. (2003) to identify structural changes in linear regression models. The package **changepoint** (Killick and Eckley, 2014) provides numerous methods for estimating change-points in a univariate time series, containing the Binary Segmentation (BS) and the pruned exact linear time (PELT) algorithm as described in Killick et al. (2012), and the segment neighbourhoods algorithm in Auger and Lawrence (1989). The package **mosum** (Meier et al., 2021) executes the moving sum (MOSUM) procedure introduced by Eichinger and Kirch (2018) for univariate time series. It can implement MOSUM with a single bandwidth parameter and also allows multiple bandwidths via either bottom-up merging or localized pruning. The package **cpss** (Wang and Zou, 2023) focuses on change-point estimation in various generalized linear models utilizing the sample-split strategy proposed by Zou et al. (2020). We note that there are also packages targeting nonparametric distributional changes, e.g. the package **ecp** (James and Matteson, 2014) and **cpm** (Ross, 2015).

However, the aforementioned methods, as well as their implementation packages, are subject to certain limitations when applied to change-point estimation in multivariate time series. First, most packages only provide functions to detect specific types of changes (e.g. mean or variance). Although it may be possible to modify these functions to cover other types of changes (e.g. quantiles), such a generalization is usually not easy and requires non-trivial effort. This means that for the same dataset, different methods or substantial modifications to the existing codes may be required for estimating different types of changes, which may incur inconvenience of implementation for practitioners. Second, many packages implement methods that assume temporal independence among data, which may not be realistic in practice, and thus may suffer from issues such as false positive detections.

Recently, Zhao et al. (2022) have developed a new framework called self-normalization based change-point estimation (SNCP) to overcome the above limitations. The most appealing feature of SNCP is its versatility as it allows for change-point estimation in a broad class of parameters (such as mean, variance, correlation, quantile and their combinations) in a unified fashion. The basic idea of SNCP is to augment the conventional cumulative sum (CUSUM) statistics with the technique called self-normalization (SN). SN is originally introduced by Shao (2010) for confidence interval construction of a general parameter in the stationarity time series setting, and is later extended to change-point testing by Shao and Zhang (2010). It can bypass the issue of bandwidth selection in the consistent long-run variance estimation. See Shao (2015) for a review. SNCP is fully nonparametric, robust to temporal dependence and applicable universally for various parameters of interest for a multivariate time series. Furthermore, based on a series of carefully designed nested local-windows, SNCP can isolate each true change-point adaptively and achieves the goal of multiple change-point estimation with respectable detection power and estimation accuracy.

In this paper, we introduce the **R** package **SNSeg** (Sun et al., 2023), which implements the SNCP framework in Zhao et al. (2022) for univariate and multivariate time series segmentation. This is achieved by the functions `SNSeg_Uni()` and `SNSeg_Multi()`, respectively. Another contribution of this paper is to extend the SNCP framework to change-point estimation in the mean vector of a high-dimensional time series. Since SNCP is only applicable to fixed-dimensional time series, a new procedure based on U-statistics (Wang et al., 2022), termed as SNHD, is proposed by modifying the original SNCP in Zhao et al. (2022). The implementation of SNHD is available through the function `SNSeg_HD()`. Graphical options are also allowed for plotting the estimated change-points and associated test statistics.

The rest of the paper is organized as follows. We first provide the background of SN based statistics and the SNCP/SNHD procedures for change-point estimation in Section 2.2. In Section

2.3, we demonstrate the core functions of the package **SNSeg** by various examples of change-point estimation problems. Additional simulation results and comparison with existing packages are provided in Section 2.4. Section 2.5 concludes.

## 2 SNCP Framework

This section gives necessary statistical backgrounds of SNCP in change-point estimation problems. We first demonstrate how an SN based CUSUM test statistic works for estimating a single change-point, and then introduce the nested local-window based SNCP algorithm for multiple change-point estimation. The extension of SNCP to change-point estimation in high-dimensional mean problem is also provided and we term the related algorithm as SNHD. The issue of how to choose tuning parameters is also discussed.

### 2.1 Single Change-Point Estimation

Let $\{Y_t\}_{t=1}^n$ be a sequence of multivariate time series of dimension $p$, which is assumed to be fixed for now. We aim to detect whether there is a change-point in the quantities $\{\theta_t\}_{t=1}^n$ defined by $\theta_t = \theta(F_t) \in \mathbb{R}^d$, where $F_t$ denotes the distribution function of $Y_t$, and $\theta(\cdot)$ is a general functional such as mean, variance, auto-covariance, quantiles, etc. More specifically, if there is no change-point, then

$$\theta_1 = \cdots = \theta_n. \tag{1}$$

Otherwise, we assume there is an unknown change-point $k^* \in \{1, \cdots, n-1\}$ defined by

$$\theta_1 = \cdots = \theta_{k^*} \neq \theta_{k^*+1} = \cdots = \theta_n, \tag{2}$$

and our interest is to recover the location $k^*$.

The above setting allows for at most one change-point in $\{\theta_t\}_{t=1}^n$. A commonly used statistic for testing the existence of change-points is based on the CUSUM process, defined by

$$D_n(k) = \frac{k(n-k)}{n^{3/2}} \left(\hat{\theta}_{1,k} - \hat{\theta}_{k+1,n}\right), \ k \in \{1, 2, \cdots, n-1\}, \tag{3}$$

where for any $1 \leq a < b \leq n$, $\hat{\theta}_{a,b} = \theta(\hat{F}_{a,b})$ estimates the model parameter with $\hat{F}_{a,b}$ being the empirical distribution of $\{Y_t\}_{t=a}^b$. For example, when $\theta(\cdot)$ is the mean functional, it can be shown that

$$D_n(k) = \frac{1}{\sqrt{n}} \sum_{t=1}^{k} (Y_t - \bar{Y}), \quad \bar{Y} = n^{-1} \sum_{t=1}^{n} Y_t.$$

The CUSUM process in (3) sequentially compares the estimates before and after a time point $k$, and its norm is expected to attain the maximum when $k = k^*$. Intuitively, if (1) holds, then $D_n(k)$ should fluctuate around zero; otherwise if (2) holds, then $\hat{\theta}_{1,k}$ and $\hat{\theta}_{k+1,n}$ are consistent estimators for $\theta_1$ and $\theta_n$, respectively at $k = k^*$, and the resulting contrast $\|D_n(k^*)\|$ would be most informative about the change signal $\Delta_n = \theta_{k^*+1} - \theta_{k^*}$. Therefore, it is natural to estimate the change-point location via

$$\tilde{k} = \arg \max_{k=1,\cdots,n-1} \|D_n(k)\|^2. \tag{4}$$

However, analyzing the asymptotic distribution of CUSUM process $\{D_n(\lfloor nr \rfloor)\}_{r\in[0,1]}$ for time series data is difficult, as it typically depends on a nuisance parameter called long-run variance (Newey and West, 1987; Andrews, 1991). As mentioned before, the estimation of long-run variance is quite challenging even in a stationary time series, let alone the scenario when a change-point is present.

To bypass this issue, Zhao et al. (2022) propose to estimate the change-point via the self-normalized version of (4), i.e.

$$\hat{k} = \arg \max_{k=1,\cdots,n-1} T_n(k), \quad T_n(k) = D_n(k)' V_n^{-1}(k) D_n(k), \tag{5}$$

where

$$V_n(k) = \sum_{i=1}^{k} \frac{i^2(k-i)^2}{n^2k^2} (\hat{\theta}_{1,i} - \hat{\theta}_{i+1,k})^{\otimes 2} + \sum_{i=k+1}^{n} \frac{(n-i+1)^2(i-k-1)^2}{n^2(n-k)^2} (\hat{\theta}_{i,n} - \hat{\theta}_{k+1,i-1})^{\otimes 2}, \tag{6}$$

is defined as the self-normalizer of $D_n(k)$ with $a^{\otimes 2} = aa^\top$ for a vector $a$. The self-normalizer $V_n(k)$ is proportional to long run variance, which gets canceled out in the limiting null distribution of $T_n(k)$. Thus, the testing/estimation of a single change-point is completely free of tuning parameters.

In practice, we may not know whether the series $\{\theta_t\}_{t=1}^n$ contains a change-point or not, so a testing step is called for prior to the estimation step. Formally speaking, given a pre-specified threshold $K_n$, we declare the existence of a change-point when

$$SN_n := \max_{k=1,\cdots,n-1} T_n(k) > K_n,$$

and then estimate the single change-point via (5). Otherwise, if $SN_n$ is below the threshold $K_n$, we declare no change-points.

### 2.2 Multiple Change-Point Estimation

Section 2.2.1 introduces how SNCP works in the single change-point setting. In this section, we further discuss its implementation for multiple change-point estimation. Compared with the single change-point setting, the main difficulty of multiple change-point estimation lies in how to isolate one change-point from another. In SNCP, this is achieved by a nested local-window approach.

We first introduce some notations. Assume there are $m_0 \geq 0$ unknown number of change-points with $k_0 = 0 < k_1 < \cdots < k_{m_0} < n = k_{m_0+1}$ that partition $Y_t$ into $m_0 + 1$ stationary segments with constant quantity of interest $\theta^{(i)}$ in the $i$th segment, for $i = 1, \cdots, m_0 + 1$. In other words,

$$\theta_t = \theta^{(i)}, \ \ k_{i-1} + 1 \leq t \leq k_i, \text{ for } i = 1, \cdots, m_0 + 1.$$

Similar to the single change-point estimation framework, for $1 \leq t_1 < k < t_2 \leq n$, we define an SN based test statistic

$$T_n(t_1, k, t_2) = D_n(t_1, k, t_2)' V_n^{-1}(t_1, k, t_2) D_n(t_1, k, t_2), \tag{7}$$

where $D_n(t_1, k, t_2) = \frac{(k-t_1+1)(t_2-k)}{(t_2-t_1+1)^{3/2}}(\hat{\theta}_{t_1,k} - \hat{\theta}_{k+1,t_2})$, $V_n(t_1, k, t_2) = L_n(t_1, k, t_2) + R_n(t_1, k, t_2)$ and

$$L_n(t_1, k, t_2) = \sum_{i=t_1}^{k} \frac{(i - t_1 + 1)^2 (k - i)^2}{(t_2 - t_1 + 1)^2 (k - t_1 + 1)^2} (\hat{\theta}_{t_1,i} - \hat{\theta}_{i+1,k})^{\otimes 2},$$

$$R_n(t_1, k, t_2) = \sum_{i=k+1}^{t_2} \frac{(t_2 - i + 1)^2 (i - 1 - k)^2}{(t_2 - t_1 + 1)^2 (t_2 - k)^2} (\hat{\theta}_{i,t_2} - \hat{\theta}_{k+1,i-1})^{\otimes 2}.$$

Here $T_n(t_1, k, t_2)$ plays the same role as $T_n(k)$ in (5), except for the fact that it is defined on the subsample $\{Y_t\}_{t=t_1}^{t_2}$. In other words, $T_n(t_1, k, t_2) = T_n(k)$ if $t_1 = 1$ and $t_2 = n$.

We now combine the SN framework with a nested local-window segmentation algorithm in Zhao et al. (2022) for multiple change-point estimation. For each $k$, instead of using the global statistic $T_n(1, k, n)$ which is computed with all observations, we compute a maximal SNCP test statistic based on a collection of nested windows covering $k$. Specifically, we fix a small trimming parameter $\epsilon \in (0, 1/2)$ and define the window size $h = \lfloor n\epsilon \rfloor$. For each $k = h, \cdots, n - h$, we define the nested local-window set $H_{1:n}(k)$ as

$$H_{1:n}(k) = \{(t_1, t_2) | t_1 = k - j_1 h + 1, j_1 = 1, \cdots, \lfloor k/h \rfloor; t_2 = k + j_2 h, j_2 = 1, \cdots, \lfloor (n-k)/h \rfloor.\} \tag{8}$$

Note that for $k < h$ and $k > n - h$, we have $H_{1:n}(k) = \emptyset$. In Figure 1, we plot a graphical illustration of the nested local-windows in $H_{1:n}(k)$, where local windows are constructed by combining every pair of the red left bracket [ and the blue right bracket ].

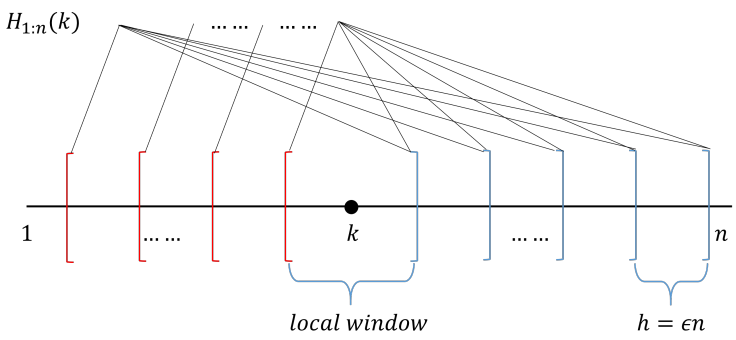


**Figure 1:** Graphical illustration of the nested local-windows in $H_{1:n}(k)$. Each pair of the red left bracket [ and the blue right bracket ] represents a local-window in $H_{1:n}(k)$.

For each $k = 1, \cdots, n-1$, based on its nested local-window set $H_{1:n}(k)$, we define a maximal SN test statistic such that

$$T_{1,n}(k) = \max_{(t_1,t_2)\in H_{1:n}(k)} T_n(t_1, k, t_2), \tag{9}$$

where we set $\max_{(t_1,t_2)\in\emptyset} T_n(t_1, k, t_2) = 0$.

Intuitively, with a sufficiently small trimming $\epsilon$, the nested local-window framework ensures that for a true change-point location, say $k^*$, there exists some local window set denoted by $(t_1^*, t_2^*)$ containing $k^*$ as the only change-point. In other words, $k^*$ is isolated by the interval $(t_1^*, t_2^*)$ so that the procedure in the single change-point scenario as Section 2.2.1 can be applied. This suggests that for at least one pair of $(t_1, t_2)$ in $H_{1:n}(k)$, $T_n(t_1, k^*, t_2)$ is large. The detection power is further enhanced by taking the maximum of these test statistics.

Based on the maximal test statistic $T_{1,n}(k)$ and a pre-specified threshold $K_n$, SNCP proceeds as follows. Starting with the full sample $\{Y_t\}_{t=1}^n$, we calculate $T_{1,n}(k), k = 1, \cdots, n$. Given that $\max_{k=1,\ldots,n} T_n(k) \le K_n$, SNCP declares no change-point. Otherwise, SNCP sets $\widehat{k} = \arg\max_{k=1,\ldots,n} T_{1,n}(k)$ and we recursively perform SNCP on the subsample $\{Y_t\}_{t=1}^{\widehat{k}}$ and $\{Y_t\}_{t=\widehat{k}+1}^n$ until no change-point is declared. Denote $W_{s,e} = \{(t_1, t_2) | s \le t_1 < t_2 \le e\}$ and $H_{s:e}(k) = H_{1:n}(k) \bigcap W_{s,e}$, which is the nested window set of $k$ on the subsample $\{Y_t\}_{t=s}^e$. Define the subsample maximal SN test statistic as $T_{s,e}(k) = \max_{(t_1,t_2)\in H_{s:e}(k)} T_n(t_1, k, t_2)$. Algorithm 1 states the formal description of SNCP in multiple change-points estimation.

We note that SNCP shares some similarity with binary segmentation (BS) in the sense that both algorithms search for change-points in a *sequential* fashion. However, they are also quite different. In particular, the SN test statistic in SNCP is computed over a set of nested local-windows instead of over a single interval. In contrast, in the classical change-point literature, BS is usually coupled with a global CUSUM statistic computed over the entire dataset $[1, n]$. As is documented in the literature (Shao, 2010), the main drawback of BS with CUSUM statistic is its power loss under non-monotonic change, which is caused by the use of the global CUSUM statistic coupled with the sequential search. However, due to the use of the nested local-window based SN test statistic, SNCP does not suffer from this power loss phenomenon as long as $\epsilon$ is chosen to be smaller than the minimum spacing between two change-points. On the other hand, due to the sequential search nature, both SNCP and BS may encounter the multiple testing problem. In addition, their power may be lesser compared to a global search algorithm, such as dynamic programming or PELT, which is again due to the sequential nature of the search algorithm.

As pointed out by a referee, another way of interpreting our method is to view SNCP as the test statistic and BS as the search algorithm. Therefore, the use of BS is not necessarily problematic when there are multiple change-points in the data and it depends on what test statistics BS is combined with. This points to a potentially interesting research direction, which is to develop locally adaptive test statistic that can accommodate multiple change-points and combine it with BS.

**Algorithm 1:** SNCP procedures for multiple change-point estimation

**Input:** Time series $\{Y_t\}_{t=1}^n$, threshold $K_n$, window size $h = \lfloor n\epsilon \rfloor$
**Output:** Estimated change-points $\widehat{\mathbf{k}} = (\hat{k}_1, \cdots, \hat{k}_{\hat{m}})$
**Initialization:** SNCP$(1, n, K_n, h)$, $\widehat{\mathbf{k}} = \emptyset$
**Procedure:** SNCP$(s, e, K_n, h)$
**if** $e - s + 1 < 2h$ **then**
  stop
**else**
  $\hat{k}^* = \arg\max_{k=s,\cdots,e} T_{s,e}(k)$;
  **if** $T_{s,e}(\hat{k}^*) \le K_n$ **then**
    stop
  **else**
    $\widehat{\mathbf{k}} = \widehat{\mathbf{k}} \cup \hat{k}^*$;
    run SNCP$(s, \hat{k}^*, K_n, h)$ and SNCP$(\hat{k}^* + 1, e, K_n, h)$;
  **end**
**end**

### 2.3 Multiple Change-Point Estimation for High-Dimensional Mean

In this section, we modify the SNCP framework in Section 2.2.2 to design a new algorithm called SNHD for multiple change-point estimation in the mean vector of a high-dimensional time series.

Different from the subsample test statistic $T_n(t_1, k, t_2)$ used in SNCP for a low-dimensional time series, SNHD is designed based on the high-dimensional U-statistic proposed by Wang et al. (2022). Given a $p$-dimensional time series $\{Y_t\}_{t=1}^n$, we define the subsample contrast statistic as

$$D_n^U(t_1, k, t_2) = \sum_{\substack{t_1 \le j_1, j_3 \le k \\ j_1 \ne j_3}} \sum_{\substack{k+1 \le j_2, j_4 \le t_2 \\ j_2 \ne j_4}} (Y_{j_1} - Y_{j_2})^\top (Y_{j_3} - Y_{j_4}). \tag{10}$$

Note that $D_n^U(t_1, k, t_2)$ is a two-sample U-statistic estimating the squared $L_2$-norm of the difference between the means of $\{Y_t\}_{t=t_1}^k$ and $\{Y_t\}_{t=k+1}^{t_2}$ (up to some normalizing constant), and therefore targets dense changes in high-dimensional mean. The statistic in (10) is only applicable to high-dimensional time series with temporal independence, and in the presence of temporal dependence, a trimming parameter needs to be introduced to alleviate the bias due to serial dependence; see Wang et al. (2022). Define the self-normalizer as

$$V_n^U(t_1, k, t_2) = \frac{1}{n}\Big[\sum_{t=t_1+1}^{k-2} D_n^U(t_1, t, k)^2 + \sum_{t=k+2}^{t_2-2} D_n^U(k+1, t, t_2)^2\Big]. \tag{11}$$

The subsample SNHD test statistic at time point $k$, in the same spirit as (9), is defined as

$$T_{1,n}^U(k) = \max_{(t_1,t_2)\in H_{1:n}(k)} T_n^U(t_1, k, t_2), \quad T_n^U(t_1, k, t_2) = D_n^U(t_1, k, t_2)^2 / V_n^U(t_1, k, t_2), \tag{12}$$

where $H_{1:n}(k)$ is the nested local-window set defined in (8). With a pre-specified threshold $K_n^U$, a change-point is detected at $\hat{k} = \arg\max_{k=1,\cdots,n} T_{1,n}^U(k)$ if $\max_{k=1,\cdots,n} T_{1,n}^U(k)$ is above $K_n^U$. For multiple change-point estimation, SNHD proceeds similarly as SNCP in Algorithm 1.

### 2.4 Choice of Trimming Parameter $\epsilon$ and Threshold $K_n$

For practical implementation of SNCP and SNHD, there are still two tuning parameters that one has to choose, namely the trimming parameter $\epsilon$ and the change-point detection threshold $K_n$. The choice of $\epsilon$ reflects one's belief of the minimum (relative) spacing between two consecutive change-points. This is usually set to be a small constant such as 0.05, 0.10, 0.15. The theoretical validity of our approach requires the minimum spacing between change-points to be of order $O(n)$, and opting for an overly small value of $\epsilon$ may result in sub-optimal performance in finite sample as the nested local-window may not contain sufficient observations. On the other hand, an overly large value of $\epsilon$ may increase the potential risk of under-estimating change-points if $\epsilon$ is larger than the minimum spacing between two true change-points. In practice, we recommend using 0.05 as a default value when no prior knowledge of minimum spacing between change-points is available.

A nice feature of using SN is that the limiting distributions for SNCP or SNHD under the no change-point scenario are pivotal and furthermore reflect the impact of the choice of $\epsilon$, see Theorem 3.1 in Zhao et al. (2022), and Section S.2.9 in Zhao et al. (2021), respectively. Since $K_n$ and $K_n^U$ are used to balance one's tolerance of type-I and type-II errors, this implies that we can choose $K_n$ and $K_n^U$ as the $q \times 100\%$ quantiles (i.e. the critical value) of the limiting null distribution with $q$ typically set as 0.9, 0.95, 0.99. The threshold value $K_n$ also increases with the dimension of the quantity $\theta$, and we refer to Table 1 in Zhao et al. (2022) for details. In the **SNSeg** package, we offer users a wide range of $\epsilon$ and $q$ to choose from. Details are given in the following section.

In Section 2.4.1, we further conduct a sensitivity analysis, which suggests that the performance of SNCP in general is robust w.r.t. the choice of $(\epsilon, K_n)$.

## 3 The SNSeg Package

In this section, we introduce the functions within the **SNSeg** package for multiple change-point estimation. In particular, `SNSeg_Uni()` in Section 2.3.1 implements the SNCP procedure of change-point estimation for a univariate time series with changes in a single or multiple parameters, such as mean, variance, auto-correlations, quantiles or even their combinations. It can also be implemented for detecting change-points in other quantities, with a user-defined function as input. The function `SNSeg_Multi()` in Section 2.3.2 utilizes the SNCP algorithm for change-point estimation in mean or covariance matrix of a multivariate time series. In Section 2.3.3, the function `SNSeg_HD()` estimates

change-points in mean of a high-dimensional time series using the SNHD procedure. In addition to these major functions for change-point estimation, we further introduce `max_SNsweep()` in Section 2.3.4, which helps obtain the SN test statistics and create a segmentation plot based on the output of the above functions. Followed by the graphical options, the function `SNSeg_estimate()` generates parameter estimates within each segment separated by the estimated change-points.

### 3.1 SNCP for Univariate Time Series

For a univariate time series, change-point estimation in a single or multiple functionals can be implemented through the function `SNSeg_Uni()`. This function is also capable of detecting change-points associated with the change in correlation between bivariate time series. The **R** code is given as:

```
SNSeg_Uni(ts, paras_to_test, confidence = 0.9, grid_size_scale = 0.05,
          grid_size = NULL, plot_SN = TRUE, est_cp_loc = TRUE)
```

It takes the following input arguments.

- `ts`: Input time series $\{Y_t\}_{t=1}^n$, i.e., a univariate time series expressed as a numeric vector with length $n$. However, when the argument `paras_to_test` is specified as `bivcor`, which stands for the correlation between bivariate time series, the input `ts` must be an $n \times 2$ matrix.
- `paras_to_test`: The parameters that SNCP aims to examine, which are presented as a string, a number, a combination of both, or a user-defined function that defines a specific functional. Available options of `paras_to_test` include:
  - `"mean"`: The function performs change-point estimation on the mean of the time series.
  - `"variance"`: The function performs change-point estimation on the variance.
  - `"acf"`: The function performs change-point estimation on the autocorrelation of order 1.
  - `"bivcor"`: The function performs change-point estimation on the bivariate correlation.
  - A numeric quantile value within the range (0,1): The function performs change-point estimation on the quantile level specified by the numeric value.
  - A vector containing characters `"mean"`, `"variance"`, `"acf"`, and one or more numerical quantile levels. Therefore, `SNSeg_Uni()` is capable of estimating change-points in either a single parameter or a combination of multiple parameters.
  - A user-defined `R` function that returns a numeric value. Existing functions in **R** such as `mean()` and `var()` can also be used. This option provides additional flexibility for the users to define a specific functional that they are interested in and is not covered by our built-in options. The input argument `paras_to_test` should possess the form of `function(ts){...}`.
- `confidence`: A numeric value that specifies the confidence level of the SN test. Available choices of confidence levels contain `0.9`, `0.95`, `0.99`, `0.995` and `0.999`. It automatically obtains the threshold ($K_n$, the critical value) corresponding to the input confidence level. The default value of `confidence` is set at 0.9.
- `grid_size_scale`: A numeric value that specifies the trimming parameter $\epsilon$ and only in use if `grid_size = NULL`. Available choices include 0.05, 0.06, 0.07, 0.08, 0.09, 0.1, 0.11, 0.12, 0.13, 0.14, 0.15, 0.2, 0.25, 0.3, 0.35, 0.4, 0.45 and 0.5. The default value of `grid_size_scale` is 0.05.
  - In the function, any input less than 0.05 will be set to exactly 0.05 and similarly, any input greater than 0.5 will be set to 0.5. In such case, a warning that "Detected the grid_size_scale is greater than 0.5" or "less than 0.05" will be generated.
- `grid_size`: A numeric value that specifies the local window size $h$. It should be noted that $h = \lfloor n \times \epsilon \rfloor$, i.e., `grid_size`$= \lfloor n \times$ `grid_size_scale`$\rfloor$. By default, the value of `grid_size` is set to NULL, and the function computes the critical value $K_n$ using the argument `grid_size_scale`. However, users have the option to set the `grid_size` manually, in which case the function computes the corresponding `grid_size_scale` via dividing `grid_size` by $n$, and then determines the critical value using this computed `grid_size_scale` value.
- `plot_SN`: A Boolean value that specifies whether to plot the time series or not. The default setting is TRUE.
- `est_cp_loc`: A Boolean value that specifies whether to plot a red vertical line for each estimated change-point. The default setting is TRUE.

The function `SNSeg_Uni()` provides users with flexibility by allowing them to select parameter types that they want to target at. Additionally, users can specify the window size or choose an appropriate value for $\epsilon$ to achieve the desired theoretical critical value. However, if $\epsilon$ happens to be larger than the true minimum spacing between change-points, the function carries the risk of missing some of them. This limitation also applies to the functions `SNSeg_Multi()` and `SNSeg_HD()`. In practice, we suggest $\epsilon = 0.05$ with no prior knowledge. If the calculated trimming parameter $\epsilon$ by any of the two arguments falls within the range [0.05, 0.5], but is not in the pre-specified set of available values for `grid_size_scale`, the function performs a linear interpolation by identifying two nearest `grid_size_scale` that are below and above the calculated $\epsilon$ and then computes the weighted average of the corresponding critical values $K_n$. The resulting interpolated value is used as the final critical value for the SN test.

When called, `SNSeg_Uni()` returns an S3 object of class `SNSeg_Uni` containing the following entries.

- `ts`: The input time series `ts`.
- `paras_to_test`: The parameter(s) examined in change-point estimation.
- `grid_size`: A numeric value of the local window size $h$.
- `SN_sweep_result`: A list of $n$ matrices where the $k$th matrix stores the SN test statistic $T_n(t_1, k, t_2)$ computed for all $(t_1, t_2) \in H_{1:n}(k)$ as in (9). In particular, the $k$th matrix consists of four columns: 1. the SN test statistic $T_n(t_1, k, t_2)$ computed via (7); 2. the location $k$; 3. the left endpoint $t_1$; and 4. the right endpoint $t_2$.
- `est_cp`: A numeric vector containing the locations of the estimated change-points.
- `confidence`: The confidence level of the SN test.
- `critical_value`: The critical value of the SN test given $\epsilon$ and the confidence level.

It is worth noting that the output of the function `SNSeg_Uni()` can serve as an input of the function `max_SNsweep()` (to be described in Section 2.3.4) to generate a segmentation plot for SN test statistics, and the same also holds for functions `SNSeg_Multi()` and `SNSeg_HD()`. Additionally, S3 objects of class `SNSeg_Uni` are supported by `print()`, `summary()` and `plot()` functions. The S3 function `print()` can be used to display the estimated change-points, `summary()` presents information such as change-point locations and other details listed in the output of `SNSeg_Uni()`, and `plot()` facilitates the generation of time series segmentation plots, providing an alternative option to the argument `plot_SN = TRUE` for users. These functions can also be applied to outputs of functions `SNSeg_Multi()` and `SNSeg_HD()`, which will be introduced below.

To illustrate, in the following, we present examples demonstrating multiple change-point estimation in both single and multiple parameters.

**Example 1: variance change in univariate time series**

We start with the example of the change-point model (V1) in Section S.2.5 in the supplement of Zhao et al. (2022), where two variance changes occur at $k = 400$ and 750, respectively. Specifically,

$$(\text{V1}): \quad Y_t = \begin{cases} 0.5Y_{t-1} + \epsilon_t, & t \in [1, 400], \\ 0.5Y_{t-1} + 2\epsilon_t, & t \in [401, 750], \\ 0.5Y_{t-1} + \epsilon_t, & t \in [751, 1024], \end{cases}$$

where $\epsilon_t$ is a sequence of i.i.d. $N(0, 1)$ random variables.

We set `grid_size_scale` at $\epsilon = 0.05$, which corresponds to a `grid_size` of $\lfloor 0.05 * 1024 \rfloor = 51$, and set `confidence` at 90%. Subsequently, we visualize the input time series by setting `plot_SN` as `TRUE`, and generate an SN test statistics segmentation plot using the `max_SNsweep()` function (to be introduced in Section 2.3.4).

```
# Generate model (V1)
set.seed(7)
ts <- MAR_Variance(reptime = 1, type = "V1") # generate model (V1)
par(mfcol = c(2, 1), mar = c(4, 2.5, 2.5, 0.5))

# SNCP in the change of variance
result1 <- SNSeg_Uni(ts, paras_to_test = "variance", confidence = 0.9,
                     grid_size_scale = 0.05, grid_size = NULL, plot_SN = TRUE,
                     est_cp_loc = TRUE)

# Segmentation plot for SN-based test statistics
SNstat1 <- max_SNsweep(result1, plot_SN = TRUE, est_cp_loc = TRUE, critical_loc = TRUE)
```

The estimated locations of the change-points $\widehat{\mathbf{k}}$, the local window size $h$, and the critical value $K_n$ used can be accessed using the following commands:

```
result1$est_cp
[1] 411 748
result1$grid_size
[1] 51
result1$critical_value
[1] 141.8941
```

The estimated change-point locations are $\hat{k} = 411$ and 748 with a local window size of $h = 51$ and critical value of $K_n = 141.8941$ when setting the trimming parameter to $\epsilon = 0.05$ and the confidence level to 90%. It is clear that the estimated change-points align closely with the true change-points, which demonstrates the accuracy of the SNCP procedure. The above outputs can also be obtained via the S3 methods `summary()` and `print()`, and are shown in the following commands:

```
# S3 method: print
print(result1)
#> The detected change-point location(s) are 411,748
# S3 method: summary
summary(result1)
#> There are 2 change-points detected at 90th confidence level based on the change in
#> the single variance parameter.
#>
#> The critical value of SN-based test is 141.8941189
#>
#> The detected change-point location(s) are 411,748 with a grid_size of 51
```

Figure 2 displays segmentation plots for the input time series and the SN test statistics regarding the changes in univariate variance. It reveals that the SN statistics associated with the detected change-points surpass the critical value, and can be deemed as plausible changes. The upper plot (SN segmentation plot of the time series) can also be achieved through the command `plot(result1)`.

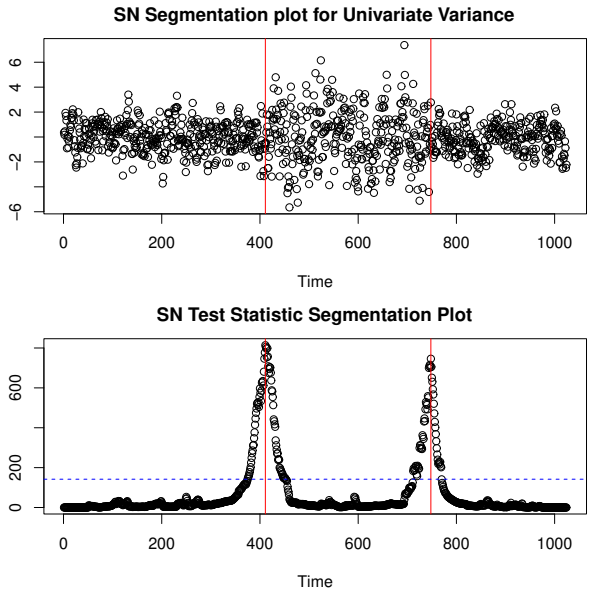


**Figure 2:** An example of simulated time series of model (V1). The upper panel illustrates the detection of change-points for the input time series, and the lower panel displays the segmentation of the SN test statistics using the estimated change-points. The detected change-point locations are indicated by a red vertical line and the critical value is represented by a blue horizontal line.

In addition to the summary statistics and SN based segmentation plots, the function `SNSeg_estimate()` provides parameter estimates within each segment that is separated by the estimated change-points. To illustrate this, we apply the following command to the same example.

```
SNSeg_estimate(SN_result = result1)
```

```
# output
$variance
[1] 1.438164 5.065029 1.341965

attr(,"class")
[1] "SNSeg_estimate"
```

We can also manually specify the value of `grid_size` to calculate the critical value $K_n$ and estimate change-points. The function `SNSeg_Uni()` is applied to the same time series, with the only difference being that the window size $h$ is set to 102, which corresponds to `grid_size_scale=0.1`, instead of `NULL`. We further set `confidence` at 90%. The estimated change-points and the critical value $K_n$ can be obtained using the following commands:

```
# SNCP in the change of variance with a different grid_size and confidence level
result2 <- SNSeg_Uni(ts, paras_to_test = "variance", confidence = 0.9,
                     grid_size_scale = 0.05, grid_size = 102, plot_SN = FALSE,
                     est_cp_loc = FALSE)
result2$est_cp
[1] 411 744
result2$grid_size
[1] 102
result2$critical_value
[1] 111.1472
```

Note that since `grid_size` is not `NULL`, the argument `grid_size_scale = 0.05` will be ignored by the function `SNSeg_Uni()`. Interestingly, though we use quite different window size $h = 102$, the estimated change-points are almost the same as before, which suggests the robustness of SNCP. The critical value differs from the previous example, due to the variation in the window size $h$ (or equivalently $\epsilon$). In other words,the threshold $K_n$ used in SNCP reflects the influence of the chosen window size $h$ (or $\epsilon$), which makes the change-point detection more robust and accurate.

**Example 2: second moment change in univariate time series with a user-defined function**

In addition to the built-in parameter choices, the function `SNSeg_Uni()` allows users to customize the input parameter using their own function.

For instance, if users are interested in examining changes in the second moment, they can create a function that yields the mean square as a numeric value and designate this function to the input argument for `paras_to_test`.

To illustrate, we consider the model (V1) from **Example 1**. The function `SNSeg_Uni()` is utilized with the default input configuration, except that we now assess the change in the second moment of the data. The specified `paras_to_test`, along with the execution time and the resultant estimated change-points, can be acquired using the following commands:

```
# define a function for paras_to_test
# change in 2nd moment
second_moment <- function(ts){
  result <- mean(ts^2)
  return(result)
}
start.time <- Sys.time()
result.general <- SNSeg_Uni(ts, paras_to_test = second_moment, confidence = 0.9,
                            grid_size_scale = 0.05, grid_size = NULL,
                            plot_SN = FALSE, est_cp_loc = TRUE)
end.time <- Sys.time()
as.numeric(difftime(end.time,start.time)) # execution time (in minutes)
result.general$est_cp # change-point estimates

# Output
> as.numeric(difftime(end.time,start.time)) # execution time (in minutes)
[1] 1.083779
> result.general$est_cp # change-point estimates
[1] 411 749
```

As evident from the above results, SNCP detected two change-points at $\hat{k} = 411, 749$ when examining changes in the second moment. The estimated change-points are close to the locations of the true change-point locations, 400 and 750, respectively.

To illustrate the computational efficiency of the built-in choices of `paras_to_test` (i.e., `"mean"`, `"variance"`, etc.) another example is given to detect changes in variance but using the user-defined functional `var()`, which is then compared with the built-in option `paras_to_test = "variance"` in terms of the execution time. The comparison result can be accessed using the following commands:

```
start.time <- Sys.time()
result1 <- SNSeg_Uni(ts, paras_to_test = "variance", confidence = 0.9,
                     grid_size_scale = 0.05, grid_size = NULL, plot_SN = TRUE,
                     est_cp_loc = TRUE)
end.time <- Sys.time()
difftime(end.time,start.time) # built-in parameter time

# user defined variance
paras_to_test <- function(ts){
  var(ts)
}
start.time <- Sys.time()
result.general <- SNSeg_Uni(ts, paras_to_test = paras_to_test, confidence = 0.9,
                            grid_size_scale = 0.05, grid_size = NULL,
                            plot_SN = FALSE, est_cp_loc = TRUE)
end.time <- Sys.time()
difftime(end.time,start.time) # general functional parameter time
c(result1$est_cp,result.general$est_cp)

# output
> difftime(end.time,start.time) # built-in parameter time
Time difference of 4.702668 secs
> difftime(end.time,start.time) # general functional parameter time
Time difference of 1.154525 mins
> result1$est_cp # built-in parameter estimate
[1] 411 748
> result.general$est_cp # general functional estimate
[1] 411 748
```

Both methods can accurately estimate change-points, but utilizing the built-in parameter significantly reduces computation time compared to using user-defined function. The former method optimizes efficiency by leveraging the linear structure of variance calculation and is implemented via dynamic programming with the `cumsum()` function. In contrast, the latter method, employing a user-defined function, does not utilize the linear structure of variance (since it takes a general functional as an input which may not have a specific structure) and instead recursively calculates all subsample variances, leading to redundant calculations and increased computational time.

**Example 3: multiple-parameter change in univariate time series**

In addition to identifying changes in a single parameter, `SNSeg_Uni()` also allows for estimating change-points by simultaneously combining information across multiple parameters. This can be done by modifying the `paras_to_test` argument. For example, users can specify `paras_to_test = c("mean", "acf", 0.6, 0.9)` to simultaneously detect changes in mean, autocorrelation, 60% and 90% quantile of the input time series.

We consider the simulated univariate time series of model (MP1) in Section 4.4 of Zhao et al. (2022), where the true change-points are located at $k = 333$ and 667. In particular,

$$(\text{MP1}): Y_t = \begin{cases} X_t, & t \in [1, 333] \\ F^{-1}\left(\Phi\left(X_t\right)\right), & t \in [334, 667] \\ X_t, & t \in [668, 1000], \end{cases}$$

where $\{X_t\}_{t=1}^n$ follows an AR(1) process with $X_t = 0.2X_{t-1} + \sqrt{1-\rho^2}\epsilon_t$, $\epsilon_t$ is a sequence of i.i.d. $N(0,1)$ random variables, $\Phi(\cdot)$ denotes the CDF of $N(0,1)$, and $F(\cdot)$ denotes a mixture of a truncated normal and a generalized Pareto distribution (GPD). In particular, $F(x) = 0.5F_1(x) + 0.5F_2(x)$, where $F_1(x) = 2\Phi(x), x \leq 0$ is a standard normal distribution truncated at 0 and $F_2(x) = 1 - (1 + \xi(x - \mu)/\sigma)_+^{-1/\xi}$ is a GPD distribution with the location parameter $\mu = 0$, scale parameter $\sigma = 2$ and tail index $\xi = 0.125$. Note that $F^{-1}(q) = \Phi^{-1}(q)$ for $q \leq 0.5$ and $F^{-1}(q) \neq \Phi^{-1}(q)$ for $q > 0.5$.

To showcase the versatility of SNCP, we first detect change-points based on the 90% quantiles, where we set the `grid_size_scale` at 0.1 and `confidence` at 0.9.

```
set.seed(7)
```

```
require(truncnorm)
require(evd)
mix_GauGPD <- function(u, p, trunc_r, gpd_scale, gpd_shape) {
    # function for generating a mixture of truncated normal + GPD
    indicator <- (u < p)
    rv <- rep(0, length(u))
    rv[indicator > 0] <- qtruncnorm(u[indicator > 0] / p, a = -Inf, b = trunc_r)
    rv[indicator <= 0] <- qgpd((u[indicator <= 0] - p) / (1 - p), loc = trunc_r,
                               scale = gpd_scale, shape = gpd_shape)
    return(rv)
}

# Generate model (MP1)
n <- 1000
cp_sets <- c(0, 333, 667, 1000)
rho <- 0.2
ts <- MAR(n, 1, rho) * sqrt(1 - rho ^ 2) # generate AR(1)
trunc_r <- 0
p <- pnorm(trunc_r)
gpd_scale <- 2
gpd_shape <- 0.125
ts[(cp_sets[2] + 1):cp_sets[3]] <-
    mix_GauGPD(u = pnorm(ts[(cp_sets[2] + 1):cp_sets[3]]), p, trunc_r, gpd_scale, gpd_shape)

# SNCP in the change of 90% quantile
result_q9 <- SNSeg_Uni(ts, paras_to_test = c(0.9), confidence = 0.9,
                       grid_size_scale = 0.1, plot_SN = FALSE, est_cp_loc = FALSE)
# Output
result_q9$est_cp
[1] 332 667
result_q9$grid_size
[1] 100
result_q9$critical_value
[1] 110.9993
```

As observed, the estimated change-points take place at $\hat{k} = 332$ and 667, which are close to the locations of the true change-points. We can further use SNCP to examine if there is any change in the variance for the same time series.

```
# SNCP in the change of variance
result_v <- SNSeg_Uni(ts, paras_to_test = c('variance'), confidence = 0.9,
                      grid_size_scale = 0.1, plot_SN = FALSE, est_cp_loc = FALSE)
# Output
result_v$est_cp
[1] 329 665
result_v$grid_size
[1] 100
result_v$critical_value
[1] 110.9993
```

The estimated change-points in variance take place at $\hat{k} = 329$ and 665, which are close to the estimated change-points in the 90% quantile. To reconcile the two sets of estimated change-points, we can further examine changes in variance the 90% quantile simultaneously using SNCP, which gives a final estimate of 331 and 667.

```
# SNCP in the change of variance and 90% quantile
result_q9v <- SNSeg_Uni(ts, paras_to_test = c(0.9, 'variance'), confidence = 0.9,
                        grid_size_scale = 0.1, plot_SN = TRUE, est_cp_loc = TRUE)
# Output
result_q9v$est_cp
[1] 331 667
result_q9v$grid_size
[1] 100
result_q9v$critical_value
[1] 167.4226
```

For practitioners, how to further identify which component(s) in "paras_to_test" changes is an interesting question. A natural strategy is as follows. For each detected change-point $\hat{\tau}$, we first construct a local window centered around it, e.g. $[\hat{\tau} - \epsilon n, \hat{\tau} + \epsilon n]$, where the local window should only contain a single change-point (with high probability). Within the local window, we then apply `SNSeg_Uni()` for each parameter in "`paras_to_test`" and test if it changes.

### 3.2 SNCP for Multivariate Time Series

The SN based change-point estimation for multivariate time series can be implemented via the function `SNSeg_Multi()`. In particular, SNSeg_Multi() allows change-point detection in multivariate means or covariance matrix of the input time series. The **R** code is given as:

```
SNSeg_Multi(ts, paras_to_test = "mean", confidence = 0.9, grid_size = NULL,
            grid_size_scale = 0.05, plot_SN = FALSE, est_cp_loc = TRUE)
```

The input arguments of `confidence`, `grid_size`, `grid_size_scale` and `est_cp_loc` are the same as those in the function `SNSeg_Uni()`, and the difference lies in `ts`, `plot_SN` and `paras_to_test`.

- `ts`: Input time series $\{Y_t = (Y_{t1}, \cdots, Y_{tp})\}_{t=1}^{n}$ as a matrix, i.e., a multivariate time series represented as a matrix with $n$ rows and $p$ columns, where each column is a univariate time series. The dimension $p$ for `ts` should be at least 2.
- `paras_to_test`: A string that specifies the parameter that SNCP aims to examine. Available options of `paras_to_test` include:
  - `"mean"`: The function performs change-point estimation on the mean of the multivariate time series.
  - `"covariance"`: The function performs change-point estimation on the covariance matrix of the multivariate time series.
- `plot_SN`: A Boolean value that specifies whether to generate time series segmentation plot or not. `SNSeg_Multi` returns a plot for each individual time series if `plot_SN = TRUE`.

When necessary, the function `SNSeg_Multi()` applies the same linear interpolation rule as `SNSeg_Uni()` to determine the critical value for the SN test. When called, `SNSeg_Multi()` returns an S3 object of class `SNSeg_Multi` comprising the `grid_size`, `SN_sweep_result`, `est_cp`, `confidence` and `critical_value`, which have already been described in the context of the function `SNSeg_Uni()`. It also generates plots for each time series when `plot_SN = TRUE`. Similar to `SNSeg_Uni`, S3 objects of class `SNSeg_Multi` are also supported by `print()`, `summary()` and `plot()` functions.

**Example 4: mean change in multivariate time series**

We consider model (M2) in Section 4.2 of Zhao et al. (2022), which is generated by

$$(\text{M2}) : Y_t = \begin{cases} -3/\sqrt{5} + X_t, & t \in [1, 75], [526, 575], \\ 0 + X_t, & t \in [76, 375], [426, 525], [576, 1000], \\ 3/\sqrt{5} + X_t, & t \in [376, 425]. \end{cases}$$

where $X_t$ is a 5-dimensional VAR(1) process with $X_t = 0.5X_{t-1} + \epsilon_t$, and $\epsilon_t$ is a sequence of i.i.d. $N(0, \mathbf{I}_5)$ random vectors.

The five true change-points occur at $k = 75, 375, 425, 525$ and 575. We analyze it by examining the change in multivariate means using `grid_size_scale` at 0.05 and `confidence` at 0.9. The code implementation is as follows:

```
# Generate model (M2)
set.seed(7)
d <- 5
n <- 1000
cp_sets <- c(0, 75, 375, 425, 525, 575, 1000)
mean_shift <- c(-3, 0, 3, 0, -3, 0) / sqrt(d)
rho_sets <- 0.5
sigma_cross <- list(diag(d))
ts <- MAR_MTS_Covariance(n, 1, rho_sets, cp_sets = c(0, n), sigma_cross)[[1]] # generate VAR(1)
no_seg <- length(cp_sets) - 1

for (index in 1:no_seg) { # Mean shift
    tau1 <- cp_sets[index] + 1
    tau2 <- cp_sets[index + 1]
```

```
    ts[, tau1:tau2] <- ts[, tau1:tau2] + mean_shift[index]
}

par(mfrow=c(2,3))
# SNCP in the change of multivariate mean
result_multimean <- SNSeg_Multi(ts, paras_to_test = "mean", confidence = 0.9,
                                grid_size_scale = 0.05, plot_SN = TRUE,
                                est_cp_loc = TRUE)

# Output
result_multimean$est_cp
[1] 80 373 423 526 576
result_multimean$grid_size
[1] 50
result_multimean$critical_value
[1] 415.8649
```

The estimated change-points are $\hat{k} = 80, 373, 423, 526$ and 576 with a window size $h = 50$ and a critical value of 415.8649. This result again closely aligns with the true change-point locations. The output of `SNSeg_Multi()` also allows for the use of function `max_SNsweep()` for plotting the segmentation of the SN test statistics. The code implementation is as follows:

```
SNstat_multimean <- max_SNsweep(result_multimean, plot_SN = TRUE, est_cp_loc = TRUE,
                                critical_loc = TRUE)
plot(ts[1, ], main = 'SN Segmentation Plot for the First Time Series')
abline(v = result_multimean$est_cp, col = 'red')
```

Figure 3 plots the associated SN test statistics and estimated change-points. For illustration, we also plot the first time series $\{Y_{t,1}\}_{t=1}^{n}$ along with the estimated change-points.

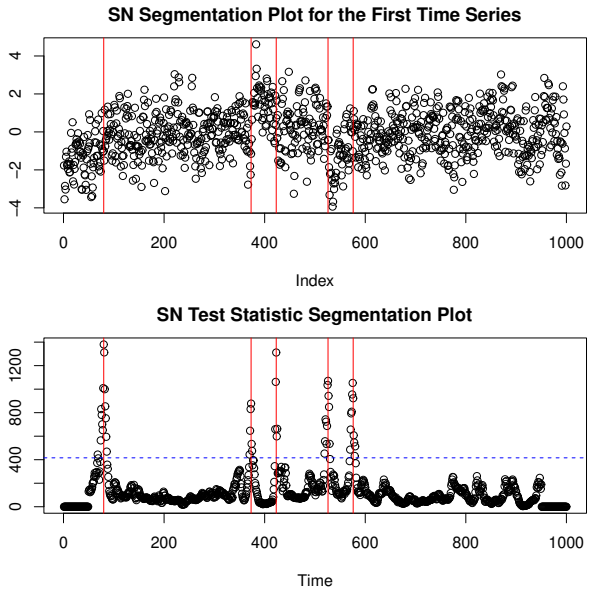


**Figure 3:** The segmentation of the SN test statistics using the estimated change-points. The detected change-point locations are indicated by a red vertical line and the critical value is represented by a blue horizontal line.

### 3.3 SNHD for High-Dimensional Time Series

The function `SNSeg_HD()` is specifically designed to estimate change-points in the mean functional of high-dimensional time series. The **R** code is given as:

```
SNSeg_HD(ts, confidence = 0.9, grid_size_scale = 0.05, grid_size = NULL,
         plot_SN = FALSE, est_cp_loc = TRUE, ts_index = c(1:5)
```

Its input arguments are the same as the function `SNSeg_Multi()` except for the followings:

- `ts`: The dimension of the input time series `ts` should be at least 10 to ensure a decent finite sample performance of the asymptotic theory.
- `plot_SN`: A Boolean value that specifies whether to return a plot for individual time series.
- `ts_index`: A positive integer or a vector of positive integers that specifies which individual time series to plot given `plot_SN = TRUE`. The default value is `c(1:5)`, and under the default setting, the function will plot the first 5 time series.

When called, `SNSeg_HD()` returns an S3 object of class `SNSeg_HD` containing `grid_size`, `SN_sweep_result`, `est_cp`, `confidence` and `critical_value` that are similar to those described by `SNSeg_Uni()` and `SNSeg_Multi()`. It also generates a plot for the time series specified by the argument `ts_index` when `plot_SN = TRUE`. Additionally, S3 objects of class `SNSeg_HD` are supported by `plot()`, `summary()` and `print()` functions. Similar to the core function `SNSeg_HD()`, the `plot()` method incorporates the option `ts_index`, enabling users to visualize the desired time series.

**Example 5: mean change in high-dimensional time series**

We generate high-dimensional time series data based on the following simulation setting:

$$(\mathrm{HD}): Y_t = \mu_i + X_t, \ \ \tau_{i-1} + 1 \le t \le \tau_i, \ \ i = 1, 2, \cdots, 6,$$

where $X_t$ is a sequence of i.i.d. $N(0, I_{100})$ random vectors and the five change-points are evenly located at $(\tau_1, \tau_2, \cdots, \tau_5) = (100, 200, \cdots, 500)$, with $\tau_0 = 0$ and $\tau_6 = 600$. We set $\mu_1 = \mathbf{0}_{100}$, $\theta_i = \mu_{i+1} - \mu_i$, $\theta_i = (-1)^i(\mathbf{1}_5^\top, \mathbf{0}_{95}^\top)^\top \times \sqrt{4/5}$ for $i = 1, 2, \cdots, 5$. We apply `SNSeg_HD()` to analyze this time series with `grid_size_scale` set at 0.05 and `confidence` set at 0.9.

```
# Generate model (HD)
set.seed(7)
p <- 100
n <- 600
cp_sets <- c(0, 100, 200, 300, 400, 500, 600)
mean_shift <- c(0, sqrt(4 / 5), 0, sqrt(4 / 5), 0, sqrt(4 / 5))
ts <- matrix(rnorm(n * p, 0, 1), n, p)
no_seg <- length(cp_sets) - 1
for (index in 1:no_seg) { # Mean shift
    tau1 <- cp_sets[index] + 1
    tau2 <- cp_sets[index + 1]
    ts[tau1:tau2, 1:5] <- ts[tau1:tau2, 1:5] + mean_shift[index]
}

# SNHD for high-dimensional means
par(mfrow=c(2,2))
result_hd <- SNSeg_HD(ts, confidence = 0.9, grid_size_scale = 0.05,
                      plot_SN = TRUE, est_cp_loc = TRUE,
                      ts_index = c(1:4))

# Output
result_hd$est_cp
[1]  105 203 302 397 500
```

Figure 4 plots the first four individual time series and the estimated change-points as requested by the argument `ts_index = c(1:4)`. As observed in the example, `SNSeg_HD()` successfully detects all the change-points in this high-dimensional time series. This result demonstrates the effectiveness and feasibility of using SN algorithms for change-point detection in high-dimensional time series.

### 3.4 Generate the SN Test Statistics

As discussed in Section 2.2, the success of SNCP and SNHD depends on the local SN test statistic $T_{1:n}(k)$ and $T_{1,n}^U(k)$ for $k = 1, \cdots, n$. To facilitate further analysis, the function `maxSNsweep()` allows the users to compute and plot these test statistics along with the identified change-points. The **R** code is given as:

```
max_SNsweep(SN_result, plot_SN = TRUE, est_cp_loc = TRUE, critical_loc = TRUE)
```

It takes the following arguments:

- `SN_result`: A list generated as the output of the functions `SNSeg_Uni()`, `SNSeg_Multi()`, or `SNSeg_HD()`.

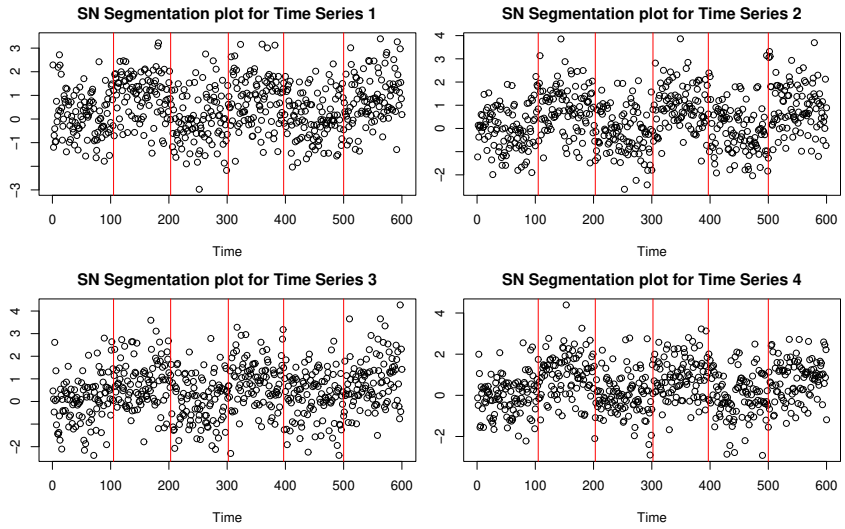


**Figure 4:** An example of estimating changes in high-dimensional mean for model (HD). The detected change-point locations are indicated by a red vertical line for the first 4 time series.

- `plot_SN`: A Boolean value that specifies whether to return an SN test statistics segmentation plot.
- `est_cp_loc`: A Boolean value that specifies whether to plot a red vertical line for each estimated change-point.
- `critical_loc`: A Boolean value that specifies whether to plot a blue horizontal line for the critical value $K_n$ or $K_n^U$ used in the SN test.

When called, `max_SNsweep()` returns the maximal SN test statistic, namely $T_{1,n}(k)$ or $T_{1,n}^U(k)$, for each time point $k$. In addition, it can provide a segmentation plot based on these SN test statistics. Users are able to determine whether to mark the change-point locations and the critical value on the plot.

As an illustration of `max_SNsweep()`, suppose we apply `SNSeg_Uni()` to estimate change-points and save the output as `result1`. The following code can be used to generate the SN test statistic $T_{1:n}(k)$ for each $k = 1, \cdots, n$ and in addition the segmentation plot, which is already given in the lower panel of Figure 2.

```
SNstat1 <- max_SNsweep(result1, plot_SN = TRUE, est_cp_loc = TRUE, critical_loc = TRUE)
```

As delineated in Section 2.3.1, 2.3.2 and 2.3.3, functions `SNSeg_Uni()`, `SNSeg_Multi()`, and `SNSeg_HD()` serve as the foundation for the `SNSeg_estimate()` function, which facilitates the computation of parameter estimates for individual segments separated by the identified change-points. When called, `SNSeg_estimate()` returns an S3 object of class `"SNSeg_estimate"` containing the parameter estimate of each segment. The **R** code is given as:

```
SNSeg_estimate(SN_result)
```

It takes the following argument:

- `SN_result`: an S3 object with class `"SNSeg_Uni"`, `"SNSeg_Multi"` or `"SNSeg_HD"`. The input of `SN_result` must be the output from one of the functions in `SNSeg_Uni()`, `SNSeg_Multi()` and `SNSeg_HD()`.

We refer back to **Example 1** for an illustration of its use.

## 4 Additional Numerical Results

This section provides additional numerical results of **SNSeg**. In Section 2.4.1, we conduct a sensitivity analysis of `SNSeg_Uni()` across various input parameters. Section 2.4.2 compares with other popular change-point estimation packages. In Section 2.4.3. we demonstrate the usefulness of employing multiple parameters and contrasts it with detecting changes in a single parameter. We also provide brief explanations and recommendations on the selection of quantiles.

In this section, we measure the accuracy of change-point estimation by counting the difference between the number of estimated change-points and true values $\hat{m} - m_o$, the Hausdorff distance

$d_H$, and adjusted Rand index (ARI). The Hausdorff distance is defined as follows. Denote the set of true change-points as $\tau_o$ and the set of estimated change-points as $\hat{\tau}$, we define $d_1(\tau_o, \hat{\tau}) = \max_{\tau_1 \in \hat{\tau}} \min_{\tau_2 \in \tau_o} |\tau_1 - \tau_2|$ and $d_2(\tau_o, \hat{\tau}) = \max_{\tau_1 \in \tau_o} \min_{\tau_2 \in \hat{\tau}} |\tau_1 - \tau_2|$, where $d_1(\tau_o, \hat{\tau})$ measures the over-segmentation error of $\hat{\tau}$ and $d_2(\tau_o, \hat{\tau})$ measures the under-segmentation error of $\hat{\tau}$. The Hausdorff distance is $d_H(\tau_o, \hat{\tau}) = \max\{d_1(\tau_o, \hat{\tau}), d_2(\tau_o, \hat{\tau})\}$. The ARI is originally proposed in Morey and Agresti (1984) as a measure of similarity between two different partitions of the same observations for evaluating the accuracy of clustering. Under the change-point setting, we calculate the ARI between partitions of the time series given by $\hat{\tau}$ and $\tau_o$. Ranging from 0 to 1, a higher ARI indicates more coherence between the two partitions by $\hat{\tau}$ and $\tau_o$ and thus more accurate change-point estimation. We further note that all numerical results in this section are implemented on a laptop with 1.7 GHz 12th Gen Intel Core i7 CPU and 64 GB of RAM.

### 4.1 Sensitivity analysis of SNCP

We first examine the performance variations resulted from choices of the trimming parameter $\epsilon$ and threshold $K_n$ (reflected by the confidence level $q$) when multiple change-points exist. Specifically, we generate the data according to model (SA):

$$(\text{SA}) : n = 1200, \rho = 0.5, Y_t = \begin{cases} 0 + X_t, & t \in [1, 150], [301, 450], [601, 750], [901, 1050] \\ \delta + X_t, & t \in [151, 300], [451, 600], [751, 900], [1051, 1200], \end{cases}$$

where $\{X_t\}_{t=1}^n$ is generated from a unit-variance AR(1) process with $X_t = X_{t-1}/2 + \sqrt{3}\epsilon_t/2$, and $\{\epsilon_t\}$ is *i.i.d.* $N(0, 1)$. We vary $\delta \in \{\sqrt{3}, \sqrt{6}\}$ to compare the results under low and high signal-to-noise ratios.

**Table 1:** Sensitivity analysis of $\epsilon$ and $q$ for time series model (SA) with $\delta \in \{\sqrt{3}, \sqrt{6}\}$. $\delta$: magnitude of change in (SA); $q$: confidence level of SNCP; $\epsilon$: trimming parameter (the value of `grid_size_scale`); $\hat{m} - m_o$: the difference between the estimated number and the true number of change-points; *ARI*: Average Adjusted Rand Index; $d_1$: Average over-segmentation error; $d_2$: Average under-segmentation error; $d_H$: Average Hausdorff distance. The average time is presented in units of seconds. The optimal result is bolded for each comparison metric.

| | | $\hat{m} - m_o$ | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $\delta$ | $(q, \epsilon)$ | $\leq -3$ | $-2$ | $-1$ | 0 | 1 | 2 | $\geq 3$ | ARI | $d_1$ | $d_2$ | $d_H$ | time(s) |
| $\sqrt{3}$ | 0.90, 0.05 | 0 | 1 | 89 | 882 | 27 | 1 | 0 | **0.933** | 1.12 | 2.18 | 2.18 | 1.59 |
| | 0.95, 0.05 | 0 | 26 | 175 | 788 | 11 | 0 | 0 | 0.918 | 1.01 | 3.38 | 3.38 | 1.59 |
| | 0.90, 0.08 | 0 | 33 | 195 | 772 | 0 | 0 | 0 | 0.911 | 0.97 | 3.65 | 3.65 | 0.59 |
| | 0.95, 0.08 | 16 | 85 | 302 | 597 | 0 | 0 | 0 | 0.880 | 0.92 | 5.92 | 5.92 | 0.59 |
| | 0.90, 0.10 | 0 | 4 | 58 | **938** | 0 | 0 | 0 | 0.931 | 1.02 | **1.75** | **1.75** | 0.36 |
| | 0.95, 0.10 | 0 | 24 | 120 | 856 | 0 | 0 | 0 | 0.919 | 0.99 | 2.74 | 2.74 | 0.36 |
| | 0.90, 0.12 | 0 | 4 | 130 | 866 | 0 | 0 | 0 | **0.933** | 0.77 | 2.17 | 2.17 | 0.24 |
| | 0.95, 0.12 | 1 | 14 | 159 | 826 | 0 | 0 | 0 | 0.926 | 0.76 | 2.69 | 2.69 | 0.24 |
| | 0.90, 0.15 | 1000 | 0 | 0 | 0 | 0 | 0 | 0 | 0.002 | **0.00** | 49.98 | 49.98 | **0.13** |
| | 0.95, 0.15 | 1000 | 0 | 0 | 0 | 0 | 0 | 0 | 0.001 | **0.00** | 50.01 | 50.01 | **0.13** |
| $\sqrt{6}$ | 0.90, 0.05 | 0 | 0 | 4 | 964 | 31 | 1 | 0 | **0.965** | 0.77 | **0.60** | 0.77 | 1.60 |
| | 0.95, 0.05 | 0 | 0 | 11 | 968 | 21 | 0 | 0 | **0.965** | 0.66 | 0.79 | 0.79 | 1.60 |
| | 0.90, 0.08 | 0 | 0 | 16 | 984 | 0 | 0 | 0 | 0.963 | 0.58 | 0.77 | 0.77 | 0.67 |
| | 0.95, 0.08 | 0 | 2 | 48 | 950 | 0 | 0 | 0 | 0.958 | 0.57 | 1.17 | 1.17 | 0.67 |
| | 0.90, 0.10 | 0 | 0 | 2 | **998** | 0 | 0 | 0 | 0.961 | 0.62 | 0.65 | **0.65** | 0.37 |
| | 0.95, 0.10 | 0 | 0 | 8 | 992 | 0 | 0 | 0 | 0.961 | 0.62 | 0.72 | 0.72 | 0.37 |
| | 0.90, 0.12 | 0 | 0 | 34 | 966 | 0 | 0 | 0 | 0.958 | 0.59 | 0.95 | 0.95 | 0.25 |
| | 0.95, 0.12 | 0 | 0 | 34 | 966 | 0 | 0 | 0 | 0.958 | 0.59 | 0.95 | 0.95 | 0.25 |
| | 0.90, 0.15 | 1000 | 0 | 0 | 0 | 0 | 0 | 0 | 0.000 | **0.00** | 50.01 | 50.01 | **0.16** |
| | 0.95, 0.15 | 1000 | 0 | 0 | 0 | 0 | 0 | 0 | 0.000 | **0.00** | 50.01 | 50.01 | **0.16** |

We vary $\epsilon \in \{0.05, 0.08, 0.10, 0.12, 0.15\}$ and $q \in \{0.90, 0.95\}$, and study how they affect the performance of SNCP. The numerical result over 1000 replications is summarized in Table 1 for reader's convenience. From the table, we find that as long as the window size $\epsilon$ is smaller than the minimum spacing $\epsilon_o = 0.125$, the performance of SNCP is quite robust and stable across the choices of $\epsilon$ and $q$. However, SNCP fails to detect changes with the window size $\epsilon = 0.15 > \epsilon_o$, highlighting the importance of selecting an appropriate value for $\epsilon$. Furthermore, we find that the execution time increases with diminishing values of $\epsilon$, while no discernible disparity in execution time is found across

various thresholds.

We also briefly study the execution time of SNCP using `SNSeg_Uni()` across multiple model parameters including mean, variance, autocorrelation (ACF), 90% quantile, and a multi-parameter scenario with both variance and 90% quantile. For model (V1) from **Example 1** and (MP1) from **Example 3**, Table 2 presents the averaged execution time over 100 replications of `SBSeg_Uni()`. The execution time of SNCP varies in the order of mean, variance, ACF, and quantile, progressing from the lowest to the highest. Notably, the multi-parameter scenario requires a longer runtime compared to the single-parameter cases.

**Table 2:** Execution time (in seconds) of SNCP for different parameters when applied to the models (V1) and (MP1) averaged over 100 replications.

| Model | mean | variance | ACF | quantile | multi-parameter |
|---|---|---|---|---|---|
| V1 | 1.75 | 5.86 | 8.83 | 17.80 | 31.83 |
| MP1 | 1.68 | 5.82 | 8.52 | 16.56 | 31.06 |

### 4.2 Comparison: SNCP vs BinSeg, PELT, MOSUM and ECP

We next compare SNCP with BinSeg, PELT, MOSUM and ECP in terms of the accuracy of change-point estimates, especially when data exhibits temporal dependence. BinSeg and PELT are implemented by the package **changepoint** (the functions `cpt.mean()` and `cpt.var()`, respectively), MOSUM utilizes the package **mosum** (the function `mosum()`), and ECP adopts the package **ecp** (the function `e.divisive()`). For SNCP, we set the trimming parameter $\epsilon = 0.05$ and confidence level $q = 0.9$, adhering to its default configuration. The input parameters for other methods are also chosen as default values. In particular, the default thresholds for MOSUM and ECP are based on critical values of asymptotic null distribuions under confidence level 0.95; while that for BinSeg and PELT are non-asymptotic. For the bandwidth parameter `G` that requires manual selection in the function `mosum()` for MOSUM, we let `G=100`. This choice aligns with the recommendation in Section 3.5 from Eichinger and Kirch (2018) that `G` should be half the minimal distance between two change-points. In the case of model (M) below, this minimum distance is 200. The details can be found in the Appendix.

We first compare the performance of all methods under the no change-point scenario, where the time series is stationary with no change-point. We simulate a stationary univariate time series $\{Y_t\}_{t=1}^{n=1000}$ from a unit-variance AR(1) process $Y_t = \rho Y_{t-1} + \sqrt{1-\rho^2}\epsilon_t$ where $\{\epsilon_t\}$ is *i.i.d.* $N(0,1)$. We vary $\rho \in \{0, 0.4, 0.7\}$ to investigate the robustness of SNCP (and other methods) against false positives (i.e. type-I error) under different levels of temporal dependence.

The experiment is repeated 1000 times for each $\rho$, and the results are documented in Table 3. In general, under their respective default settings, all methods provide satisfactory type-I error control when there is no temporal dependence ($\rho = 0$) whereas MOSUM and ECP are prone to produce false positives when dependence is moderate ($\rho = 0.4$). Under strong temporal dependence ($\rho = 0.7$), all tests exhibit high false-positive rates and SNCP is the most robust option.

**Table 3:** Number of change-points detected by each method when there is no change-point. A higher value of $\rho$ indicates a stronger temporal dependence. The optimal method is bolded for each $\rho$.

| $n = 1000$ | $\rho = 0$ | | | $\rho = 0.4$ | | | $\rho = 0.7$ | | |
|---|---|---|---|---|---|---|---|---|---|
| $\hat{m}$ | 0 | 1 | $\geq 2$ | 0 | 1 | $\geq 2$ | 0 | 1 | $\geq 2$ |
| SNCP | 910 | 82 | 8 | 884 | 111 | 5 | **744** | 210 | 46 |
| BinSeg | **1000** | 0 | 0 | **963** | 35 | 2 | 558 | 240 | 202 |
| PELT | **1000** | 0 | 0 | 943 | 30 | 27 | 152 | 66 | 782 |
| MOSUM | 954 | 43 | 3 | 292 | 330 | 378 | 7 | 16 | 977 |
| ECP | 952 | 24 | 24 | 145 | 72 | 783 | 0 | 0 | 1000 |

We further examine their power performance under model (M):

$$(\mathrm{M}): n = 1000, \rho \in \{0, 0.4, 0.7\}, Y_t = \begin{cases} X_t, & t \in [1, 200], [401, 600], [801, 1000] \\ 2 + X_t, & t \in [201, 400], [601, 800]. \end{cases}$$

Here $\{X_t\}_{t=1}^{n}$ is generated from a unit-variance AR(1) process that $X_t = \rho X_{t-1} + \sqrt{1-\rho^2}\epsilon_t$, and $\{\epsilon_t\}$ is *i.i.d.* $N(0,1)$. The true change-points occur at 200, 400, 600 and 800. Table 4 summarizes the results

over 1000 replications.

From the table, we observe that in the absence of temporal dependence ($\rho = 0$), all the methods perform well, with PELT being the most effective. It should be noted that, due to the use of self-normalizer, SNCP may experience some decrease in estimation accuracy compared to other methods. When the dependence is moderate at $\rho = 0.4$, SNCP demonstrates robust performance, while other competing methods tend to overestimate the number of change-points. With stronger dependence ($\rho = 0.7$), SNCP exhibits the best performance based on the distribution of $\hat{m} - m_o$ along with $d_H$ and ARI, while all the other methods severely overestimate the number of change-points. In terms of the computational speed, we find that BinSeg, PELT, and MOSUM are more efficient than SNCP. Consequently, we recommend employing SNCP for change-point estimation when data exhibit moderate or strong dependence, while opting for BinSeg or PELT in cases with no or weak dependence.

Here, we only compare the results under mean shifts, and we refer the interested readers to Zhao et al. (2022) for results in other settings. Broadly speaking, our findings indicate that the SNCP exhibits greater robustness to temporal dependence compared to competing methods. However, it's worth noting that other methods might demonstrate superior performance in instances where temporal dependence is weak. For instance, BinSeg, PELT, and MOSUM are adept at handling frequent change-points with fast computational speed.

We note that the unsatisfactory performance associated with BinSeg, PELT, MOSUM and ECP in the presence of strong temporal dependence is to be expected since these methods are developed for time series with temporal independence. To accommodate temporal dependence, some of these above-mentioned methods offer choices to modify their built-in penalty, which may avoid the over-segmentation and under-segmentation issues with some proper choice of tuning parameters. In addition, we further acknowledge that there exist several R packages such as **AR1seg** (Levy Leduc, 2014), **EnvCpt** (Killick et al., 2021) or **fastcpd** (Li and Zhang, 2024), which contain change-point detection methods allowing for temporal dependence.

**Table 4:** Performance of different methods for the time series model (M). $\rho$: The strength of temporal dependence; $\hat{m} - m_o$: the difference between the estimated number and the true number of change-points; $ARI$: Average Adjusted Rand Index; $d_1$: Average over-segmentation error; $d_2$: Average under-segmentation error; $d_H$: Average Hausdorff distance. The average time is presented in units of seconds. The optimal result is bolded for each comparison metric.

| | | $\hat{m} - m_o$ | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $\rho$ | Method | $\leq -3$ | $-2$ | $-1$ | 0 | 1 | 2 | $\geq 3$ | ARI | $d_1$ | $d_2$ | $d_H$ | time (s) |
| $\rho = 0$ | SNCP | 0 | 0 | 0 | 991 | 9 | 2 | 0 | 0.983 | 4.13 | 3.42 | 4.13 | 6.85 |
| | BinSeg | 0 | 0 | 0 | 961 | 39 | 0 | 0 | 0.988 | 3.83 | 3.26 | 3.83 | **0.01** |
| | **PELT** | 0 | 0 | 0 | **999** | 1 | 0 | 0 | **0.994** | **1.73** | **1.69** | **1.73** | 0.11 |
| | MOSUM | 0 | 0 | 0 | 993 | 7 | 0 | 0 | 0.992 | 3.09 | 2.09 | 3.09 | 0.00 |
| | ECP | 0 | 0 | 0 | 937 | 56 | 7 | 0 | 0.984 | 7.04 | 2.24 | 7.04 | 46.85 |
| $\rho = 0.4$ | **SNCP** | 0 | 0 | 0 | **972** | 27 | 1 | 0 | 0.956 | **8.10** | 5.73 | **8.10** | 5.78 |
| | BinSeg | 0 | 0 | 0 | 744 | 256 | 0 | 0 | 0.963 | 15.16 | 5.60 | 15.16 | **0.00** |
| | PELT | 0 | 0 | 0 | 862 | 109 | 29 | 0 | **0.964** | 14.03 | **3.81** | 14.03 | 0.05 |
| | MOSUM | 0 | 0 | 0 | 779 | 199 | 20 | 2 | 0.959 | 38.19 | 4.81 | 38.19 | **0.00** |
| | ECP | 0 | 0 | 0 | 170 | 184 | 217 | 429 | 0.828 | 87.05 | 4.14 | 87.05 | 54.62 |
| $\rho = 0.7$ | **SNCP** | 0 | 2 | 59 | **865** | 69 | 4 | 1 | **0.934** | **17.83** | 23.69 | **29.74** | 6.01 |
| | BinSeg | 0 | 0 | 0 | 201 | 799 | 0 | 0 | 0.930 | 55.11 | 11.30 | 55.11 | **0.00** |
| | PELT | 0 | 0 | 0 | 104 | 162 | 195 | 539 | 0.867 | 99.50 | 8.95 | 99.50 | 0.06 |
| | MOSUM | 0 | 0 | 0 | 209 | 414 | 281 | 96 | 0.909 | 126.3 | 11.06 | 126.3 | **0.00** |
| | ECP | 0 | 0 | 0 | 1 | 0 | 1 | 998 | 0.537 | 146.4 | **8.31** | 146.4 | 92.2 |

### 4.3 Single vs Multiple Parameters

As outlined in Section 2.3.1, the function `SNSeg_Uni()` enables users to examine changes in either a single parameter or multiple parameters. We first provide a simple example which shows that using multiple parameters may not necessarily be significantly inferior to using a single parameter, in terms of change-point estimates. This observation holds true even when the change solely stems from the single parameter. For example, we consider model (M) with $\rho = 0.4$ from Section 2.4.2, which is solely driven by mean changes.

Table 5 summarizes the numerical results over 1000 replications. For clarity, we specify these cases using names beginning with "SN". For instance, SNM denotes SNCP for estimating changes in a single mean, $\text{SNMQ}_{20}$ represents SNCP for estimating changes in both mean and the 20% quantile,

**Table 5:** Performance of SNCP based on a single parameter and multiple parameters for (M). Beginning with "SN", M, V, $Q_{20}$ represent mean, variance and 20% quantile, respectively. $\hat{m} - m_o$: the difference between the estimated number and the true number of change-points; $ARI$: Average Adjusted Rand Index; $d_1$: Average over-segmentation error; $d_2$: Average under-segmentation error; $d_H$: Average Hausdorff distance. The average time is presented in units of seconds. The optimal result is bolded for each comparison metric.

| | | $\hat{m} - m_o$ | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Model | Method | $\leq -3$ | $-2$ | $-1$ | 0 | 1 | 2 | $\geq 3$ | ARI | $d_1$ | $d_2$ | $d_H$ | time (s) |
| (M) | **SNM** | 0 | 0 | 0 | **972** | 28 | 0 | 0 | **0.969** | **8.54** | **6.40** | **8.54** | **1.49** |
| | $SNMQ_{20}$ | 0 | 0 | 3 | 958 | 39 | 0 | 0 | 0.967 | 9.71 | 7.33 | 10.25 | 28.6 |
| | SNMV | 0 | 0 | 3 | 934 | 62 | 1 | 0 | 0.963 | 12.53 | 8.06 | 13.11 | 18.2 |

and SNMV targets the mean and variance changes simultaneously. From the table, we find that all three methods yield rather similar results, albeit mild overestimation by SNMV. This indicates that introducing additional parameters does not necessarily hinder the performance of SNCP.

We then provide another example that examining multiple parameters can outperform examining a single parameter. Specifically, we compare the performance of SNCP based on a single variance or quantile (90% or 95%) and their multi-parameter combination under the setting (MP1) from **Example 3**. Recall that for (MP1), the change originates from the upper quantiles and the actual change-points take place at 333 and 667. The numerical result of (MP1) over 1000 replications is taken from Table 5 of Zhao et al. (2022), and summarized in Table 6 here for readers' convenience. Similar to Table 5, we specify the parameter settings using names beginning with "SN". For instance, SNV denotes the change in a single variance, $SNQ_{90}$ represents the change in the 90% quantile, and $SNQ_{90}V$ targets the variance and 90% quantile changes simultaneously. We observe that for (MP1), $SNQ_{90}$ and $SNQ_{95}$ performs well with a high estimation accuracy since the change of (MP1) originates from upper quantiles. By integrating changes in variance and quantiles, improvements are observed across estimation accuracy, ARI, and Hausdorff distance $d_H$. Notably, the combined-parameter setting $SNQ_{90,95}V$ achieves the optimal performance compared to all the other parameter configurations.

**Table 6:** Performance of SNCP based on the change in a single parameter and multiple parameters for (MP1). $Q_{90}, Q_{95}, V$ represent the change in the 90th and the 95th quantile as well as the variance respectively. $\hat{m} - m_o$: the difference between the estimated number and the true number of change-points; $ARI$: Average Adjusted Rand Index; $d_1$: Average over-segmentation error; $d_2$: Average under-segmentation error; $d_H$: Average Hausdorff distance. The optimal result is bolded for each comparison metric.

| | | $\hat{m} - m_o$ | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Model | Method | $\leq -3$ | $-2$ | $-1$ | 0 | 1 | 2 | $\geq 3$ | ARI | $d_1$ | $d_2$ | $d_H$ |
| (MP1) | $SNQ_{90}$ | 0 | 10 | 132 | 805 | 50 | 3 | 0 | 0.839 | 3.25 | 7.26 | 7.85 |
| | $SNQ_{95}$ | 0 | 5 | 100 | 820 | 73 | 2 | 0 | 0.868 | 3.16 | 5.70 | 6.62 |
| | SNV | 0 | 2 | 110 | 832 | 54 | 2 | 0 | 0.869 | 2.45 | 5.47 | 6.06 |
| | $SNQ_{90,95}$ | 0 | 3 | 82 | 850 | 62 | 3 | 0 | 0.878 | 3.01 | 4.88 | 5.67 |
| | $SNQ_{90}V$ | 0 | 0 | 56 | 869 | 70 | 5 | 0 | 0.891 | 3.04 | 3.95 | 4.77 |
| | $SNQ_{95}V$ | 0 | 2 | 64 | 861 | 68 | 5 | 0 | 0.889 | **2.92** | 4.30 | 5.14 |
| | $\mathbf{SNQ_{90,95}V}$ | 0 | 2 | 48 | **882** | 66 | 2 | 0 | **0.894** | 2.95 | **3.79** | **4.58** |

Overall, our findings illustrate that employing multiple parameters does not always diminish performance of SNCP compared to using a single parameter, even when the change is primarily driven by a single parameter. Nevertheless, it is important to recognize that incorporating prior information on change types can enhance the effectiveness of SNCP.

Another aspect that is of interest is the choice of quantile for SNCP. As delineated in Section 2.3.1, the `SNSeg_Uni()` function provides users with the capability to assess variations in either a single or multiple quantiles. Particularly for practitioners, an appropriate selection of the quantile becomes pivotal when the true quantile that may change remains unknown. Table 6 for model (MP1) also offers valuable insights in this regard. Given that (MP1) experiences changes in upper quantiles, the usage of the 90% or the 95% quantile yields satisfactory results. Furthermore, the application of both 90% and 95% quantiles in combination results in an improvement compared to utilizing a single quantile.

Consequently, in cases where the specific quantile that is changing is unknown, we recommend users visually inspect their time series for signals such as peaks or troughs to assess the potential range of quantiles where changes might occur, and further employ multiple quantiles within this range

for more robust change-point estimation. In other words, if one knows that the change happens in a specific range of the distribution (for example, the upper tail), we recommend he/she target several quantiles in this range (for example, targeting 90%, 95%) simultaneously, instead of picking only one quantile. In practice, it is seldom that only a particular quantile changes, while the other quantile levels near this quantile exhibit no change. Hence, testing several quantiles together can boost power and estimation accuracy to the best degree.

## 5 Summary

In this paper, we introduced the **R** package **SNSeg**, which provides implementations of the SN-based procedures for change-points estimation in univariate, multivariate, and high-dimensional time series. We described the main functions of the package, namely `SNSeg_Uni()`, `SNSeg_Multi()`, `SNSeg_HD()`, which enable the detection of change-points in a single or multiple parameter(s) of the time series. Furthermore, we presented examples demonstrating the usage of the package, including visualizing both the time series data and the segmentation plots of the SN test statistics, as well as the computation of parameter estimates within the segments that are separated by the estimated change-points.

The **SNSeg** package offers a comprehensive set of tools to effectively identify change-points in time series data. We hope the availability of **SNSeg** on CRAN can help facilitate the analysis and understanding of temporal patterns and dynamics for both researchers and practitioners.

## References

D. W. Andrews. Heteroskedasticity and autocorrelation consistent covariance matrix estimation. *Econometrica*, pages 817–858, 1991. [p47]

I. E. Auger and C. E. Lawrence. Algorithms for the optimal identification of segment neighborhoods. *Bulletin of mathematical biology*, 51(1):39–54, 1989. [p46]

B. Eichinger and C. Kirch. A mosum procedure for the estimation of multiple random change points. *Bernoulli*, 24(1):526–564, 2018. [p46, 62]

N. A. James and D. S. Matteson. ecp: An R package for nonparametric multiple change point analysis of multivariate data. *Journal of Statistical Software*, 62(7):1–25, 2014. [p46]

R. Killick and I. A. Eckley. changepoint: An R package for changepoint analysis. *Journal of Statistical Software*, 58(3):1–19, 2014. [p46]

R. Killick, P. Fearnhead, and I. A. Eckley. Optimal detection of changepoints with a linear computational cost. *Journal of the American Statistical Association*, 107(500):1590–1598, 2012. [p46]

R. Killick, C. Beaulieu, S. Taylor, and H. Hullait. Package 'envcpt'. 2021. [p63]

C. Levy Leduc. Package r ar1seg (available on the cran). 2014. [p63]

X. Li and X. Zhang. fastcpd: Fast change point detection in r. *arXiv preprint arXiv:2404.05933*, 2024. [p63]

A. Meier, C. Kirch, and H. Cho. mosum: A package for moving sums in change-point analysis. *Journal of Statistical Software*, 97(8):1–42, 2021. [p46]

L. C. Morey and A. Agresti. The measurement of classification agreement: An adjustment to the rand statistic for chance agreement. *Educational and Psychological Measurement*, 44(1):33–37, 1984. [p61]

W. K. Newey and K. D. West. A simple, positive semi-definite, heteroskedasticity and autocorrelation-consistent covariance matrix. *Econometrica*, 55:703–708, 1987. [p47]

G. J. Ross. Parametric and nonparametric sequential change detection in R: The cpm package. *Journal of Statistical Software*, 66(3):1–20, 2015. [p46]

X. Shao. A self-normalized approach to confidence interval construction in time series. *Journal of the Royal Statistical Society: Series B*, 72(3):343–366, 2010. [p46, 49]

X. Shao. Self-normalization for time series: a review of recent developments. *Journal of the American Statistical Association*, 110:1797–1817, 2015. [p46]

X. Shao and X. Zhang. Testing for change points in time series. *Journal of the American Statistical Association*, 105(491):1228–1240, 2010. [p46]

S. Sun, Z. Zhao, F. Jiang, and X. Shao. *SNSeg: Self-Normalization(SN) Based Change-Point Estimation for Time Series*, 2023. URL https://CRAN.R-project.org/package=SNSeg. R package version 1.0.0. [p46]

G. Wang and C. Zou. cpss: an package for change-point detection by sample-splitting methods. *Journal of Quality Technology*, 55:61–74, 2023. [p46]

R. Wang, C. Zhu, S. Volgushev, and X. Shao. Inference for change points in high-dimensional data via selfnormalization. *The Annals of Statistics*, 50(2):781–806, 2022. [p46, 50]

A. Zeileis, F. Leisch, K. Hornik, and C. Kleiber. strucchange: An r package for testing for structural change in linear regression models. *Journal of Statistical Software*, 7(2):1–38, 2002. [p46]

A. Zeileis, C. Kleiber, W. Krämer, and K. Hornik. Testing and dating of structural changes in practice. *Computational Statistics & Data Analysis*, 44(1–2):109–123, 2003. [p46]

Z. Zhao, F. Jiang, and X. Shao. Segmenting time series via self-normalization. *arXiv preprint https://arxiv.org/pdf/2112.05331v1.pdf*, 2021. [p50]

Z. Zhao, F. Jiang, and X. Shao. Segmenting time series via self-normalisation. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, 84(5):1699–1725, 2022. [p46, 47, 48, 50, 52, 55, 57, 63, 64]

C. Zou, G. Wang, and R. Li. Consistent selection of the number of change-points via sample-splitting. *Annals of Statistics*, 48(1):413, 2020. [p46]

*Shubo Sun*
*University of Miami*
*Herbert Business School*
*Coral Gables, FL, USA*
sxs3935@miami.edu

*Zifeng Zhao*
*University of Notre Dame*
*Mendoza College of Business*
*Notre Dame, IN, USA*
zifeng.zhao@nd.edu

*Feiyu Jiang*
*Fudan University*
*School of Management*
*Shanghai, China*
jiangfy@fudan.edu.cn

*Xiaofeng Shao*
*University of Illinois at Urbana-Champaign*
*Department of Statistics*
*Champaign, IL, USA*
xshao@illinois.edu

# fmeffects: An R Package for Forward Marginal Effects

*by Holger, Christian A. Scholbeck, Christian Heumann, Bernd Bischl, and Giuseppe Casalicchio*

**Abstract** Forward marginal effects have recently been introduced as a versatile and effective model-agnostic interpretation method particularly suited for non-linear and non-parametric prediction models. They provide comprehensible model explanations of the form: if we change feature values by a pre-specified step size, what is the change in the predicted outcome? We present the R package fmeffects, the first software implementation of the theory surrounding forward marginal effects. The relevant theoretical background, package functionality and handling, as well as the software design and options for future extensions are discussed in this paper.

## 1 Introduction

A growing number of disciplines are adopting black box machine learning (ML) models to make predictions, including medicine (Rajkomar et al., 2019; Boulesteix et al., 2020), psychology (Dwyer et al., 2018), economics (Mullainathan and Spiess, 2017; Athey and Imbens, 2019), or the earth sciences (Dueben and Bauer, 2018). Although one can often observe a superior predictive performance of black box models (such as neural networks, gradient boosting, random forests, or support vector machines) over intrinsically interpretable models (such as generalized linear or additive models), their lack of transparency or interpretability is considered a major drawback (Breiman, 2001). This has been a major driver in the development of model-agnostic explanation techniques, which are often referred to by the umbrella terms of interpretable ML (Molnar, 2022) or explainable artificial intelligence (Kamath and Liu, 2021).

Marginal effects (MEs) (Williams, 2012) have been a mainstay of model interpretations in many applied fields such as econometrics (Greene, 2019), psychology (McCabe et al., 2022), or medical research (Onukwugha et al., 2015). MEs explain the effect of features on the predicted outcome in terms of derivatives w.r.t. a feature or forward differences in prediction. They are typically averaged to an average marginal effect (AME) for an entire data set, which serves as a global (scalar-valued) feature effect measure (Bartus, 2005). To explain feature effects for non-linear models, Scholbeck et al. (2024) introduced a unified definition of forward marginal effects (FMEs), a non-linearity measure (NLM) for FMEs, and the conditional average marginal effect (cAME). The NLM is an auxiliary model diagnostic to avoid interpreting local changes in prediction as linear effects. The cAME aims to describe the model via regional FME averages for subgroups with similar FMEs, which can, for instance, be found by recursive partitioning (RP). FMEs, therefore, represent a means to explain models on a local, regional, and global level.

**Contributions:** We present the R package **fmeffects**, the first software implementation of the theory surrounding FMEs, including the NLM and the cAME. The user interface only requires a pre-trained model and an evaluation data set. The package is designed according to modular principles, making it simple to maintain and extend. This paper introduces the relevant theoretical background of FMEs, demonstrates the usage of the package in the context of a practical use case, and explains the software design.

## 2 Background on forward marginal effects

FMEs can be used for model explanations on the local, regional (also referred to as semi-global), and global level. These differ with respect to the region of the feature space that the explanation refers to. The local level explains a model/prediction for single observations, the regional level for a certain subspace (or subgroups of observations), and the global level for the entire feature space. Increasing the scope of the explanation requires increasing amounts of aggregations of local explanations (see the illustration by Scholbeck et al. (2020) of aggregations of local explanations to global ones for various methods). This can be problematic for non-parametric models where local explanations can be highly heterogeneous due to non-linear effects or interactions.

### 2.1 Notation

Let $\widehat{f} : \mathcal{X} \to \mathbb{R}$ be the prediction function of a learned model where $\mathcal{X} \subset \mathbb{R}^p$ denotes the feature space. While our definition naturally covers regression models, for classification models, we assume that $\widehat{f}$ returns the score or probability for a predefined class of interest. A subspace of the feature space is denoted by $\mathcal{X}_{[\,]} \subseteq \mathcal{X}$. The random feature vector is denoted by[1] $\boldsymbol{X} = (X_1, \ldots, X_p)$. Observations are denoted by $\boldsymbol{x} = (x_1, \ldots, x_p) \in \mathcal{X}$. A set of feature indices is denoted by $S \subseteq \{1, \ldots, p\}$. We often index (random) vectors as $\boldsymbol{x}_S$ or $\boldsymbol{X}_S$. We denote set complements by $-S = \{1, \ldots, p\} \setminus S$. With slight abuse of notation, we represent the partitioning of a vector into two arbitrary but disjoint groups by $\boldsymbol{x} = (\boldsymbol{x}_S, \boldsymbol{x}_{-S})$, regardless of the order of features. For a single feature of interest, the set $S$ is replaced by an integer index $j$. We usually assume an evaluation data set $\mathcal{D} = \left(\boldsymbol{x}^{(i)}\right)_{i=1}^{n}$, with $\boldsymbol{x}^{(i)} \in \mathcal{X}$, which may consist of both training and test data.

### 2.2 Forward marginal effects

The FME can be considered a basic, local unit of interpretation. Given an observation $\boldsymbol{x}$, it tells us how the prediction changes if we change a subset of feature values $\boldsymbol{x}_S$ by a vector of step sizes $\boldsymbol{h}_S$.

$$\text{FME}_{\boldsymbol{x},\boldsymbol{h}_S} = \widehat{f}(\boldsymbol{x}_S + \boldsymbol{h}_S, \boldsymbol{x}_{-S}) - \widehat{f}(\boldsymbol{x}) \qquad \text{for continuous features } \boldsymbol{x}_S$$

Scholbeck et al. (2024) introduced an observation-specific categorical FME whose definition is congruent with the FME for continuous features. The categorical FME corresponds to the change in prediction when replacing $x_j$ by the reference category $c_j$:

$$\text{FME}_{\boldsymbol{x},c_j} = \widehat{f}(c_j, \boldsymbol{x}_{-j}) - \widehat{f}(\boldsymbol{x}) \qquad \text{for categorical } x_j$$

Note that this definition of a categorical ME differs from the one that is typically found in fields like econometrics (Williams, 2012), where we set $x_j$ to a reference category for all observations and then record the change in prediction resulting from changing the reference category to another category.

Furthermore, it is common to globally average MEs to an average marginal effect (AME) to estimate the expected local effect. For FMEs, this corresponds to:

$$\begin{aligned} \text{AME}_{\mathcal{D},\boldsymbol{h}_S} &= \widehat{\mathbb{E}_{\boldsymbol{X}}\left[\text{FME}_{\boldsymbol{X},\boldsymbol{h}_S}\right]} \\ &= \frac{1}{n}\sum_{i=1}^{n}\left[\widehat{f}\left(\boldsymbol{x}_S^{(i)} + \boldsymbol{h}_S, \boldsymbol{x}_{-S}^{(i)}\right) - \widehat{f}\left(\boldsymbol{x}^{(i)}\right)\right] \end{aligned} \tag{1}$$

Note that for categorical feature changes and observations where $x_j = c_j$, the FME equals 0. In the **fmeffects** package, the categorical AME only consists of observations whose observed feature value differs from the selected category. This approach is motivated by our goal to explain the effects of *changing feature values* on the predicted outcome. For instance, in Fig. 11, we demonstrate the effect of rainfall on the daily number of bike rentals in Washington D.C. by switching each non-rainy day's precipitation status to rainfall. Considering all observations, including rainy days, would obfuscate the interpretation we desire from our model. However, it is important to remember that every AME comprises a different set of points.

### 2.3 Step size selection

The selection of step sizes is determined by contextual and data-related considerations (Scholbeck et al., 2024). First, the FME allows us to investigate the model according to specific research questions. For instance, we might be interested in the effects of a specific change in a patient's body weight on the predicted individual disease risk. Often, we are interested in an interpretable or intuitive step size. In the case of body weight, typically expressed in kilograms, we could use a 1kg change (for instance, instead of 1g) as a natural increment. Without contextual information, we could use a unit change as a reasonable default; or dispersion-based measures such as one standard deviation, percentages of the interquartile range, or the mean/median absolute deviation.

### 2.4 Non-linearity measure

For continuous features, we can consider $\boldsymbol{x}_S + \boldsymbol{h}_S$ a continuous transition of feature values. The associated change in prediction may be misinterpreted as a linear effect. This is counteracted by the

[1] Bold letters denote vectors.

NLM, which corresponds to a continuous coefficient of determination $R^2$ between the prediction function and the linear secant that intersects $\boldsymbol{x}$ and $(\boldsymbol{x}_S + \boldsymbol{h}_S, \boldsymbol{x}_{-S})$ (see Fig. 1). The continuous transition through the feature space is first parameterized as a fraction $t \in [0, 1]$ of the multivariate step size $\boldsymbol{h}_S$:

$$\gamma(t) = \begin{pmatrix} x_1 \\ \vdots \\ x_p \end{pmatrix} + t \cdot \begin{pmatrix} h_1 \\ \vdots \\ h_s \\ 0 \\ \vdots \\ 0 \end{pmatrix} \quad , \quad t \in [0, 1]$$

The value of the linear secant $g_{\boldsymbol{x},\boldsymbol{h}_S}(t)$ corresponds to:

$$g_{\boldsymbol{x},\boldsymbol{h}_S}(t) = \begin{pmatrix} x_1 + t \cdot h_1 \\ \vdots \\ x_s + t \cdot h_s \\ \vdots \\ x_p \\ \widehat{f}(\boldsymbol{x}) + t \cdot \text{FME}_{\boldsymbol{x},\boldsymbol{h}_S} \end{pmatrix}$$

The mean prediction $\widehat{f}_{\text{mean}}$ on the interval $t \in [0, 1]$ is given by:

$$\begin{aligned} \widehat{f}_{\text{mean}} &= \frac{\int_0^1 \widehat{f}(\gamma(t)) \left|\left| \frac{\partial \gamma(t)}{\partial t} \right|\right|_2 dt}{\int_0^1 \left|\left| \frac{\partial \gamma(t)}{\partial t} \right|\right|_2 dt} \\ &= \int_0^1 \widehat{f}(\gamma(t)) \, dt \end{aligned}$$

The NLM compares the squared deviation between the prediction function and the linear secant to the squared deviation between the prediction function and the mean prediction:

$$\text{NLM}_{\boldsymbol{x},\boldsymbol{h}_S} = 1 - \frac{\int_0^1 \left(\widehat{f}(\gamma(t)) - g_{\boldsymbol{x},\boldsymbol{h}_S}(t)\right)^2 \left|\left| \frac{\partial \gamma(t)}{\partial t} \right|\right|_2 dt}{\int_0^1 \left(\widehat{f}(\gamma(t)) - \widehat{f}_{\text{mean}}\right)^2 \left|\left| \frac{\partial \gamma(t)}{\partial t} \right|\right|_2 dt} \quad \in (-\infty, 1]$$

Fig. 2 illustrates the setting for multivariate feature changes. The NLM can be approximated via numerical integration, e.g., via Simpson's rule.

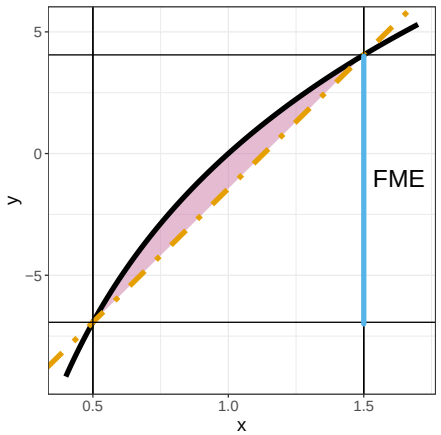


**Figure 1:** Illustration by Scholbeck et al. (2024) of a univariate FME (blue) given the prediction function (black) and linear secant (orange, dashed). The NLM indicates how well the secant can explain the prediction function (inversely proportional to the purple area) compared to how well the most uninformative baseline model (the average prediction) can explain the prediction function.

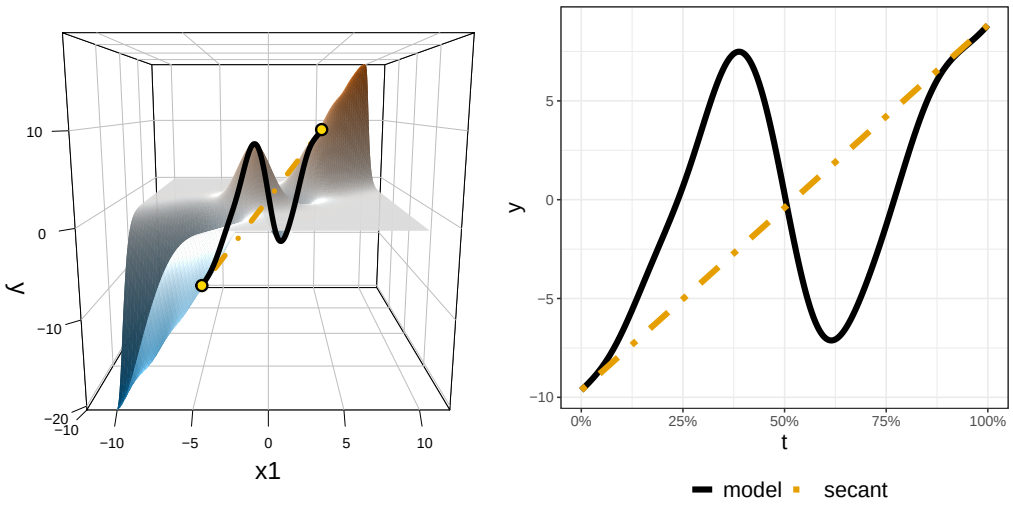


**Figure 2:** Illustration of the multivariate NLM by Scholbeck et al. (2024). **Left:** An exemplary bivariate prediction function and two points to compute an FME. Consider an observation $\boldsymbol{x} = (-5, -5)$ and step size vector $\boldsymbol{h}_S = (10, 10)$. We create the shortest path through the feature space to reach the point (5, 5), which consists of directly proportional changes in both features. Above the path, we see the linear secant (orange, dashed) and the non-linear prediction function (black). **Right:** The multivariate change in feature values can be parameterized as a percentage $t$ of the step size $\boldsymbol{h}_S$. The deviation between the prediction function and the linear secant, as well as the deviation between the prediction function and mean prediction, both correspond to a line integral.

The NLM indicates how well the linear secant can explain the prediction function, compared to the baseline model of using the mean prediction. A value of 1 indicates perfect linearity, where the linear secant is identical to the prediction function. For a value of 0, the mean prediction can explain the prediction function to the same degree as the secant. For negative values, the mean prediction better explains the prediction function than the linear secant (severe non-linearity).

It is, therefore, easiest to interpret FMEs with NLM values close to 1. Although every FME always represents the exact change in prediction, an FME with a low NLM value does not fully describe the behavior of the model in that specific locality. In contrast, an FME with an NLM close to 1 is a sufficient descriptor of the (linear) model behavior. In other words, the NLM serves as an auxiliary diagnostic tool, indicating trust in how well the FME describes the local change in prediction.

### 2.5 Conditional average marginal effect

To receive a global model explanation akin to a beta coefficient in linear models, local FMEs can be averaged to the AME. Mehrabi et al. (2021) define an *aggregation bias* as drawing false conclusions about individuals from observing the entire population. Given a data set $\mathcal{D}$, the conditional average marginal effect (cAME) estimator applies to a subgroup of $n_{[\,]}$ observations, denoted by $\mathcal{D}_{[\,]}$:

$$\begin{aligned} \text{cAME}_{\mathcal{D}_{[\,]},\boldsymbol{h}_S} &= \mathbb{E}_{\widehat{X_{[\,]}}}\left[\text{FME}_{X_{[\,]},\boldsymbol{h}_S}\right] \\ &= \frac{1}{n_{[\,]}} \sum_{i:\boldsymbol{x}^{(i)} \in \mathcal{D}_{[\,]}} \left[\widehat{f}\left(\boldsymbol{x}_S^{(i)} + \boldsymbol{h}_S, \boldsymbol{x}_{-S}^{(i)}\right) - \widehat{f}\left(\boldsymbol{x}^{(i)}\right)\right] \end{aligned} \tag{2}$$

Although this estimator can be applied to arbitrary subgroups, we aim to find subgroups with cAMEs that counteract the aggregation bias. Desiderata for such subgroups include within-group effect homogeneity, between-group effect heterogeneity, full segmentation, non-congruence, confidence, and stability (Scholbeck et al., 2024). In other words, we aim to partition the data into subgroups that explain variability in the FMEs. A viable option to partition $\mathcal{D}$ is to run RP on $\mathcal{D}$ with FMEs as the target. For instance, in **fmeffects**, both **rpart** (Therneau and Atkinson, 2019) and `ctree()` from **partykit** (Hothorn and Zeileis, 2015) are supported to find subgroups.

# 3 Related work

## 3.1 Model-agnostic interpretations

The basic mechanism behind model-agnostic methods is to probe the model with different feature values, a methodology similar to a model sensitivity analysis (Scholbeck et al., 2020, 2023). The basis of explaining models is to determine the direction and magnitude of the effect of features on the predicted outcome (Casalicchio et al., 2019; Scholbeck et al., 2020, 2024). The individual conditional expectation (ICE) (Goldstein et al., 2015), partial dependence (PD) (Friedman, 2001), accumulated local effects (ALE) (Apley and Zhu, 2020), Shapley values (Štrumbelj and Kononenko, 2010; Lundberg and Lee, 2017; Covert et al., 2020) and local interpretable model-agnostic explanations (LIME) (Ribeiro et al., 2016) are some of the most popular model-agnostic explanation methods for ML models. Notably, counterfactual explanations (Wachter et al., 2018) represent the reverse of the FME, indicating the smallest necessary change in feature values to reach a targeted prediction.

FMEs complement the literature by allowing for a unique combination of local, regional, and global model explanations. Furthermore, while most methods (including the ICE, PD, ALE, or Shapley values) provide explanations in terms of prediction *levels*, FMEs provide explanations in terms of prediction *changes*. LIME is based on training a local and interpretable surrogate model whose coefficients can also provide an interpretation in terms of prediction changes. Scholbeck et al. (2024) highlighted differences between both approaches: notably, while surrogate models introduce additional uncertainty connected with the estimation of the surrogate, FMEs are motivated by the goal of stable and comprehensible model insight. Furthermore, locally estimated FMEs can be aggregated within subgroups and entire data sets for regional and global explanations. Around the same time, regional aggregations have also been introduced for ICE curves, for example (Britton, 2019; Herbinger et al., 2022; Molnar et al., 2024).

## 3.2 Relationship between individual conditional expectation and forward marginal effect

Scholbeck et al. (2024) illustrated a relationship between the ICE / PD and the FME / AME. In general, the FME can be seen as the difference between two locations on an ICE. The AME corresponds to the difference between two locations on the PD only for a function that is linear in the feature of interest. Therefore, the following relationship between the ICE and FME is worth noting here. The ICE can be considered a one-way sensitivity function that indicates the effects of varying a set of features indexed by $S$ while the remaining ones are kept constant:

$$\text{ICE}_{\boldsymbol{x},S}(\boldsymbol{x}_S^*) = \widehat{f}(\boldsymbol{x}_S^*, \boldsymbol{x}_{-S})$$

For an instance $\boldsymbol{x}$, the prediction after increasing $\boldsymbol{x}_S$ by $\boldsymbol{h}_S$ is also a value of the ICE:

$$\begin{aligned}\text{FME}_{\boldsymbol{x},\boldsymbol{h}_S} &= \widehat{f}(\boldsymbol{x}_S + \boldsymbol{h}_S, \boldsymbol{x}_{-S}) - \widehat{f}(\boldsymbol{x}) \\ &= \text{ICE}_{\boldsymbol{x},S}(\boldsymbol{x}_S + \boldsymbol{h}_S) - \text{ICE}_{\boldsymbol{x},S}(\boldsymbol{x}_S)\end{aligned}$$

## 3.3 Related work on marginal effects

MEs have a long history in applied statistics and the Stata programming language (StataCorp, 2023). Initially implemented by Bartus (2005), the `margins()` command is now fully integrated into Stata and provides comprehensive capabilities for various computations and visualizations of statistical models such as (generalized) linear models (Williams, 2012). MEs are typically defined in terms of derivatives of the model w.r.t. a feature. For instance, this variant is the default approach to interpret models in econometrics (Greene, 2019). The FME is the less commonly used definition (Scholbeck et al., 2024; Mize et al., 2019). Note that—in contrast to forward differences—derivatives are not suitable to explain piecewise constant prediction functions such as tree-based models.

In recent years, MEs have gained traction in the R community. The R package **margins** (Leeper, 2018) was the first port of Stata's `margins()` command to R. Other packages related to MEs include **ggeffects** (Lüdecke, 2018) and **marginaleffects** (Arel-Bundock, 2023). In particular, **marginaleffects** can also return FMEs (although under different terminology). Our package, **fmeffects**, mainly differs from **marginaleffects** in two aspects:

**Implementing new theory surrounding FMEs:** The **fmeffects** package is the first software implementation of the theory surrounding model-agnostic FMEs as introduced by Scholbeck et al. (2024). Although packages such as **marginaleffects** support the computation of FMEs and

other quantities, **fmeffects** is specifically designed for FMEs with unique features such as implementations of the NLM, the cAME via RP, and novel visualization methods.

**Model-agnostic black box interpretations:** It follows that **fmeffects** is targeted at model-agnostic explanations of non-linear or intransparent models. Whereas existing theory on MEs (and packages such as **marginaleffects**) focuses on classical statistical modeling in combination with statistical inference (see, for instance, Breiman (2001) comparing statistical modeling culture with ML), FMEs (and thus **fmeffects**) are comparable to methods and software from the literature on interpretable ML such as the ICE, PD, ALE, or LIME. This does not imply that **marginaleffects** cannot be used for black box interpretations. As mentioned in the previous point, it also supports the computation of FMEs, e.g., in combination with **mlr3**, but the focus of **fmeffects** lies on the interpretation of black box models through a specialized and targeted range of novel capabilities.

## 4 Advantages and limitations of forward marginal effects

### 4.1 Advantages

Although the ICE and the FME are closely related, the latter provides several novel ways to generate insights into the model:

- **Univariate changes in feature values:** FMEs are comparable to ICE curves for univariate changes in feature values. In certain scenarios, however, they may provide more comprehensible visualizations of effects for individual instances (see Fig. 4 for an example).
- **Bivariate changes in feature values:** The ICE and PD also provide insight into the sensitivity of the model prediction for variations in two features, which is visualized as a heatmap (see Fig. 7). However, it is difficult to visually compare the ICE of many different observations (which correspond to heatmaps as well). Although the ICE provides insight into a larger variation in feature values, while the FME only considers a single tuple of changes in feature values, bivariate FMEs can be easily compared visually (see Fig. 6).
- **Higher-order changes in feature values:** If we evaluate the sensitivity of the prediction for changes in more than two feature values, virtually every visualization method breaks down. In this case, FMEs still provide comprehensible model explanations that can be aggregated in various ways (see Fig. 10).
- **Local fidelity assessment:** The locally restricted change in feature values for the FME facilitates evaluations of the fidelity of the model explanation (e.g., via the NLM). In other words, the NLM allows us to describe how well the FME summarizes the local shape of the prediction function in a single value. See Fig. 8 for a visualization of NLM values for different observations.
- **Comprehensible regional explanations:** Although regional explanations have been first proposed in the context of grouping ICE curves (Herbinger et al., 2022; Britton, 2019), they more easily apply to scalar model explanations such as FMEs. Essentially, a regional model explanation represents a group of observations or a subspace of the feature space where model explanations are relatively homogeneous. Such groupings are easily achievable via RP or other techniques that do not require functional target values such as ICEs.
- **Avoiding extrapolation:** The ICE is computed on the entire feature range (see, e.g., Fig. 4), which is likely to result in model extrapolations. By its nature, the FME is typically used with small step sizes relative to the feature range, which naturally avoids model extrapolations.

### 4.2 Limitations

- **Step size selection:** The step size fundamentally influences effects and the model interpretation. Although FMEs for different step sizes can be computed and visualized in an exploratory manner, some level of prior reasoning about what step sizes to use is recommended.
- **Decision tree instability for cAME:** Although not a shortcoming of the FME itself, subgroups found by RP to compute cAMEs are subject to a high variance. This may be counteracted by stabilizing the split search, e.g., by considering statistical significance of tree splits or resorting to different algorithms to find subgroups.
- **Non-linearity assessment for proportional feature changes:** For multi-dimensional feature changes, the NLM only considers equally proportional changes in all features.

## 5 On causal interpretations and avoiding model extrapolations

Note that model-agnostic techniques, including FMEs, explain associations between the target and the features within the model. In the absence of additional assumptions, such associations cannot be interpreted as causes and effects (Molnar et al., 2022). For instance, increasing the value of a feature $x_1$ may always be accompanied by an increase in the target, but it may be the target $y$ that causes $x_1$ to increase. Another typical scenario is the presence of confounding factors that influence both $y$ and $x_1$. Finally, $x_1$ may only (or also) influence a mediator $x_2$, which in turn influences $y$.

This does not, however, make model interpretations obsolete. More importantly, as highlighted by Adadi and Berrada (2018), model interpretations can be used to gain knowledge, debug, audit, or justify the model and its predictions. Throughout this paper, we will model the effects of environmental influences on the number of daily bike rentals in Washington, D.C. For our estimated model, a drop in humidity by 10 percentage points has a considerable effect on the predicted number of daily bike rentals (see Fig. 5). This effect cannot be assumed to be causal, as humidity is physically influenced by the outside temperature, which will also affect people's choice to rent a bike. Here, temperature is a confounder that influences both humidity and daily bike rentals. However, the business renting out bikes can still use the associations found by a model with a good predictive performance to control the optimal number of bikes at their disposal. This is conditional on the model's ability to accurately predict the target for the given feature vector, requiring us to avoid model extrapolations, which correspond to predictions within areas of the feature space where the model has not seen much or any training data. This issue is closely linked to the multivariate distribution of the training data; in our example, a change in humidity is likely to be accompanied by a change in temperature as well, which we somewhat circumvent (depending on the magnitude of the step size) when making isolated changes to humidity. One may disregard this issue and deliberately predict in areas of the feature space the model has not seen during training. The resulting FMEs will still be valid model descriptions but, as explained above, they are likely to be bad descriptions of the data generating process.

Model extrapolations negatively impact many model-agnostic interpretation methods (Hooker, 2004b,a, 2007; Hooker et al., 2021; Molnar et al., 2022). For example, Apley and Zhu (2020) demonstrated how PD plots suffer from extrapolation issues and introduced ALE plots as a solution to this problem. Scholbeck et al. (2024) illustrated the perils of model extrapolations for FMEs specifically and discussed possible options. One option in particular is also implemented in **fmeffects**: points outside the multivariate envelope (meaning the Cartesian product of all observed feature ranges) of the training data can be excluded from the analysis. This directly relates to the selection of small step sizes relative to the feature range, as large step sizes will result in a point falling outside the envelope.

When using extrapolation prevention methods, note that we consider different sets of points for different step sizes, which differs from the usage of MEs in other contexts (see, for instance, the package **marginaleffects** for a comparison). The exclusion of points only impacts aggregations of FMEs, i.e., the cAME and AME. As discussed in the section on Forward marginal effects, this also affects the computation of categorical AMEs. In Eq. (1) and Eq. (2), the AME and cAME are formulated as estimators of the expected global or regional (concerning a subspace) effects. The fewer observations we are considering for an average, the larger the variance of the estimate.

## 6 User interface and package handling

### 6.1 Local explanations

The `fme()` function is the central user interface. It mainly requires a pre-trained model and a data set (see section Design and options for extensions for details). Further control parameters include a list of features and step sizes, whether to compute NLM values for each FME, and an extrapolation detection method. The `fme()` function initiates the construction and computations of a `ForwardMarginalEffect` object without requiring the user to know **R6** (Chang, 2021) syntax.

For this use case, we train a random forest from the **randomForest** package (Liaw and Wiener, 2002) on the bike sharing data set (Fanaee-T, 2013) using **mlr3**. Note that models trained via **tidymodels** and **caret** are also supported, as well as models trained via `lm()`, `glm()`, and `gam()`. We aim to predict and explain the daily bike rental demand in Washington, D.C., based on features such as the outside temperature, wind speed, or humidity. We first train the model:

```
> library(fmeffects)
> data(bikes, package = "fmeffects")
> library(mlr3verse)
> library(mlr3extralearners)
> forest = lrn("regr.randomForest")
```

```
> task = as_task_regr(x = bikes, id = "bikes", target = "count")
> forest$train(task)
```

Then, we simply pass the trained model, evaluation data, and remaining parameters to the `fme()` function. It returns a `ForwardMarginalEffect` object, which can be analyzed via `summary()` and visualized via `plot()` (see Fig. 3). Here, the outside temperature is raised by 5 degrees Celsius ceteris paribus. To avoid overplotting values, each hexagon represents a local average of FMEs. Users can easily access the data used by all plot functions to implement their own visualizations.

Let us single out the observation with the largest associated FME. This observation corresponds to a single day with a recorded temperature of 8 degrees Celsius. Increasing the temperature by 5 degrees Celsius on this particular day results in 2563 additional predicted bike rentals. We plot such model explanations for the entire data set and average FMEs to receive a global model explanation. The AME—the global average of FMEs—is 304: an increase in temperature by 5 degrees Celsius results in an average increase of 304 predicted daily bike rentals.

```
> effects.univariate.temp = fme(
+   model = forest,
+   data = bikes,
+   features = list("temp" = 5),
+   ep.method = "envelope")

> summary(effects.univariate.temp)

Forward Marginal Effects Object

Step type:
  numerical

Features & step lengths:
  temp, 5

Extrapolation point detection:
  envelope, EPs: 48 of 731 obs. (7 %)

Average Marginal Effect (AME):
  304.1722

> plot(effects.univariate.temp)
```

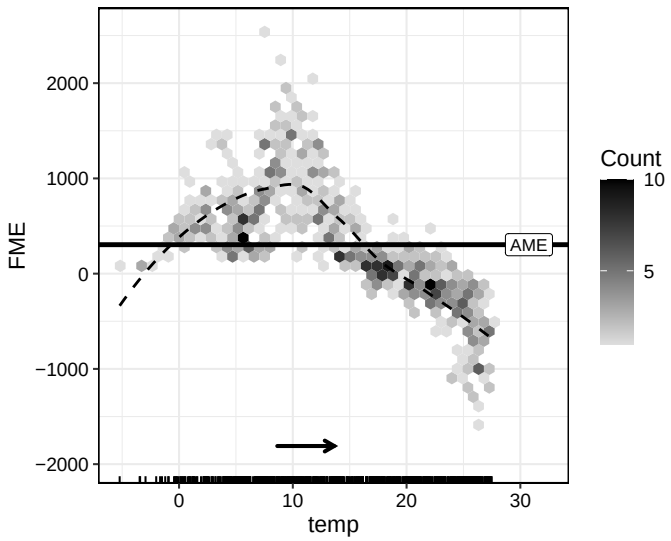


**Figure 3:** Plot of univariate FMEs for feature '`temp`' and step size 5. Each hexagon represents a local FME average. The horizontal value represents the observed feature value of '`temp`'. Each observation's '`temp`' value is moved according to the arrow's direction and length. The vertical value of each hexagon indicates the FME value associated with that feature change. The horizontal bar indicates the AME. The shade of the hexagon implies how many observations it contains. A smoothing function facilitates interpretations by modeling an approximate pattern of FMEs across the feature range.

Let us take a moment to compare the FME plot with the combined ICE and PD plot generated by the R package **iml** (Molnar et al., 2018) (see Fig. 4). This is one of the most popular and established model-agnostic ways to interpret predictive models (Molnar, 2022). The ICE is a local model explanation and represents the prediction for an observation where only the features of interest are varied (in this case, only 'temp'). The PD is the average of ICEs (in the univariate case, the vertical average) and indicates the global, average prediction when a subset of features is varied for all observations. Although we can see a rough trajectory of the feature influence on local and average predictions, it is difficult to pinpoint the exact effects of changing 'temp' on the prediction for single observations. Furthermore, ICE curves are more likely to be subject to model extrapolations, a result of predicting in areas where the model was not trained on a sufficient amount of data.

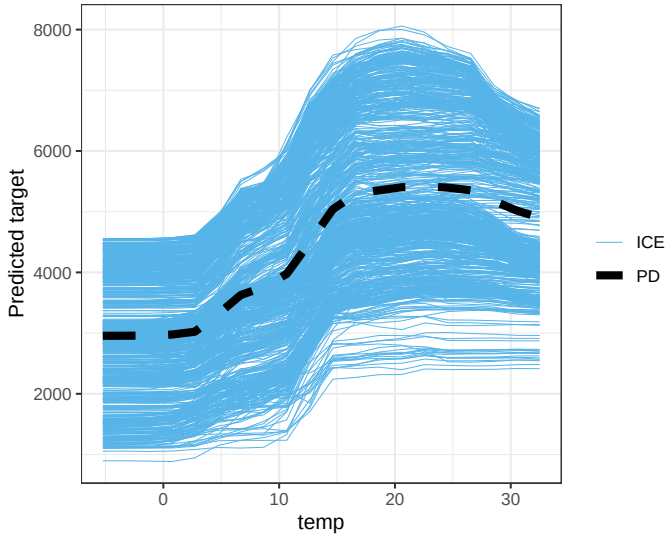


**Figure 4:** An ICE and PD plot for feature 'temp' generated by the R package **iml**. Each solid blue curve (an ICE) represents predictions for a single instance while only 'temp' varies. The dashed black curve (the PD) is the vertical average of ICEs and represents the average, isolated influence of 'temp'.

FMEs allow for positive or negative step sizes. For instance, let us investigate the effects of an isolated drop in humidity by 10 percentage points. We can observe an AME of 103 additional predicted bike rentals a day. Individual effects tend to be larger the higher the humidity on that particular day.

```
> effects.univariate.humidity = fme(
+   model = forest,
+   data = bikes,
+   features = list("humidity" = -0.1),
+   ep.method = "envelope")

> summary(effects.univariate.humidity)

Forward Marginal Effects Object

Step type:
  numerical

Features & step lengths:
  humidity, -0.1

Extrapolation point detection:
  envelope, EPs: 1 of 731 obs. (0 %)

Average Marginal Effect (AME):
  102.9158

> plot(effects.univariate.humidity)
```

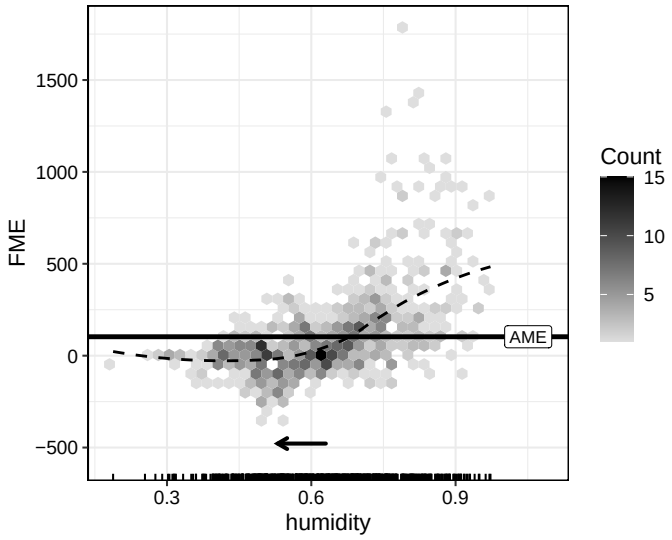


**Figure 5:** Univariate FMEs for a drop in humidity by 10 percentage points. Especially for high humidity values, the drop results in a considerable increase in predicted daily bike rentals.

In many applications, we are interested in interactions of features on the prediction. Until now, we only analyzed the univariate effects of 'temp' and 'humidity' on the predicted amount of bike rentals. However, potential interactions between features may exist. We evaluate an increase in temperature by 5 degrees Celsius and a simultaneous drop in humidity by 10 percentage points (see Fig. 6). For a bivariate change in feature values, the two arrows depict the direction and magnitude of the feature change in the respective variable. As in the univariate case, we plot local averages within hexagons to avoid overplotting values. The location of the hexagon is determined by the observations' observed feature values in the provided data set. Its color indicates the FME associated with the bivariate feature change. An increase in the outside temperature by 5 degrees Celsius and a simultaneous drop in humidity by 10 percentage points is associated with an AME of 403. The univariate AMEs roughly add up to the bivariate AME, indicating that, on average, there is no additional interaction between both feature changes on the prediction.

```
> effects.bivariate = fme(
+   model = forest,
+   data = bikes,
+   features = list("temp" = 5, "humidity" = -0.1),
+   ep.method = "envelope")

> summary(effects.bivariate)

Forward Marginal Effects Object

Step type:
  numerical

Features & step lengths:
  temp, 5
  humidity, -0.1

Extrapolation point detection:
  envelope, EPs: 49 of 731 obs. (7 %)

Average Marginal Effect (AME):
  403.0714

> plot(effects.bivariate)
```

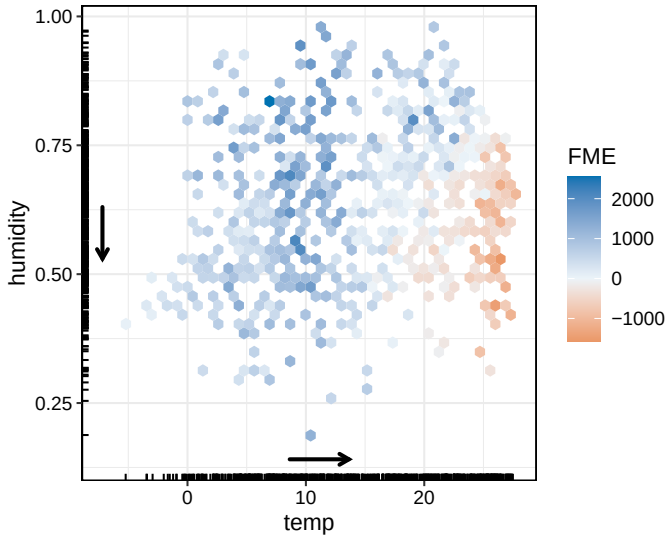


**Figure 6:** Visualizing bivariate FMEs for an increase in 'temp' by 5 degrees Celsius and a simultaneous drop in 'humidity' by 10 percentage points. FMEs are highly heterogeneous. We can see mostly positive effects, especially for observations with combinations of medium 'temp' and 'humidity' values.

Let us repeat the same procedure as for univariate feature changes and compare the FME plot to an alternative option, the bivariate PD plot (see Fig. 7). As opposed to the novel visualization with FMEs, the PD plot only visualizes the average, global effect of changing both features on the predicted amount of bike rentals. It does not inform us about the distribution of observed feature values, thus not allowing us to evaluate the effects of increasing one feature and decreasing another simultaneously.

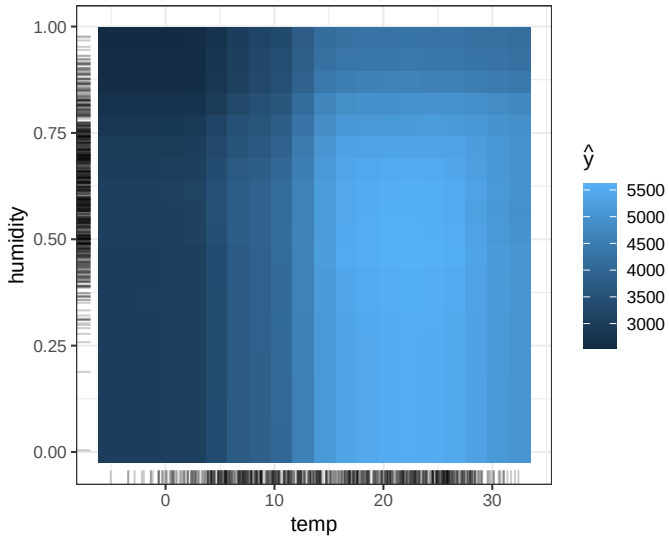


**Figure 7:** A bivariate PD plot (created via the R package **iml**), visualizing the global interaction between 'temp' and 'humidity' on the predicted amount of bike rentals. Plugging in medium to large values for 'temp' and low to medium values for 'humidity', ceteris paribus, results in more predicted bike rentals on average. As opposed to bivariate FMEs, we cannot investigate multiple local effects, nor can we see the actual distribution of observed feature values. As a result, we cannot evaluate the effects of increasing one feature and decreasing another simultaneously.

Let us now proceed to investigate non-linearity. Non-linearity can be visually assessed for ICE curves (see Fig. 4), but it is hard to quantify and would be somewhat meaningless for a large variation in the feature of interest. Furthermore, for bivariate or higher-dimensional changes in feature values, we lose any option for visual diagnoses of non-linearity. In contrast, the NLM can be computed for FMEs with continuous step sizes, regardless of dimensionality. The average non-linearity measure (ANLM) is 0.34, indicating that the linear secant, on average, is a bad descriptor of the FME.

```
> effects.bivariate.nlm = fme(
+   model = forest,
+   data = bikes,
+   features = list("temp" = 5, "humidity" = -0.1),
+   ep.method = "envelope",
+   compute.nlm = TRUE)

> effects.bivariate.nlm

Forward Marginal Effects Object

Features & step lengths:
  temp, 5
  humidity, -0.1

Average Marginal Effect (AME):
  403.0714

Average Non-Linearity Measure (ANLM):
  0.34

> plot(effects.bivariate.nlm, with.nlm = TRUE)
```

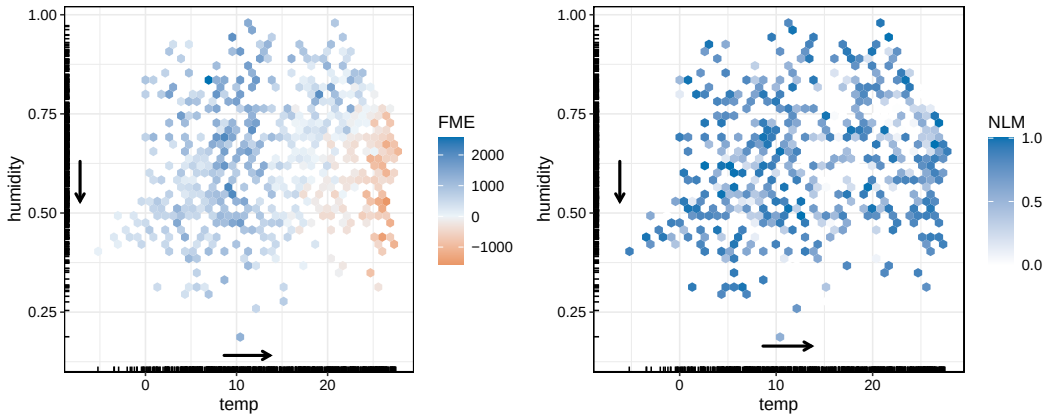


**Figure 8:** Adding NLM computations to the FME plot. Each hexagon in the left and right plots represents a local average of FME and NLM values, respectively.

Fig. 8 simply contrasts FME values with the corresponding NLM values. In this case, we can see both non-linear FMEs (whiter NLM) and linear FMEs (deep blue-colored NLM). We could now, for instance, focus on interpreting linear FMEs. All FMEs depicted in Fig. 9 have an NLM of 0.9 or higher, meaning that they almost fully describe the model prediction for proportional changes in '`temp`' and '`humidity`'.

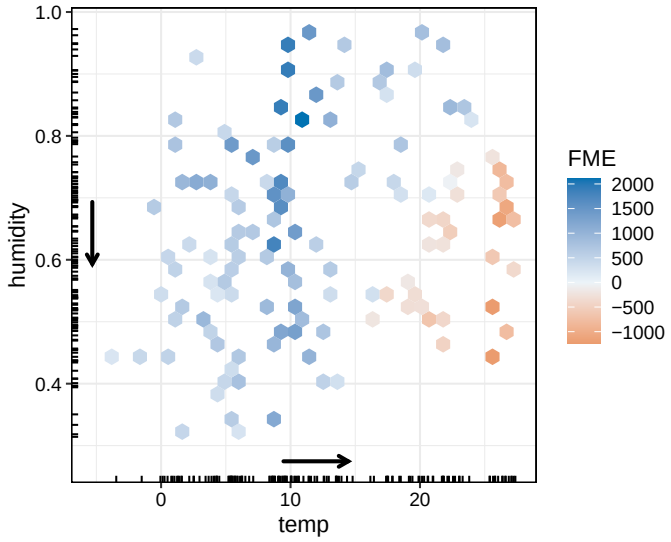


**Figure 9:** Visualizing FMEs with an NLM $\geq 0.9$.

An advantage of FMEs is their ability to provide comprehensible model insight even when exploring higher-order feature changes. Let us factor in a third feature change, now simultaneously reducing windspeed by 5 miles per hour, and visualize the distribution of FME and NLM values. We can see that in addition to an increase in temperature and a decrease in humidity, a decrease in windspeed further boosts the average number of predicted daily bike rentals.

```
> effects.trivariate.nlm = fme(
+   model = forest,
+   data = bikes,
+   features = list("temp" = 5, "humidity" = -0.1, "windspeed" = -5),
+   ep.method = "envelope",
+   compute.nlm = TRUE)

> summary(effects.trivariate.nlm)

Forward Marginal Effects Object

Step type:
  numerical

Features & step lengths:
  temp, 5
  humidity, -0.1
  windspeed, -5

Extrapolation point detection:
  envelope, EPs: 117 of 731 obs. (16 %)

Average Marginal Effect (AME):
  515.2608

Average Non-Linearity Measure (ANLM):
  0.31

> plot(effects.trivariate.nlm, with.nlm = TRUE)
```

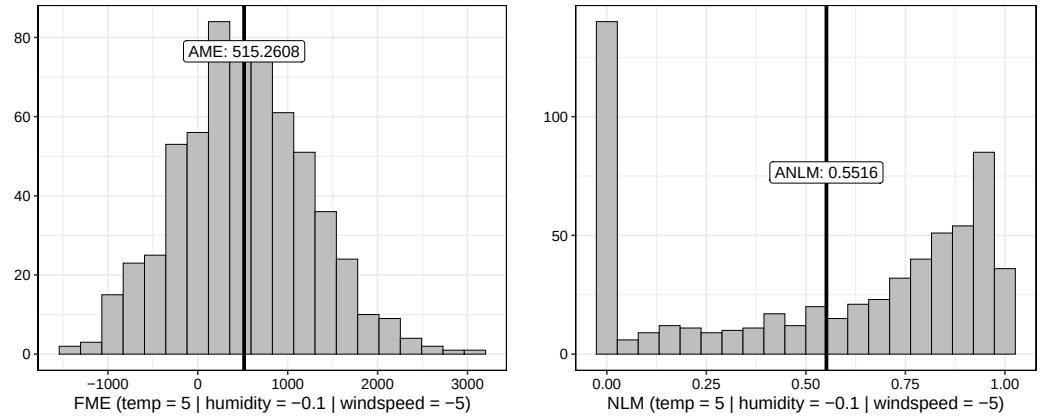


**Figure 10:** Adding a third feature change, a drop in windspeed by 5 miles per hour, and visualizing the distribution of FME and NLM values. For the NLM plot, negative NLMs are binned as 0. It follows that the ANLM value in the plot differs from the raw ANLM in the summary output.

So far, we have only evaluated changes in continuous features. In many applications, we are concerned with switching categories of categorical features, a way of counterfactual thinking inherent to the human thought process. Note that despite the allure of switching categories of categorical features, one needs to be aware of potential model extrapolations. To illustrate this, we switch each non-rainy day's precipitation status to rainfall. Rainfall has an average isolated effect of lowering daily rentals by 699 bikes (see Fig. 11).

```
> effects.categ = fme(
+   model = forest,
+   data = bikes,
+   features = list("weather" = "rain"))

> summary(effects.categ)

Forward Marginal Effects Object

Step type:
  categorical

Feature & reference category:
  weather, rain

Extrapolation point detection:
  none, EPs: 0 of 710 obs. (0 %)

Average Marginal Effect (AME):
  -699.4915

> plot(effects.categ)
```

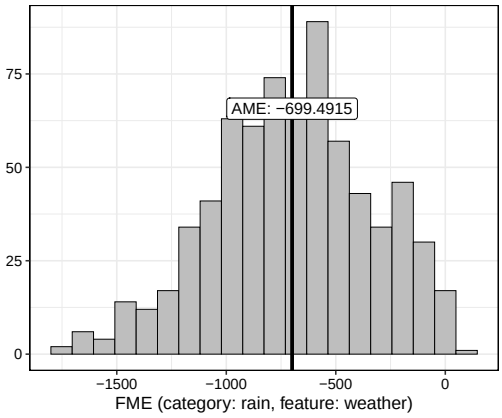


**Figure 11:** Distribution of categorical FMEs resulting from switching each non-rainy day's precipitation status to rain. On average, rainfall lowers predicted bike rentals by 699 bikes per day.

### 6.2 Regional explanations

In our examples, we can see highly heterogeneous local effects. The more heterogeneous FMEs are, the less information the AME carries. In many practical applications, we are interested in compactly describing the behavior of the predictive model across the feature space, akin to a beta coefficient in a linear model. This is where regional explanations come into play. We aim to find subgroups with more homogeneous FME values, thereby describing the behavior of the model not in terms of a global average but in terms of multiple regional averages (cAMEs).

In **fmeffects**, this can be achieved by further processing the `ForwardMarginalEffect` object containing FMEs (and optionally NLM values) using the `came()` function. This returns a `Partitioning` object (in this case, an object of the class `"PartitioningCTREE"`, a subclass of the abstract class `"Partitioning"`, see later section on Design and options for extensions).

For the univariate change in temperature by 5 degrees Celsius, we decide to search for precisely 2 subgroups[2] (for a description of this algorithm, see the following section on Design and options for extensions). A summary of the created object informs us about the number of observations, cAME, and standard deviation (SD) of FMEs inside the root node and leaf nodes (the found subgroups). We succeeded in finding subgroups with lower SDs while maintaining an appropriate sample size. The root node SD of 620 can be successfully split down to 442 and 369 within the subgroups. By visualizing the tree, we can see how the data was partitioned. For cooler outside temperatures equal to or lower than $\approx 16$ degrees Celsius, we can observe a positive cAME of 730 additional bike rentals per day. On warmer days with a temperature above $\approx 16$ degrees Celsius, the model predicts 205 less bike rentals a day when the outside temperature increases by 5 degrees.

```
> subspaces = came(effects = effects.univariate.temp, number.partitions = 2)
> summary(subspaces)

PartitioningCtree of an FME object

Method:  partitions = 2

   n      cAME  SD(fME)
 683  304.1722 620.4775 *
 372  729.8519 441.6201
 311 -205.0011 368.8368
---
* root node (non-partitioned)

AME (Global): 304.1722

> plot(subspaces)
```

[2]This value is to be set by the user depending on how many regional explanations are to be found. Alternatively, we can search for a pre-defined SD of FMEs inside the terminal nodes. How many subgroups can be found depends on the data and predictive model.

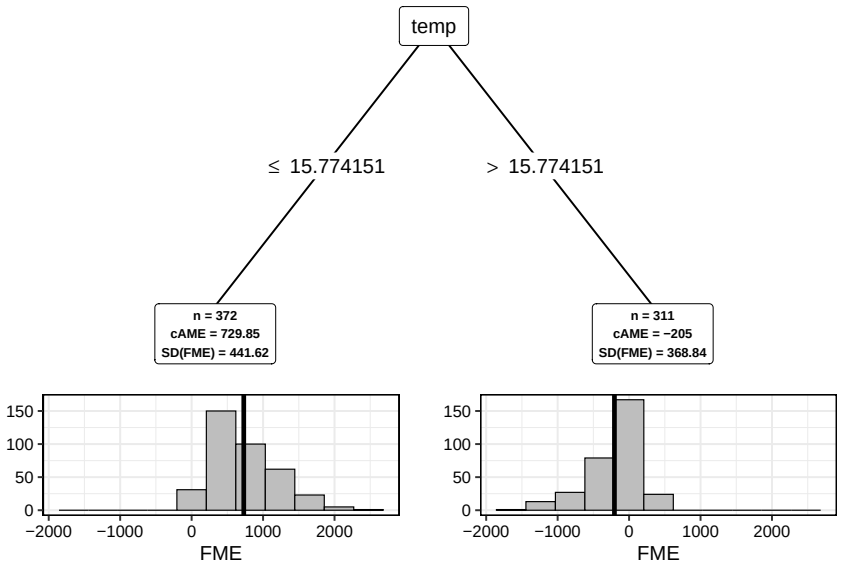


**Figure 12:** Using a decision tree to find subgroups of observations with more homogeneous FMEs of increasing 'temp' by 5 degrees Celsius. Each leaf node visualizes one subgroup, the number of observations, the cAME, and the SD of FMEs indicating FME homogeneity.

### 6.3 Global explanations

When to search for regional explanations thus depends on the heterogeneity of local effects. The `ame()` function provides an appropriate summary for the entire model. It uses a default step size of 1 or 0.01 for small feature ranges. For categorical FMEs, it uses every observed category as a reference category. Alternatively, custom step sizes and subsets of features can be used. The `summary()` function prints a compact model summary of each feature, a default step size, the AME, the SD of FMEs, 25% and 75% quantiles of FMEs, as well as the number of observations left after excluding extrapolation points (EPs). A large dispersion indicates heterogeneity of FMEs and thus a small fidelity of the AME and possible benefits from searching for subgroups with varying cAMEs. A different workflow can, therefore, also consist of starting with the table generated by `ame()` and deciding which feature effects can be described by AMEs and which might be better describable by subgroups and cAMEs. If this has been unsuccessful, we can resort to local model explanations. Recall our example from the previous section on Regional explanations where we split FMEs associated with increasing temperature by 5 degrees Celsius. From the `ame()` summary, we see that 'temp' has a relatively large SD in relation to its AME (here calculated with a step size of 1), and the interquartile range indicates a wide spread of FMEs from -21 in the 25% quantile up to 107 in the 75% quantile, which makes it a promising candidate to find subgroups with more homogeneous FMEs.

```
> ame.results = ame(model = forest, data = bikes)
> summary(ame.results)

Model Summary Using Average Marginal Effects:

      Feature step.size        AME       SD       0.25       0.75   n
1      season    winter  -894.4673 456.3625 -1248.2476  -586.5656 550
2      season    spring   141.6627 557.8672  -242.9194   652.8917 547
3      season    summer   538.4263 627.8606    45.0598  1196.3612 543
4      season      fall   493.7475 581.3166     8.7096  1101.5087 553
5        year         0 -1890.8318 641.3168 -2377.7961 -1496.2576 366
6        year         1   1785.563 508.6759  1412.6724  2183.8292 365
7     holiday        no   165.2367 213.3036    72.5637   194.7954  21
8     holiday       yes  -122.4971 141.9902  -189.0043   -22.1315 710
9     weekday    Sunday   107.3675 199.4931   -33.8124   218.4856 626
10    weekday    Monday  -127.8842  232.482   -260.735    23.9211 626
11    weekday   Tuesday  -110.9437 219.9664  -216.2248    29.4189 626
```

```
12    weekday Wednesday   -16.5913 204.4574  -113.3341   118.8563 627
13    weekday  Thursday    27.4835 189.9021   -85.0117   140.1993 627
14    weekday    Friday     53.982 194.2184   -65.3866   170.0411 627
15    weekday  Saturday   110.8837 191.1073    -7.7049   231.8014 627
16 workingday        no    -41.222 115.1556  -126.4856    45.9121 500
17 workingday       yes    42.5305 154.5266   -67.1033   134.7876 231
18    weather     misty  -236.5115 327.3365   -442.211   -71.8195 484
19    weather     clear   368.2611 325.1541   145.7027   459.1031 268
20    weather      rain  -699.4915 362.8458  -943.5127  -454.9041 710
21       temp         1    56.6478 167.5781   -21.1847   106.6103 731
22   humidity      0.01   -20.3705  58.2372   -35.0143      8.289 731
23  windspeed         1   -24.3256  73.3227   -50.7023    12.0791 731
```

## 7 Design and options for extensions

The **fmeffects** package is built on a modular design for improved maintainability and future extensions. Fig. 13 provides a visual overview of the core design. The greatest emphasis is placed on the strategy and adapter design patterns (Gamma et al., 1994). Simply put, the strategy pattern decouples the source code for algorithm selection at runtime into separate classes. We repeatedly implement this pattern throughout the package by creating abstract classes whose subclasses implement specific functionalities. The adapter design pattern (also called a "wrapper") creates an interface for communication between two classes.

- `"Predictor"`: An abstract class that implements the adapter pattern to accommodate future implementations of storing a predictive model. `"PredictorMLR3"`, `"PredictorParsnip"`, and `"PredictorCaret"` are subclasses that store an **mlr3**, **parsnip** (Kuhn and Vaughan, 2023) (part of **tidymodels**), or **caret** model object. This allows users of **fmeffects** to use numerous predictive models such as random forests, gradient boosting, support vector machines, or neural networks. `"PredictorLM"` stores models returned by `lm()`,`glm()`,`or gam()`. The package can be extended with novel model types by implementing a new subclass that stores the model, data, target, and is able to return predictions.
- `"AverageMarginalEffects"`: A class to compute AMEs for each feature in the data (or a subset of features). Internally, a new `"ForwardMarginalEffect"` object is used to compute and aggregate FMEs. For convenience, we implement a wrapper function `ame()` to facilitate object creation and to initiate computations without requiring user input in the form of **R6** syntax.
- `"ForwardMarginalEffect"`: The centerpiece class of the package. It keeps access to a `Predictor`, stores important information to create FMEs, and after the computations are completed, stores results and gives access to visualization methods. For convenience, the wrapper function `fme()` can be used.
- `"FMEPlot"`: An abstract class for code decoupling of different plot categories into distinct classes. Subclasses include `"FMEPlotUnivariate"`, `"FMEPlotBivariate"`, `"FMEPlotHigherOrder"`, `"FMEPlotCategorical"`.
- `"ExtrapolationDetector"`: Identifies (and excludes) EPs. The current implementation supports the method "envelope", excluding points outside the multivariate envelope of the training data.
- `"NonLinearityMeasure"`: For the NLM, we need to approximate three line integrals, e.g., via Simpson's 3/8 rule. The general definition of Simpson's 3/8 rule for a univariate function f(x) and integration interval $[a, b]$ corresponds to:

$$\int_a^b f(x) \approx \frac{b-a}{8}\left[f(a) + 3f\left(\frac{2a+b}{3}\right) + 3f\left(\frac{a+2b}{3}\right) + f(b)\right] \tag{3}$$

  We make use of a composite Simpson rule, which divides up the interval $[a, b]$ into $n$ subintervals of equal size and approximates each subinterval with Eq. (3).
- `"Partitioning"`: An abstract class, allowing for various implementations of finding subgroups for cAMEs. For convenience, the wrapper function `came()` can be used. The current implementation supports RP via the **rpart** and **partykit** (CTREE algorithm) packages (classes `"PartitioningRPart"` and `"PartitioningCTREE"`).

  We believe there are two criteria that should guide this process: FME homogeneity within each subgroup and the number of subgroups. A low number of subgroups is generally preferred. In certain applications, we may want to search for a predefined number of subgroups, akin to the search for a predefined number of clusters in clustering problems. Many RP algorithms do not support searching for a number of subgroups, which is what the `"Pruner"` class is intended for.

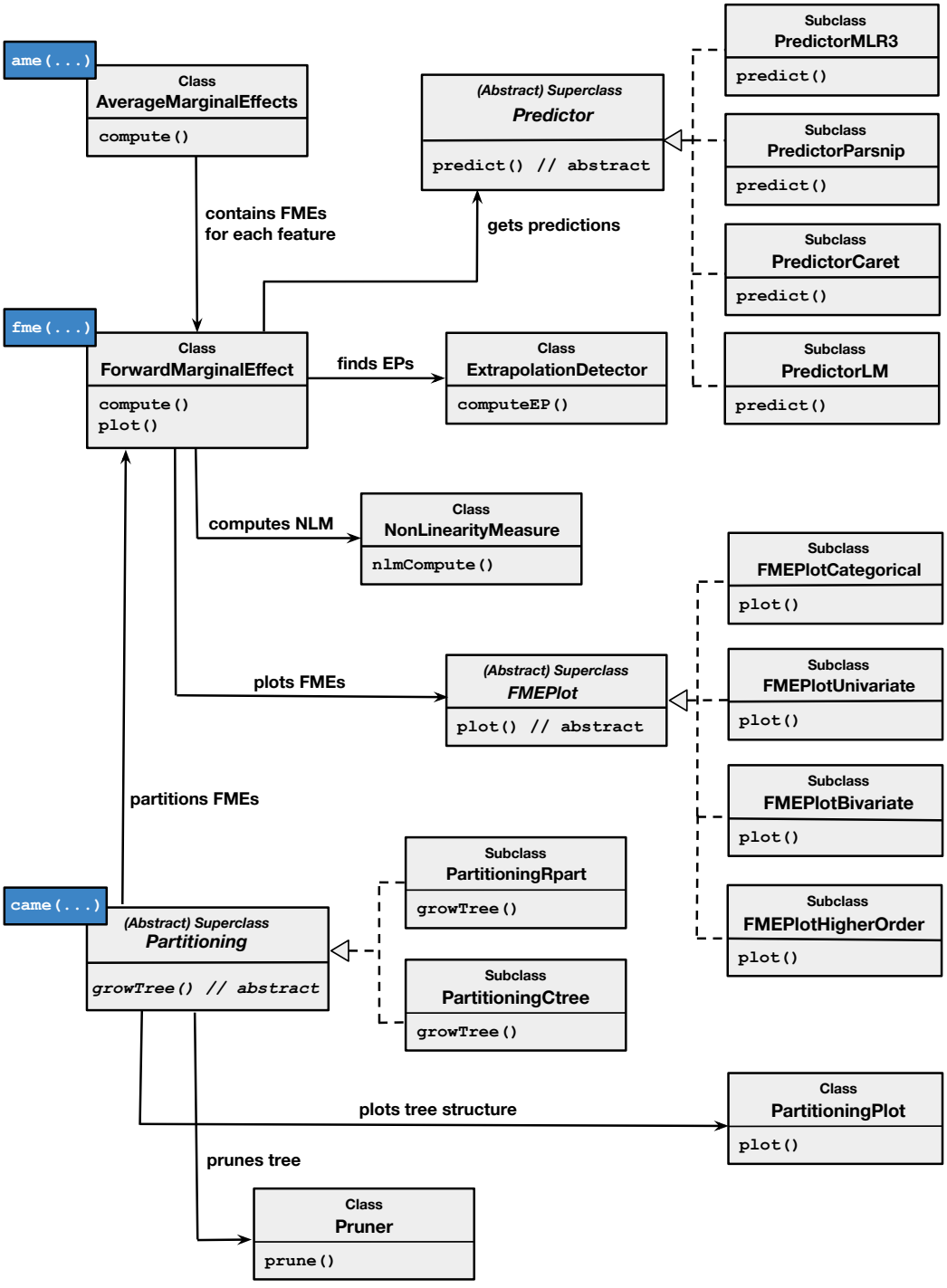


**Figure 13:** Design overview of the **fmeffects** package, including methods that implement the main functionality of each class. Classes may contain more methods than depicted. Blue boxes indicate wrapper functions to instantiate objects of the respective class.

- `"Pruner"`: To receive a predefined number of subgroups for arbitrary RP algorithms, we follow a two-stage process: grow a large tree by tweaking tree-specific hyperparameters and then prune it back to receive the desired number of subgroups. A `"Partitioning"` subclass is implemented such that it can first grow a large tree, e.g., with a low complexity parameter for **rpart**. Then `"Pruner"` iteratively computes the SD of FMEs for each parent node of the current terminal nodes and removes all terminal nodes of the parent with the lowest SD.
- `"PartitioningPlot"`: Decouples visualizations of the separation of $\mathcal{D}$ into subgroups from specific implementations of the `"Partitioning"` subclass. Here, we make use of a dependency on **partykit** for a tree data structure. This allows visualizations of any partitioning with the same methods. The package **ggparty** (Borkovec and Madin, 2019) creates tree figures that illustrate the partitioning, descriptive statistics for each terminal node, and histograms of FMEs (and optionally NLM values).

## 8 Conclusion

This paper introduces the R package **fmeffects**, the first software implementation of the theory surrounding FMEs. We showcase the package functionality with an applied use case and discuss design choices and implications for future extensions. FMEs are a versatile model-agnostic interpretation method and give us comprehensible model explanations in the form of: if we change $x$ by an amount $h$, what is the change in predicted outcome $\widehat{y}$? FMEs equip stakeholders, including those without ML expertise, with the ability to understand feature effects for any model. We therefore hope that this package will work towards a more widespread adoption of FMEs in practice.

Software development is an ongoing process. As the theory surrounding FMEs evolves, so should the **fmeffects** package. As noted by Scholbeck et al. (2024), possible directions for future research include the development of techniques to better quantify extrapolation risk for the selection of step sizes; furthermore, the subgroup search for cAMEs is subject to uncertainties, which may be able to be quantified; and lastly, we may be able to spare computations by searching for representative FMEs, similar to prototype observations that are representative of clusters of observations (Tan et al., 2019). Future performance improvements may also be made via parallel computing, which at this point is only implemented for NLM computations.

## References


A. Adadi and M. Berrada. Peeking inside the black-box: A survey on explainable artificial intelligence (XAI). *IEEE Access*, 6:52138–52160, 2018. URL https://doi.org/10.1109/access.2018.2870052. [p73]

D. W. Apley and J. Zhu. Visualizing the effects of predictor variables in black box supervised learning models. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, 82(4):1059–1086, 2020. URL https://doi.org/10.1111/rssb.12377. [p71, 73]

V. Arel-Bundock. *marginaleffects: Predictions, Comparisons, Slopes, Marginal Means, and Hypothesis Tests*, 2023. URL https://CRAN.R-project.org/package=marginaleffects. R package version 0.11.1. [p71]

S. Athey and G. W. Imbens. Machine learning methods that economists should know about. *Annual Review of Economics*, 11(1):685–725, 2019. URL https://doi.org/10.1146/annurev-economics-080217-053433. [p67]

T. Bartus. Estimation of marginal effects using margeff. *The Stata Journal*, 5(3):309 – 329, 2005. [p67, 71]

M. Borkovec and N. Madin. *ggparty: 'ggplot' Visualizations for the 'partykit' Package*, 2019. URL https://CRAN.R-project.org/package=ggparty. R package version 1.0.0. [p85]

A.-L. Boulesteix, M. N. Wright, S. Hoffmann, and I. R. König. Statistical learning approaches in the genetic epidemiology of complex diseases. *Human Genetics*, 139(1):73–84, 2020. URL https://doi.org/10.1007/s00439-019-01996-9. [p67]

L. Breiman. Statistical modeling: The two cultures (with comments and a rejoinder by the author). *Statistical Science*, 16(3):199 – 231, 2001. URL https://doi.org/10.1214/ss/1009213726. [p67, 72]

M. Britton. Vine: Visualizing statistical interactions in black box models. arXiv, 2019. URL https://doi.org/10.48550/arXiv.1904.00561. [p71, 72]

G. Casalicchio, C. Molnar, and B. Bischl. Visualizing the feature importance for black box models. In M. Berlingerio, F. Bonchi, T. Gärtner, N. Hurley, and G. Ifrim, editors, *Machine Learning and Knowledge Discovery in Databases*, pages 655–670. Springer International Publishing, Cham, 2019. URL https://doi.org/10.1007/978-3-030-10925-7_40. [p71]

W. Chang. *R6: Encapsulated Classes with Reference Semantics*, 2021. URL https://CRAN.R-project.org/package=R6. R package version 2.5.1. [p73]

I. C. Covert, S. Lundberg, and S.-I. Lee. Understanding global feature contributions with additive importance measures. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS'20, Red Hook, NY, USA, 2020. Curran Associates Inc. [p71]

P. D. Dueben and P. Bauer. Challenges and design choices for global weather and climate models based on machine learning. *Geoscientific Model Development*, 11(10):3999–4009, 2018. URL https://doi.org/10.5194/gmd-11-3999-2018. [p67]

D. B. Dwyer, P. Falkai, and N. Koutsouleris. Machine learning approaches for clinical psychology and psychiatry. *Annual Review of Clinical Psychology*, 14(1):91–118, 2018. URL https://doi.org/10.1146/annurev-clinpsy-032816-045037. [p67]

H. Fanaee-T. Bike Sharing Dataset. UCI Machine Learning Repository, 2013. URL https://doi.org/10.24432/C5W894. [p73]

J. H. Friedman. Greedy function approximation: A gradient boosting machine. *Ann. Statist.*, 29(5): 1189–1232, 2001. URL https://doi.org/10.1214/aos/1013203451. [p71]

E. Gamma, R. Helm, R. Johnson, and J. M. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional, 1st edition, 1994. [p83]

A. Goldstein, A. Kapelner, J. Bleich, and E. Pitkin. Peeking inside the black box: Visualizing statistical learning with plots of individual conditional expectation. *Journal of Computational and Graphical Statistics*, 24(1):44–65, 2015. URL https://doi.org/10.1080/10618600.2014.907095. [p71]

W. Greene. *Econometric Analysis*. Pearson International, 8th edition, 2019. [p67, 71]

J. Herbinger, B. Bischl, and G. Casalicchio. Repid: Regional effect plots with implicit interaction detection. In G. Camps-Valls, F. J. R. Ruiz, and I. Valera, editors, *Proceedings of The 25th International Conference on Artificial Intelligence and Statistics*, volume 151 of *Proceedings of Machine Learning Research*, pages 10209–10233. PMLR, 2022. [p71, 72]

G. Hooker. Diagnosing extrapolation: Tree-based density estimation. In *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '04, page 569–574, New York, NY, USA, 2004a. Association for Computing Machinery. [p73]

G. Hooker. Discovering additive structure in black box functions. In *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '04, pages 575–580, New York, NY, USA, 2004b. ACM. URL http://doi.acm.org/10.1145/1014052.1014122. [p73]

G. Hooker. Generalized functional ANOVA diagnostics for high-dimensional functions of dependent variables. *Journal of Computational and Graphical Statistics*, 16(3):709–732, 2007. URL https://doi.org/10.1198/106186007X237892. [p73]

G. Hooker, L. Mentch, and S. Zhou. Unrestricted permutation forces extrapolation: Variable importance requires at least one more model, or there is no free variable importance. *Statistics and Computing*, 31 (6):82, 2021. URL https://doi.org/10.1007/s11222-021-10057-z. [p73]

T. Hothorn and A. Zeileis. partykit: A modular toolkit for recursive partytioning in R. *Journal of Machine Learning Research*, 16(118):3905–3909, 2015. [p70]

U. Kamath and J. Liu. Introduction to interpretability and explainability. In *Explainable Artificial Intelligence: An Introduction to Interpretable Machine Learning*, pages 1–26. Springer International Publishing, Cham, 2021. URL https://doi.org/10.1007/978-3-030-83356-5_1. [p67]

M. Kuhn and D. Vaughan. *parsnip: A Common API to Modeling and Analysis Functions*, 2023. URL https://CRAN.R-project.org/package=parsnip. R package version 1.1.1. [p83]

T. J. Leeper. *margins: Marginal effects for model objects*, 2018. URL https://CRAN.R-project.org/package=margins. R package version 0.3.23. [p71]

A. Liaw and M. Wiener. Classification and regression by randomforest. *R News*, 2(3):18–22, 2002. URL https://CRAN.R-project.org/doc/Rnews/. [p73]

S. M. Lundberg and S.-I. Lee. A unified approach to interpreting model predictions. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 4768–4777, Red Hook, NY, USA, 2017. Curran Associates Inc. [p71]

D. Lüdecke. ggeffects: Tidy data frames of marginal effects from regression models. *Journal of Open Source Software*, 3(26):772, 2018. URL https://doi.org/10.21105/joss.00772. [p71]

C. J. McCabe, M. A. Halvorson, K. M. King, X. Cao, and D. S. Kim. Interpreting interaction effects in generalized linear models of nonlinear probabilities and counts. *Multivariate Behavioral Research*, 57 (2-3):243–263, 2022. URL https://doi.org/10.1080/00273171.2020.1868966. [p67]

N. Mehrabi, F. Morstatter, N. Saxena, K. Lerman, and A. Galstyan. A survey on bias and fairness in machine learning. *ACM Comput. Surv.*, 54(6), 2021. URL https://doi.org/10.1145/3457607. [p70]

T. D. Mize, L. Doan, and J. S. Long. A general framework for comparing predictions and marginal effects across models. *Sociological Methodology*, 49(1):152–189, 2019. URL https://doi.org/10.1177/0081175019852763. [p71]

C. Molnar. *Interpretable Machine Learning*. 2nd edition, 2022. URL https://christophm.github.io/interpretable-ml-book. [p67, 75]

C. Molnar, B. Bischl, and G. Casalicchio. iml: An R package for interpretable machine learning. *JOSS*, 3(26):786, 2018. URL https://doi.org/10.21105/joss.00786. [p75]

C. Molnar, G. König, J. Herbinger, T. Freiesleben, S. Dandl, C. A. Scholbeck, G. Casalicchio, M. Grosse-Wentrup, and B. Bischl. General pitfalls of model-agnostic interpretation methods for machine learning models. In A. Holzinger, R. Goebel, R. Fong, T. Moon, K.-R. Müller, and W. Samek, editors, *xxAI - Beyond Explainable AI. xxAI 2020. Lecture Notes in Computer Science, vol 13200*, Cham, 2022. Springer. URL https://doi.org/10.1007/978-3-031-04083-2_4. [p73]

C. Molnar, G. König, B. Bischl, and G. Casalicchio. Model-agnostic feature importance and effects with dependent features: A conditional subgroup approach. *Data Mining and Knowledge Discovery*, 38(5): 2903–2941, 2024. URL https://doi.org/10.1007/s10618-022-00901-9. [p71]

S. Mullainathan and J. Spiess. Machine learning: An applied econometric approach. *Journal of Economic Perspectives*, 31(2):87–106, 2017. URL https://doi.org/10.1257/jep.31.2.87. [p67]

E. Onukwugha, J. Bergtold, and R. Jain. A primer on marginal effects—part I: Theory and formulae. *PharmacoEconomics*, 33(1):25–30, 2015. URL https://doi.org/10.1007/s40273-014-0210-6. [p67]

A. Rajkomar, J. Dean, and I. Kohane. Machine learning in medicine. *New England Journal of Medicine*, 380(14):1347–1358, 2019. URL https://doi.org/10.1056/NEJMra1814259. [p67]

M. T. Ribeiro, S. Singh, and C. Guestrin. "Why should I trust you?": Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, page 1135–1144, New York, NY, USA, 2016. Association for Computing Machinery. URL https://doi.org/10.1145/2939672.2939778. [p71]

C. A. Scholbeck, C. Molnar, C. Heumann, B. Bischl, and G. Casalicchio. Sampling, intervention, prediction, aggregation: A generalized framework for model-agnostic interpretations. In P. Cellier and K. Driessens, editors, *Machine Learning and Knowledge Discovery in Databases*, volume 1167 of *Communications in Computer and Information Science*, pages 205–216. Springer International Publishing, Cham, 2020. URL https://doi.org/10.1007/978-3-030-43823-4_18. [p67, 71]

C. A. Scholbeck, J. Moosbauer, G. Casalicchio, H. Gupta, B. Bischl, and C. Heumann. Position paper: Bridging the gap between machine learning and sensitivity analysis. arXiv, 2023. URL https://doi.org/10.48550/arXiv.2312.13234. [p71]

C. A. Scholbeck, G. Casalicchio, C. Molnar, B. Bischl, and C. Heumann. Marginal effects for non-linear prediction functions. *Data Mining and Knowledge Discovery*, 38(5):2997–3042, 2024. URL https://doi.org/10.1007/s10618-023-00993-x. [p67, 68, 69, 70, 71, 73, 85]

StataCorp. *Stata: Release 18*. College Station, TX: StataCorp LLC., 2023. [p71]

E. Štrumbelj and I. Kononenko. An efficient explanation of individual classifications using game theory. *Journal of Machine Learning Research*, 11(1):1–18, 2010. [p71]

P.-N. Tan, A. Karpatne, M. Steinbach, and V. Kumar. *Introduction to Data Mining: Global Edition*. Pearson, 2019. [p85]

T. Therneau and B. Atkinson. *rpart: Recursive Partitioning and Regression Trees*, 2019. URL https://CRAN.R-project.org/package=rpart. R package version 4.1-15. [p70]

S. Wachter, B. Mittelstadt, and C. Russell. Counterfactual explanations without opening the black box: Automated decisions and the GDPR. *Harvard Journal of Law and Technology*, 31(2):841–887, 2018. [p71]

R. Williams. Using the margins command to estimate and interpret adjusted predictions and marginal effects. *Stata Journal*, 12(2):308–331(24), 2012. [p67, 68, 71]

*Holger*
*Ludwig-Maximilians-Universität in Munich*
*Germany*
hbj.loewe@gmail.com

*Christian A. Scholbeck*
*Ludwig-Maximilians-Universität in Munich*
*Munich Center for Machine Learning (MCML)*
*Germany*
https://orcid.org/0000-0001-6607-4895
christian.scholbeck@stat.uni-muenchen.de

*Christian Heumann*
*Ludwig-Maximilians-Universität in Munich*
*Germany*
christian.heumann@stat.uni-muenchen.de

*Bernd Bischl*
*Ludwig-Maximilians-Universität in Munich*
*Munich Center for Machine Learning (MCML)*
*Germany*
bernd.bischl@stat.uni-muenchen.de

*Giuseppe Casalicchio*
*Ludwig-Maximilians-Universität in Munich*
*Munich Center for Machine Learning (MCML)*
*Germany*
giuseppe.casalicchio@stat.uni-muenchen.de

# GSSTDA: Implementation in an R Package of the Progression of Disease with Survival Analysis (PAD-S) that Integrates Information on Genes Linked to Survival in the Mapper Filter Function

*by Miriam Esteve, Raquel Bosch-Romeu, Antonio Falco, Jaume Fores, and Joan Climent*

**Abstract** GSSTDA is a new package for R that implements a new analysis for trascriptomic data, the Progression Analysis of Disease with Survival (PAD-S) by Fores-Martos et al. (2022), which allows to identify groups of samples differentiated by both survival and idiosyncratic biological features. Although it was designed for transcriptomic analysis, it can be used with other types of continuous omics data. The package implements the main algorithms associated with this methodology, which first removes the part of expression that is considered physiological using the Disease-Specific Genomic Analysis (DSGA) and then analyzes it using an unsupervised classification scheme based on Topological Data Analysis (TDA), the Mapper algorithm. The implementation includes code to perform the different steps of this analysis: data preprocessing by DSGA, the selection of genes for further analysis and a new filter function, which integrates information about genes related to survival, and the Mapper algorithm for generating a topological invariant Reeb graph. These functions can be used independently, although a function that performs the entire analysis is provided. This paper describes the methodology and implementation of these functions, and reports numerical results using an extract of real data base application.

## 1 Introduction

This paper presents the implementation of Progression Analysis of Disease with Survival (PAD-S) (Fores-Martos et al. (2022)) a new analysis, based on Progression Analysis of Disease (PAD), whose main novelty is that it integrates information on genes linked to survival. The PAD, that was developed by Nicolau et al. (2011), allows a set of transcriptomic data samples to be summarised in a combinatorial graph whose nodes are subsets of the samples. In this analysis, data pre-processed using the Disease-Specific Genomic Analyses (DSGA) is subjected to the Mapper algorithm. Although both DSGA and PAD were initially used with microarray data, they can be used on other types of continuous omics data. This extends to PAD-S as well.

This DSGA is a mathematical analysis for trascriptomic data that isolates the component of data relevant to disease by defining a transformation that measures the extent to which diseased tissue samples deviates from healthy tissue samples (Nicolau et al. (2007)). On the other hand, Mapper is a Topological Data Analysis tool (Carlsson (2009)). TDA is intended to find the topological structure of the data using tools for algebraic topology and computational geometry. Two of its approaches is persistent homology (Edelsbrunner and Harer (2022)) and Mapper (Singh et al. (2007); Lum et al. (2013)). Persistent homology adopts concepts from abstract algebra in order to extract characteristics in the data, such as the presence of higher-dimensional holes and the number of connected components. Mapper however was designed as a visualization method although it can also be considered as an unsupervised classification methodology. It allows to simplify and visualize the information of high dimensional data sets into a combinatorial or complex simplicial graph, which is called the dataset skeleton. To do so, it employs preassigned guiding functions called filters. Mapper has been applied in biological and biomedical research. It has been used successfully to study aortic stenosis, where through the construction of Mapper graph is able to identify disease subtypes with a higher resolution than standard approaches (Casaclang-Verzosa et al. (2019)). Another application in the clinic has been its use in asthma, where clinical examination and biochemical and cellular parameters in biological samples are introduced as input for Mapper in order to identify subgroup of healthy patients and subgroups of asthmatic patients (Hinks et al. (2015)). In addition, it has also been used in other cases to analyze omics data, such as single-cell RNA sequencing data to study cell differentiation and development (Rizvi et al. (2017)) or to study breast cancer using RNA sequencing data (Mathews et al. (2019)).

The PAD, using DSGA and Mapper and developing an own filter function and a way to select the most relevant genes for Mapper, puzzles out the geometry characteristics of the data that are

obscured when using cluster analysis and deliver a simple representation of the trascriptomic data set. Our new analysis, the PAD-S, integrates in its new filter function information about the relationship between gene expression level and survival and also uses this information to select the genes to be used in Mapper. This adapted filter function aims to capture the expected survival associated with each patient from the information obtained from the survival analyses.

Mapper performs clustering by strata of different value ranges of this filter function, an then, it is determined if there is a relationship between clusters of different strata, obtaining a topological invariant Reeb graph. As PAD-S filtering function captures the expected survival of each individual, we obtain a graph that captures the shape of the data along the values of the "survival". In this graph, in addition to possible clusters of interest, it is possible to study structures that appear in the graph, such as branches or loops, that can uncover underlying biological patterns. This could show new relevant groups of patients with good or bad prognosis and specific biological characteristics not detected by other types of analysis.

In PAD-S, this ability of Mapper is strengthened by DSGA preprocessing, as it facilitates only the portion of gene expression that is not present in healthy tissue to be used in clustering, thus streamlining data processing and enhancing the extraction of biologically relevant information. In turn, gene selection allows the results to meet the proposed objectives in the most efficient way by selecting the most relevant survival-related genes or that have greater variability. This could lead to the identification of potential biomarkers.

In the landscape of omics data analysis methods, PAD-S stands out for its unique approach rooted in TDA. Mapper can be defined as an atypical unsupervised classification method and, like PCA and traditional clustering, it is not necessary to have prior knowledge of the data and the possible subgroups that form it. However, the Mapper's approach provides a holistic view of the data, allowing for the identification of data subsets, non-linear relationships and intricate patterns reflected in the structures of the graph that may be missed by these methods.

Many methods for analysis of omics data are nowadays available in Bioconductor, a collection of almost 1000 packages for the analysis and comprehension of high-throughput biological data, in R packages such as: **rtracklayer** (Lawrence et al., 2009), for interacting with multiple genome browsers (UCSC) and manipulating annotation tracks in various formats (GFF, BED, bedGraph, BED15, WIG, BigWig and 2bit); **Rsubread** (Liao et al., 2019), for read mapping, read counting, SNP calling, structural variant detection and gene fusion discovery; or **survcomp** (Schroder et al., 2011), for performance assessment and comparison of survival models; **scMappR** (Sokolowski et al., 2021), for providing experimentally relevant cell-type specific information to a list of differentially expressed genes (DEG). Regarding the availability of the R implementation of the analyses used in this study, in GitHub, we can find implementations of the Mapper algorithm, **Mapper** [`https://github.com/peekxc/Mapper`] or **TDAMapper** [`https://github.com/paultpearson/TDAmapper`]. No implementations found in CRAN repository. In relation to the DSGA, the implementation provided by the developers is no longer available.

In this paper, we introduce a new package in R, called **GSSTDA** which is the implementation of the already presented Progression Analysis of Disease with Survival (PAD-S). In particular, **GSSTDA** includes a complete set of baseline functions, covering DSGA method and Mapper tool applying the filter function with information on survival-related genes. We make PAD-S available as a R package, with options for DSGA only, filtering function with selection of genes, Mapper only, or a combination of the three. **GSSTDA** is available as free software, under the GNU General Public License version 3, and can be downloaded from the Comprehensive R Archive Network (CRAN) at `https://cran.r-project.org/web/packages/GSSTDA`, including supplementary material as data sets or vignettes to replicate all the results presented in this paper. In addition, **GSSTDA** is hosted on an open source repository on GitHub at `https://github.com/MiriamEsteve/GSSTDA`.

The functions included in the **GSSTDA** package are summarized in 1. This table comprises two columns divided into four subsections: three of them correspond to the three stages of the Gene Structure Survival using Topological Data Analysis and one to the full analysis. The first column is the name of the functions and the second one is the description of the functions.

The paper is organized as follows. The following subsection Background summaries the three parts of the methodology in the **GSSTDA** package: DSGA, gene selection and mapper process. Section Data structure describes the structure of the functions, the results and briefly explains which data are used to illustrate the package. Section Basic functions of the package presents the basic methods explained in Background. Finally, Section Conclusion concludes.

**Table 1:** GSSTDA package functions

| Function | Description |
|---|---|
| 'dsga' | For 'dsga object'. It allows the calculation of the 'disease component' of a expression matrix which consists of, through linear models, eliminating the part of the data that is considered normal or healthy and keeping only the component that is due to the disease. It is intended to precede other techniques like classification or clustering. For more information see @Nicolau2007. |
| 'results_dsga' | For 'dsga object'. It calculates the 100 genes with the highest variability in the matrix disease component between samples and use them to draw the heat map. |
| 'gene_selection' | For 'gene_selection object'. It fittings a Cox proportional hazard model to each gene, then it makes a selection of genes according to both: their variability within the database and their relationship with survival. Subsequently, with the genes selected, it calculates the values of the filtering functions for each patient. The filter function allows to summarise each vector of each individual in a single data. This function takes into account the survival associated with each gene. In particular, the implemented filter function performs the vector magnitude in the Lp-norm (as well as k powers of this magnitude) of the vector resulting of weighting each element of the column vector by the Z score obtained in the cox proportional hazard model. |
| 'mapper' | For 'mapper object'. It condenses the information of high-dimensional data sets into a combinatory graph or simplicial complex that is referred to as the skeleton of the data set. This implementation is the mapper of one dimension, i.e. using only one filter function value. |
| 'plot_mapper' | For 'mapper object'. It produces an interactive network plot using visNetwork function from the mapper results. |
| 'gsstda' | For 'gsstda object'. It integrate the three parts of the process: the preprocessing of the data [dsga process], the gene selection and the filter function [gene selection process], and the mapper algorithm [mapper process]. |

## 2 Background

### 2.1 DSGA: Disease-Specific Genomic Analysis

In PAD-S analysis the first step is use the DSGA method to transform the original data. DSGA was first developed for transcriptome array data although it can be used for other continuous omics data such as methylation data. It is intended to precede other techniques like clustering. DSGA methods obtains the amount of deviation that is present in a diseased sample compared to healthy control tissues, by isolating and separating the disease component $(D_c)$ from the normal-component $(N_c)$ portion of the data. This method involves three steps: $(i)$ Flat construction $(ii)$ Healthy State Model (HSM) construction, and $(iii)$ Disease component computation for the original data set.

#### Flat constuction

The flat construction reduces or smooth characteristics of the data that are idiosyncratic to each normal or healthy tissue sample. The data used in this step is filtering with the healthy or normal tissue samples and is organized by columns. This matrix is denote as $N$. The $R$ normal tissue gene expression vectors, denoted as $N_1, N_2, ..., N_R$, are computed as flattened vectors of the form $\hat{N}_1, \hat{N}_2, ..., \hat{N}_R$. The flattened vector $\hat{N}_i$ is calculated by fitting a 0-intercept least-squares linear model from all other $N_i$-normal tissue vectors $N_1, N_2, ..., N_{i-1}, N_{i+1}, ..., N_R$ as predictor variables, that is,

$$\hat{N}_i = \sum_{\substack{j=1 \\ j \neq i}}^{R} \beta_j N_j.$$

In this equation, $\hat{N}_i$ is the $i$-th flat vector and $\beta_j$ is the coefficient associated with normal tissue sample $N_j$. Then, after carrying out this step separately for each healthy tissue sample, we obtain the flat matrix in which each column represents a flatted healthy tissue sample as:

$$\hat{N} = \left[\hat{N}_1, \hat{N}_2, ..., \hat{N}_R\right].$$

This flat data matrix $\hat{N}$ could be expressed as the sum of a signal matrix $\hat{N}_{true}$ and a noise matrix. In the next step, a singular value decomposition will be applied to denoise it. The estimated matrix $\hat{N}_{true}$ is the so-called Healthy State Model.

#### Healthy State Model (HSM) generation

Healthy State Model (HSM) computation describes, predicts, and quantitatively captures the functioning of health systems. In this step, the singular value decomposition (SVD) of the flat data matrix $\hat{N}$ is performed. In the implementation presented in this article, a different method developed by Gavish and Donoho (2014) is used for the selection of the number of singular values than in the original DSGA, which allows the process to be automated.

In Gavish and Donoho (2014), it is assumed that the number of rows in the matrix to be decomposed must be smaller than the number of columns. This is not true for the flat data matrix $\hat{N}$ obtained in the previous step, an $\ell$-by-$R$ matrix, as we assume a larger number of genes ($\ell$) than healthy tissue samples ($R$). For this reason, in this section we work with the transpose matrix $\hat{N}^T$, an $R$-by-$\ell$ matrix, and so $R < \ell$.

Following this method, the matrix $\hat{N}^T$ can be expressed as the sum of a signal matrix and a noise matrix:

$$\hat{N}^T = \hat{N}^T_{true} + \gamma \hat{N}^T_{noise},$$

where $\hat{N}^T$ is the transpose flat matrix; $\hat{N}^T_{true}$ is an underlying low-rank matrix that contains the true signal; $\hat{N}^T_{noise}$ is a noise matrix in which entries are assumed to be a sample of i.i.d. random variables extracted from a Gaussian distribution, with zero mean and unit variance; and $\gamma$ is a parameter that indicates the magnitude of the noise.

In the DSGA, the estimation of $\hat{N}_{true}$ corresponds to the previously introduced HSM. Our aim is to estimate it using a truncated SVD of $\hat{N}^T$ which is the default way of estimating it.

The singular value decomposition of the transpose of the flat matrix is denoted as:

$$\hat{N}^T = UDV^T,$$

where $D$ is a diagonal matrix in which the elements of the diagonal are the singular values of $\hat{N}^T$

$\sigma_1, \sigma_2, ...\sigma_R$, with $R$ being the number of rows of $\hat{N}^T$, ordered from high to low; and $U$ and $V^T$ are matrices containing the left and right singular vectors in columns and rows, respectively.

As mentioned above, $\hat{N}^T_{true}$ can be estimated by truncated SVD. One option to carry out this truncated SVD is to determine a hard threshold from which singular values in $D$ are selected. This threshold depends on $\gamma$:

$$\hat{N}^T_{true} \simeq HSM^T = UD_\gamma V^T.$$

For rectangular matrices with known $\gamma$ the optimal hard threshold for singular value selection is:

$$\tau = \lambda(\beta)\sqrt{\ell}\gamma,$$

where:

- $\ell$ is the number of columns of the matrix $\hat{N}^T$.
- $\beta$ is the aspect ratio of our input matrix $\hat{N}^T$, that is, $\beta = \frac{R}{\ell}$, with $R$ and $\ell$ being the number of rows and colums respectively.
- $\lambda(\beta)$ is obtained through the following expression:

$$\lambda(\beta) = \left(2(\beta+1) + \frac{8\beta}{(\beta+1) + (\beta^2 + 14\beta + 1)^{1/2}}\right)^{1/2}.$$

On the other hand, for rectangular matrices with unknown $\gamma$ the optimal hard threshold for singular value selection is:

$$\tau = \omega(\beta)\sigma_{med},$$

where $\sigma_{med}$ is the median of the values $\{\sigma_1, \sigma_2, ...\sigma_R\}$ and $\omega(\beta)$ is defined by:

$$\omega(\beta) = \frac{\lambda(\beta)}{\mu_\beta}.$$

$\lambda(\beta)$ has already been defined above. $\mu_\beta$ is found by computing numerically the upper bound of the following definite integral in the range $[(1-\beta)^2, (1+\beta)^2]$:

$$\int_{(1-\beta)^2}^{\mu_\beta} \frac{\left[((1+\sqrt{\beta})^2 - t)\,(t - (1-\sqrt{\beta})^2)\right]^{1/2}}{2\pi t\beta} dt = \frac{1}{2}.$$

Only those singular values larger than the threshold obtained by the respective method will be kept. It is important to note that $\gamma$ is not known or predetermined in standard analyses. Nevertheless, our package is intentionally designed to enable users to assess the impact of varying $\gamma$ values, guided by external insights or theoretical assumptions about noise levels.

As already introduced, the estimation of $\hat{N}_{true}$, the new de-noised matrix, is the Healthy State Model (HSM). To construct it assume we need a diagonal matrix termed $D_\gamma = \text{diag}(\sigma_1^*, \ldots, \sigma_R^*)$ which is obtained from $D = \text{diag}(\sigma_1, \ldots, \sigma_R)$ by using that equation:

$$\sigma_i^* = \begin{cases} \sigma_i & \text{if } \sigma_i > \tau, \\ 0 & \text{otherwise.} \end{cases}$$

So, the `HSM` is defined as:

$$\hat{N}^T_{true} \simeq UD_\gamma V^T,$$
$$HSM \simeq \hat{N}_{true}.$$

### Disease component

The computation of the disease vectors for each sample in the original data set is applied. The original matrix including healthy and disease samples is expressed as $O$. Then, for each $O_i$ data vector present in $O$, a zero-intercept linear model is fitted to the columns of the $HSM$ matrix, i.e,

$$O_i = \sum_k \eta_k^{(i)} HSM_k + \varepsilon_i$$

where $\eta_k^{(i)}$ are the coefficients obtained from the best approximation of the column space of the matrix $HSM$ to $O_i$ and $\varepsilon_i$ is its corresponding residual. The estimation of $\eta_k^{(i)}$ aims to minimize the error

between disease vectors and the projection of the observed disease vectors onto the HSM. Least squares regression is used in our package but other options such as ridge regression could also be employed adapting the estimation method based on data complexity and considerations like multicollinearity and regularization needs.

Now, considering the matrix

$$O_{DSGA} = [\varepsilon_1 \cdots \varepsilon_p],$$

we remove the portion of each sample that best mimics the expression patterns of healthy tissues and remain with the vector of residuals that carry out the information about how much each gene of a particular sample deviates from the values observed in healthy tissues.

### 2.2 Selection of the genes

Once the disease component matrix $O_{DSGA}$ has been constructed, the next step consists of the selection of genes for downstream analysis. The gene selection procedure is based on the variability of the expression of each gene across the $O_{DSGA}$ matrix and the degree of association of each gene with either disease-free or overall survival.

Associations between the levels of the expression of each gene with survival are computed employing univariate Cox proportional hazard models using the vectors corresponding to the pathological tissue samples of the original expression matrix. The $Z$-scores derived from the Cox proportional hazard models fits representing the degree of association of each gene with survival are then stored in the vector $Z_{cox}$. While it is often essential to check compliance with the proportional hazards assumption, we believe it is not for the PAD-S analysis, as the $Z$ scores are merely used as weights.

The standard deviation of each gene in the pathological tissue samples is then calculated using the $O_{DSGA}$ matrix. The vector of standard deviations is then stored as $O_{sd}$. To avoid values between 1 and $-1$, $+1$ is added to all positive values in vectors $Z_{cox}$ and $O_{sd}$, and $-1$ is added to all negative values of vector $Z_{cox}$. The element-wise product between $Z_{cox}$ and $O_{sd}$ matrices is computed. Two methods of gene selection are proposed. In the first of them, the top and bottom $n$ genes of the distribution of this product are selected for further analysis. In the other option, the $n$ genes with the highest absolute value of this product are chosen.

Then, the $O_{DSGA}$ matrix is filtered keeping the selected genes. This matrix constitute the input for the Mapper algorithm.

### 2.3 Mapper

Mapper (Singh et al. (2007), Lum et al. (2013)) is a tool derived from TDA that allows to condense high-dimensional data sets into a combinatorial graph capturing the shape of the data. Some of its properties is that it is insensitivity to metric, noise robustness and it allows multiscale representations (Carlsson (2009)).

This section explains $(i)$ the PAD-S filter function that needs to be applied before Mapper, $(ii)$ the the One Dimensional Mapper Algorithm itself and $(iii)$ the different options for selecting the optimal number of clusters that can be used in the cluster step.

#### Filter function in PAD-S

To use the Mapper, a filter function denoted as $f$ is required. This function summarizes each point in our data set denoted by $X$ to $\mathbb{R}$.

$$f : X \to \mathbb{R}.$$

The selection of this filter function is of particular relevance and must be adapted to the nature of the problem under study. While more than one filter function can be used, resulting in a multi-dimensional mapper, the PAD-S methodology employs a single filter function to focus analysis on survival-associated gene expressions, simplifying the computational model, enhancing interpretability for clinical decision-making, and ensuring robustness through theoretical and empirical validation. This choice ensures a targeted analysis that is directly relevant to the biological and clinical questions at hand. The mapper developers suggest using density estimators, eccentricity, or graph Laplacians as filter functions (Singh et al. (2007)), although any function deemed appropriate can be used. In PAD analysis the filter function of choice was the vector of magnitude in the $L^p$ norm, as well as $k$ powers of this magnitude:

$$f(O^i_{DSGA}) = f(O^i_{DSGA}; p, k) = \left[\sum_r |g_r^{(i)}|^p\right]^{k/p}.$$

where $O^i_{DSGA}$ denotes the $i$-th column vector of our disease component matrix (corresponding to individual $i$), which contains the values of each selected gene in that sample with coordinates $g_r^{(i)}$. Note that if $k = 1$ and $p = 2$, the function simply computes the standard (Euclidean) vector magnitude of each column.

In PAD-S analysis the filter function takes account the magnitude of the association between the expression level of a particular gene and survival. In particular the filter function is defined as follows:

$$f(O^i_{DSGA}; p, k) = \left[\sum_r |z_r \cdot g_r^{(i)}|^p\right]^{k/p},$$

where $z_r$ is the $z$-value derived from the Cox proportional hazard models analysis and $g_r^{(i)}$ the $i$-th disease component value (corresponding to individual $i$) for $r$-th feature.

Note that in this case, to avoid values between 1 and $-1$, $+1$ is added to all positive values in vectors $Z_{cox}$ and $O^i_{DSGA}$, and $-1$ is added to all negative values of both vectors. Therefore, the amount of deviation of each gene to the `HSM` is multiplied by the degree of association of this particular gene with survival.

### The One Dimensional Mapper Algorithm

In PAD-S methodology, the Mapper is subsequently applied on $O_{DSGA}$ using its filter function defined in the previous section Filter function in PAD-S.

In addition, Mapper requires the use of a specific distance metric and clustering type for the clustering step. To this end, the **GSSTDA** package implementation allows choosing between correlation and Euclidean distances as distance metrics and among single linkage, average linkage, complete linkage or k-medioids as clustering methods.

Once the values of the filter function have been calculated, the range of filter function values is divided into overlapping intervals. Subsequently, for each interval, the clustering of the individuals that have a value of the filter function that is within that interval is performed. After clustering each interval, the graph is constructed. Each cluster is a node and those that share at least one individual are joined by an edge. This is possible because the intervals are overlapping.

The output graph $G = G(V, E)$ is then defined putting each cluster as a node (or vertex) of the nodes that share samples are connected with an edge.

### Methods for the identification of the optimal number of clusters

Two different options for the selection of the optimal number of clusters are offered in the package presented. One of them is the method originally used in Mapper, which is also the one used by the PAD analysis. It selects the number of clusters by constructing a histogram of the cluster edge lengths using $k$ bins. An empty interval is usually generated in this histogram. The edge length of the start of this interval is chosen. The clusters with a greater edge length than this one are chosen. In addition to this method, the option of using the Silhouettes (Rousseeuw (1987)) method is also available.

In this method, the first step is as follows. For a particular data point $x$ included in cluster $C_i$ we first define $a(x)$ as the average distance of data point $x$ to all other points $y$ in the cluster. Thus,

$$a(x) = \frac{1}{|C_i| - 1} \sum_{\substack{y \in C_i \\ j \neq i}} d(x, y).$$

Then, $b(x)$ is defined as the minimum mean difference of point $x$ to any other cluster $C$ of which $x$ is not a member, as the average distance from $x$ to all points $y$ in $C$ where $C \neq C_i$.

$$b(x) = \min_{\substack{1 \leq k \leq \ell \\ k \neq i}} \frac{1}{|C_k|} \sum_{y \in C_k} d(x, y).$$

The cluster which has the minimum average distance to point $x$ is said to be the neighbor cluster of $x$.

In the context of our analysis, the function $d(x, y)$ represents the distance measure used to quantify the similarity between data points $x$ and $y$ in our dataset. This distance metric is critical for clustering and analyzing the structure of the data. Specifically, $d(x, y)$ could be any metric that suits the nature of the data and the specific requirements of the analysis, such as Euclidean, Manhattan, or cosine similarity. In this manuscript, unless otherwise specified, we use the Euclidean distance, which is

defined as

$$d(x,y) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2},$$

where $x_i$ and $y_i$ are the components of vectors $x$ and $y$ respectively. This choice is motivated by its geometric interpretability and computational efficiency in handling numerical data typical in omics studies.

The concept of Silhouette for point $x$ is defined by

$$s(x) = \frac{b(x) - a(x)}{\max\{a(x), b(x)\}}, \quad \text{if} \quad |C_i > 1|,$$

that is, if the number of elements in the cluster is larger than 1. When a particular cluster $C_i$ contains only a single object, it is unclear how $a(x)$ should be defined, and therefore we set the silhouette to 0.

$$s(x) = 0, \quad \text{if} \quad |C_i| = 1.$$

The average of the $s(x)$ values of all points grouped in a particular cluster $C_i$, i.e.

$$\bar{s}_i = \frac{1}{|C_i|} \sum_{x \in C_i} s(x)$$

indicates how well grouped are the members of this cluster, whereas the average of all data points

$$\bar{s} = \frac{1}{|\cup_i C_i|} \sum_{x \in \cup_i C_i} s(x)$$

indicates how well the available data points have been clustered in general.

To select the optimum number of clusters within each interval of the filter function, the average silhouette values $\bar{s}$ are computed for all possible partitions from 2 to $n-1$, where $n$ is the number of samples within a specific interval. Then the $n$ that produces the highest value of $\bar{s}$ and that exceeds a specific threshold is selected as the optimum number of clusters. The threshold of 0.25 for $\bar{s}$ has been chosen based on standard practice, recognizing it as a moderate value that reflects adequate separation and cohesion within clusters, which is crucial for ensuring both statistical significance and biological or clinical relevance of the clusters. If a different threshold is considered, it should be adjusted based on the study's specific objectives and the dataset's characteristics, with higher thresholds used for stronger delineation between clusters and lower thresholds suitable for exploratory analyses or overlapping data categories. To implement an alternative threshold, one should: 1) Analyze the distribution of silhouette scores for various thresholds to understand the impact on cluster structures, 2) Validate the stability and validity of these clusters using additional internal metrics, 3) Consult with domain experts to align the threshold with biological or clinical importance, and 4) Conduct a sensitivity analysis to ensure robustness of the results. If no partition produces an $\bar{s}$ exceeding the chosen threshold, all samples are then assigned to a unique cluster, facilitating the clear identification of distinct groupings within the data.

## 3 Data structure

Data are managed as a regular R `matrix` in the **GSSTDA** functions. The main functions of the GSSTDA package are `dsga()`, `gene_selection()`, `mapper()` and `gsstda()`, which return structured objects named `dsga_object`, `gene_selection_object`, `mapper_object` and `GSSTDA_object`, respectively. These objects contain fields with relevant information such as the genes selected for the mapper algorithm or the arguments introduced by the user in the function call.

The main fields of the all objects are the following:

- `full_data`: Matrix containing normalized gene expression data. The columns correspond to the patients and the rows to the genes.
- `survival_time`: Numerical vector of the same length as the number of columns of `full_data`. In addition, the patients must be in the same order as in `full_data`. For the patients whose sample is pathological should be indicated the time between the disease diagnosis and event (death, relapse or other). If the event has not occurred it should be indicated the time until the end of follow-up. Patients whose sample is from healthy tissue must have an `NA` value.
- `survival_event`: Numerical vector of the same length as the number of columns of `full_data`. Patients must be in the same order as in `full_data`. For the the patients with pathological sample should be indicated whether the event has occurred (1) or not (0). Only these values are

valid and healthy patients must have an NA value.
- case_tag: Character vector of the same length as the number of columns of full_data. Patients must be in the same order as in full_data. It must be indicated for each patient whether its sample is from pathological or healthy tissue. One value should be used to indicate whether the patient's sample is healthy and another value should be used to indicate whether the patient's sample is pathological. The user will then be asked which one indicates whether the patient is healthy. Only two values are valid in the vector in total.

### 3.1 Data set

```
# We load the data
data("full_data")
data("survival_time")
data("survival_event")
data("case_tag")
```

We illustrate all the functions presented in this paper resorting to a single data set (full_data) with its corresponding vectors (survival_time, survival_event and case_tag) available in the **GSSTDA** package. The original data are from the study GSE42568 available in https://www.ncbi.nlm.nih.gov/geo/query/acc.cgi?acc=GSE42568. This data was processed normalizing the genes expression. We carry out background correction, summarizing, and quantile normalization of data set using the fRMA method implemented in the **fRMA** package. Our data set consists of 20,825 genes and 121 patients. Compliant with CRAN publication requirements, which necessitate demonstrating our package's functionality on a manageable subset of data, we have strategically reduced the number of genes in our dataset to 4165. This ensures that our analysis remains computationally feasible while maintaining a representative sample. The examples were executed in the posit (RStudio Cloud) environment, which has a system configuration including 3.75 GB RAM, Intel Xeon E5-2673 v3 @ 2.40 GHz processor, and Ubuntu 20.04 operating system. The calculations were completed in 78 seconds, demonstrating the package's efficiency. The dataset thus includes gene expression profiling from 104 breast cancer and 17 normal breast biopsies, exemplifying the package's utility across typical clinical samples without compromising the integrity and representativeness of the results. The first four patients and their ten genes are shown below for the four data sets:

| | GSM1045191 | GSM1045192 | GSM1045193 | GSM1045194 |
|---|---|---|---|---|
| A1BG | 5.769 | 5.912 | 5.829 | 5.868 |
| A1BG-AS1 | 5.340 | 5.417 | 4.986 | 5.059 |
| A1CF | 4.732 | 4.780 | 4.870 | 5.780 |
| A2M | 11.053 | 10.794 | 5.897 | 11.029 |
| A2M-AS1 | 5.410 | 5.223 | 5.063 | 4.740 |
| A2ML1 | 4.375 | 4.680 | 4.888 | 4.629 |
| A2MP1 | 4.712 | 6.083 | 6.943 | 6.452 |
| A4GALT | 6.791 | 7.086 | 7.562 | 7.604 |
| A4GNT | 4.456 | 4.331 | 4.720 | 4.966 |
| AA06 | 4.990 | 5.903 | 5.741 | 5.886 |

| Vectors | Extract of the values |
|---|---|
| 'survival_time' | NA NA NA 99.417 24.805 99.023 13.339 73.101 8.279 70.242 80.46 |
| 'survival_event' | NA NA NA 0 1 0 1 0 1 0 0 |
| 'case_tag' | NT NT NT T T T T T T T T |

## 4 Basic functions of the package

In this section, we introduce the main functions of the library related to Gene Structure Survival using Topological Data Analysis.

### 4.1 DSGA object

The basic model of Disease-Specific Genomic Analysis that we explained in subsection DSGA: Disease-Specific Genomic Analysis can be implemented in R using the function dsga():

```
dsga_object <- dsga(full_data, survival_time, survival_event, case_tag, gamma)
```

The minimum arguments of this function are `full_data`, `survival_time`, `survival_event` and `case_tag`, which were explained in the previous section Data structure. The parameter `gamma` is optional. It indicates the magnitude of the noise assumed in the flat data matrix for the generation of the Healthy State Model. If it takes the value `NA` the magnitude of the noise is assumed to be unknown. We recommend using the default value `NA`. Additionally, the function requests the following information from the user at runtime:

- Enter "yes" if the columns are the patients and the row the genes or "no" in other cases. In the "no" case, the library will automatically change the columns and rows to fulfill this condition.
- Enter which is the tag of the patients whose sample is from healthy tissue that is stored in the `case_tag` vector. It is also possible to introduce it in the function as a parameter.

This function returns an object composed of:

- `normal_space`: The matrix with the normal space (linear space generated from normal tissue samples).
- `matrix_disease`: The disease component matrix that contains the disease component of all patients (`dsga_object[["matrix_disease_component"]]`).

As an example, using data and vectors from subsection Data structure, we next process the genes expression using the suitable code as follows. Results are returned as an `dsga object`, as explained in section Data structure.

```
dsga_object <- dsga(full_data = full_data,
                    survival_time = survival_time,
                    survival_event = survival_event,
                    case_tag = case_tag)

dsga_information <- results_dsga(
                   matrix_disease_component = dsga_object[["matrix_disease_component"]],
                    case_tag = case_tag)
```

The DSGA information are plotted using `results_dsga()` function (see Figure 1).

```
print(dsga_information)
```

```
#>   [1] "CPB1"     "AGR3"     "BMPR1B"   "ANKRD30A" "CP"       "CEACAM6"
#>   [7] "ADH1B"    "CXCL13"   "COL11A1"  "CHI3L1"   "CYP4Z1"   "AGR2"
#>  [13] "CD36"     "CLIC6"    "CYP4X1"   "C19orf33" "AREG"     "APOD"
#>  [19] "ADIPOQ"   "BEX1"     "AGTR1"    "CALML5"   "CLDN8"    "CYP4B1"
#>  [25] "CRISP3"   "DACH1"    "CXCL14"   "AFF3"     "CYP2B7P"  "CAPN8"
#>  [31] "COMP"     "CLSTN2"   "CCL19"    "CFB"      "CYP2T1P"  "CSTA"
#>  [37] "AZGP1"    "CRISPLD1" "CLGN"     "ANXA3"    "CGA"      "CHGB"
#>  [43] "CXCL9"    "AKR1C2"   "ACTG2"    "ALOX15B"  "AQP3"     "CLCA2"
#>  [49] "COL2A1"   "CXCL11"   "CA12"     "C15orf48" "CA2"      "CYP4Z2P"
#>  [55] "ASPN"     "AR"       "CHRDL1"   "AKR1C1"   "BBOX1"    "ABCA8"
#>  [61] "CYP4F8"   "CT83"     "CXCL10"   "ABAT"     "CEACAM5"  "ADAMTS15"
#>  [67] "ANLN"     "CLDN1"    "CPE"      "DCLK1"    "CELSR1"   "COL10A1"
#>  [73] "CFD"      "CNTNAP2"  "CLDN11"   "APOBEC3B" "ALDH3B2"  "C16orf54"
#>  [79] "CDO1"     "ANKRD30B" "COL14A1"  "ARHGAP36" "CECR2"    "BAMBI"
#>  [85] "CCND1"    "ADGRG6"   "AKR1C3"   "ABCC13"   "C16orf89" "CCL8"
#>  [91] "CNKSR3"   "DCD"      "CCL5"     "CEP55"    "CST6"     "ARNT2"
#>  [97] "BEX5"     "COL4A5"   "CLEC3A"   "ARMT1"
```

### 4.2 Gene selection object

The basic model of gene selection that we explained in subsection Selection of the genes can be implemented in R using the function `gene_selection()`. Furthermore, this function for convenience calculates the filter function values for mapper. It uses the filter function developed for the PAD-S:

```
gene_selection_object <- gene_selection(data, gen_select_type, percent_gen_select)
```

The fields of the `gene_selection()` are the following:

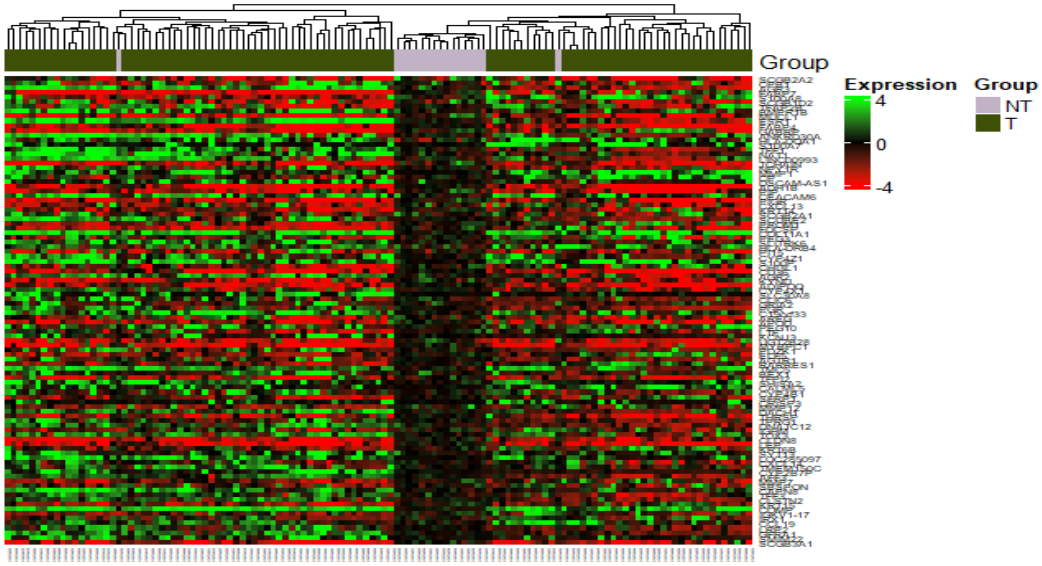


**Figure 1:** Heatmap of the disease component matrix (result of the 'dsga' function) after selecting the 100 genes with the highest variability between samples. Each column represents a patient sample and each row represents a gene. The color scale reflects the values of the disease component, with warmer colors indicating higher expression values than in healthy tissue and cooler colors indicating lower expression. Key: 'NT' stands for 'Non-Tumor' – representing normal breast biopsy samples; 'T' stands for 'Tumor' – representing breast cancer biopsy samples. In addition, the result of the hierarchical clustering of the samples using the euclidean distance and the complete method is included. This visualization aids in distinguishing between the disease component profiles of tumor and non-tumor samples, highlighting the difference between the two types after preprocessing with DSGA.

- `gen_select_type`: Options on how to select the genes to be used in the mapper:
  - `Abs`: The genes with the highest absolute value are chosen.
  - `Top_Bot`: Half of the selected genes are those with the highest value (positive value, i.e. worst survival prognosis) and the other half are those with the lowest value (negative value, i.e. best prognosis). "Top_Bot" default option.
- `percent_gen_select`: Percentage (from zero to one hundred) of genes to be selected to be used in mapper. 10 is the default option.
- `data`: The `data` argument could be two options:
  - `dsga_object`: This is use if `dsga()` function was previously executed.
  - `data_object`: Create a object `data_object` with the require information (see following R example).

This function returns a `gene_selection object` composed of:

- `cox_all_matrix`: A matrix with the results of the application of proportional hazard models: the regression coefficients, the odds ratios, the standard errors of each coefficient, the Z values and the p-values for each Z value)
- `genes_selected`: A vector with the name of the selected genes,
- `genes_disease_ component`: The matrix of disease components with only the rows of the selected genes.
- `filter_values`: The vector of the values of the filter function.

We introduce two distinct options for data input when utilizing our analytical package. These input methods are designed to accommodate different data structures and user preferences, ensuring flexibility and accessibility for a variety of research needs.

### First option: dsga_object

The first option for data input involves using a predefined object type, `dsga_object`. This object is specifically tailored for users who have pre-processed their data using the Disease-Specific Genomic Analysis (DSGA) method. The `dsga_object` contains all necessary attributes and methods

for subsequent analysis steps within our package, ensuring that users can seamlessly integrate their DSGA-processed data.

As an example, we next create a `gene_selection object` for the first option of data input:

```
dsga_object <- dsga(full_data = full_data,
                    survival_time = survival_time,
                    survival_event = survival_event,
                    case_tag = case_tag)
gene_selection_object <- gene_selection(data =  dsga_object,
                                        gen_select_type = "Top_Bot",
                                        percent_gen_select = 10)
```

### Second option: data_object

The second option allows users to input their data as a generic data_object. This approach is intended for users who may have their data prepared in different formats or who require a more general input method that does not depend on the specific preprocessing steps like those required by the DSGA. The data_object should contain all the necessary data fields, such as gene expression data, survival data, and any other relevant clinical parameters, formatted in a way that can be directly utilized by our package.

The example of the second option of data input is:

```
# Create data object
data_object <- list(full_data = full_data,
                    survival_time = survival_time,
                    survival_event = survival_event,
                    case_tag = case_tag)
class(data_object) <- "data_object"

#Select gene from data object
gene_selection_object <- gene_selection(data = data_object,
                                        gen_select_type = "Top_Bot",
                                        percent_gen_select = 10)
```

Each of these input options is designed to provide the flexibility needed to handle diverse data types and preprocessing techniques. This ensures that our package can be effectively used in various scenarios, catering to both advanced users with specific preprocessing needs and those seeking more general data input methods.

## 4.3 Mapper object

The basic model of mapper that we explained in subsection Mapper can be implemented in R using the function `mapper()`:

```
mapper_object <- mapper(data,
                        filter_values,
                        num_intervals,
                        percent_overlap,
                        distance_type,
                        clustering_type,
                        num_bins_when_clustering,
                        linkage_type,
                        optimal_clustering_mode,
                        silhouette_threshold)
```

The fields of the mapper object are the following:

- `data`: Input matrix with which the analysis is performed. The user is asked whether the columns correspond to patients or features. In GSSTDA, this matrix corresponds to the matrix of disease components with only the rows of the selected genes. It can be any other matrix to which mapper is to be applied.

- `filter_values`: Vector obtained after applying the filtering function to the input matrix, i.e, a vector with the filtering function values for each included sample.
- `num_intervals`: Number of intervals used to create the first sample partition based on filtering values. "5" is the default option.
- `percent_overlap`: Percentage of overlap between intervals. Expressed as a percentage. "40" is the default option.
- `distance_type`: Type of distance to be used for clustering.
  - `correlation`: "correlation" is the default option.
  - `euclidean`
- `clustering_type`: Type of clustering method.
  - `hierarchical`: "hierarchical" is the default option.
  - `PAM`: "PAM" ("partition around medoids") option.
- `num_bins_when_clustering`: Number of bins to generate the histogram employed by the standard optimal number of cluster finder method. Parameter not necessary if the `optimal_clustering_mode` option is "silhouette" or the `clustering_type` is "PAM". "10" is the default option.
- `linkage_type`: Linkage criteria used in hierarchical clustering. Only necessary for hierarchical clustering.
  - `single`: Single-linkage clustering. "single" is the default option.
  - `complete`: Complete-linkage clustering.
  - `average`: Average linkage clustering (or UPGMA).
- `optimal_clustering_mode`: Method for selection optimal number of clusters. It is only necessary if the chosen type of algorithm is "hierarchical". In this case, choose between "standard" (the method used in the original mapper article) or "silhouette". In the case of the "PAM" algorithm, the method will always be "silhouette".
- `silhouette_threshold`: Minimum value of $\bar{s}$ that a set of clusters must have to be chosen as optimal. Within each interval of the filter function, the average silhouette values $\bar{s}$ are computed for all possible partitions from 2 to $n-1$, where $n$ is the number of samples within a specific interval. The $n$ that produces the highest value of $\bar{s}$ and that exceeds a specific threshold is selected as the optimum number of clusters. If no partition produces an $\bar{s}$ exceeding the chosen threshold, all samples are then assigned to a unique cluster. We recommend to use the default value of 0.25.

This function returns a `mapper object` which contains:

- `interval_data`: The values of the intervals.
- `sample_in_level`: The samples included in each interval.
- `clustering_all_levels`: The information about the cluster to which the individuals in each interval belong.
- `node_samples`: A list including the individuals contained in each detected node.
- `node_sizes`: Their size.
- `node_average_filt`: The average of the filter function values of the individuals of each node.
- `adj_matrix`: The adjacency matrix linking the nodes.

As an example, using results of `dsga()` and `gene_selection()`, we next process the `mapper()` using the suitable code as follows:

The information obtained from the mapper object are showed using `print`:

```
print(mapper_object)

#> $interval_data
#> $interval_data$Level_1
#> [1] 612.7152 627.1164
#>
#> $interval_data$Level_2
#> [1] 621.3959 635.6971
#>
#> $interval_data$Level_3
#> [1] 629.9766 644.2778
#>
#> $interval_data$Level_4
#> [1] 638.5573 652.8585
```

```
#>
#> $interval_data$Level_5
#> [1] 647.1381 661.4393
#>
#> $interval_data$Level_6
#> [1] 655.7188 670.0200
#>
#> $interval_data$Level_7
#> [1] 664.2995 678.6007
#>
#> $interval_data$Level_8
#> [1] 672.8802 687.1814
#>
#> $interval_data$Level_9
#> [1] 681.4609 695.7622
#>
#> $interval_data$Level_10
#> [1] 690.0417 704.4429
#>
#>
#> $sample_in_level
#> $sample_in_level$Level_1
#> [1] "GSM1045193" "GSM1045217" "GSM1045302"
#>
#> $sample_in_level$Level_2
#> [1] "GSM1045192" "GSM1045217" "GSM1045288"
#>
#> $sample_in_level$Level_3
#> [1] "GSM1045192" "GSM1045194" "GSM1045221"
#>
#> $sample_in_level$Level_4
#> [1] "GSM1045194" "GSM1045211" "GSM1045225" "GSM1045243" "GSM1045250"
#> [6] "GSM1045282"
#>
#> $sample_in_level$Level_5
#>  [1] "GSM1045211" "GSM1045213" "GSM1045225" "GSM1045242" "GSM1045243"
#>  [6] "GSM1045250" "GSM1045282" "GSM1045285" "GSM1045293" "GSM1045301"
#>
#> $sample_in_level$Level_6
#>  [1] "GSM1045203" "GSM1045206" "GSM1045209" "GSM1045212" "GSM1045213"
#>  [6] "GSM1045214" "GSM1045224" "GSM1045227" "GSM1045237" "GSM1045241"
#> [11] "GSM1045242" "GSM1045248" "GSM1045255" "GSM1045264" "GSM1045266"
#> [16] "GSM1045274" "GSM1045285" "GSM1045293" "GSM1045301"
#>
#> $sample_in_level$Level_7
#>  [1] "GSM1045200" "GSM1045201" "GSM1045203" "GSM1045206" "GSM1045209"
#>  [6] "GSM1045218" "GSM1045222" "GSM1045227" "GSM1045229" "GSM1045232"
#> [11] "GSM1045233" "GSM1045237" "GSM1045239" "GSM1045240" "GSM1045241"
#> [16] "GSM1045244" "GSM1045252" "GSM1045255" "GSM1045257" "GSM1045259"
#> [21] "GSM1045264" "GSM1045269" "GSM1045270" "GSM1045276" "GSM1045278"
#> [26] "GSM1045294" "GSM1045298" "GSM1045305" "GSM1045306"
#>
#> $sample_in_level$Level_8
#>  [1] "GSM1045191" "GSM1045195" "GSM1045196" "GSM1045197" "GSM1045198"
#>  [6] "GSM1045199" "GSM1045200" "GSM1045201" "GSM1045204" "GSM1045205"
#> [11] "GSM1045207" "GSM1045210" "GSM1045216" "GSM1045218" "GSM1045219"
#> [16] "GSM1045222" "GSM1045228" "GSM1045231" "GSM1045232" "GSM1045233"
#> [21] "GSM1045238" "GSM1045239" "GSM1045240" "GSM1045245" "GSM1045246"
#> [26] "GSM1045252" "GSM1045254" "GSM1045257" "GSM1045258" "GSM1045259"
#> [31] "GSM1045260" "GSM1045261" "GSM1045263" "GSM1045267" "GSM1045270"
#> [36] "GSM1045271" "GSM1045275" "GSM1045280" "GSM1045283" "GSM1045284"
#> [41] "GSM1045287" "GSM1045296" "GSM1045298" "GSM1045300" "GSM1045303"
#> [46] "GSM1045306" "GSM1045309"
#>
```

```
#> $sample_in_level$Level_9
#>  [1] "GSM1045191" "GSM1045195" "GSM1045196" "GSM1045197" "GSM1045198"
#>  [6] "GSM1045199" "GSM1045202" "GSM1045204" "GSM1045205" "GSM1045208"
#> [11] "GSM1045215" "GSM1045220" "GSM1045228" "GSM1045231" "GSM1045235"
#> [16] "GSM1045236" "GSM1045246" "GSM1045249" "GSM1045253" "GSM1045256"
#> [21] "GSM1045258" "GSM1045260" "GSM1045262" "GSM1045265" "GSM1045267"
#> [26] "GSM1045268" "GSM1045271" "GSM1045273" "GSM1045277" "GSM1045279"
#> [31] "GSM1045280" "GSM1045281" "GSM1045286" "GSM1045289" "GSM1045290"
#> [36] "GSM1045291" "GSM1045292" "GSM1045295" "GSM1045297" "GSM1045299"
#> [41] "GSM1045304" "GSM1045307" "GSM1045309" "GSM1045310"
#>
#> $sample_in_level$Level_10
#>  [1] "GSM1045202" "GSM1045215" "GSM1045220" "GSM1045223" "GSM1045226"
#>  [6] "GSM1045230" "GSM1045234" "GSM1045235" "GSM1045236" "GSM1045247"
#> [11] "GSM1045249" "GSM1045251" "GSM1045256" "GSM1045262" "GSM1045265"
#> [16] "GSM1045268" "GSM1045272" "GSM1045273" "GSM1045277" "GSM1045279"
#> [21] "GSM1045286" "GSM1045289" "GSM1045291" "GSM1045292" "GSM1045297"
#> [26] "GSM1045304" "GSM1045307" "GSM1045308" "GSM1045310" "GSM1045311"
#>
#>
#> $clustering_all_levels
#> $clustering_all_levels$Level_1
#> GSM1045193 GSM1045217 GSM1045302
#>          1          2          2
#>
#> $clustering_all_levels$Level_2
#> GSM1045192 GSM1045217 GSM1045288
#>          1          1          1
#>
#> $clustering_all_levels$Level_3
#> GSM1045192 GSM1045194 GSM1045221
#>          1          1          1
#>
#> $clustering_all_levels$Level_4
#> GSM1045194 GSM1045211 GSM1045225 GSM1045243 GSM1045250 GSM1045282
#>          1          1          1          1          1          1
#>
#> $clustering_all_levels$Level_5
#> GSM1045211 GSM1045213 GSM1045225 GSM1045242 GSM1045243 GSM1045250 GSM1045282
#>          1          1          1          1          1          1          1
#> GSM1045285 GSM1045293 GSM1045301
#>          1          1          1
#>
#> $clustering_all_levels$Level_6
#> GSM1045203 GSM1045206 GSM1045209 GSM1045212 GSM1045213 GSM1045214 GSM1045224
#>          1          1          2          3          4          5          6
#> GSM1045227 GSM1045237 GSM1045241 GSM1045242 GSM1045248 GSM1045255 GSM1045264
#>          7          4          2          7          3          2          3
#> GSM1045266 GSM1045274 GSM1045285 GSM1045293 GSM1045301
#>          5          8          4          3          9
#>
#> $clustering_all_levels$Level_7
#> GSM1045200 GSM1045201 GSM1045203 GSM1045206 GSM1045209 GSM1045218 GSM1045222
#>          1          1          1          1          1          1          1
#> GSM1045227 GSM1045229 GSM1045232 GSM1045233 GSM1045237 GSM1045239 GSM1045240
#>          1          1          1          1          1          1          1
#> GSM1045241 GSM1045244 GSM1045252 GSM1045255 GSM1045257 GSM1045259 GSM1045264
#>          1          1          1          1          1          1          1
#> GSM1045269 GSM1045270 GSM1045276 GSM1045278 GSM1045294 GSM1045298 GSM1045305
#>          1          1          1          1          1          1          1
#> GSM1045306
#>          1
#>
#> $clustering_all_levels$Level_8
```

```
#> GSM1045191 GSM1045195 GSM1045196 GSM1045197 GSM1045198 GSM1045199 GSM1045200
#>          1          1          1          1          1          1          1
#> GSM1045201 GSM1045204 GSM1045205 GSM1045207 GSM1045210 GSM1045216 GSM1045218
#>          1          1          1          1          1          1          1
#> GSM1045219 GSM1045222 GSM1045228 GSM1045231 GSM1045232 GSM1045233 GSM1045238
#>          1          1          1          1          1          1          1
#> GSM1045239 GSM1045240 GSM1045245 GSM1045246 GSM1045252 GSM1045254 GSM1045257
#>          1          1          1          1          1          1          1
#> GSM1045258 GSM1045259 GSM1045260 GSM1045261 GSM1045263 GSM1045267 GSM1045270
#>          1          1          1          1          1          1          1
#> GSM1045271 GSM1045275 GSM1045280 GSM1045283 GSM1045284 GSM1045287 GSM1045296
#>          1          1          1          1          1          1          1
#> GSM1045298 GSM1045300 GSM1045303 GSM1045306 GSM1045309
#>          1          1          1          1          1
#>
#> $clustering_all_levels$Level_9
#> GSM1045191 GSM1045195 GSM1045196 GSM1045197 GSM1045198 GSM1045199 GSM1045202
#>          1          1          1          1          1          1          1
#> GSM1045204 GSM1045205 GSM1045208 GSM1045215 GSM1045220 GSM1045228 GSM1045231
#>          1          1          1          1          1          1          1
#> GSM1045235 GSM1045236 GSM1045246 GSM1045249 GSM1045253 GSM1045256 GSM1045258
#>          1          1          1          1          1          1          1
#> GSM1045260 GSM1045262 GSM1045265 GSM1045267 GSM1045268 GSM1045271 GSM1045273
#>          1          1          1          1          1          1          1
#> GSM1045277 GSM1045279 GSM1045280 GSM1045281 GSM1045286 GSM1045289 GSM1045290
#>          1          1          1          1          1          1          1
#> GSM1045291 GSM1045292 GSM1045295 GSM1045297 GSM1045299 GSM1045304 GSM1045307
#>          1          1          1          1          1          1          1
#> GSM1045309 GSM1045310
#>          1          1
#>
#> $clustering_all_levels$Level_10
#> GSM1045202 GSM1045215 GSM1045220 GSM1045223 GSM1045226 GSM1045230 GSM1045234
#>          1          1          1          1          1          1          2
#> GSM1045235 GSM1045236 GSM1045247 GSM1045249 GSM1045251 GSM1045256 GSM1045262
#>          1          1          1          1          1          1          1
#> GSM1045265 GSM1045268 GSM1045272 GSM1045273 GSM1045277 GSM1045279 GSM1045286
#>          1          1          1          1          1          1          1
#> GSM1045289 GSM1045291 GSM1045292 GSM1045297 GSM1045304 GSM1045307 GSM1045308
#>          1          1          1          1          1          1          1
#> GSM1045310 GSM1045311
#>          1          1
#>
#>
#> $node_samples
#> $node_samples$Node_1
#> [1] "GSM1045193"
#>
#> $node_samples$Node_2
#> [1] "GSM1045217" "GSM1045302"
#>
#> $node_samples$Node_3
#> [1] "GSM1045192" "GSM1045217" "GSM1045288"
#>
#> $node_samples$Node_4
#> [1] "GSM1045192" "GSM1045194" "GSM1045221"
#>
#> $node_samples$Node_5
#> [1] "GSM1045194" "GSM1045211" "GSM1045225" "GSM1045243" "GSM1045250"
#> [6] "GSM1045282"
#>
#> $node_samples$Node_6
#>  [1] "GSM1045211" "GSM1045213" "GSM1045225" "GSM1045242" "GSM1045243"
#>  [6] "GSM1045250" "GSM1045282" "GSM1045285" "GSM1045293" "GSM1045301"
```

```
#>
#> $node_samples$Node_7
#> [1] "GSM1045203" "GSM1045206"
#>
#> $node_samples$Node_8
#> [1] "GSM1045209" "GSM1045241" "GSM1045255"
#>
#> $node_samples$Node_9
#> [1] "GSM1045212" "GSM1045248" "GSM1045264" "GSM1045293"
#>
#> $node_samples$Node_10
#> [1] "GSM1045213" "GSM1045237" "GSM1045285"
#>
#> $node_samples$Node_11
#> [1] "GSM1045214" "GSM1045266"
#>
#> $node_samples$Node_12
#> [1] "GSM1045224"
#>
#> $node_samples$Node_13
#> [1] "GSM1045227" "GSM1045242"
#>
#> $node_samples$Node_14
#> [1] "GSM1045274"
#>
#> $node_samples$Node_15
#> [1] "GSM1045301"
#>
#> $node_samples$Node_16
#>  [1] "GSM1045200" "GSM1045201" "GSM1045203" "GSM1045206" "GSM1045209"
#>  [6] "GSM1045218" "GSM1045222" "GSM1045227" "GSM1045229" "GSM1045232"
#> [11] "GSM1045233" "GSM1045237" "GSM1045239" "GSM1045240" "GSM1045241"
#> [16] "GSM1045244" "GSM1045252" "GSM1045255" "GSM1045257" "GSM1045259"
#> [21] "GSM1045264" "GSM1045269" "GSM1045270" "GSM1045276" "GSM1045278"
#> [26] "GSM1045294" "GSM1045298" "GSM1045305" "GSM1045306"
#>
#> $node_samples$Node_17
#>  [1] "GSM1045191" "GSM1045195" "GSM1045196" "GSM1045197" "GSM1045198"
#>  [6] "GSM1045199" "GSM1045200" "GSM1045201" "GSM1045204" "GSM1045205"
#> [11] "GSM1045207" "GSM1045210" "GSM1045216" "GSM1045218" "GSM1045219"
#> [16] "GSM1045222" "GSM1045228" "GSM1045231" "GSM1045232" "GSM1045233"
#> [21] "GSM1045238" "GSM1045239" "GSM1045240" "GSM1045245" "GSM1045246"
#> [26] "GSM1045252" "GSM1045254" "GSM1045257" "GSM1045258" "GSM1045259"
#> [31] "GSM1045260" "GSM1045261" "GSM1045263" "GSM1045267" "GSM1045270"
#> [36] "GSM1045271" "GSM1045275" "GSM1045280" "GSM1045283" "GSM1045284"
#> [41] "GSM1045287" "GSM1045296" "GSM1045298" "GSM1045300" "GSM1045303"
#> [46] "GSM1045306" "GSM1045309"
#>
#> $node_samples$Node_18
#>  [1] "GSM1045191" "GSM1045195" "GSM1045196" "GSM1045197" "GSM1045198"
#>  [6] "GSM1045199" "GSM1045202" "GSM1045204" "GSM1045205" "GSM1045208"
#> [11] "GSM1045215" "GSM1045220" "GSM1045228" "GSM1045231" "GSM1045235"
#> [16] "GSM1045236" "GSM1045246" "GSM1045249" "GSM1045253" "GSM1045256"
#> [21] "GSM1045258" "GSM1045260" "GSM1045262" "GSM1045265" "GSM1045267"
#> [26] "GSM1045268" "GSM1045271" "GSM1045273" "GSM1045277" "GSM1045279"
#> [31] "GSM1045280" "GSM1045281" "GSM1045286" "GSM1045289" "GSM1045290"
#> [36] "GSM1045291" "GSM1045292" "GSM1045295" "GSM1045297" "GSM1045299"
#> [41] "GSM1045304" "GSM1045307" "GSM1045309" "GSM1045310"
#>
#> $node_samples$Node_19
#>  [1] "GSM1045202" "GSM1045215" "GSM1045220" "GSM1045223" "GSM1045226"
#>  [6] "GSM1045230" "GSM1045235" "GSM1045236" "GSM1045247" "GSM1045249"
#> [11] "GSM1045251" "GSM1045256" "GSM1045262" "GSM1045265" "GSM1045268"
#> [16] "GSM1045272" "GSM1045273" "GSM1045277" "GSM1045279" "GSM1045286"
```

```
#> [21] "GSM1045289" "GSM1045291" "GSM1045292" "GSM1045297" "GSM1045304"
#> [26] "GSM1045307" "GSM1045308" "GSM1045310" "GSM1045311"
#>
#> $node_samples$Node_20
#> [1] "GSM1045234"
#>
#>
#> $node_sizes
#>  Node_1  Node_2  Node_3  Node_4  Node_5  Node_6  Node_7  Node_8  Node_9 Node_10
#>       1       2       3       3       6      10       2       3       4       3
#> Node_11 Node_12 Node_13 Node_14 Node_15 Node_16 Node_17 Node_18 Node_19 Node_20
#>       2       1       2       1       1      29      47      44      29       1
#>
#> $node_average_filt
#> $node_average_filt$Node_1
#> [1] 612.8152
#>
#> $node_average_filt$Node_2
#> [1] 619.2167
#>
#> $node_average_filt$Node_3
#> [1] 628.0636
#>
#> $node_average_filt$Node_4
#> [1] 637.1298
#>
#> $node_average_filt$Node_5
#> [1] 649.4243
#>
#> $node_average_filt$Node_6
#> [1] 654.6199
#>
#> $node_average_filt$Node_7
#> [1] 667.8046
#>
#> $node_average_filt$Node_8
#> [1] 669.0753
#>
#> $node_average_filt$Node_9
#> [1] 663.2821
#>
#> $node_average_filt$Node_10
#> [1] 662.3884
#>
#> $node_average_filt$Node_11
#> [1] 662.0506
#>
#> $node_average_filt$Node_12
#> [1] 662.4023
#>
#> $node_average_filt$Node_13
#> [1] 663.1383
#>
#> $node_average_filt$Node_14
#> [1] 663.8281
#>
#> $node_average_filt$Node_15
#> [1] 658.3845
#>
#> $node_average_filt$Node_16
#> [1] 672.296
#>
#> $node_average_filt$Node_17
#> [1] 680.2087
```

```
#>
#> $node_average_filt$Node_18
#> [1] 688.619
#>
#> $node_average_filt$Node_19
#> [1] 693.9702
#>
#> $node_average_filt$Node_20
#> [1] 697.2491
#>
#>
#> $adj_matrix
#>         Node_1 Node_2 Node_3 Node_4 Node_5 Node_6 Node_7 Node_8 Node_9 Node_10
#> Node_1       1      0      0      0      0      0      0      0      0       0
#> Node_2       0      1      1      0      0      0      0      0      0       0
#> Node_3       0      0      1      1      0      0      0      0      0       0
#> Node_4       0      0      0      1      1      0      0      0      0       0
#> Node_5       0      0      0      0      1      1      0      0      0       0
#> Node_6       0      0      0      0      0      1      0      0      1       1
#> Node_7       0      0      0      0      0      0      1      0      0       0
#> Node_8       0      0      0      0      0      0      0      1      0       0
#> Node_9       0      0      0      0      0      0      0      0      1       0
#> Node_10      0      0      0      0      0      0      0      0      0       1
#> Node_11      0      0      0      0      0      0      0      0      0       0
#> Node_12      0      0      0      0      0      0      0      0      0       0
#> Node_13      0      0      0      0      0      0      0      0      0       0
#> Node_14      0      0      0      0      0      0      0      0      0       0
#> Node_15      0      0      0      0      0      0      0      0      0       0
#> Node_16      0      0      0      0      0      0      0      0      0       0
#> Node_17      0      0      0      0      0      0      0      0      0       0
#> Node_18      0      0      0      0      0      0      0      0      0       0
#> Node_19      0      0      0      0      0      0      0      0      0       0
#> Node_20      0      0      0      0      0      0      0      0      0       0
#>         Node_11 Node_12 Node_13 Node_14 Node_15 Node_16 Node_17 Node_18 Node_19
#> Node_1        0       0       0       0       0       0       0       0       0
#> Node_2        0       0       0       0       0       0       0       0       0
#> Node_3        0       0       0       0       0       0       0       0       0
#> Node_4        0       0       0       0       0       0       0       0       0
#> Node_5        0       0       0       0       0       0       0       0       0
#> Node_6        0       0       1       0       1       0       0       0       0
#> Node_7        0       0       0       0       0       1       0       0       0
#> Node_8        0       0       0       0       0       1       0       0       0
#> Node_9        0       0       0       0       0       1       0       0       0
#> Node_10       0       0       0       0       0       1       0       0       0
#> Node_11       1       0       0       0       0       0       0       0       0
#> Node_12       0       1       0       0       0       0       0       0       0
#> Node_13       0       0       1       0       0       1       0       0       0
#> Node_14       0       0       0       1       0       0       0       0       0
#> Node_15       0       0       0       0       1       0       0       0       0
#> Node_16       0       0       0       0       0       1       1       0       0
#> Node_17       0       0       0       0       0       0       1       1       0
#> Node_18       0       0       0       0       0       0       0       1       1
#> Node_19       0       0       0       0       0       0       0       0       1
#> Node_20       0       0       0       0       0       0       0       0       0
#>         Node_20
#> Node_1        0
#> Node_2        0
#> Node_3        0
#> Node_4        0
#> Node_5        0
#> Node_6        0
#> Node_7        0
#> Node_8        0
#> Node_9        0
```

```
#> Node_10       0
#> Node_11       0
#> Node_12       0
#> Node_13       0
#> Node_14       0
#> Node_15       0
#> Node_16       0
#> Node_17       0
#> Node_18       0
#> Node_19       0
#> Node_20       1
#>
#> $n_sizes
#> [1] 20
#>
#> $average_nodes
#> [1] 9.7
#>
#> $standard_desviation_nodes
#> [1] 14.78655
#>
#> $number_connections
#> [1] 16
#>
#> $proportion_connections
#> [1] 0.08421053
#>
#> $number_ramifications
#> [1] 7
#>
#> attr(,"class")
#> [1] "mapper_object"
```

In addition, the mapper information are plotted using `plot_mapper()` function (see Figure 2). Hovering the mouse over each node in the interactive graph displays the number of samples that form the node.

To show more clearly the resizing options provided by *plot_mapper*(), the same graph is shown by varying the values of the function parameters (see Figure 3):

```
plot_mapper(mapper_object, trans_node_size = TRUE, exp_to_res = 1/1.5)
```

### 4.4 GSSTDA object

The analysis of `gsstda()` function consists of the three previous parts, i.e., a preprocessing of the data {`dsga()`], the gene selection and the filter function [`gene_selection()`], and the mapper algorithm [`mapper()`]. So, this function integrates the explained functions above. Results are returned as an `gsstda object`, which is composed by all the results of `dsga()`, `gene_selection()` and `mapper()` functions.

The basic model of GSSTDA (dsga + gene selection + mapper) can be implemented in R using the function `gsstda()`:

```
gsstda_object <- gsstda(full_data, survival_time, survival_event, case_tag,
                        gamma, gen_select_type, percent_gen_select,
                        num_intervals, percent_overlap, distance_type,
                        clustering_type, num_bins_when_clustering,
                        linkage_type, optimal_clustering_mode,
                        silhouette_threshold)
```

This function returns a `gsstda object` composed of:

- `normal_space`: The matrix with the normal space (linear space generated from normal tissue samples).

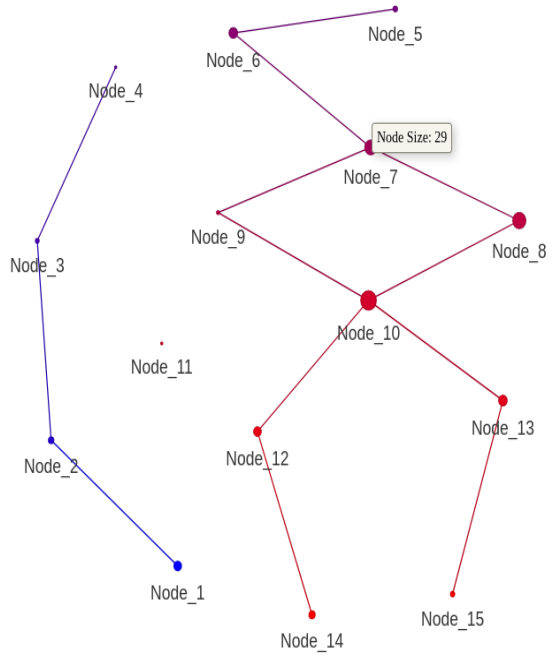


**Figure 2:** Mapper result graph. Mapper visualization depicting clusters of samples based on gene expression profiles along the value of the filter function, with nodes representing clusters and edges indicating overlap, facilitating the identification of nodes of different gene expression patterns and survival. The intensity of the color within each node reflects the average level of filter function within the cluster, aiding in the characterization of biological significance. Low filter function values, represented in blue, are associated with better survival. High values, in red, are associated with worse survival. Additionally, annotation allows for the identification of a cluster of interest, assisting in the extraction of insight into potential biomarkers or gene expression patterns associated with specific sample groups. The default option used in this case resizes the node sizes as $No.samples^{1/2}$.

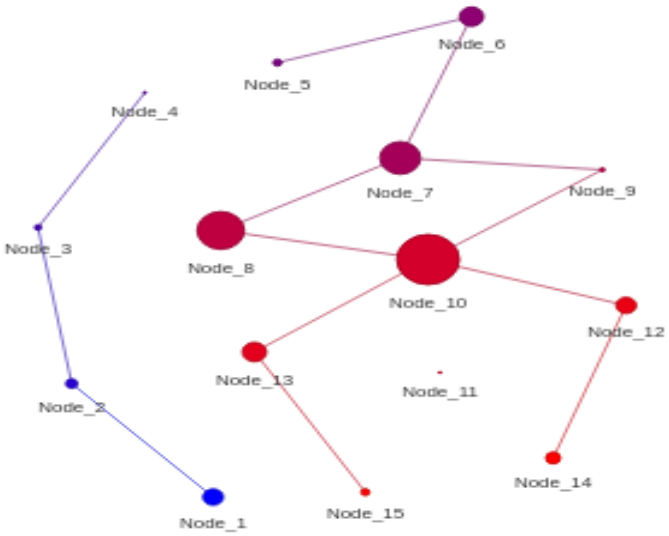


**Figure 3:** Mapper result graph after varying the parameter *exp_to_res* to 1/1.5 ($No.samples^{1/1.5}$). This way the resizing of the node size is smoother.

- `matrix_disease_components`: The matrix of the disease components (the transformed `full_data` matrix from which the normal component has been removed).
- `cox_all_matrix`: A matrix with the results of the application of proportional hazard models for each gene.
- `genes_selected`: The genes selected for `gene_selection()`.
- `genes_disease_component`: The matrix of the disease components with information from these genes only.
- `mapper_object`: The mapper object explained in the subsection Mapper object.

As an example, using arguments of `dsga()` and `gene_selection()`, we next process the `gsstda()` using the suitable code as follows:

```
gsstda_object <- gsstda(full_data = full_data,
                        survival_time = survival_time,
                        survival_event = survival_event,
                        case_tag = case_tag,
                        gen_select_type = "Top_Bot",
                        percent_gen_select = 10,
                        num_intervals = 10,
                        percent_overlap = 40,
                        distance_type = "correlation",
                        clustering_type = "hierarchical",
                        linkage_type = "single")
```

The information obtained from the GSSTDA object are showed using `print` for `dsga` and `mapper` information. In addition, the mapper information obtained by GSSTDA object are plotted using `plot_mapper()` function (see Figure 2).

## 5 Conclusions

The **GSSTDA** package allows transcriptomic data to be analysed (although it could be used for other types of omics data) integrating the information related on the degree of association of each gene with survival through the use of a specific feature selection procedure and a new adapted filter function. This allows to obtain a graph reflecting groups of samples that are not only differentiated by similarity in expression pattern but also by similarity in survival.

Specifically, this package is the implementation of the Progression Analysis of Disease with Survival. This methodology pre-processes the data using the DSGA which isolates the part of the expression that is considered pathological. This allows only this part to be used in further analysis. The genes to be used in the Mapper analysis are then selected on the basis of their association with survival. This information is also integrated into the filter function. Using the values of this filter function and the data processed through the DSGA, this information is condensed into a graph using the Mapper algorithm. One of the advantages of using Mapper over classical clustering techniques is that it has been shown in numerous applications to reveal aspects of the data that classical clustering does not detect.

In the broad field of oncology, the study of survival is essential, and omics sciences are increasingly important. GSSTDA allows the integration of both, so it could be useful in any oncological pathology and the results obtained can contribute to the development of personalized medicine. It can aid in the identification of prognostic biomarkers, therapeutic targets, or molecular subtypes. As has been done in this work with survival, any function of the data that reflects a clinical or biological characteristic of interest could be used as a filtering function so that the applications of possible adaptations of this package could be numerous.

Additionally, the combined use of our package with other analytical tools could show its versatility and potential for interdisciplinary research collaborations. Concrete examples or case studies demonstrating the application of the package in the analysis of real-world data sets and the discovery of novel biological insights can further illustrate its practical value and impact on biomedical research.

Several functions have been implemented in the **GSSTDA** package: for processing the data using DSGA method, selecting the genes and calculating the values of the new filter function taking into account the association of each gene with survival, and generating a topological invariant Reeb graph applying the Mapper algorithm. The package provides a function to perform the full PAD-S analysis but also allows the DSGA and Mapper analyses, which can have numerous applications, to be performed independently.

Throughout the paper, we have also shown how to organize the data, use the available functions, and interpret the results. In particular, to illustrate the different functions implemented in the package, we applied all of them on a common empirical example so that results can easily be compared. In this way, we believe that the **GSSTDA** package is a valid self-contained R package for grouping transcriptomics data samples according to their survival and gene expression by integrating information on genes linked to survival in the process of gene selection and in the Mapper filter function from the popular topological technique: Topological Data Analysis.

Diving into potential future directions for expanding this R package, an array of promising avenues beckon. Initially, existing functionalities could be improved by refining data preprocessing algorithms, optimizing parameter selection methods, and bolstering computational efficiency to accommodate larger datasets more adeptly. Another possible parallel option is to explore new alternatives to carry out these processes and adapt them to other types of omics data. To complement its applications, functions that allow exploring the results obtained could be integrated into the package.Exploring integrative strategies for analyzing multi-omics data stands out as another compelling trajectory, offering deeper insights into biological mechanisms and disease pathology. Adapting the method to other problems by modifying the feature selection process and the filtering function represents another possible direction for future work. Additionally, fostering community engagement and collaboration will be pivotal for nurturing ongoing growth and evolution, inviting users to contribute feedback, suggestions, and innovative applications to enrich the package's capabilities and impact. Through these concerted efforts, the aim is to equip researchers with a versatile and robust toolkit for exploring complex biological datasets and advancing biomedical research.

## 6 Acknowledgments

R. Bosch-Romeu, A. Falco and M. Esteve thank the grant TED2021-129347B-C22 funded by Ministerio de Ciencia e Innovación/ Agencia Estatal de Investigación. Additionally, R. Bosch-Romeu and A. Falco gratefully acknowledges the financial support from CEU Cardenal Herrera University and Santander under a predoctoral grant, from Scientific Foundation of the Spanish Association Against Cancer under a predoctoral grant, and the Spanish Ministry of Science, Innovation and Universities under grant *Ayudas para la Formación de Profesorado Universitario FPU 2022* (FPU 22/02810).

## References

G. Carlsson. Topology and data. *Bulletin of the American Mathematical Society*, 46(2):255–308, 2009. [p90, 95]

G. Casaclang-Verzosa, S. Shrestha, M. J. Khalil, J. S. Cho, M. Tokodi, S. Balla, M. Alkhouli, V. Badhwar, J. Narula, J. D. Miller, et al. Network tomography for understanding phenotypic presentations in aortic stenosis. *JACC: Cardiovascular Imaging*, 12(2):236–248, 2019. [p90]

H. Edelsbrunner and J. L. Harer. *Computational topology: an introduction*. American Mathematical Society, 2022. [p90]

J. Fores-Martos, B. Suay-Garcia, R. Bosch-Romeu, M. C. Sanfeliu-Alonso, A. Falco, and J. Climent. Progression analysis of disease with survival (pad-s) by survmap identifies different prognostic subgroups of breast cancer in a large combined set of transcriptomics and methylation studies. *bioRxiv*, pages 2022–09, 2022. [p90]

M. Gavish and D. L. Donoho. The optimal hard threshold for singular values is $4/\sqrt{3}$. *IEEE Transactions on Information Theory*, 60(8):5040–5053, 2014. [p93]

T. S. Hinks, X. Zhou, K. J. Staples, B. D. Dimitrov, A. Manta, T. Petrossian, P. Y. Lum, C. G. Smith, J. A. Ward, P. H. Howarth, et al. Innate and adaptive t cells in asthmatic patients: relationship to severity and disease mechanisms. *Journal of allergy and clinical immunology*, 136(2):323–333, 2015. [p90]

M. Lawrence, R. Gentleman, and V. Carey. rtracklayer: an r package for interfacing with genome browsers. *Bioinformatics*, 25(14):1841, 2009. [p91]

Y. Liao, G. K. Smyth, and W. Shi. The r package rsubread is easier, faster, cheaper and better for alignment and quantification of rna sequencing reads. *Nucleic acids research*, 47(8):e47–e47, 2019. [p91]

P. Y. Lum, G. Singh, A. Lehman, T. Ishkanov, M. Vejdemo-Johansson, M. Alagappan, J. Carlsson, and G. Carlsson. Extracting insights from the shape of complex data using topology. *Scientific reports*, 3 (1):1236, 2013. [p90, 95]

J. C. Mathews, S. Nadeem, A. J. Levine, M. Pouryahya, J. O. Deasy, and A. Tannenbaum. Robust and interpretable pam50 reclassification exhibits survival advantage for myoepithelial and immune phenotypes. *NPJ Breast Cancer*, 5(1):30, 2019. [p90]

M. Nicolau, R. Tibshirani, A.-L. Børresen-Dale, and S. S. Jeffrey. Disease-specific genomic analysis: identifying the signature of pathologic biology. *Bioinformatics*, 23(8):957–965, 02 2007. ISSN 1367-4803. doi: 10.1093/bioinformatics/btm033. URL https://doi.org/10.1093/bioinformatics/btm033. [p90]

M. Nicolau, A. J. Levine, and G. Carlsson. Topology based data analysis identifies a subgroup of breast cancers with a unique mutational profile and excellent survival. *10.1073/pnas.1102826108*, 2011. [p90]

A. H. Rizvi, P. G. Camara, E. K. Kandror, T. J. Roberts, I. Schieren, T. Maniatis, and R. Rabadan. Single-cell topological rna-seq analysis reveals insights into cellular differentiation and development. *Nature biotechnology*, 35(6):551–560, 2017. [p90]

P. J. Rousseeuw. Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *Journal of computational and applied mathematics*, 20:53–65, 1987. [p96]

M. S. Schroder, A. C. Culhane, J. Quackenbush, and B. Haibe-Kains. survcomp: an r/bioconductor package for performance assessment and comparison of survival models. *Bioinformatics*, 27(22): 3206–3208, 2011. [p91]

G. Singh, F. Mémoli, G. E. Carlsson, et al. Topological methods for the analysis of high dimensional data sets and 3d object recognition. *PBG@ Eurographics*, 2:091–100, 2007. [p90, 95]

D. J. Sokolowski, M. Faykoo-Martinez, L. Erdman, H. Hou, C. Chan, H. Zhu, M. M. Holmes, A. Goldenberg, and M. D. Wilson. Single-cell mapper (scmappr): using scrna-seq to infer the cell-type specificities of differentially expressed genes. *NAR Genomics and Bioinformatics*, 3(1):lqab011, 2021. [p91]

*Miriam Esteve*
*Universidad Cardenal Herrera-CEU*
*Department of Matematicas, Fisica y Ciencias Tecnologicas, 03203 Carmelitas, 3 (Elche), Spain*
*ORCiD: 0000-0002-5908-0581*
miriam.estevecampello@uchceu.es

*Raquel Bosch-Romeu*
*Universidad Cardenal Herrera-CEU*
*Department of Matematicas, Fisica y Ciencias Tecnologicas, San Bartolome 55, Alfara del Patriarca (Valencia), Spain*
*ORCiD: 0000-0001-9126-3241*
raquel.boschromeu@uchceu.es

*Antonio Falco*
*Universidad Cardenal Herrera-CEU*
*Department of Matematicas, Fisica y Ciencias Tecnologicas, ESI International Chair at CEU UCH, 03203 Carmelitas, 3 (Elche) Spain*
*ORCiD: 0000-0001-6225-0935*
afalco@uchceu.es

*Jaume Fores*
*Universidad Cardenal Herrera-CEU*
*Department of Matematicas, Fisica y Ciencias Tecnologicas, San Bartolome 55, Alfara del Patriarca (Valencia), Spain*
*ORCiD: 0000-0002-9025-4877*
fores.martos.jaume@gmail.com

*Joan Climent*
*Universidad Cardenal Herrera-CEU*

*Departamento de Producción y Sanidad Animal, Salud Pública Veterinaria y Ciencia y Tecnología de los Alimentos (PASAPTA), C/ Tirant lo Blanc, 7. 46115, Valencia*
*ORCiD: 0000-0002-8927-6614*
joan.climentbataller@uchceu.es

# Kernel Heaping - Kernel Density Estimation from regional aggregates via measurement error model

*by Lorena Gril, Laura Steinkemper, Marcus Groß, and Ulrich Rendtel*

**Abstract** The phenomenon of "aggregation" often occurs in the regional dissemination of information via choropleth maps. Choropleth maps represent areas or regions that have been subdivided and color-coded proportionally to ordinal or scaled quantitative data. By construction discontinuities at the boundaries of rigid aggregation areas, often of administrative origin, occur and inadequate choices of reference areas can lead to errors, misinterpretations and difficulties in the identification of local clusters. However, these representations do not reflect the reality. Therefore, a smooth representation of georeferenced data is a common goal. The use of naive non-parametric kernel density estimators based on aggregates positioned at the centroids of the areas result also in an inadequate representation of reality. Therefore, an iterative method based on the Simulated Expectation Maximization algorithm was implemented in the Kernelheaping package. The proposed approach is based on a partly Bayesian algorithm treating the true unknown geocoordinates as additional parameters and results in a corrected kernel density estimate.

## 1 Introduction

The data represented by area aggregates do not offer the precise geocoordinates, but they rather refer to areas, usually of varying sizes, such as states, provinces, municipalities, electoral districts, ZIP codes or other statistical spatial references. These data are mostly displayed by choropleth maps. On the one hand, the reason for this type of presentation is that data were not collected at a more granular level. For example, election results can only be traced back to the corresponding constituency in order to maintain election secrecy of the voters. On the other hand, the data with precise geoinformation are often aggregated due to privacy reasons of the participants of a survey or a census survey. Here, data aggregation is a simple strategy of data anonymization. One drawback of choropleth maps are strong variations of the distribution density at the borders of the reference areas which make it difficult to identify regional clusters and lead to misinterpretations of choropleth maps. The representation of data as aggregates leads to the fact that this underlies the specific areal map base. In the so-called support problem, one would like to construct a map based on a different area system which is not hierarchically nested in the original area system. For example, we want aggregates for administrative districts, but we only have access to aggregates at the ZIP code level. It is by no means obvious how the original total numbers are distributed across the preferred areas.

From the statistical point of view, both problems could be addressed with a solid methodological input. The first step is to search for statistical tools that solve the above problems, given full access to the geocoordinates of each observation. A good candidate is the two-dimensional kernel density estimate, which provides a smooth regional distribution of the variable of interest without discontinuities. Moreover, the kernel density is not linked to a specific reference areas. However, to construct a kernel density estimate, we need to know the original geocoordinates. When using aggregated data by means of a naive kernel density estimator, which allocates all observations at the centroids of the respective area, the kernel density estimates are biased and give often a dismal spiky representation of the reality, depending on the used bandwidth. The knowledge of exact geocoordinates solves also the change of support problem. We consider the measurement of the area to which the observation belongs as an imperfect measurement of the exact geocoordinate. In a standard measurement error situation, only limited knowledge of the measurement process is available. Therefore, identifying assumptions must be made in order to draw conclusions about the true value, e.g. the independence of the measurement error from the true value. In the case of aggregation, however, the measurement process is known. This knowledge opens a lane for the standard statistical approach in situations with latent, i.e. unobserved, data: the EM algorithm. However, to account for the aggregation process, the algorithm is extended by a statistical simulation concept. The so-called *Stochastic Expectation Maximisation Algorithm* (SEM) was proposed by Celeux et al. (1996).

The proposed algorithm delivers a kernel density estimation which takes the measurement errors, i.e. the aggregation process, into account. As a by-product of the algorithm, we obtain simulated geocoordinates that are consistent with the final density estimate. With these artificial coordinates the

computation of aggregates in any arbitrary area system can be done without additional effort, so that the support change problem can be solved immediately.

In addition to aggregation, there are other ways in which measurement errors can affect the observed variable. Carroll and Stefanski (1990) introduced the deconvolution kernel density estimator which utilizes the property of the characteristic function. Thereby, the density of the measurement error process is known, so the Fourier inversion theorem can be applied. In addition, Delaigle (2014) introduced a kernel-based method for dealing with heteroscedastic errors in this context. Basulto-Elias et al. (2021) applied bivariate kernel deconvolution to panel data. In the context of differential privacy, Farokhi (2020) introduced a deconvoluting kernel density estimator to remove the effect of privacy-preserving additive noise. Despite the possibility of using this method, the use of SEM by drawing pseudo-samples is more intuitive in the context of population aggregation and leads to the proposed implementation in the case of aggregation as a measurement error.

This paper is dedicated to the **R** package **Kernelheaping** providing a partly Bayesian algorithm, which treats the true unknown values as additional parameters and estimates the aggregation parameters to give a corrected kernel density estimate proposed by Groß and Rendtel (2016) and Groß et al. (2016). In Groß and Rendtel (2016) and Walter et al. (2022) the kernel heaping method is proposed based on one-dimensional survey data, i.e. asymmetric rounding occurring in self-reported survey answers such as income, weight or height due to social desirability. An extension to the two-dimensional case was published in Groß et al. (2016) reversing the rounding respectively aggregation process by using a measurement error model. This method was applied to Berlin register data of residents for deriving density estimates of ethnic minorities and aged people as well as a simulation study was made.

By the development and implementation of this approach, further relevant articles could be published. In the article by Rendtel and Ruhanen (2018), the need of childcare in Berlin is analyzed. For this purpose, first a good representation of the population below 18 was obtained by the above mentioned kernel heaping method. Then, a demand analysis was made comparing children's living area to actual geocoordinates of kindergartens, schools and pediatrician's offices. In addition, high density areas of newborns were considered in temporal context. In the context of mobile phone data Hadam et al. (2020) compared user density estimates based on grid cell counts and on the kernel heaping algorithm. A special feature here are the strong size differences of the observed grid cells, which make visual comparisons difficult. The already mentioned change of support problem was discussed in the article by Groß et al. (2020) on the basis of student resident numbers in ZIP areas. However, for planning purposes the aggregates were needed on municipality districts which are in a non-hierarchical relationship to the ZIP areas. Rendtel et al. (2021) showed, apart from choropleth maps, a smooth representation of the spatial and temporal spread of the Corona pandemic. This is a central topic of epidemiological research and also in the interest of public media. The temporally spatially animated kernel density maps, which were created using the kernel heaping algorithm, show soft transitions in the corresponding counties and thus clusters of COVID-19 infections in Germany can be better identified. In contrast, the choropleth maps give a rather unstable impression of the development of the pandemic. The spatio-temporal animated maps can be found here. In an article by Erfurth et al. (2021) election results of the 2016 Berlin House of Representatives election were analyzed. The evaluation is based on a local voter register with known anonymized addresses. Some modifications respectively extensions of the basic method proposed by Groß et al. (2016) were made. A boundary correction was introduced to eliminate the underestimation of the kernel density estimate at the boundaries of the population, and uninhabited areas are also taken into account. In addition, an algorithm for the calculation of local percentages is adapted. The extensions to the algorithm are already implemented in the **Kernelheaping** package.

## 2 Statistical Method

In this section, first the multivariate kernel density estimation is introduced. Then the main idea, namely the use of the Stochastic Expectation Maximization algorithm in combination with aggregated data, is presented. Often, however, one faces boundaries in relation to georeferenced data, either uninhabited or unobserved areas, which makes an extension with a boundary correction unavoidable. As a result, the SEM algorithm must also be adapted. The method can also be used to calculate proportions, which makes a further extension of the algorithm necessary.

### 2.1 Multivariate Kernel Density Estimation

The multivariate kernel density estimation (KDE) is a non-parametric approach to estimate the probability distribution of a continuous variables. This popular method produces smooth density estimates and can be seen as an improvement of traditional histograms. For the case presented here with respect to geocoordinates, it is sufficient to consider $X = \{X_1, X_2, \ldots, X_n\}$ a sample of size $n$ from a bivariate random variable, i.e. $X_i = (X_{i1}, X_{i2})$ as longitude and latitude coordinates, with unknown density $f(x)$. The kernel density estimator at point $x$ is given by

$$\hat{f}_H(x) = \frac{1}{n|H|} \sum_{i=1}^{n} K\Big(H^{-1}(x - X_i)\Big), \tag{1}$$

where $K(\cdot)$ is a multivariate kernel function, $H$ denotes a symmetric positive definite bandwidth matrix and $|\cdot|$ denotes the determinant. In this setting, the kernel function $K(\cdot)$ is the multivariate Gaussian kernel.
The choice of bandwidth $H$ is very important for the performance of a KDE. Several bandwidth selecting approaches have been discussed in the literature, see Izenman (1991) and Silverman (1986). The plug-in approach of Wand and Jones (1994) is used hereafter as a bandwidth selector due to computational efficiency as seen in the simulation study. The performance of $\hat{f}_H$ is measured by the Mean Integrated Squared Error $MISE(H) = \mathbb{E} \int_R^2 (\hat{f}_H(x) - f(x))^2 dx$. Thus, the optimal bandwidth in the space of symmetric and positive definite $2 \times 2$ matrices $\mathcal{H}$ would be the one that minimizes the MISE. Since the optimal bandwidth does not have a closed-form solution, asymptotic analysis is used. Additionally, since the true density is unknown, a data-driven approach is used to iteratively estimate the bandwidth matrix. For bivariate kernels, under the assumptions (A1) the density $f$ has all second-order partial derivatives bounded, continuous and squared integrable; (A2) the kernel $K$ is the bivariate Gaussian kernel; (A3) all entries of $H$ and $n^{-1}|H|^{-1/2}$ tend to 0 for $n \to \infty$; Duong and Hazelton (2003) stated

$$\begin{aligned} MISE(H) =& AMISE(H) + o(n^{-1}|H|^{-1/2} + tr^2(H)) \\ AMISE(H) =& n^{-1}|H|^{-1/2} R(K) + 1/4 \mu_2(K)^2 (vech^T H) \psi_4 (vech H), \end{aligned}$$

where $R(K) = \int_{R^2} K(x)^2 dx$, $\mu_2(K) I = \int_{R^2} x x^T K(x) dx$, with $\mu_2(K) < \infty$, $vech$ is the vector half operator and $\psi_4 = \int_{R^2} vech(2D^2 f(x) - dgD^2 f(x)) vech^T (2D^2 f(x) - dgD^2 f(x)) dx$ with $D^2 f(x)$ being the Hessian matrix of $f$ and $dgA$ being matrix A with all of its non-diagonal elements set to zero. The plug-in method further states explicitly $\psi_4$ by a function $\psi_r$ and makes use of the tractability of AMISE by seeking to estimate $H_{AMISE} = \arg\min_{H \in \mathcal{H}} AMISE(H)$. An pilot estimation of $\psi_r$ is plugged in to obtain an estimate of $\psi_4$. Hence, an estimate of the $AMISE(H)$ is obtained that can be numerically minimized to give the plug-in bandwidth $\hat{H}$.
A detailed review of automatic or data-driven bandwidth selection methods is given by Heidenreich et al. (2014).

### 2.2 SEM algorithm for the estimation of densities based on aggregated data

For aggregated data, however, using a naive multivariate KDE returns an unrealistic representation of the reality. The proposed approach needs as an input aggregated data as well as the predefined centroids, i.e. midpoints of the areas, and the corresponding absolute value of the aggregate. Furthermore, the grid size corresponds to the specified discretization of the geocoordinates. Let $x_g$, $g = 1, \ldots, G$ be the geocoordinates of the $G$ grid points with $\Delta_1$ and $\Delta_2$ be the distance between two grid points in the direction of longitude and latitude. For each grid point it have to be determined in which of the $A$ areas it lies. This divides the grid points into $A$ subsets $\mathcal{G}_a = \{x_g | g = 1, \ldots, G, x_g \in a\}$, where $a = 1, \ldots, A$ corresponds to the areas. Note that the subdivision is disjoint, i.e. the set of grid points $\mathcal{G} = \cup_{a=1}^{A} \mathcal{G}_a$. The midpoints of the area $a$ are denoted by $W_a$, and the total number of observed values by $N_a$ for $a = 1, \ldots, A$.

Step 0 Initialize of burn-in B, sample size R, grid size and corresponding subsets $\mathcal{G}_a$ with respect to all areas $a$.

Step 1 Calculate the naive KDE $\hat{f}^{(0)}$ using equation 1 and Gaussian Kernel from the sample $s^{(0)}$, which contains $N_a$ times the midpoint $W_a$, $a = 1, \ldots, A$. Additionally, set the bandwidth parameters $h_1^{(0)}$ and $h_2^{(0)}$ sufficiently large such that no spikes occur in the estimation. The bandwidth parameter are chosen in relation to the grid size and the amount of areas, i.e. $h_i^{(0)} = \left(\frac{\Delta_i}{A}\right)^2$, $i = 1, 2$.

Step 2 Draw a sample $s^{(t)}$ consisting of $N_a$ grid points of the set $\mathcal{G}_a$, $a = 1, \ldots, A$. The sampling

procedure is with replacement and the sampling weights of each grid point $x_g$ is proportional to $\hat{f}^{(t-1)}(x_g)$. Thus, the sampling is proportional size with respect to the preceding density estimate.

Step 3 Calculate $\hat{f}^{(t)}$ using equation 1 and Gaussian Kernel from the sample $s^{(t)}$ using smoothing parameters $h_1^{(t)}$ and $h_2^{(t)}$ obtained by the plug-in estimator of Wand and Jones (1994).

Step 4 Repeat Step 2 and Step 3 B+R times.

Step 5 Calculate the final density estimate $\hat{f}(x)$ by

$$\hat{f}(x) = \frac{1}{R}\sum_{r=1}^{R} \hat{f}^{(B+r)}(x).$$

### 2.3 Boundary correction of kernel density estimation

The KDE of bounded maps raises the question of how to handle the boundaries, since a part of the estimate may be outside the region of interest. As the KDE should not cover areas outside the boundary, a possible approach proposed by Jones (1993) is to restrict the kernel function to the area within the boundary denoted by $\mathcal{S}$. The rescaling factors $w$ controlling the areas within the boundaries can by calculated for each point $x$ by

$$w_x = \int_{\mathcal{S}} \frac{1}{|H|} K\Big(H^{-1}(x-y)\Big)dy.$$

Hence, the rescaled KDE $\hat{f}_{rs}(x)$ at each point $x$ is

$$\hat{f}_{rs}(x) = \frac{1}{N|H|}\sum_{x_s\in\mathcal{S}} \frac{1}{w_x} K\Big(H^{-1}(x-x_s)\Big). \tag{2}$$

In the case of geocoordinates, the set of grid points lying in settled area $\mathcal{S}$ are denoted by $\mathcal{G}_{\mathcal{S}}$ and $\Delta_{\mathcal{G}}$ the area of control of a grid point. The rescaling factors $w_x$ at each geocoordinate $x = (x_1, x_2)$ by using the Gaussian Kernel can be approximated by

$$\begin{aligned} w_x &\approx \sum_{x_s\in\mathcal{S}} \frac{1}{|H|} K\Big(H^{-1}(x-x_s)\Big)\Delta_{\mathcal{G}} \\ &= \frac{\Delta_{\mathcal{G}}}{\sqrt{2\pi}} \frac{1}{h_1 h_2} \sum_{(x_{s1},x_{s2})\in\mathcal{G}_{\mathcal{S}}} \exp\left\{-0.5\left(\frac{(x_1-x_{s1})^2}{h_1} + \frac{(x_2-x_{s2})^2}{h_2}\right)\right\}. \end{aligned} \tag{3}$$

### 2.4 SEM algorithm for kernel density estimation with boundary correction

Step 0 Initialize burn-in B, sample size R, grid size, $\mathcal{G}_{a|\mathcal{S}}$ grid points lying in area $a$, $a = 1, \ldots, A$ and restricted to settled area $\mathcal{S}$.

Step 1 Determine sample set $s^{(0)}$ consisting of $N_a$ times $W_a$, $a = 1, \ldots, A$, whereas $W_a \in \mathcal{S}$ must hold. Determine $h_1^{(0)}$ and $h_2^{(0)}$ sufficiently large and rescaling factors $w_x^{(0)}$ for all grid points $x \in \mathcal{G}_{\mathcal{S}}$ using equation 3. Calculate KDE $\hat{f}_{rs}^{(0)}$ according to equation 2 for $x \in \mathcal{G}_{\mathcal{S}}$.

Step 2 Draw a sample $s^{(t)}$ consisting of $N_a$ grid points of the set $\mathcal{G}_{a|\mathcal{S}}, a = 1, \ldots, A$. The sampling procedure is with replacement and the sampling weights of each grid point $x_s \in \mathcal{S}$ is proportional to $\hat{f}_{rs}^{(t-1)}(x_s)$.

Step 3 Calculate $\hat{f}_{rs}^{(t)}$ using equation 2 and Gaussian Kernel from the sample $s^{(t)}$. Calculate the smoothing parameters $h_1^{(t)}$ and $h_2^{(t)}$ obtained by the plug-in estimator of Wand and Jones (1994) and recalculate the correction weights $w_x^{(t)}$ according equation 3.

Step 4 Repeat Step 2 and Step 3 B+R times.

Step 5 Calculate the final density estimate $\hat{f}_{rs}(x)$ by

$$\hat{f}_{rs}(x) = \frac{1}{R}\sum_{r=1}^{R} \hat{f}_{rs}^{(B+r)}(x).$$

### 2.5 Estimation of local proportions

Proportions of the population with a certain characteristic are often geographically expressed by choropleth maps. Proportion means a percentage of the total population with a certain characteristic on the overall population, e.g. the proportion of voters of a certain party among all voters. The calculation based on the choropleth maps is the number of persons with the characteristic in the area $a$ divided by all persons in the area $a$ for $a = 1, \ldots, A$. In terms of density estimation, let $f_P$ and $f_C$ be the density of the entire population and the density of the population with a certain characteristic, respectively. Let $N_P$ and $N_C$ be the total number of people in the population or the number of people with a certain characteristic, respectively, with $N_P \geq N_C$. Let $x = (x_1, x_2)$ be a coordinate in the observed map. The local proportion of the people with a certain characteristic within a rectangle of size $\Delta_{x_1} \times \Delta_{x_2}$ is obtained by

$$r(x) = \frac{N_C f_C(x)(\Delta_{x_1} \times \Delta_{x_2})}{N_P f_P(x)(\Delta_{x_1} \times \Delta_{x_2})} = \frac{N_C f_C(x)}{N_P f_P(x)}.$$

A non-parametric estimator of the local ratio is the Nadaraya-Watson estimator $\hat{r}_{NW}$ showing to be the ratio of two KDE with equal smoothing factor

$$\hat{r}_{NW} = \frac{N_C \hat{f}_C(x)}{N_P \hat{f}_P(x)}, \tag{4}$$

where $\hat{f}_C$ and $\hat{f}_P$ are the KDE with respect to the entire population and the population with a certain characteristic, see Härdle (1990).

### 2.6 SEM algorithm estimating local proportions

For the sampling phase of the SEM algorithm the sample of the persons with the characteristic of interest is a subsample of the population sample, i.e. $s_P^{(t)} \subset s_C^{(t)}$. For the calculation of the joint smoothing parameter we take the smaller sample, i.e. the sample $s_C^{(t)}$. This results in larger smoothing parameters and therefore more stable ratios in equation 4.

Step 0 Initialize burn-in B, sample size R, grid size, and corresponding subsets $\mathcal{G}_a$ with respect to all areas $a$.

Step 1 Calculate the naive KDEs $\hat{f}_P^{(0)}$ and $\hat{f}_C^{(0)}$ using equation 1 and Gaussian Kernel from the samples $s_P^{(0)}$ and $s_C^{(0)}$, which contains of $N_{P,a}$ respectively $N_{C,a}$ times the the midpoint $W_a$, $a = 1, \ldots, A$. Additionally, set the bandwidth parameters $h_1^{(0)}$ and $h_2^{(0)}$ sufficiently large such that no spikes occur in the estimation.

Step 2 Draw a sample $s_P^{(t)}$ consisting of $N_{P,a}$ grid points of the set $\mathcal{G}_a$, $a = 1, \ldots, A$. The sampling procedure is with replacement and the sampling weights of each grid point $x_g$ is proportional to $\hat{f}_P^{(t-1)}(x_g)$.

Then, take a subsample $s_C^{(t)}$ from $s_P^{(t)}$ consisting of $N_{C,a}$ grid points. The subsampling is proportional to $\hat{f}_C^{(t-1)}(x_g)$.

Step 3 Calculate $\hat{f}_P^{(t)}$ and $\hat{f}_C^{(t)}$ using equation 1 and Gaussian Kernel from the sample $s_P^{(t)}$ respectively $s_C^{(t)}$ using the joint smoothing parameters $h_1^{(t)}$ and $h_2^{(t)}$ for the sample $s_C^{(t)}$ obtained by the plug-in estimator of Wand and Jones (1994).

Step 4 Repeat Step 2 and Step 3 B+R times. After the burn-in phase, i.e. after B replications, compute for the $(B + r)^{th}$ sample the ratio

$$\hat{f}_{C|P}^{B+r}(x) = \frac{\hat{f}_C^{B+r}(x)}{\hat{f}_P^{B+r}(x)}$$

for all grid points.

Step 5 Calculate the final density estimate $\hat{f}_{C|P}(x)$ for all grid points by

$$\hat{f}_{C|P}(x) = \frac{1}{R} \sum_{r=1}^{R} \hat{f}_{C|P}^{(B+r)}(x).$$

## 3 Functions in the Kernelheaping package

Tabel 1 gives an overview of significant functions within **Kernelheaping** package. The original idea is based on one-dimensional survey data that is rounded or summarized into classes. The method has been extended to a two or three dimensional application. The main functions used in the following case study are shown in bold.

| Univariate Methods | |
|---|---|
| `createSim.Kernelheaping()` | Creates one dimensional sample data from a given distribution and rounds these data. |
| `dheaping()` | Smooth KDE on heaped data based on the same rounding value. |
| `dclass()` | Smooth KDE on data in different sized heaping classes. |
| Multivariate Methods | |
| `dbivr()` | Insert two dimensional heaped data, both coordinates need to be heaped by the same rounding value. Returns a smooth KDE based on heaped data. |
| **`dshapebivr()`** | Aggregated population data based on area system provided as shape file is represented smoothly by an iterative 2D KDE method. |
| **`dshapebivrProp()`** | Aggregated overall population data and data with a specific characteristic based on area system provided as shape file is represented as smooth local proportions by an iterative 2D KDE method. |
| `toOtherShape()` | Aggregates the pseudo-samples obtained by `dshapebivr()` on another area system provided by the user. |
| `dshape3dProp()` | Several different aggregated observations based on an area system is smoothly represented by a 3D KDE. |

**Table 1:** Overview of the functions in the Kernelheaping package

## 4 Case Study

The first part of the case study concentrates on the standard form of **bivariate kernel density estimation** and in the second part the focus lies in calculating **proportions**. The third part focuses on identifying regions with the **highest density**. The kernel heaping algorithm utilizes various packages providing tools to generate plots, maps or nested functions. The packages **fields**, **ggplot2**, **RColorBrewer**, **dplyr**, **terra**, **sp**, **sf**, **patchwork**, **rmapshaper** and **Kernelheaping** are needed to execute the code by using the `install.packages()` and `library()` functions in **R**.

Data from the Office of Statistics Berlin-Brandenburg and from Open Street Map are used in the following examples. Spatially, the focus is on the lowest administrative planning levels of Berlin, the so-called **LORs** (German abbreviation for Lebensweltlich orientierte Räume). Berlin is therefore subdivided into 447 LORs, which are characterized by a high degree of internal homogeneity. The shape file of Berlin LORs is available at `https://www.statistik-berlin-brandenburg.de/opendata/RBS_OD_LOR_2015_12.zip`.

### 4.1 Part 1: Bivariate kernel density estimation

To illustrate the **bivariate kernel density estimation**, we focus on the density of elderly individuals between the age of 65 and 80 in the lowest administrative areas (LORs) of Berlin. The dataset of Berlin's population can be downloaded from `https://www.statistik-berlin-brandenburg.de/opendata/EWR201512E_Matrix.csv`. The first step involves loading the data for the LORs. The CSV (comma-separated values) dataset, referred to as `data` in the code below, contains information from the Berlin Residents' Registration Office such as gender, age group, migration background and nationality. The key variable in the dataset is `RAUMID`, which is used to identify the specific LORs. Additionally, there is a shapefile named `Berlin` that encompasses the administrative units' shapes. A shapefile is a two-dimensional data format that is commonly used for geospatial vector data. Geographical objects within a shapefile can be represented and displayed in various forms, such as points, lines, or, in our specific case, as polygons. Once the two datasets are loaded in the **R** workspace, the Berlin

shapefiles need to be georeferenced via the command `st_transform`, using the world geodetic system 1984 (WGS 84) as a spatial reference.

```
setwd("~/Downloads")
# Load data matrix with population numbers
data <- read.csv2("EWR201512E_Matrix.csv")
# Load shape file of Berlin
Berlin <- read_sf("RBS_OD_LOR_2015_12/RBS_OD_LOR_2015_12.shp")
Berlin <- st_transform(Berlin, CRS("+proj=longlat +datum=WGS84"))
```

To prepare plotting the chorpleth map, the data used in our analysis is added to the `Berlin` dataset. Note that you have to be careful to assign the correct values to the respective LORs. In this case, the order of the data is the same as the order of the given polygons. This allows us to compute the density of individuals over 65 years old and below 80. To create a plot by using the package **ggplot2**, the visualisation of the LORs is done by `geom_sf`. In Figure 1 the choropleth map of the total number of inhabitants between 65 and 80 is shown. Choropleth maps assume a homogeneous distribution within each LOR and represent discrete color levels, resulting in discontinuities at the area boundaries. Each LOR respectively polygon can be filled either with the absolute number of people between the age of 65 and 80 (E_E65U80) or the density over all areas (E_E65U80density).

```
# Add the population data to the shape file
Berlin$E_E65U80 <- data$E_E65U80
Berlin$E_E65U80density <- Berlin$E_E65U80 / sum(Berlin$E_E65U80)

# Plot Density of the population data as choropleth map
ggplot(Berlin) +
  geom_sf(data = Berlin, mapping = aes(fill = E_E65U80density)) +
  ggtitle("Proportion of Inhabitants between 65 and 80 years by LOR",
          "Choropleth Map") +
  scale_fill_gradientn(colours = c("#FFFFFF", "#5c87c2", "#19224e"), "Density") +
  xlab("Longitude")+ylab("Latitude")+
  coord_sf()
```

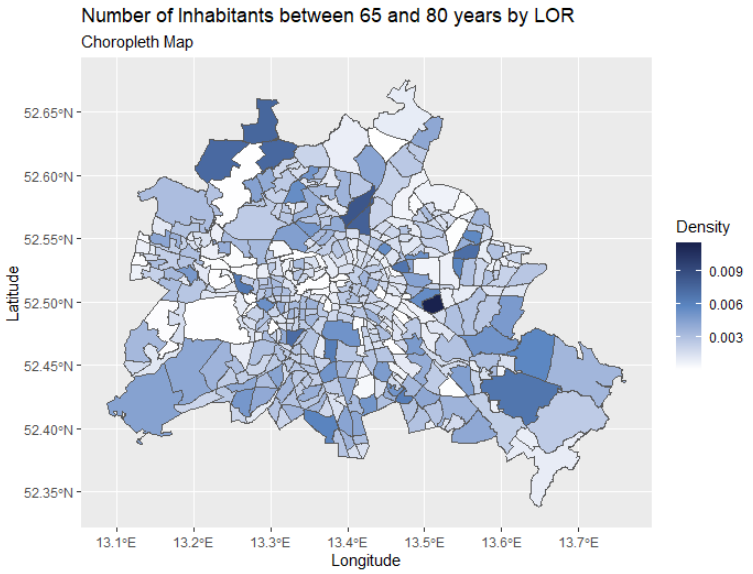


**Figure 1:** Choroleth map of the proportion of inhabitants between 65 and 80 years by LOR in Berlin

As the discontinuities at the boundaries reflect an unrealistic scenario, the kernel heaping algorithm is applied to the data. First the data must be prepared for this application. One of the input variables of the `dshapebivr`, which is used to evaluate the proposed approach, are the centroids of each LOR. Therefore, the `Berlin` data of type `sf` is converted to a `Large SpatialPolygonsDataFrame`. The `lapply` function allows in an easy way to obtain from each polygon the centroids, and the number of people between 65 and 80 years in the form of a data matrix. The midpoint of the $i^{th}$ polygon out of the 447 are given at `Berlin@polygons[[i]]@Polygons[[1]]@labpt` and the total number of people in this certain age are given in `data$E_E65U80[i]`. In this case, the function being used is `function(x) x@lapt`,

which obtains the midpoint of each polygon. Additionally, the total number of peoples between 65 and 80 are added to the data matrix.

```
# Create data frame with midpoint of areas and counts per area
Berlin.shp <- as_Spatial(Berlin)
dataIn <- lapply(Berlin.shp@polygons, function(x) x@labpt) %>%
  do.call(rbind, .) %>% cbind(data$E_E65U80)
head(dataIn)
         [,1]     [,2] [,3]
[1,] 13.34749 52.50644  562
[2,] 13.35580 52.51386   24
[3,] 13.35870 52.50350  639
[4,] 13.36822 52.50211  261
[5,] 13.36617 52.50796  120
[6,] 13.38190 52.51102  282
```

The result of this procedure is stored in the data matrix `dataIn`. Considering the calculated table, it can be deduced that the first two columns represent the latitude and longitude of the midpoint coordinate of each polygon and the third column represents the total number of people of the observed age group in the area.

The most important component of this code is the function `dshapebivr` of the package **Kernelheaping**, which iteratively calculates the bivariate kernel density with respect to the people in a certain age range using the SEM algorithm. The function needs as an input the data frame obtained in the last step as well as the shapefile `Berlin.shp`. In the case study, the `gridsize` for evaluating the KDE is set to 325. Furthermore, it is important to choose an adequate number of burn-in iterations and further sample iterations for the kernel heaping function to yield accurate results. The proposed algorithm requires a few steps to converge to a realistic kernel density. In this case, the first 5 burn-in iterations are ignored, and the average of the last 10 iterations is calculated and returned as an estimate. The `adaptive` parameter controls whether an adaptive smoothing factor is applied for boundary correction. In this case, it is set to `FALSE`, indicating that no further smoothing is utilized. The parameter `boundary` refers to the adjustment of the kernel density estimation near the boundaries. Jones (1993) emphasizes the significance of this correction: "If a probability density function has bounded support, kernel density estimates often overspill the boundaries and are consequently especially biased at and near these edges." Therefore, correcting the density close to boundaries can reduce the induced bias. The bandwidth matrix $H$ is computed by the plug-in method proposed by Wand and Jones (1994).

```
# Use aggregated data to obtain KDE
est <- dshapebivr(data = dataIn, burnin = 5, samples = 10,
                  adaptive = FALSE, shapefile = Berlin.shp,
                  gridsize = 325, boundary = TRUE)
```

The result of the algorithm was stored in the data list `est`. In `est$Mestimate` the averaged KDEs over the sampling phase is stored. The KDEs in the algorithm are calculated by the function `kde` of the package **ks**. In `est$resultDensity` and `est$resultX` the matrices with estimated density for each iteration respectively true latent values X of the estimated geocoordinates are stored. The input shapefile `Berlin.shp` and data matrix `dataIn` are also stored in `est`. The grid points needed to evaluate the KDE can be found in `est$gridx` and `est$gridy`. Additionally, the values for sample size, burn-in and adaptive are stored in `est`.

In order to plot the result, the data frame `Kdata` with all combinations of grid points with respect to their longitude and latitude coordinates as well as the corresponding density estimation obtained by the proposed algorithm is needed. The function `expand.grid` creates a data frame from all combinations of the supplied variables. Then the normed density per grid cell is added to the data frame. Additionally, the function `filter` ensures that only grid points with positive densities are included in the data frame. The data `Kdata` can now be visualized using the **ggplot2** package, which provides various options for graphical representations.

```
# Prepare KDE as data frame for plot
Kdata <- data.frame(expand.grid(long = est$Mestimates$eval.points[[1]],
                                lat = est$Mestimates$eval.points[[2]]),
                    Density = est$Mestimates$estimate %>% as.vector) %>%
  filter(Density > 0)
Kdata$Density <- Kdata$Density*(est$gridx[2] - est$gridx[1])*
  (est$gridy[2] - est$gridy[1])

# Plot KDE obtained by aggregated data
```

```
ggplot(Kdata) +
  geom_raster(aes(long, lat, fill = Density)) +
  ggtitle("Density of Inhabitants between 65 and 80 Years",
          "obtained by using Kernel Heaping Algorithm") +
  scale_fill_gradientn(colours = c("#FFFFFF", "#5c87c2", "#19224e"))+
  xlab("Longitude")+ylab("Latitude")+
  geom_sf(data = Berlin, fill = NA) +
  coord_sf()
```

Figure 2 illustrates the bivariate density of elderly inhabitants by using the proposed approach. The impact of kernel heaping becomes evident when comparing Figure 2 with the conventional choropleth maps commonly used in official statistics shown in Figure 1. Figure 1 appears unrealistic compared to Figure 2 due to the abrupt color shifts at the boundaries of the regions. The discrete color display obscures information, i.e. regional concentrations within the districts remain hidden.

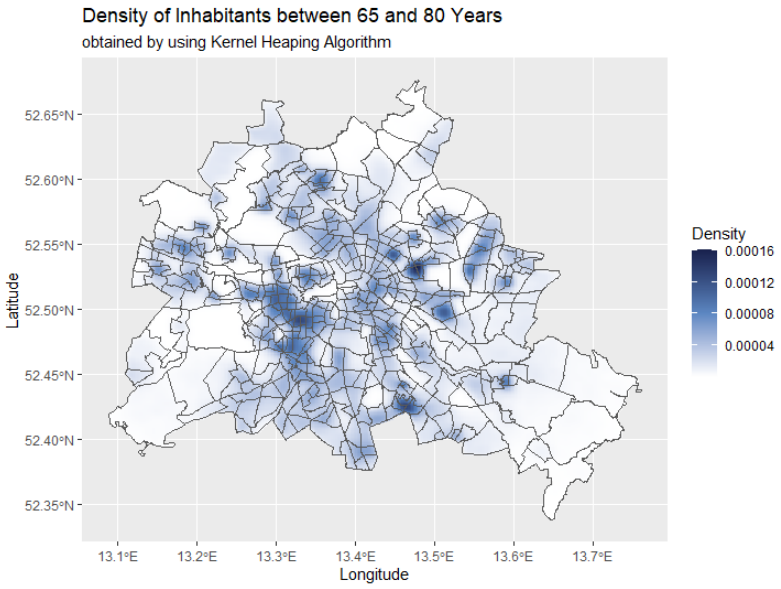


**Figure 2:** Bivariate density of the number of inhabitants between 65 and 80 years in Berlin obtained by the SEM algorithm

### 4.2 Part 2: Creating spatial maps for proportions

This section focuses on creating maps which calculate **proportions** using data of individuals with a Turkish migration background within the population with migration background of Berlin.

To ensure that only settled areas are considered, it is necessary to exclude unsettled regions such as parks, lakes or forests. The corresponding shape files for uninhabited areas can be downloaded from http://download.geofabrik.de/europe/germany/berlin.html. Similar to the previous section, the shape file containing the LORs of Berlin is uploaded in **R**. Additionally, land use data `BerlinN` and water areas `BerlinWater` are loaded in **R**. Land use characterizes what an area is used for, e.g., residential, commercial activities, agriculture, education, recreation, etc. and water areas refers to areas with rivers, lakes etc. The residential areas are then exempted from `BerlinN` and further divided into green areas and other areas, to distinguish between different land uses within the unsettled regions.

```
# Load Berlin shape files
Berlin <- read_sf("RBS_OD_LOR_2015_12/RBS_OD_LOR_2015_12.shp")
Berlin <- st_transform(Berlin, CRS("+proj=longlat +datum=WGS84"))

# Load shape files of green area, water areas and others
BerlinN <- read_sf("berlin-latest-free.shp/gis_osm_landuse_a_free_1.shp")
BerlinWater <- read_sf("berlin-latest-free.shp/gis_osm_water_a_free_1.shp")

BerlinN <- BerlinN[!(BerlinN$fclass == "residential"), ]
BerlinGreen <- BerlinN[(BerlinN$fclass %in% c("forest", "grass", "nature_reserve",
                                              "park", "cemetery","allotments",
```

```
                                                    "farm", "meadow","orchard",
                                                    "vineyard","heath")), ]
BerlinOther <- BerlinN[!(BerlinN$fclass %in% c("forest", "grass", "nature_reserve",
                                                "park", "cemetery","allotments",
                                                "farm", "meadow","orchard",
                                                "vineyard","heath")), ]
```

After reading in the shape file corresponding to the unsettled areas by using `read_sf`, the data is again transformed into WGS84 system by `st_transform`. Additionally, to reduce the complexity of the polygons, they are simplified by the Ramer-Douglas-Peucker algorithm using `ms_simplify`. The algorithm decimates a curve consisting of line segments to a similar curve with fewer points. This is done for this case study due to computational efficiency. Since the data extend beyond the area of Berlin, i.e. the data reach as far as Brandenburg, the data must be restricted to Berlin using `st_intersection`. The functions `dshapebivr` and `dshapebivrProp` require a single shape file as input data and therefore water, nature and other shape files are united by `bind_rows` and transformed into a `Large SpatialPolygonsDataFrame` using `as_Spatial`.

```
# Simplify the uninhabited areas for computational purpose
BerlinGreen <- st_transform(
  rmapshaper::ms_simplify(BerlinGreen, keep = 0.001, keep_shapes=FALSE),
                          CRS("+proj=longlat +datum=WGS84"))
BerlinOther <- st_transform(
  rmapshaper::ms_simplify(BerlinOther, keep = 0.001, keep_shapes=FALSE),
                          CRS("+proj=longlat +datum=WGS84"))
BerlinWater <- st_transform(
  rmapshaper::ms_simplify(BerlinWater, keep = 0.001, keep_shapes=FALSE),
                          CRS("+proj=longlat +datum=WGS84"))

BerlinOther <- st_intersection(BerlinOther, Berlin)
BerlinGreen <- st_intersection(BerlinGreen, Berlin)
BerlinWater <- st_intersection(BerlinWater, Berlin)

BerlinUnInhabitated <- bind_rows(BerlinWater, BerlinGreen, BerlinOther)
BerlinUnInhabitated <- as_Spatial(BerlinUnInhabitated)
```

The differences between the simplified polygons and the original one strongly depend on the parameters. However, also small changes can achieve run time reductions. In Figure 3 the polygons with respect to the water areas of Berlin are plotted. It can be seen that the tuning parameter was chosen to large, hence it had a significant impact on the areas. For to enhance the readability, the code for Figure 3 is not provided.

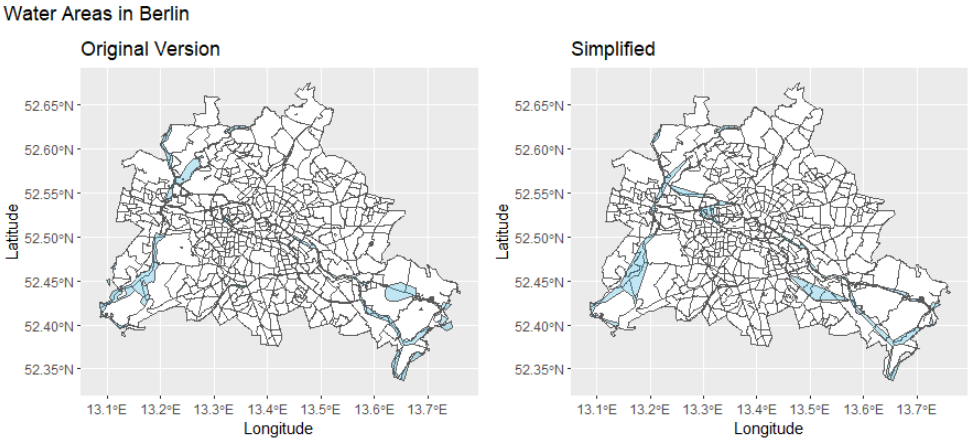


**Figure 3:** Original data of the water areas in Berlin (left) based on polygons and simplified polygons using Ramer-Douglas-Peucker algorithm with a tolerance value of 0.002 (right)

The data for this application contains the total number of migrants in Berlin are uploaded in **R**. The CSV file can be found under https://www.statistik-berlin-brandenburg.de/opendata/EWRMIGRA201512H_Matrix.csv. At this stage, the key variable `RAUMID` is converted into a character

variable. Additionally, as the length of the variable RAUMID is between 7 and 8, all variables are transformed to the length of 8. Therefore, the leading digits of variables of length 7 in the RAUMID is padded with a 0 to ensure consistent formatting. Finally, the number of migrants can be sorted in ascending order based on the RAUMID.

Similar to the first part, a data matrix is determined to connect the midpoints of the polygons with the corresponding total number of Turkish migrants. Therefore, the longitude and latitude information of the midpoints are connected with the corresponding number of Turkish migrants as well as the total number of migrants. Note that in this application the Large SpatialPolygonsDataFrame called Berlin.shp is used to determine the midpoints of each LOR.

```
# Load data frame with migrant data
BerlinMigration <- read.csv2("EWRMIGRA201512H_Matrix.csv")
BerlinMigration$RAUMID <- as.character(BerlinMigration$RAUMID)
BerlinMigration$RAUMID[nchar(BerlinMigration$RAUMID) == 7] <-
  paste0("0", BerlinMigration$RAUMID[nchar(BerlinMigration$RAUMID) == 7])
BerlinMigration <- BerlinMigration[order(BerlinMigration$RAUMID), ]

# Add proportion of Turkish mirgants to shape file
Berlin$turkDensity <- BerlinMigration$HK_Turk / BerlinMigration$MH_E
```

In Figure 4 the proportion of Turkish migrants with respect to all migrants is shown as a chorophlet map. Note that in the KDE evaluation using SEM algorithm the unsettled areas have been excluded. However, when visualizing the proportions of migrants using choropleth maps, these areas are not excluded. Therefore, in Figure 4 the intersections between the visualized proportions of migrants and unsettled areas are displayed. This is an additional disadvantage of the common used chorpleth map. In dark green Berlin's Green areas (forests, parks, etc.), in blue Berlin's Water Areas (rivers, lakes, etc.) and in grey Berlin's other unsettled areas are visualized.

```
# Plot proportions of Turkish mirgants as choropleth map
ggplot() +
  geom_sf(data = Berlin, mapping = aes(fill = turkDensity)) +
  ggtitle("Proportion of Turks in Population with Migration Background",
          subtitle = "Choropleth Map") +
  scale_fill_gradientn(colours = c("#FFFFFF", "coral1"), "Density") +
  geom_sf(fill = "grey20", alpha = .25,color = NA,
              data = BerlinOther) +
  geom_sf(fill = "darkolivegreen3", color = NA,
              data = BerlinGreen, alpha = 0.25) +
  geom_sf(fill = "deepskyblue3",color = NA,
              data = BerlinWater, alpha = 0.25) +
  xlab("Longitude")+ ylab("Latitude")+
  coord_sf()


# Create data frame with midpoint of areas, counts of overall population
# and Turkish population per area
dataTurk <- cbind(t(sapply(1:length(Berlin.shp@polygons),
                           function(x) Berlin.shp@polygons[[x]]@labpt)),
                  BerlinMigration$HK_Turk, BerlinMigration$MH_E)
```

The function dshapebivrProp shares many similarities with dshapebivr as both are used to estimate kernel density for data classified in polygons by using SEM algorithm based on heaped data. Most of the parameters used in dshapebivrProp are the same as in dshapebivr. However, dshapebivrProp specifically focuses on estimating the density proportions. Additionally, in this application uninhabited areas are removed using the parameter deleteShapes.

```
# Use aggregated data to obtain local proportions
EstTurk <- dshapebivrProp(data = dataTurk, burnin = 5, samples = 10,
                          adaptive = FALSE, deleteShapes = BerlinUnInhabitated,
                          shapefile = Berlin.shp, gridsize = 325, boundary = TRUE,
                          numChains = 4, numThreads = 4)
```

The output EstTurk is similar to the output est of the function dshapebivr in Part 1. EstTurk consist of input information such as shapefile, burnin, sample, deleteShape etc. Most important is, however, Mestimate the corrected KDE using the proposed approach.

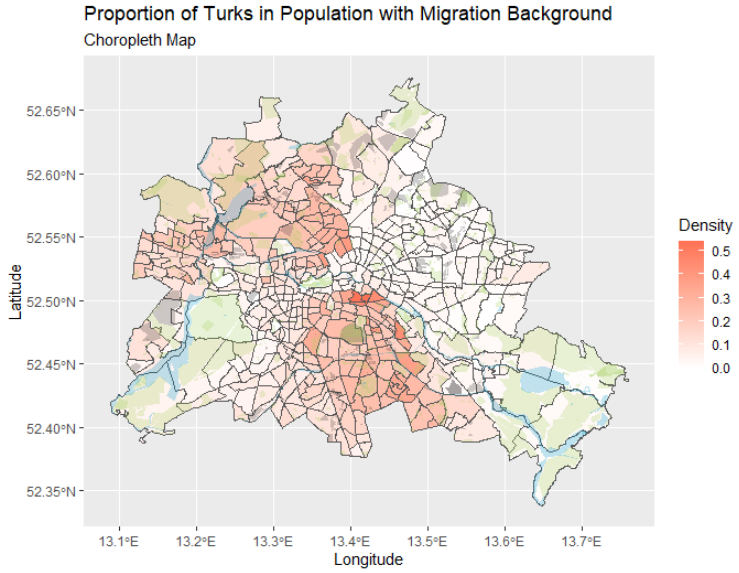


**Figure 4:** Choropleth map of the proportion of Turkish people in the population with migration background per LOR

For visualizing the KDE, all grid points are obtained by getting all combinations of longitude and latitude coordinates using `expand.grid`. Again, negative proportion values are filter out. In Figure 5 the KDE using the proposed approach of the proportions of Turkish migrants with respect to all migrants are visualized. As uninhabited areas are excluded from the estimation, in comparison to the choropleth map in Figure 4, no values occur in these areas as well as smooth transitions of the proportions are visible.

```
# Prepare local proportions as data frame for plot
gridBerlin <- expand.grid(long = EstTurk$Mestimates$eval.points[[1]],
                          lat = EstTurk$Mestimates$eval.points[[2]])
KdataTurk <- data.frame(gridBerlin,
                        Proportion = EstTurk$proportions %>% as.vector)  %>%
                        filter(Proportion > 0)

# Plot local proportions obtained from aggregated data
ggplot() +
  geom_tile(data = KdataTurk, aes(long, lat, fill = Proportion)) +
  ggtitle("Proportion of Turks in Population with Migration Background",
          "obtained by using Kernel Heaping Algorithm") +
  scale_fill_gradientn(colours = c("#FFFFFF", "coral1"), "n") +
  geom_sf(fill = "grey20", alpha = .25,color = NA,
          data = BerlinOther) +
  xlab("Longitude")+ ylab("Latitude")+
  geom_sf(fill = "darkolivegreen3", color = NA,
          data = BerlinGreen, alpha = .25) +
  geom_sf(fill = "deepskyblue3",color = NA,
          data = BerlinWater, alpha = .25) +
  xlab("Longitude")+ ylab("Latitude")+
  coord_sf()
```

### 4.3 Part 3: Identification of high density areas

In the final part, the focus lies on the identification of high-density regions using the example of different migration groups in Berlin.

Similar to the previous part, the centers of the polygons are linked with the total number of Arabic, former Soviet Union, and Polish migrants. The bivariate kernel density by the proposed approach is then evaluated using `dhapebivr` with similar input parameters as in Part 1.

```
# Create data frame with midpoint of areas, counts of overall population
```

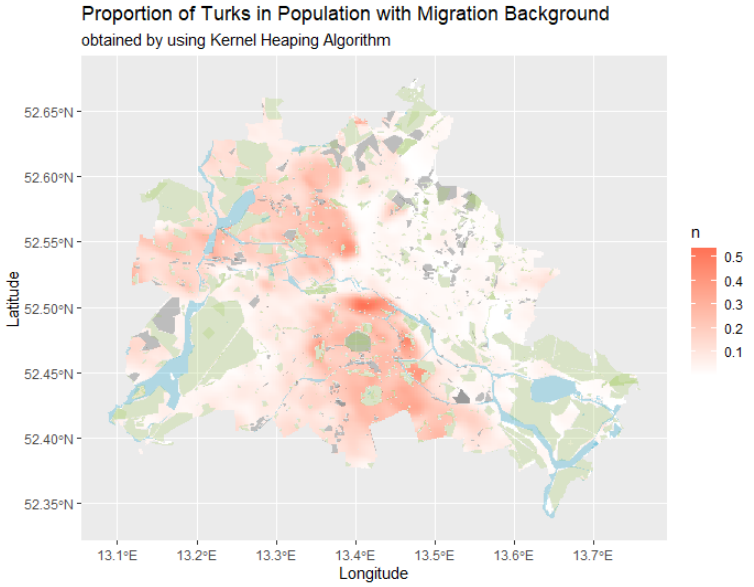


**Figure 5:** Kernel Density of the Proportion of Turkish people in the population with migration background using proposed approach

```
# and Arabic, former Soviet and Polish population per area
dataArab <- cbind(t(sapply(1:length(Berlin.shp@polygons),
                           function(x) Berlin.shp@polygons[[x]]@labpt)),
                           BerlinMigration$HK_Arab)
dataSU <- cbind(t(sapply(1:length(Berlin.shp@polygons),
                         function(x) Berlin.shp@polygons[[x]]@labpt)),
                         BerlinMigration$HK_EheSU)
dataPol <- cbind(t(sapply(1:length(Berlin.shp@polygons),
                          function(x) Berlin.shp@polygons[[x]]@labpt)),
                          BerlinMigration$HK_Polen)

# Use aggregated data to obtain local propotions
EstArab <- dshapebivr(data = dataArab, burnin = 5, samples = 10, adaptive = FALSE,
                      shapefile = Berlin.shp, gridsize = 325, boundary = TRUE)
EstSU <- dshapebivr(data = dataSU, burnin = 5, samples = 10, adaptive = FALSE,
                    shapefile = Berlin.shp, gridsize = 325, boundary = TRUE)
EstPol <- dshapebivr(data = dataPol, burnin = 5, samples = 10, adaptive = FALSE,
                     shapefile = Berlin.shp, gridsize = 325, boundary = TRUE)
```

To identify "hot spots" or regions with significantly higher density than others, a predetermined quantile needs to be established. In this case, we set the `visquantile` to 0.95. This means that in 95 percent of the grid points the estimate of the percentage is lower than the given value.

The data is prepared in the same manner as in the previous two parts.

```
# prepare local proportions as data frame for plot
# Plot only high density regions
Visquantile <- 0.95
KdataArab <-
    data.frame(gridBerlin, Density = EstArab$Mestimates$estimate %>% as.vector) %>%
        filter(Density > 0) %>%
        filter(Density > quantile(Density, Visquantile)) %>%
        mutate(Density = "Arabian countries")
KdataSU <-
    data.frame(gridBerlin, Density = EstSU$Mestimates$estimate %>% as.vector) %>%
        filter(Density > 0) %>%
        filter(Density > quantile(Density, Visquantile)) %>%
        mutate(Density = "Former Soviet Union")
KdataPol <-
    data.frame(gridBerlin, Density = EstPol$Mestimates$estimate %>% as.vector) %>%
        filter(Density > 0) %>%
```

```
        filter(Density > quantile(Density, Visquantile)) %>%
        mutate(Density = "Poland")

# Plot local proportions by aggregated data
ggplot() +
  geom_sf(data = Berlin) +
  geom_tile(aes(long, lat, fill = Density), data = KdataArab, alpha = 0.6) +
  geom_tile(aes(long, lat, fill = Density), data = KdataSU, alpha = 0.6) +
  geom_tile(aes(long, lat, fill = Density), data = KdataPol, alpha = 0.6) +
  ggtitle("Hotspots of Inhabitants with different Migration Background",
          "obtained by using Kernel Heaping Algorithm") +
  geom_sf(fill = "grey20", alpha = .25,color = NA,
          data = BerlinOther) +
  xlab("Longitude")+ ylab("Latitude")+
  geom_sf(fill = "darkolivegreen3", color = NA,
          data = BerlinGreen, alpha = .25) +
  geom_sf(fill = "deepskyblue3",color = NA,
          data = BerlinWater, alpha = .25) +
  xlab("Longitude")+ ylab("Latitude")+
coord_sf()
```

The result is a hot spot map displaying the density of former Soviet Union in green, Arab in red, and Polish migrants in blue visualized in Figure 6. By employing standard bivariate kernel density estimation, proportions estimation and hotspot identification techniques, it is possible to address the limitations of choropleth maps. The new resulting maps exhibit enhanced visual representation due to the incorporation of kernel heaping. This approach effectively highlights regional concentrations within the life-oriented regions (LORs) and hotspots can be identified.

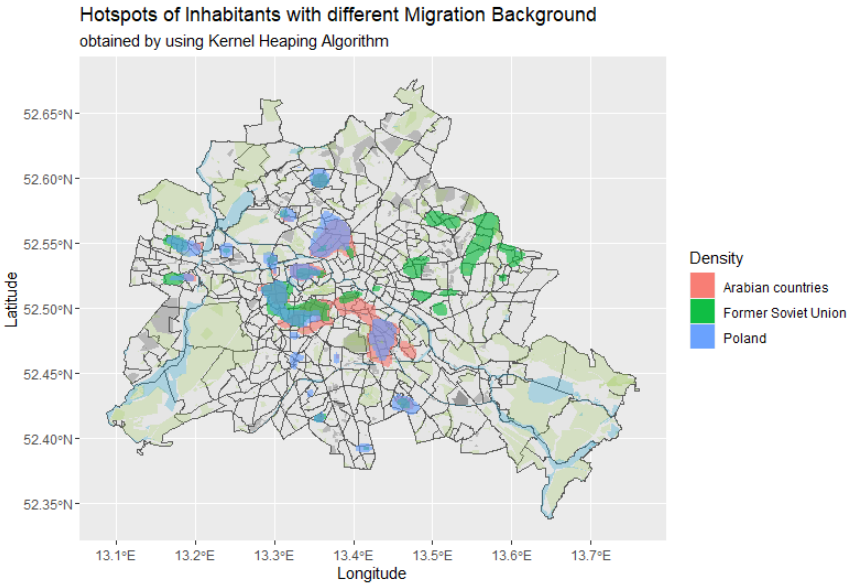


**Figure 6:** Hot spots of migrants from the former Soviet Union as well as Arab, and Polish migrants constructed from the Kernel density estimates.

## 5 Robustness analysis based on a simulation study

In order to validate the method based on different data structures, sample data points in the form of coordinates have been randomly generated with various spreads, i.e. sample data were generated from a mixture of 111 normal distributions with different standard deviations. For each of the 111 midpoints with varying the standard deviation, 800 coordinates were generated. The standard deviation varies from $3 \cdot 10^{-4.8} \cdot I_2$, for highly agglomerated data, $3 \cdot 10^{-4.55} \cdot I_2$, for large agglomeration clusters, to $3 \cdot 10^{-3} \cdot I_2$, for nearly uniformly distributed data. Additionally, a combination of these three standard deviations each associated with 37 midpoints has been used for data generation and named $C_{NMD}$. Since green areas, water areas and uninhabited areas in Berlin are known, those points that

were randomly placed in one of these areas or fell outside of Berlin were excluded. Each setting has been randomly simulated 500 times. Figure 7 shows the data variations and is intended to represent the highly agglomerated data ($3 \cdot 10^{-4.8} \cdot I_2$), less agglomerated data ($3 \cdot 10^{-4.55} \cdot I_2$), nearly equally distributed data ($3 \cdot 10^{-3} \cdot I_2$) and data resulting from the combination of 3 different standard deviations ($C_{NMD}$).

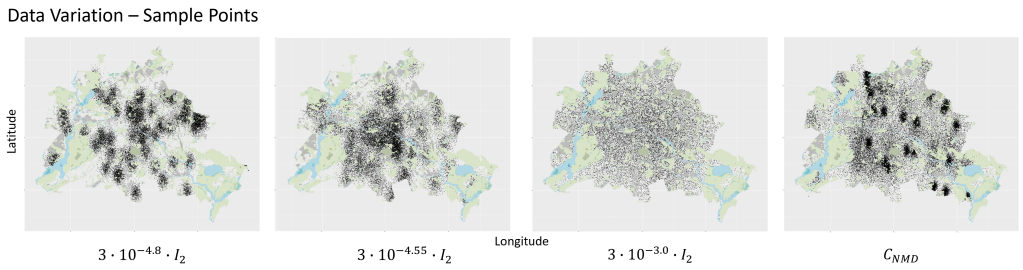


**Figure 7:** Different data generation processes by using Gaussian Mixture distribution with various variances

Based on these data structures, the randomly drawn sample points have been aggregated to a wide range of area systems, i.e. from districts with 12 areas, and Prognosis areas with 60 areas to Zip codes with 193 areas and Lifeworld-oriented spaces with 442 areas. Based on aggregated data, the kernel heaping method aiming for a smooth representation is applied. For the comparison of two densities over the considered domain, the Root Mean Integrated Squared Error ($RMISE$) is used, which is defined for the true density $f$ and the estimated density $\hat{f}_H$ as

$$RMISE(f, \hat{f}_H) = \frac{1}{G} \sum_{g=1}^{G} (f(x_g) - \hat{f}_H(x_g))^2 \Delta_G.$$

Figure 8 shows the information loss for different variants of the data structures on the horizontal axis, as well as aggregation error and simulated smoothing error by kernel heaping as separated plots. The information loss is determined by normalizing the considered error by the sampling error, which occurs through drawing samples from the known mixture of normal distributions. The aggregation of the sampling data points on area systems with fewer areas highly increases the aggregation error. Additionally, the data structure is influential. The kernel heaping algorithm uses the aggregated information to generate a smooth density through the SEM algorithm. This smooth density drastically reduces the loss of information which is enhanced by the smoothing effect of the algorithm. Nevertheless, for highly agglomerated data the effect is less because highly agglomerated data clusters cannot be reconstructed, i.e. when multiple clusters have been aggregated into one area. The simulation study shows that an information loss between 25% and 50% can be improved with respect to $RMISE$, depending on the underlying data structure.

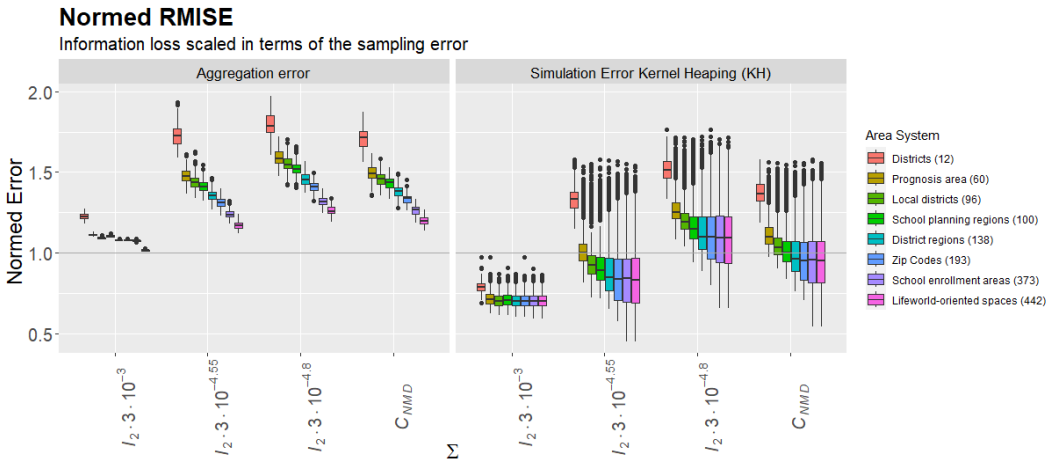


**Figure 8:** Information Loss by Normed Root Mean Integrated Squared Error with respect to the Sampling Error for Aggregation processes as well as Kernel Heaping compared to true density for different area systems and data gerneration processes.

The computation time depends strongly on the grid used to evaluate the kernel density. The finer the grid, the longer the computation time. To generate a 100 × 100 meter grid over Berlin, which has an extension of 45 km in north-south direction and 38 km in east-west direction, the area has to be divided into 450 and 380 grid cells per direction, respectively. As the number of grid cells decreases, the computation time decreases. In general, as the number of areas on which the aggregation is based increases, the computation time increases, as seen in the table 2. The data structure on which the aggregation is based plays a minor role and causes the results to vary by a few seconds. Note that a burn-in phase of 5 iterations and a sampling phase of 10 iterations are used. The computations are based on a 12th Gen Intel(R) Core(TM) i5-12500, 3000 MHz, 6 cores processor.

| Number of Areas | 12 | 60 | 96 | 100 | 138 | 193 | 373 | 442 |
|---|---|---|---|---|---|---|---|---|
| 100m × 100m | 270.99 | 282.96 | 304.64 | 277.95 | 287.09 | 316.71 | 309.90 | 301.75 |
| 200m × 200m | 62.02 | 62.87 | 69.86 | 64.09 | 64.41 | 70.34 | 70.33 | 67.95 |
| 500m × 500m | 14.33 | 13.98 | 14.78 | 14.45 | 15.01 | 14.46 | 14.73 | 15.11 |

**Table 2:** Overview of the mean computation time (in seconds) for different area systems and grid sizes

### 5.1 Kernel Heaping under different Bandwidth selectors

An automatic bandwidth selection via the plug-in bandwidth by Wand and Jones (1994) is implemented in the **Kernelheaping** package. The calculation of the KDE in each iteration step is based on the `kde()` function of the **ks** package, which offers other automatic bandwidth calculation methods besides the plug-in bandwidth. In addition to the plug-in bandwidth, the biased cross-validation (BCV) bandwidth matrix selector by Stephan R. Sain and Scott (1994), the smoothed cross-validation (SCV) bandwidth selector by Duong and Hazelton (2005) and the least-squares cross-validation (LSCV) bandwidth selector by Bowman (1984) are evaluated. Instead of minimizing $AMISE(H)$ by an plug-in approach, cross-validation based approaches attempt to minimize the $MISE$ by using data for KDE computation and evaluation of its performance. To avoid dependencies the data for computation is not used for evaluation. The least-squares cross validation selector uses a leave-one-out approach and numerical minimization. The biased cross validation combines the plug-in approach with the idea of cross validation. By considering to minimize the $AMISE(H)$ an estimation of $\phi_4$ is modified by an leave-out-diagonals approach. For details see Stephan R. Sain and Scott (1994). Note that there are two BCV estimates, where the first is used in this simulation. The smoothed cross-validation can be viewed as a combination of the LSCV and BCV Duong and Hazelton (2005).

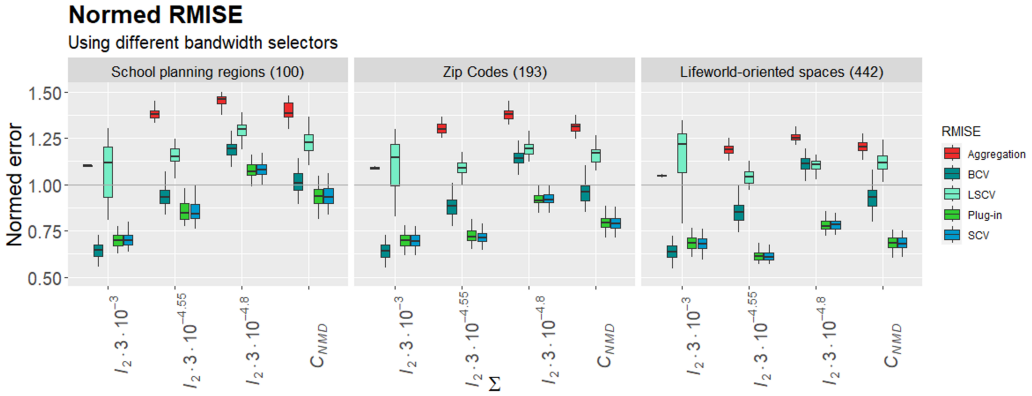


**Figure 9:** Information Loss by Normed Root Mean Integrated Squared Error for Aggregation process and Kernel Heaping using different bandwidth selectors under selected area systems and different data generation processes

In Figure 9 the information loss based on the normed $RMISE$ for different automatic bandwidth selectors is compared with the information loss of the aggregation process. As before, sample data from a Gaussian mixture distribution have been aggregated to three selected area systems. Based on the aggregated data the kernel heaping algorithm is applied with different automatic bandwidth selectors on a 500 × 500 meter grid. The plug-in bandwidth selector which used in the **Kernelheaping** is compared to the BCV, LSCV and SCV. Expect from nearly equally distributed data combined with the LSCV, the kernel heaping algorithm could reduce the information loss. The LSCV bandwidth selector performed worse followed by the BCV. The plug-in bandwidth selector and the SCV bandwidth

selector perform similarly well in the context of the global error measure. Nevertheless, the mean computation time of the SCV is approximately 3.6 times higher compared to the plug-in bandwidth selector. For school planning regions, for example, the mean computation time of the $C_{NMD}$ setting is 15.75 seconds for the plug-in selector compared to 56.52 seconds for the SCV selector. Note that the computation time for LSCV is 112.40 seconds and for BCV 332.53 seconds. Hence, compared to these automatic bandwidth selectors the plug-in bandwidth is very efficient. The computations are based on a 12th Gen Intel(R) Core(TM) i5-12500, 3000 MHz, 6 cores processor.

## 6 Summary

The **Kernelheaping** package is a useful tool for representing georeferenced data, which are given only as aggregates, by realistic kernel density estimates. Among the two dimensional cases shown, the implementation in **R** can be used for one dimensional rounding effects as well as three dimensional data established in the function `dshape3dprop()`. The focus in the case studies lied in two dimensional aggregations of population data, and in particular in the handling on shape files, etc., which are essential for georeferenced data. The function `toOtherShape` also handles the support problem, i.e. the change between non-hierarchical area systems. Since the basic idea behind the algorithm is the aggregation of populations, and the drawing of pseudo-samples reflects the drawing of a simulated population, the algorithm can only be meaningfully applied to integer non-negative values. In addition to the use of exemplary data of the population between 65 and 80 and people with a migration background, a simulation was conducted. It was demonstrated that the performance of the algorithm depends on both the underlying data structure and the size of the aggregation areas. Furthermore, the efficient use of the plug-in bandwidth was shown in comparison to CV-based methods.

## 7 Acknowledgement

Parts of this work was supported by the collaborative project *AnigeD* under funding reference number *16KISA097*.

## References

G. Basulto-Elias, A. Carriquiry, and K. e. a. De Brabanter. Bivariate kernel deconvolution with panel data. *Sankhya B*, 83:122–151, 2021. [p116]

A. W. Bowman. An alternative method of cross-validation for the smoothing of density estimates. *Biometrika*, 71(2):353–360, 1984. ISSN 00063444. URL http://www.jstor.org/stable/2336252. [p130]

R. J. Carroll and L. A. Stefanski. Approximate quasi-likelihood estimation in models with surrogate predictors. *Journal of the American Statistical Association*, 85(411):652–663, 1990. ISSN 01621459. URL http://www.jstor.org/stable/2290000. [p116]

G. Celeux, D. Chauveau, and J. Diebolt. Stochastic versions of the EM algorithm: an experimental study in the mixture case. *Journal of Statistical Computation and Simulation*, 55(4):287–314, nov 1996. doi: 10.1080/00949659608811772. [p115]

A. Delaigle. Nonparametric kernel methods for curve estimation and measurement errors. *Proceedings of the International Astronomical Union*, 10:28 – 39, 2014. URL https://api.semanticscholar.org/CorpusID:125933456. [p116]

T. Duong and M. Hazelton. Plug-in bandwidth matrices for bivariate kernel density estimation. *Journal of Nonparametric Statistics*, 15(1):17–30, 2003. doi: 10.1080/10485250306039. [p117]

T. Duong and M. L. Hazelton. Cross-validation bandwidth matrices for multivariate kernel density estimation. *Scandinavian Journal of Statistics*, 32(3):485–506, 2005. doi: https://doi.org/10.1111/j.1467-9469.2005.00445.x. URL https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1467-9469.2005.00445.x. [p130]

K. Erfurth, M. Groß, U. Rendtel, and T. Schmid. Kernel density smoothing of composite spatial data on administrative area level. *AStA Wirtschafts- und Sozialstatistisches Archiv*, 16(1):25–49, dec 2021. doi: 10.1007/s11943-021-00298-9. [p116]

F. Farokhi. Deconvoluting kernel density estimation and regression for locally differentially private data. *Scientific Reports*, 10(1), Dec. 2020. ISSN 2045-2322. doi: 10.1038/s41598-020-78323-0. [p116]

M. Groß and U. Rendtel. Kernel Density Estimation for Heaped Data. *Journal of Survey Statistics and Methodology*, 4(3):339–361, sep 2016. doi: 10.1093/jssam/smw011. [p116]

M. Groß, U. Rendtel, T. Schmid, S. Schmon, and N. Tzavidis. Estimating the Density of Ethnic Minorities and Aged People in Berlin: Multivariate Kernel Density Estimation Applied to Sensitive Georeferenced Administrative Data Protected via Measurement Error. *Journal of the Royal Statistical Society Series A: Statistics in Society*, 180(1):161–183, feb 2016. doi: 10.1111/rssa.12179. [p116]

M. Groß, A.-K. Kreutzmann, U. Rendtel, T. Schmid, and N. Tzavidis. Switching Between Different Non-Hierachical Administrative Areas via Simulated Geo-Coordinates: A Case Study for Student Residents in Berlin. *Journal of Official Statistics*, 36(2):297–314, jun 2020. doi: 10.2478/jos-2020-0016. [p116]

S. Hadam, T. Schmid, and J. Simm. Kleinräumige Prädiktion von Bevölkerungszahlen basierend auf Mobilfunkdaten aus Deutschland. In *Schriftenreihe der ASI - Arbeitsgemeinschaft Sozialwissenschaftlicher Institute*, pages 27–44. Springer Fachmedien Wiesbaden, 2020. doi: 10.1007/978-3-658-31009-7_3. [p116]

N. Heidenreich, A. Schindler, and S. Sperlich. Bandwidth selection for kernel density estimation: a review of fully automatic selectors. *AStA Advanced Statistical Analysis*, 97:403 – 433, 2014. [p117]

W. Härdle. *Applied nonparametric regression*. Cambridge University Press, Cambridge, 1990. [p119]

A. Izenman. Recent Developments in Nonparametric Density Estimation. *Journal of the American Statistical Association*, 86(413):205–224, 1991. ISSN 0943-4062. [p117]

M. C. Jones. Simple boundary correction for kernel density estimation. *Statistics and Computing*, 3(3): 135–146, sep 1993. doi: 10.1007/bf00147776. [p118, 122]

U. Rendtel and M. Ruhanen. Die Konstruktion von Dienstleistungskarten mit Open Data am Beispiel des lokalen Bedarfs an Kinderbetreuung in Berlin. *AStA Wirtschafts- und Sozialstatistisches Archiv*, 12 (3-4):271–284, nov 2018. doi: 10.1007/s11943-018-0235-y. [p116]

U. Rendtel, A. Neudecker, and L. Fuchs. Ein neues Web-basiertes Verfahren zur Darstellung der Corona-Inzidenzen in Raum und Zeit. *AStA Wirtschafts- und Sozialstatistisches Archiv*, 15(2):93–106, jun 2021. doi: 10.1007/s11943-021-00288-x. [p116]

B. Silverman. *Density Estimation for Statistics and Data Analysis*. Chapman & Hall, London, 1986. [p117]

K. A. B. Stephan R. Sain and D. W. Scott. Cross-validation of multivariate densities. *Journal of the American Statistical Association*, 89(427):807–817, 1994. doi: 10.1080/01621459.1994.10476814. [p130]

P. Walter, M. Groß, T. Schmid, and K. Weimer. Iterative Kernel Density Estimation Applied to Grouped Data: Estimating Poverty and Inequality Indicators from the German Microcensus. *Journal of Official Statistics*, 38:599–635, 2022. doi: 10.2478/jos-2022-0027. [p116]

M. P. Wand and C. Jones. Multivariate plug-in bandwidth selection. *Computational Statistics*, 9(2): 97–116, 1994. ISSN 0943-4062. [p117, 118, 119, 122, 130]

*Lorena Gril*
*Freie Universität Berlin, FB Wirtschaftswissenschaft*
*Garystr. 21, D-14195 Berlin*
*Germany*
lorena.gril@fu-berlin.de

*Laura Steinkemper*
*Freie Universität Berlin, FB Wirtschaftswissenschaft*
*Garystr. 21, D-14195 Berlin*
*Germany*
steinkel98@zedat.fu-berlin.de

*Marcus Groß*
*INWT Statistics GmbH*

*Hauptstr.8, D-10827 Berlin*
*Germany*
Marcus.Gross@inwt-statistics.de

*Ulrich Rendtel*
*Freie Universität Berlin, FB Wirtschaftswissenschaft*
*Garystr. 21, D-14195 Berlin*
*Germany*
Ulrich.Rendtel@fu-berlin.de

# SLCARE: An R Package for Semiparametric Latent Class Analysis of Recurrent Events

*by Qi Yu and Limin Peng*

**Abstract** Recurrent event data frequently arise in biomedical follow-up studies. The concept of latent classes enables researchers to characterize complex population heterogeneity in a plausible and parsimonious way. This article introduces the R package SLCARE, which implements a robust and flexible algorithm to carry out Zhao, Peng, and Hanfelt (2022)'s latent class analysis method for recurrent event data, where semiparametric multiplicative intensity modeling is adopted. SLCARE returns estimates for non-functional model parameters along with the associated variance estimates and $p$ values. Visualization tools are provided to depict the estimated functional model parameters and related functional quantities of interest. SLCARE also delivers a model checking plot to help assess the adequacy of the fitted model.

## 1 Introduction

Recurrent event data frequently arise in biomedical follow-up studies where the event of interest, such as infection or hospitalization, occurs repeatedly over time. The event recurrence often demonstrates different patterns across individuals. To accommodate such heterogeneity, many regression methods have been developed with implementation available as R packages. To name a few, analyses based on the proportional intensity model (Andersen and Gill, 1982) can be carried out with function `coxph()` from R package **survival** (Therneau, 2023) or `cph()` from R package **rms** (Harrell Jr, 2023). Fitting Wang et al. (2001) 's method based on a multiplicative intensity model is implemented by function `reReg()` from R package **reReg** (Chiou et al., 2023). Modeling the gap time between recurrent events, Clement and Strawderman (2009) proposed conditional generalized estimating equation, and developed R package **condGEE** implementing this approach. Fine et al. (2004) 's temporal regression strategy can be applied to regress the mean function of recurrent events through using function `tpr()` from R package **tpr** (Yan and Fine, 2004).

More recently, viewing the observed data as a manifestation of latent classes or subgroups, Zhao et al. (2022) proposed a semiparametric latent class analysis (LCA) method to help reveal more realistic heterogeneity structure of recurrent event data that may not be adequately captured by a single regression model. Zhao et al. (2022) adopted latent variable mixture modeling (LVMM) while avoiding stringent parametric assumptions commonly imployed in LCA literature. Note that several R packages are available for fitting LVMM for data types other than recurrent event data. For example, R package **flexmix** (Grün and Leisch, 2008) delivers a general tool for tackling finite mixtures of regression models, such as the standard linear model and the generalized linear model, with the expectation-maximization (EM) algorithms. R package **lcmm** (Proust-Lima et al., 2017) allows for fitting joint mixture models with longitudinal and single time-to-event outcomes based on the Newton-Raphson algorithm. However, none of these packages offer options to conduct LCA oriented to recurrent event outcomes.

In this paper, we introduce the R package **SLCARE**, which delivers robust and flexible implementation of Zhao et al. (2022) 's LCA method for recurrent event data and is available from the Comprehensive R Archive Network (CRAN) at `https://CRAN.R-project.org/package=SLCARE` and Github at `https://github.com/qyxxx/SLCARE`. The package **SLCARE** offers a user-friendly software for conducting LCA of recurrent event data in R with the core function `SLCARE()`. The function `SLCARE()` enables a variety of options for specifying and estimating the flexible semiparametric latent class model of Zhao et al. (2022) . For example, the number of latent classes can be pre-specified or chosen based on an entropy-based measure. Users can opt to built-in or customized initializer. Algorithm convergence criterion can be set through specifying the maximum number of iterations or the minimum change in parameter estimates. The function `SLCARE()` also provides visual tools to aid in result presentation and interpretation as well as model checking upon **ggplot2** plotting environment (Wickham, 2016).

The remainder of this paper is organized as follows. Next section describes the methodological background for the R package **SLCARE**, including model assumptions, estimation procedure and algorithm, inference, and model evaluation. In the third section, we elaborate the structure of the package. The utility of the package is illustrated via simulated data examples in Section 4. A real application is presented in Section 5. We conclude with a few remarks in the last section.

## 2 Methodological background

### 2.1 Recurrent event data and notation

For subject $i$, let $T_i^{(j)}$ denote time to the $j^{th}$ recurrent event ($i = 1, \ldots, n$, $j = 1, 2, \ldots$). The underlying counting process of recurrent events is defined as $N_i^*(t) = \sum_{j=1}^{\infty} I(T_i^{(j)} \leq t)$ ,representing the number of events occurred before or at time $t$ for subject $i$, where $I(\cdot)$ denotes the indicator function. Suppose the observation of subject $i$ is ended at time $C_i$. Then the observed counting process is given by $N_i(t) = N_i^*(\min(t, C_i)) = \sum_{j=1}^{\infty} I(T_i^{(j)} \leq \min(t, C_i))$ ($i = 1, \ldots, n$). Let $\tilde{Z}_i$ be a $p$-dimensional vector of time-independent covariates for subject $i$. The observed data consist of $\{N_i(t), C_i, \tilde{Z}_i\}_{i=1}^n$.

### 2.2 Models and assumptions

Suppose the whole population consists of $K$ latent classes, and assume that how covariates are related to the intensity function of recurrent events are the same within each latent class while being allowed to vary across different latent classes. Zhao et al. (2022) proposed a latent class multiplicative intensity model, which assumes that $N_i^*(t)$ is a nonstationary Poisson process with the intensity function,

$$\lambda_i(t) = \sum_{k=1}^{K} I(\xi_i = k) \times \lambda_0(t) \times W_i \times \eta_{0,k} \times \exp(\tilde{Z}_i^\top \tilde{\beta}_{0,k}) \tag{1}$$

where $I(\cdot)$ denotes the indicator function, $\xi_i$ denotes the unobserved latent class membership, $\lambda_0(t)$ is an unspecified, continuous, nonnegative baseline intensity function shared by all latent classes, $W_i$ is a positive subject-specific latent variable independent of $(\xi_i, \tilde{Z}_i, C_i)$. The latent variable $W_i$ plays the same role as individual frailty reflecting more or less frequent occurrences of recurrent events. Here $\eta_{0,k}$ is a positive number that captures the class-$k$ scale shift from the baseline intensity function, and $\tilde{\beta}_{0,k}$ is the regression coefficient that represents the class-$k$ covariate effects on the intensity function. For the identifiability of $\lambda_0(t)$ and $\eta_{0,k}$, it is assumed that $E(W_i|\tilde{Z}_i, \xi_i = k) = 1$ for $k = 1, \ldots, K$ and $\int_0^{\nu^*} \lambda_0(u)du = 1$, where $\nu^*$ is a predetermined constant. As commented in Zhao et al. (2022), one may choose $\nu^*$ to be slightly smaller than the upper bound of $C_i$'s support. A different choice of $\nu^*$ only results in a scale shift to $\lambda_0(t)$ and has no influence on the regression coefficients, $\tilde{\beta}_{0,k}$'s.

Zhao et al. (2022) modeled the distribution of the latent class membership $\xi_i$ by a logistic regression model:

$$P(\xi_i = k|\tilde{Z}_i) = p_k(\alpha_0, \tilde{Z}_i) \doteq \frac{\exp(\tilde{Z}_i^\top \alpha_{0,k})}{\sum_{k=1}^{K} \exp(\tilde{Z}_i^\top \alpha_{0,k})}, \quad k = 1, \cdots, K \tag{2}$$

where $\alpha_0 = (\alpha_{0,1}^\top, \ldots, \alpha_{0,K}^\top)^\top$ with $\alpha_{0,1} = \mathbf{0}_{p\times 1}$.

### 2.3 Estimation and inference procedures

#### Estimation procedure

Let $Z_i = (1, \tilde{Z}_i^\top)^\top$ and $\beta_{0,k} = (\log \eta_{0,k}, \tilde{\beta}_{0,k}^\top)^\top$. By the model assumptions stated in Section 2, the following equalities hold:

$$E[I(\xi_i = k)Z_i\{\frac{N_i^*(C_i)}{\mu_0(C_i)} - \exp(Z_i^\top \beta_{0,k})\}] = 0, \;\; k = 1, \ldots, K, \tag{3}$$

$$E[\tilde{Z}_i\{I(\xi_i = k) - \frac{\exp(\tilde{Z}_i^\top \alpha_{0,k})}{\sum_{k=1}^{K} \exp(\tilde{Z}_i^\top \alpha_{0,k})}\}] = 0, \;\; k = 1, \ldots, K, \tag{4}$$

where $\mu_0(t) = \int_0^t \lambda_0(s)ds$, representing the cumulative baseline intensity function.

Equalities in (3) and (4) cannot be directly used to construct estimating equations for $\alpha_0$ and $\beta_{0,k}$'s because $\xi_i$'s are not observable and $\mu_0(\cdot)$ is unknown. Adapting the principle of conditional score (Stefanski and Carroll, 1987) for handling missing covariates, Zhao et al. (2022) proposed to address the unobserved $I(\xi_i = k)$ by substituting it with $\tau_{ik} \doteq E[I(\xi_i = k)|Z_i, C_i, D_i]$, where $D_i \doteq N_i(C_i)$, capturing the number of the observed recurrent events from subject $i$.

First, it follows from the definition that

$$\tau_{ik} = \frac{P(D_i = d_i|\xi_i = k, Z_i, C_i)P(\xi_i = k|Z_i, C_i)}{\sum_{l=1}^{K} P(D_i = d_i|\xi_i = l, Z_i, C_i)P(\xi_i = l|Z_i, C_i)}. \tag{5}$$

As implied by model (2) and the assumption that $C_i \perp (N_i^*(\cdot), \xi_i)|Z_i$,

$$P(\xi_i = k|Z_i, C_i) = p_k(\alpha_0, \tilde{Z}_i) = \frac{\exp(\tilde{Z}_i^\top \alpha_{0,k})}{\sum_{l=1}^{K} \exp(\tilde{Z}_i^\top \alpha_{0,l})} \tag{6}$$

and

$$\begin{aligned} &P(D_i = d_i|\xi_i = k, Z_i, C_i) \\ &= \int_0^\infty \frac{\{\exp(Z_i^\top \beta_{0,k})w \cdot \mu_0(C_i)\}^{d_i}}{d_i!} \exp\{-\exp(Z_i^\top \beta_{0,k})w \cdot \mu_0(C_i)\} \cdot f_W(w)dw, \end{aligned} \tag{7}$$

where $f_W(\cdot)$ denotes the density function of frailty $W$. Combining the results in (5)-(7), $\tau_{ik}$ can be explicitly expressed as a function of $\alpha_0 \doteq (\alpha_{0,1}^\top \ldots, \alpha_{0,K}^\top)^\top, \beta_0 \doteq (\beta_{0,1}^\top, \ldots, \beta_{0,k}^\top)^\top$ and $\mu_0(\cdot)$, which is denoted by $\tau_{ik}(\alpha_0, \beta_0, \mu_0)$.

In addition, Zhao et al. (2022) proposed a Nelson-Aalen type estimator of $\mu_0(\cdot)$, given by

$$\hat{\mu}(t) = \exp\{\hat{H}(t)\} \qquad \text{with} \quad \hat{H}(t) = -\int_t^{\nu^*} \frac{\sum_{i=1}^n dN_i(s)}{\sum_{i=1}^n I(C_i \geq s)N_i(s)}. \tag{8}$$

Replacing $I(\xi_i = k)$ and $\mu_0(\cdot)$ by $\tau_{ik}(\alpha, \beta, \hat{\mu})$ and $\hat{\mu}(\cdot)$ respectively in the empirical counterparts of (3)-(4) then leads to the following estimating equations:

$$n^{1/2} S_{1,n}(\alpha, \beta, \hat{\mu}) = 0, \tag{9}$$

$$n^{1/2} S_{2,n}(\alpha, \beta, \hat{\mu}) = 0, \tag{10}$$

where $S_{j,n}(\alpha, \beta, \hat{\mu}) = (S_{j,n,1}(\alpha, \beta, \hat{\mu})^\top, \cdots, S_{j,n,K}(\alpha, \beta, \hat{\mu})^\top)^\top$ $(j = 1, 2)$ with

$$S_{1,n,k}(\alpha, \beta, \mu) = \frac{1}{n}\sum_{i=1}^{n} \tau_{ik}(\alpha, \beta, \mu)Z_i\{\frac{N_i^*(C_i)}{\hat{\mu}(C_i)} - \exp(Z_i^\top \beta_k)\}, \quad k = 1, \ldots, K; \tag{11}$$

$$S_{2,n,k}(\alpha, \beta, \mu) = \frac{1}{n}\sum_{i=1}^{n} \tilde{Z}_i\{\tau_{ik}(\alpha, \beta, \mu) - \frac{\exp(\tilde{Z}_i^\top \alpha_k)}{\sum_{j=1}^{K} \exp(\tilde{Z}_i^\top \alpha_j)}\}, \quad k = 1, \ldots, K. \tag{12}$$

Solving equations (9) and (10) renders estimators of $\alpha_0$ and $\beta_0$, denoted by $\hat{\alpha}$ and $\hat{\beta}$. Detailed derivations of estimating equations as well as the asymptotic properties of $\hat{\alpha}$ and $\hat{\beta}$ can be found in Zhao et al. (2022) .

### Estimation algorithm

Based on the result in (8) and estimating equations in (9), and (10), we propose the following algorithm to obtain $\hat{\alpha}$ and $\hat{\beta}$, the parameter estimates for models (1) and (2).

Step 1: Compute $\hat{\mu}(\cdot)$ based on formula (8). Let $L_{iter} = 0$ and set initial values of $\alpha_0$ and $\beta_0$ as $\hat{\alpha}^*$ and $\hat{\beta}^*$ respectively. Calculate $\hat{\tau}_{ik}^* \doteq \tau_{ik}(\hat{\alpha}^*, \hat{\beta}^*, \hat{\mu})$ based on equations (5), (6) and (7).

Step2: Repeat in the loop:

(i) Solve estimating equations (9) and (10) with $\tau_{ik}(\alpha, \beta, \hat{\mu})$ fixed as $\hat{\tau}_{ik}^*$. Denote the solutions by $\check{\alpha}$ and $\check{\beta}$.

(ii) $\hat{\alpha}^* := \check{\alpha}$; $\hat{\beta}^* := \check{\beta}$; $\hat{\tau}_{ik}^* := \tau_{ik}(\hat{\alpha}^*, \hat{\beta}^*, \hat{\mu})$. Increase $L_{iter}$ by 1.

(iii) If $\max(\|\frac{\check{\beta}-\hat{\beta}^*}{\hat{\beta}^*}\|_\infty, \|\frac{\check{\alpha}-\hat{\alpha}^*}{\hat{\alpha}^*}\|_\infty)$ is smaller than a pre-specified threshold value or $L_{iter}$ is greater than a pre-specified maximum number of iteration, then exit the loop. Here $\|\cdot\|_\infty$ denotes the $L_\infty$ norm and the fraction between vectors $\alpha$ and $\beta$ is component-wise fraction.

End repeat loop.

Return $\check{\alpha}$ and $\check{\beta}$ as the final estimates for $\alpha_0$,and $\beta_0$ (i.e., $\hat{\alpha}$ and $\hat{\beta}$).

### Algorithm implementation

In the following, we present some details related to the algorithm implementation, such as how to set initial values, $\hat{\alpha}^*$ and $\hat{\beta}^*$, and how to solve the estimating equations involved in the presented algorithm.

**Setting initial values for the estimation algorithm** The function `SLCARE()` in package **SLCARE** allows users to specify the initial values, $\hat{\alpha}^*$ and $\hat{\beta}^*$, by their own choice. Unless users manually specify the initial values, `SLCARE()` implements an automated initializer, which obtains $\hat{\alpha}^*$ and $\hat{\beta}^*$ through the following procedure:

> Step 1: Perform $K$-means clustering (Lloyd, 1982) of the observed covariates and number of recurrent events, i.e., $\{\tilde{Z}_i, N_i(C_i)\}_{i=1}^n$, with R function `kmeans()`, and divide all subjects into $K$ group, where $K$ stands for the number of latent classes. Denote the assigned group membership for subject $i$ by $G_i (\in \{1, \ldots, K\})$.
> Step 2: Obtain $\hat{\alpha}^*$, the initial estimate for $\alpha_0$, as the regression coefficients from fitting the multinomial regression of $\{G_i\}_{i=1}^n$ on covariates $\{\tilde{Z}_i\}_{i=1}^n$.
> Step 3: Obtain $\hat{\beta}^*$, the initial estimate of $\beta_0$, as $(\hat{\beta}_1^{*\top}, \ldots, \hat{\beta}_K^{*\top})$, where $\hat{\beta}_k^{*\top}$ is the regression coefficient estimate from fitting Wang et al. (2001) 's multiplicative intensity model to the subset with $G_i = k$ using `reReg()` from R package **reReg** (Chiou et al., 2023).

**Specifying the distribution of frailty $W$** The `SLCARE()` function allows users to specify the distribution of frailty $W$. While Zhao et al. (2022) 's method allows the frailty distribution to be a general parametric distribution, the implementation by `SLCARE()` confines to settings where $W = 1$ or $W$ follows a distribution that is parameterized as $Gamma(k, k)$. These choices of frailty distributions cover a variety of density forms.

**Solving estimating equations** In Step 2(i) of the presented algorithm, an updated estimate for $\beta_0$, $\check{\beta}$, is obtained from solving equations, $\sum_{i=1}^n \hat{\tau}_{ik} \cdot Z_i \cdot \left\{ \frac{N_i^*(C_i)}{\hat{\mu}(C_i)} - \exp(Z_i^\top \beta_k) \right\} = 0, \quad k = 1, \ldots, K$. As suggested by Zhao et al. (2022), we solve these equations by fitting a 'pseudo' weighted Poisson regression model to an augmented dataset which includes responses, $\{\frac{N_i^*(C_i)}{\hat{\mu}(C_i)}\}_{i=1}^n$, and covariates, $\{Z_i\}_{i=1}^n$, along with weights $\hat{\tau}_{ik}$. **SLCARE** performs such Poisson regression with R function `glm.fit()` from R package **stats**.

Meanwhile, in Step 2(i) of the presented algorithm, an updated estimate for $\alpha_0$, $\check{\alpha}$, is obtained by solving equations, $\frac{1}{\sqrt{n}} \sum_{i=1}^n \tilde{Z}_i \left\{ \hat{\tau}_{ik} - \frac{\exp(\tilde{Z}_i^\top \alpha_k)}{\sum_{j=1}^K \exp(\tilde{Z}_i^\top \alpha_j)} \right\} = 0, \quad k = 2, \cdots, K$. To solve these equations, we use an alternative and yet equivalent approach that fits weighted multinomial regression model on an augmented dataset, which consists of $nK$ response-covariate pairs, $\{(1, \tilde{Z}_i), \ldots, (K, \tilde{Z}_i)\}_{i=1}^n$ with weight $\hat{\tau}_{ik}$ assigned to the pair $(k, \tilde{Z}_i)$ $(k = 1, \ldots, K)$. **SLCARE** implements the weighted multinomial regression with function `multinom()` from R package **nnet** (Venables and Ripley, 2002).

**Determination of the number of latent classes** Zhao et al. (2022) proposed to determine $K$, the number of latent classes, based on a relative entropy measure (Ramaswamy et al., 1993). That is, upon fitting models (1) and (2) that assume $M$ latent classes, the corresponding relative entropy measure is defined as

$$Entropy(M) = 1 - \frac{\sum_{i=1}^n \sum_{k=1}^M -\hat{\tau}_{ik} \log(\hat{\tau}_{ik})}{n \log(M)} \tag{13}$$

where $\hat{\tau}_{ik} = \tau_{ik}(\hat{\alpha}, \hat{\beta}, \hat{\mu})$ $(k = 1, \ldots, M)$. By definition, $Entropy(M)$ is bounded between 0 and 1 with a larger value indicating a better fit to the data (Celeux and Soromenho, 1996). Following the rationale, one may choose $K$ as the maximizer of $Entropy(\cdot)$ over a sequence of candidate values for $K$. Given a pre-specified $K$, the value of $Entropy(K)$ can be generated by S3 method `print(x, type = "Entropy")`.

**Model checking** Zhao et al. (2022) proposed a graphical method for checking the overall fit of models (1) and (2). The key idea is to compare the observed recurrent events $D_i \doteq N_i(C_i)$ versus the expected number of recurrent events under models (1)-(2). Specifically, models (1) and (2) imply $E\{N_i(C_i)|Z_i\} = E\{N_i^*(C_i)|Z_i\} = \sum_{k=1}^K \tau_{ik} \cdot \mu_0(C_i) \exp(Z_i^\top \beta_{0,k})$. Then the expected number of recurrent events under the assumed models can be approximated by

$$\hat{D}_i \doteq \sum_{i=1}^K \hat{\tau}_{ik} \cdot \hat{\mu}(C_i) \cdot \exp(Z_i^\top \hat{\beta}_k). \tag{14}$$

Therefore, in the scatter plot of $\hat{D}_i$ versus $D_i$, a major departure from the identity line $D_i = \hat{D}_i$ may suggest a lack-of-fit of the assumed models. Model checking plot can be generated by S3 method `plot(x, type = "ModelChecking")`.

**Class-specific mean function of recurrent events** To help illustrate heterogeneity in recurrent event occurrence across different latent classes, **SLCARE** computes crude estimates for the class-specific mean functions of recurrent events. Specifically, the crude estimate for the class-specific mean function of recurrent event is computed as

$$\frac{\sum_{i=1}^{n} I(\hat{\xi}_i = k) \sum_{j=1}^{K} \hat{\tau}_{ij} \cdot \hat{\mu}(t) \exp(Z_i^\top \hat{\beta}_j)}{\sum_{i=1}^{n} I(\hat{\xi}_i = k)}, \quad k = 1, \ldots, K, \tag{15}$$

where $\hat{\xi}_i = \arg\max_{1 \le k \le K} \hat{\tau}_{ik}$. A plot of the estimated mean functions stratified by latent groups can be generated by S3 method `plot(x, type = "EstMeans")`.

## 3 Package structure

The main function of R package **SLCARE** is `SLCARE()`. The dataset imported to function `SLCARE()` should take the long format, where each row corresponds to one time point of a subject at which a recurrent event is observed or censored. With a dataset coded in the wide format, where each row contains the timing information of all recurrent events observed or censored within one subject, users may convert such a dataset into the required long format by using function `pivot_longer()` from package **tidyr** (Wickham et al., 2023) or function `melt()` from package **reshape2** (Wickham, 2007). In addition to allowing users to specify initial parameter estimates manually, `SLCARE()` offers a default initializer that implements the informative selection of initial values described in Section **Estimation procedure**. The built-in initializer performs $K$-means clustering of the observed data using function `kmeans()` from package **stats** and fitting multiplicative intensity models using `reReg()` from package **reReg**. When solving the estimating equations involved in the iterative estimation algorithm, `SLCARE()` performs weighted Poisson regression using `glm.fit()` from package **stats** and weighted multinomial regression using `multinom()` from package **nnet**. After running `SLCARE()`, model fitting results can be easily extracted from the output with S3 methods `summary()`, `print()`, `predict()`, and `plot()`. Graphical results generated by `SLCARE()` are presented via **ggplot2** environment and are fully customizable.

Table 1 lists the main function `SLCARE()`, corresponding S3 methods and other functions called by this function, along with brief descriptions of their purposes.

**Table 1:** An overview of the main function and imported functions in SLCARE

| FUNCTION | PURPOSE |
|---|---|
| SLCARE() | Conduct LCA with recurrent events data based on semiparametric multiplicative intensity modeling. |
| summary() | Generic function; used to summarize estimates for model parameters along with the associated variance estimates and $p$ values. |
| print() | Generic function; used to initial estimates for the estimation algorithm convergence criterion, latent class membership probability and predicted number of recurrent events. |
| predict() | Generic function; used to predict the posterior number of recurrent events. |
| plot() | Generic function; used to generate cumulative baseline intensity function estimated mean function, model checking plot. |
| reReg() | Imported from package reReg; used to find an initial estimate for $\beta_0$. |
| multinom() | Imported from package nnet; used to find an initial estimate for $\alpha_0$; used to solve the estimating equation to update the estimate for $\alpha_0$. |
| kmeans() | Imported from package stats; used to find initial estimates for $\alpha_0$ and $\beta_0$. |
| glm.fit() | Imported from package stats; used to solve the estimating equation to update the estimate for $\beta_0$. |

# 4 Illustrations

The package **SLCARE** is designed to conduct LCA of recurrent events data based on semiparametric multiplicative intensity modeling (Zhao et al., 2022). The main function `SLCARE()` has function arguments that enable flexible model specification and method implementation. In the below, we illustrate the use of `SLCARE()` through simulated datasets.

## 4.1 Input dataset

An input dataset for `SLCARE()` is a data frame containing time to recurrent events or censoring along with time-independent covariates of interest. The input dataset should take the long format, where each row contains the information in one time interval in which a recurrent event is observed or censored. As such, multiple rows in the dataset may correspond to one subject. As mentioned in Section **Package structure**, a dataset in the wide format data can be readily converted into the long format required by `SLCARE()` with the use of function `pivot_longer()` from package **tidyr** or function `melt()` from package **reshape2**.

The package **SLCARE** has a build-in dataset, `SimData`, which is a simulated dataset perturbed from a real dataset. `SimData` contains 48 subjects. For each subject, `SimData` contains values for the following six variables: `id`, which is the unique identifier for each subject; `start`, which records the starting time in minutes of the counting process interval; `stop`, which records the ending time in minutes of the counting process interval. If the subject entered the study at time zero, this column represents the time in minutes from the baseline visit to the occurrence of event or censoring; `event`, which indicates whether a recurrent event is observed (`event = 1`) or not (`event = 0`) at the end of the counting process interval; `x1`, which is a binary covariate; `x2`, which is a continuous covariate. Below we display how the data from one example subject are recorded in `SimData`:

```
#> # A tibble: 3 x 6
#> # Groups:   id [1]
#>   id    start  stop event    x1    x2
#>   <chr> <dbl> <int> <int> <int> <dbl>
#> 1 G052      0  1950     1     1 0.444
#> 2 G052   1950  2580     1     1 0.444
#> 3 G052   2580  7085     0     1 0.444
```

As shown above, subject `G052` experienced two current events at time 1950 and time 2580 before the censoring time 7085. The time-independent covariates, `x1` and `x2`, remain unchanged across different time points within the same subject.

## 4.2 Command line and function arguments

The LCA of recurrent event data outlined in Section **Methodological background** can be carried out with a single command line `SLCARE()`. `SLCARE()` provides flexible function arguments, which are shown below:

```
#> function (formula = "x1 + x2", alpha = NULL, beta = NULL, data = data,
#>     id_col = "id", start_col = "start", stop_col = "stop", event_col = "event",
#>     K = NULL, gamma = 0, max_epochs = 500, conv_threshold = 0.01,
#>     boot = NULL)
#> NULL
```

At minimum, `SLCARE()` requires the following three arguments:

`data`: a dataframe with the format similar to `SimData`.

`K`: pre-determined number of latent classes.

`formula`: a string containing the covariates of interest in `data`.

The optional arguments of `SLCARE()` are:

`alpha`: initial estimate for $\alpha_0$ in the estimation procedure. The default is NULL, which represents the initial estimate for $\alpha_0$ resulted from the automated initializer described in Section **Estimation and inference procedures**.

`beta`: initial estimate for $\beta_0$ in the estimation procedure. The default is NULL, which represents the initial estimate for $\alpha_0$ resulted from the automated initializer described in Section **Estimation and inference procedures**.

`id_col`: name of the column that includes subject identifiers.

`start_col`: name of the column that records the start time of each at-risk time interval.

`stop_col`: name of the column that records the start time of each at-risk time interval.

`event_col`: name of the column that indicates whether a recurrent event is observed or not (i.e, 1=observed; 0=otherwise).

`gamma`: parameter that indicates the distribution of frailty $W$. The default is 0 which indicates model (1) holds without the subject-specific frailty (i.e., $W = 1$); `gamma` $= k$ indicates that $W$ follows the $Gamma(k, k)$ distribution.

`max_epochs`: maximum number of iterations for the estimation algorithm.

`conv_threshold`: convergence threshold for the estimation algorithm.

`boot`: number of bootstrap replicates used to obtain the standard error estimation. The default is NULL which indicates bootstrap is not conducted.

### 4.3 Output and illustration with a sample dataset

The following code is used to perform LCA of the recurrent event dataset `SimData` with two latent classes (i.e., $K = 2$), 20 bootstrap replicates, and covariates `x1` and `x2`.

```
set.seed(0)
model1 <- SLCARE(formula = "x1 + x2", data = SimData,
                 id_col = "id", start_col = "start", stop_col = "stop",
                 event_col = "event",  K=2, boot = 20)
```

The default output/print of `SLCARE()` includes parameter estimates and the associated variance estimates and $p$-values, along with the model's relative entropy measure. The same results can also be obtained using the generic method `summary(model1, digits = 3)`, which allows for manually setting the minimum number of significant digits.

```
model1
```

```
#> Call:
#> SLCARE(formula = "x1 + x2", data = SimData, id_col = "id", start_col = "start",
#>     stop_col = "stop", event_col = "event", K = 2, boot = 20)
#>
#> Coefficients for Beta:
#>        (intercept)      x1     x2
#> class1        3.21 -0.2350 -4.933
#> class2        2.41 -0.0413  0.309
#>
#> Std. Errors for Beta:
#>        (intercept)    x1    x2
#> class1       0.216 0.278 0.813
#> class2       0.142 0.105 0.352
#>
#> P. Values for Beta:
#>        (intercept)    x1       x2
#> class1    4.81e-50 0.398 1.28e-09
#> class2    1.85e-64 0.693 3.80e-01
#>
#> Coefficients for Alpha:
#>           x1   x2
#> class1  0.00 0.00
#> class2 -0.71 4.81
#>
#> Std. Errors for Alpha:
#>           x1   x2
#> class1 0.000 0.00
#> class2 0.928 1.88
#>
```

```
#> P. Values for Alpha:
#>           x1     x2
#> class1   NaN    NaN
#> class2 0.444 0.0106
#>
#> Relative Entropy: 0.539
```

Recall that in model (2), $\alpha_{0,1} = \mathbf{0}$, indicating that `class1` is used as the reference group for evaluating the odds ratio of being in another latent class. The numbers in the row corresponding to `class2` represent covariate effects on the log odds ratio of belonging to `class2` versus `class1`.

These results are also can be extracted separately with the generic method function `print(model1, type = "Est")`, `print(model1, type = "SE")`, `print(model1, type = "PValue")`, and `print(model1, type = "Entropy")`.

### Initial estimates for the estimation algorithm

Unless manually specifying the arguments `alpha` and `beta`, `SLCARE()` implements the automated initializer described in Section **Estimation and inference procedures**. The resulting initial estimates for $\beta_0$ and $\alpha_0$ can be found through the following generic method function `print()`:

```
print(model1, type = "Init")
```

```
#> $beta
#>        (intercept)          x1          x2
#> class1     2.01630  0.09073593 -0.45944133
#> class2     2.56397 -0.24415627  0.06202144
#>
#> $alpha
#>               x1       x2
#> class1 0.0000000 0.000000
#> class2 0.5386871 2.985595
```

### Convergence criterion

`SLCARE()` provides arguments to control the termination of the iterative estimation procedure. By default, `max\_epochs=500` and `conv_threshold=0.01`. This means that the iterations would stop when the number of iterations reaches 500 or the magnitude of the change in parameter estimates is below 0.01. In addition, users are allowed to check the change in parameter estimates in the last iteration using the following code:

```
print(model1, type = "ConvergeLoss")
```

```
#> [1] 0.00909132
```

If the output is greater than the pre-determined convergence threshold, users may consider increasing the maximum number of iterations and/or setting a larger convergence threshold.

### Latent class membership probability

`SLCARE()` generates output on the estimated probability that a subject belong to different latent classes. The following are the example code and output in the case where one is interested in the class membership probabilities of the $8^{th}$ to $10^{th}$ subjects in `SimData`.

```
print(model1, type = "ClassProb")[8:10,]
```

```
#> # A tibble: 3 x 3
#> # Groups:   ID [3]
#>   ID       class1  class2
#>   <chr>     <dbl>   <dbl>
#> 1 G052   9.94e- 1 0.00610
#> 2 UOM043 5.33e- 1 0.467
#> 3 G064   4.20e-22 1
```

### Predicted number of recurrent events

`SLCARE()` computes the predicted number of recurrent events for each subject according to equation (14). The code to extract such a result for the $8^{th}$ to $10^{th}$ subjects in `SimData` is shown below, along with the corresponding output:

```
predict(model1)[8:10,]
```

```
#>        ID PosteriorPrediction
#> 8    G052            2.231237
#> 9  UOM043            2.892228
#> 10   G064           13.333784
```

Note that `SLCARE()` returns the estimated numbers of recurrent events as float rather than integer. Users may use the argument `integer = TRUE` in the genetic function `preduct()` to round those numbers to integers.

```
predict(model1, integer = TRUE)[8:10,]
```

```
#>        ID PosteriorPrediction
#> 8    G052                  2
#> 9  UOM043                  2
#> 10   G064                 13
```

### Model checking plot

`SLCARE()` generates a model checking plot following the procedure described in Section **Methodological background** via **ggplot2** environment. The following shows the `S3` method for plotting a model checking plot:

```
plot(model1, type = "ModelChecking")
```

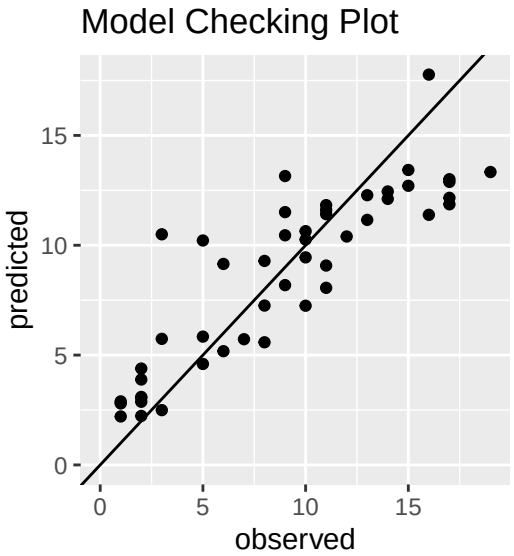


**Figure 1:** Predicted numbers of events versus the observe numbers of events.

### Estimated cumulative baseline intensity function

By equation (8), `SLCARE()` calculates $\hat{\mu}(t)$, which is an estimates for $\mu_0(t)$, cumulative baseline intensity function. A plot of $\hat{\mu}(t)$ versus $t$ can be generated with the following code:

```
plot(model1, type = "mu0")
```

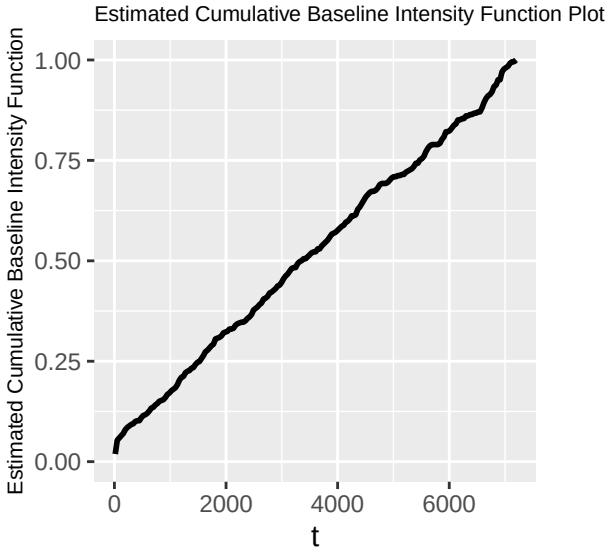


**Figure 2:** Estimated cumulative baseline intensity function.

Evaluation of $\hat{\mu}(t)$ at specific time points, for example, $t = 100, 1000$ and $5000$, can be obtained by the following command:

```
model1$est_mu0(c(100, 1000, 5000))

#> [1] 0.06086907 0.17089670 0.70936436
```

### Estimated mean function plot

Estimated mean function plots are useful for describing and comparing the expected number of recurrent events across different latent groups over time. `SLCARE()` computes the estimated mean function over time according to equation (15), which can be plotted with the following code:

```
plot(model1, type = "EstMeans")
```

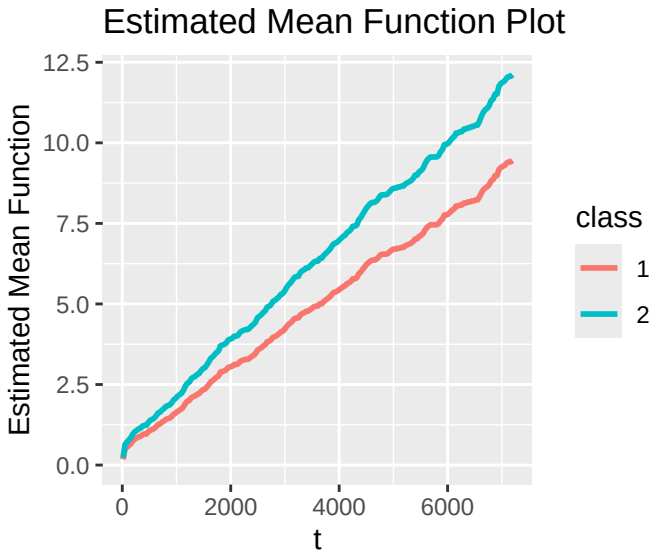


**Figure 3:** Estimated mean function for two latent classes.

## 5 A real application

We apply **SLCARE** to a dataset from a randomized phase III clinical trial FFCD 2000-05 conducted between 2002 and 2007 in patients diagnosed with advanced colorectal cancer (Ducreux et al., 2011).

The dataset is publicly available in **frailtypack** package (Rondeau et al., 2012) and is also included in the **SLCARE** package. One recurrent event of interest is the appearance of new colorectal lesions. The colorectal lesion is the region in a colon that has suffered damage through colorectal cancer. The appearance of new colorectal lesions indicates colorectal cancer progression. The main interest of our analysis is to explore the heterogeneity in the recurrence patterns of colorectal lesions.

To address this interest, we extract the dataset `colorectalSLCARE` from the `colorectal` dataset. The dataset 'colorectalSLCARE is arranged in the long format, consisting of 150 patients and 289 records.

Each record stores the value of the six variables described as follows. Variable `id` is the unique patient identifier. Variable `time0` is the start time (in years) of the recurrent event interval. Variable `time1` records the interval end time (in years) of the detection of new colorectal lesions or terminal event (death or right-censoring). Variable `new.lesions = 1` indicates that the appearance of a new colorectal lesion is observed and `new.lesions = 0` indicates that a terminal event is observed. The other two variables represent covariates of interest: (1) treatment, which is coded as `treatment = 1` if the patient received a combination chemotherapy and as `treatment = 0` if the patient received sequential chemotherapy; (2) previous resection indicator, which is coded as `prev.resection = 1` if the patient had a previous resection and as `prev.resection = 0` otherwise.

The construction and the structure of the `colorectalSLCARE` dataset are presented below.

```
data("colorectal", package = "SLCARE")
colorectalSLCARE <- colorectal |>
  dplyr::select(id, time0, time1, new.lesions, treatment, prev.resection) |>
  dplyr::mutate(id = paste0("patient", id), treatment = as.numeric(treatment) -1,
                prev.resection = as.numeric(prev.resection) -1)
str(colorectalSLCARE, vec.len = 3)

#> 'data.frame':    289 obs. of  6 variables:
#>  $ id            : chr  "patient1" "patient2" "patient3" ...
#>  $ time0         : num  0 0 0 0.525 ...
#>  $ time1         : num  0.71 1.282 0.525 0.921 ...
#>  $ new.lesions   : int  0 0 1 1 0 1 0 1 ...
#>  $ treatment     : num  0 1 0 0 0 1 1 0 ...
#>  $ prev.resection: num  0 0 0 0 0 1 1 0 ...
```

We fit models (1) and (2) to the dataset `colorectal_SLCARE` with $T_i^{(j)}$ corresponding to time to the $j$-th detection of new colorectal lesions. We consider three candidate distributions for $W$, the frailty term in model (1), which are $W = 1$ and $Gamma(r, r)$, $r = 1, 3$. For each candidate $f_W(\cdot)$, we calculate the relative entropy measure *Entropy* as described in Section **Estimation and inference procedures** using the command illustrated in Section **Illustrations**. The results are presented in Table 2.

**Table 2:** Real data example: Relative entropy calculated with different choices of $K$ and $f_W(w)$.

| | K = 2 | K = 3 |
|---|---|---|
| Gamma(1,1) | 0.785 | 0.788 |
| Gamma(3,3) | 0.802 | 0.793 |
| W = 1 | 0.462 | 0.459 |

As shown in Table 2, the maximum relative entropy is achieved with the combination of $K = 2$ and $Gamma(3, 3)$. Therefore, we set $K = 2$ and select $f_W(\cdot)$ as the density of $Gamma(3, 3)$ for the rest of the analysis.

```
set.seed(66)
finalmodel <- SLCARE(formula = "treatment + prev.resection", data = colorectalSLCARE,
      id_col = "id", start_col = "time0", stop_col = "time1", event_col = "new.lesions",
       K = 2, gamma = 3, boot = 200)
summary(finalmodel, digits = 3)

#> Call:
#> SLCARE(formula = "treatment + prev.resection", data = colorectalSLCARE,
#>     id_col = "id", start_col = "time0", stop_col = "time1", event_col = "new.lesions",
#>     K = 2, gamma = 3, boot = 200)
```

```
#>
#> Coefficients for Beta:
#>        (intercept) treatment prev.resection
#> class1       1.696    -0.415         -0.493
#> class2       0.811     0.691          0.494
#>
#> Std. Errors for Beta:
#>        (intercept) treatment prev.resection
#> class1        0.34     0.281          0.277
#> class2        0.48     0.887          0.703
#>
#> P. Values for Beta:
#>        (intercept) treatment prev.resection
#> class1    6.26e-07     0.140         0.0751
#> class2    9.11e-02     0.436         0.4823
#>
#> Coefficients for Alpha:
#>        treatment prev.resection
#> class1       0.0            0.0
#> class2     -12.3          -11.7
#>
#> Std. Errors for Alpha:
#>        treatment prev.resection
#> class1      0.00           0.00
#> class2      3.96           1.26
#>
#> P. Values for Alpha:
#>        treatment prev.resection
#> class1       NaN            NaN
#> class2   0.00196       1.51e-20
#>
#> Relative Entropy: 0.802
```

Based on the results from fitting models (1) and (2), we classify patients into two groups according to the modal class assignment rule, i.e., $\hat{\xi}_i = \arg\max_{1\le k\le K} \hat{\tau}_{ik}$. Table 3 (generated by R package **table1** (Rich, 2023)) summarizes the characteristics of the resulting two groups, Class 1 of size 127 and Class 2 of size 23. Comparing the total number of the observed colorectal lesions, captured by $N_i(C_i)$, we note that patients belong to Class 1 tend to experience more colorectal lesions as compared to those belong to Class 2. There is also notable separation in covariates `treatment` and `prev.resection`. That is, over 50% patients in Class 1 received combination chemotherapy in contrast to 0% patients in Class 2 on combination chemotherapy, and over 70% patients in Class 1 has previous resection, while all patients in Class 2 did not have resection in the past. These observations are consistent with the estimation results for $\alpha_0$ (see model summary), which suggest that patients receiving combination chemotherapy or with previous resection are much less likely to be classified to Class 2. These observations entail a plausible characterization of the two latent classes. That is, Class 1 consists of patients with more aggressive progression that incurs more intensive past or ongoing treatment, while Class 2 is featured by relatively mild disease that requires less complicated treatment regimens.

We next examine the covariate effects within each latent class. The coefficient estimates presented in model summary indicate a reasonable direction of how `treatment` and `prev.resection` influence the intensity of lesion recurrence. That is, combination chemotherapy or previous resection may help reduce the risk of lesion recurrence. However, the estimated effects which conform to our intuition are associated with p values that do not reach statistical significance.

**Table 3:** Real data example: Characteristics of the two latent classes.

| | Class 1 | Class 2 | P-value |
|---|---|---|---|
| | (N=127) | (N=23) | |
| **observed.events** | | | |
| Mean (SD) | 1.02 (0.988) | 0.391 (0.583) | <0.001 |
| Median [Min, Max] | 1.00 [0, 4.00] | 0 [0, 2.00] | |
| **treatment** | | | |
| sequential chemotherapy | 54 (42.5%) | 23 (100%) | <0.001 |
| combination chemotherapy | 73 (57.5%) | 0 (0%) | |
| **prev.resection** | | | |
| no | 37 (29.1%) | 23 (100%) | <0.001 |
| yes | 90 (70.9%) | 0 (0%) | |

In Figure 4, we plot the average estimated mean function of experiencing new colorectal lesions for the two latent classes. Figure 4 shows that Class 1 is associated with a lower frequency of experiencing new colorectal lesions than Class 2. Such a distinction may be contributed by the underlying difference in disease severity between the two latent classes, which is also manifested by the different distributions of receiving combination chemotherapy and having previous resection between these two classes demonstrated in Table 3.

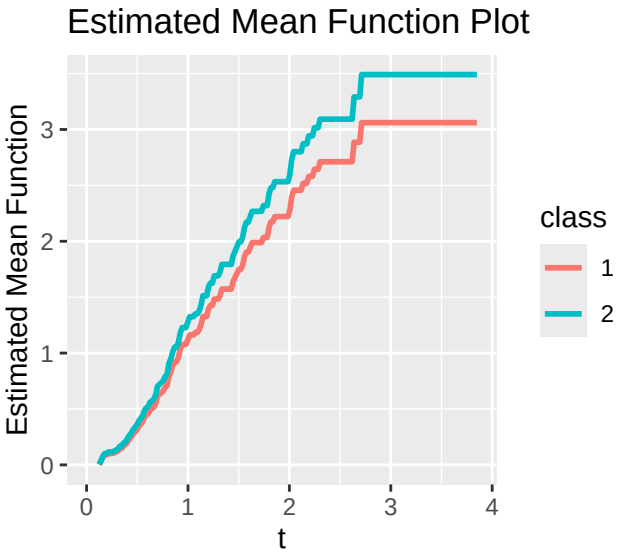


**Figure 4:** Real data example: Estimated mean function of experiencing new colorectal lesions for two classes.

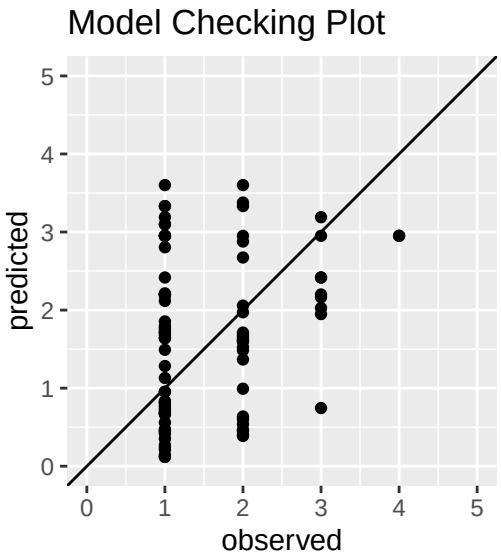


**Figure 5:** Real data example: Predicted numbers of new colorectal lesions versus the observed numbers of new colorectal lesions.

We further check the overall fit of the latent class models adopted by `SLCARE()` to the `colorectalSLCARE` dataset using the model checking plot described in Section **Methodological background**. As shown in Figure 5, the dots representing the predicted and the observed numbers of new colorectal lesions are generally balanced around the 45-degree line when there are two observed colorectal lesions. However, over-prediction and under-prediction are noted when the observed number of of colorectal lesions is one or three respectively. These observations may suggest a moderate lack-of-fit of the fitted model to this colorectical dataset.

## 6 Discussion

The R package **SLCARE** provides an easy-to-use software to conduct latent class analysis of recurrent events based on a flexible semiparametric multiplicative intensity modeling proposed by Zhao et al. (2022). A practical automated initializer is embedded in **SLCARE** to help users set initial estimates in an informative way. **SLCARE** provides a single command line function to implement full analysis with optional arguments for model customization. **SLCARE** also provides visualization tools for result summary and model evaluation with **ggplot2** plotting environment.

The R package **SLCARE** is maintained in both Comprehensive R Archive Network (CRAN) at https://CRAN.R-project.org/package=SLCARE and Github at https://github.com/qyxxx/SLCARE. There are several interesting directions to further improve this package. For example, computational speed may be improved with more efficient coding. A new option may be added to perform automated selection of $K$ and frailty distribution. Such updates of package **SLCARE** will be made available at CRAN and Github.

## References

P. K. Andersen and R. D. Gill. Cox's regression model for counting processes: a large sample study. *The annals of statistics*, 1982. URL https://www.jstor.org/stable/2240714. [p134]

G. Celeux and G. Soromenho. An entropy criterion for assessing the number of clusters in a mixture model. *Journal of Classification*, 1996. URL https://doi.org/10.1007/BF01246098. [p137]

S. H. Chiou, G. Xu, J. Yan, and C.-Y. Huang. Regression modeling for recurrent events possibly with an informative terminal event using R package reReg. *Journal of Statistical Software*, 2023. URL https://doi.org/10.18637/jss.v105.i05. [p134, 137]

D. Y. Clement and R. L. Strawderman. Conditional GEE for recurrent event gap times. *Biostatistics*, 2009. URL https://doi.org/10.1093/biostatistics/kxp004. [p134]

M. Ducreux, D. Malka, J. Mendiboure, P.-L. Etienne, P. Texereau, D. Auby, P. Rougier, M. Gasmi, M. Castaing, M. Abbas, et al. Sequential versus combination chemotherapy for the treatment of

advanced colorectal cancer (ffcd 2000–05): an open-label, randomised, phase 3 trial. *The lancet oncology*, 2011. URL https://doi.org/10.1016/S1470-2045(11)70199-1. [p143]

J. Fine, J. Yan, and M. Kosorok. Temporal process regression. *Biometrika*, 2004. URL https://doi.org/10.1093/biomet/91.3.683. [p134]

B. Grün and F. Leisch. FlexMix version 2: Finite mixtures with concomitant variables and varying and constant parameters. *Journal of Statistical Software*, 2008. URL https://doi.org/10.18637/jss.v028.i04. [p134]

F. E. Harrell Jr. *rms: Regression Modeling Strategies*, 2023. URL https://CRAN.R-project.org/package=rms. R package version 6.6-0. [p134]

S. Lloyd. Least squares quantization in PCM. *IEEE transactions on information theory*, 1982. URL https://doi.org/10.1109/TIT.1982.1056489. [p137]

C. Proust-Lima, V. Philipps, and B. Liquet. Estimation of extended mixed models using latent classes and latent processes: The R package lcmm. *Journal of Statistical Software*, 2017. URL https://doi.org/10.18637/jss.v078.i02. [p134]

V. Ramaswamy, W. S. DeSarbo, D. J. Reibstein, and W. T. Robinson. An empirical pooling approach for estimating marketing mix elasticities with PIMS data. *Marketing science*, 1993. URL https://doi.org/10.1287/mksc.12.1.103. [p137]

B. Rich. *table1: Tables of Descriptive Statistics in HTML*, 2023. URL https://CRAN.R-project.org/package=table1. R package version 1.4.3. [p145]

V. Rondeau, Y. Marzroui, and J. R. Gonzalez. frailtypack: an R package for the analysis of correlated survival data with frailty models using penalized likelihood estimation or parametrical estimation. *Journal of Statistical Software*, 2012. URL https://doi.org/10.18637/jss.v047.i04. [p144]

L. A. Stefanski and R. J. Carroll. Conditional scores and optimal scores for generalized linear measurement-error models. *Biometrika*, 1987. URL https://doi.org/10.1093/biomet/74.4.703. [p135]

T. M. Therneau. *A Package for Survival Analysis in R*, 2023. URL https://CRAN.R-project.org/package=survival. R package version 3.5-3. [p134]

W. N. Venables and B. D. Ripley. *Modern Applied Statistics with S*. 2002. URL https://www.stats.ox.ac.uk/pub/MASS4/. [p137]

M.-C. Wang, J. Qin, and C.-T. Chiang. Analyzing recurrent event data with informative censoring. *Journal of the American Statistical Association*, 2001. URL https://doi.org/10.1198/016214501753209031. [p134, 137]

H. Wickham. Reshaping data with the reshape package. *Journal of Statistical Software*, 2007. URL http://www.jstatsoft.org/v21/i12/. [p138]

H. Wickham. *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York, 2016. URL https://ggplot2.tidyverse.org. [p134]

H. Wickham, D. Vaughan, and M. Girlich. *tidyr: Tidy Messy Data*, 2023. URL https://CRAN.R-project.org/package=tidyr. R package version 1.3.0. [p138]

J. Yan and J. Fine. Estimating equations for association structures. *Statistics in medicine*, 2004. URL https://doi.org/10.1002/sim.1650. [p134]

W. Zhao, L. Peng, and J. Hanfelt. Semiparametric latent class analysis of recurrent event data. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, 2022. URL https://doi.org/10.1111/rssb.12499. [p134, 135, 136, 137, 139, 147]

*Qi Yu*
*Emory University*
*Department of Biostatistics and Bioinformatics*
*1518 Clifton Rd. NE, Atlanta, GA 30322, USA*
*ORCiD: 0009-0007-7585-1183*
qi.yu2@emory.edu

*Limin Peng*
*Emory University*
*Department of Biostatistics and Bioinformatics*
*1518 Clifton Rd. NE, Atlanta, GA 30322, USA*
*ORCiD: 0000-0002-8042-3112*
lpeng@emory.edu

# PubChemR: An R Package for Accessing Chemical Data from PubChem

*by Selcuk Korkmaz, Bilge Eren Yamasan, and Dincer Goksuluk*

**Abstract** Chemical data is a cornerstone in the fields of chemistry, pharmacology, bioinformatics, and environmental science. The PubChemR package provides a comprehensive R interface to the PubChem database, which is one of the largest and most complete repositories of chemical data. This package simplifies the process of querying and retrieving chemical information, including compound structures, properties, biological activities, and more, directly from within R. By leveraging PubChemR, users can programmatically access a wealth of chemical data, which is essential for research and analysis in the chemical sciences. The package supports various functionalities such as searching by chemical identifiers, downloading chemical structures, and retrieving bioassay results, among others. PubChemR is designed to be user-friendly, providing a intuitive experience for R users ranging from academic researchers to practitioners across various scientific disciplines. This paper presents the capabilities of PubChemR, demonstrates its use through practical examples, and discusses its potential impact on chemical data analysis.

## 1 Introduction

Chemical data serves as a foundational element in a wide spectrum of scientific research fields, from pharmacology and medicinal chemistry to materials science and environmental studies. The ability to access, query, and manipulate chemical information efficiently is essential for researchers and practitioners who rely on data-driven methodologies to advance their work. PubChem, hosted by the National Center for Biotechnology Information (NCBI), stands as one of the largest publicly available repositories of chemical data, offering free access to an abundance of information on chemical substances, compounds, and biological activities (Wang et al., 2009; Chen et al., 2009; Li et al., 2010; Wang et al., 2012; Kim et al., 2016).

The **PubChemR** package for R provides a comprehensive interface to the PubChem database, allowing users to programmatically retrieve and utilize chemical data within the R environment (Korkmaz et al., 2024). This integration facilitates a more streamlined workflow for scientists who use R for statistical analysis, data visualization, and computational modeling. By utilizing the **PubChemR** package, users can perform a variety of tasks such as searching for chemical substances, fetching compound properties, and obtaining assay data for bioactivity analysis.

Among the extensive range of R packages used in scientific research, **webchem** (Szöcs et al., 2020) and **ChemmineR** (Cao et al., 2008) have been important for accessing chemical databases. **ChemmineR** provides tools for cheminformatics in R, enabling detailed handling and analysis of chemical data. **webchem** supports access to multiple chemical databases. Additionally, other notable packages like **rcdk**, **ChemmineOB**, **BridgeDbR**, **RMassBank**, and **rgoslin** offer robust functionalities for chemical data manipulation and analysis. The **rcdk** package interfaces with the Chemistry Development Kit (CDK), providing molecular structure parsing and descriptor calculation (Guha, 2007). **ChemmineOB** interfaces with OpenBabel for chemical format conversions and molecular property calculations (Horan and Girke, 2024). **BridgeDbR** facilitates identifier mapping across biological databases, enhancing data integration (Leemans et al., 2024). **RMassBank** supports the creation and handling of mass spectrometry databases, crucial for compound identification (Stravs et al., 2013). **rgoslin** ensures accurate lipid nomenclature in lipidomics studies (Kopczynski et al., 2020).

However, when it comes to direct interaction with the PubChem database, these packages have limitations, often requiring users to work through complex API documentation or use additional tools. To address these issues, **PubChemR**, designed specifically for the PubChem database, complements the functionalities of these packages. It simplifies accessing chemical data from PubChem, using functions that make API interactions more straightforward. Users can retrieve data easily by calling these functions with the right parameters. Unlike **webchem**, which works with various databases, and **ChemmineR**, which covers a wide range of cheminformatics tasks, **PubChemR** is focused solely on PubChem, allowing for more efficient and targeted data interactions.

Here, we introduce **PubChemR**, detailing its functionality, design principles, and potential use cases. We will demonstrate how **PubChemR** can be utilized to enhance research workflows and provide examples of its application in real-world scenarios. By the end of this paper, readers will be equipped with the knowledge to integrate **PubChemR** into their data analysis toolkit, unlocking the potential to drive forward chemical and biological research with the power of R.

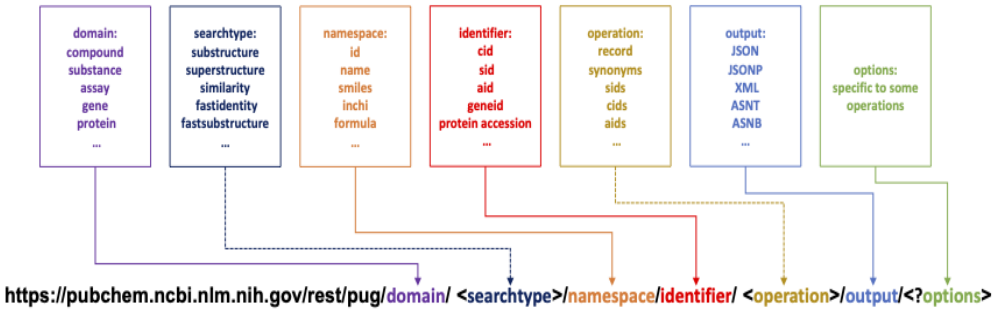


**Figure 1:** Interfacing the PubChem's PUG-REST database with the PubChemR package through URL syntax: utilizing queries as function arguments

## 2 Design

The design of PubChemR is driven by the need for a seamless and intuitive interface for R users to access the vast chemical data available in the PubChem database. Our design principles focus on simplicity, efficiency, and robustness:

- **Simplicity:** The package is designed to minimize the complexity of interacting with the PubChem API. Functions are named and structured to be self-explanatory, allowing users to intuitively understand their purposes.
- **Efficiency:** Considering the extensive size of the PubChem database, we optimized PubChemR for speed and minimal resource consumption. Our design includes efficient handling of API calls and data processing.
- **Robustness:** The package is designed to handle a wide range of user queries, from simple compound searches to complex data extractions. Exception handling and error reporting are integral, ensuring users are informed of issues in their queries or data processing.

## 3 Implementation

The implementation of PubChemR involved several key steps:

- **API Integration:** We integrated the PubChem API using R's HTTP client capabilities. This involved mapping the PubChem's RESTful services into R functions.
- **Data Processing:** The raw data from PubChem API calls are processed and transformed into user-friendly R data structures such as data frames and lists.
- **Function Development:** Each function in PubChemR corresponds to a specific type of query or data retrieval from the PubChem database, with parameters allowing for flexible and targeted searches.
- **Testing and Validation:** We performed rigorous testing to ensure accuracy and efficiency using the testthat package for unit testing each function. This approach helps us define and verify expected behavior, catch errors, and integrate well with our workflow, thereby maintaining high code quality and reliability.

## 4 Use cases

The PubChemR package interfaces with the PubChem database through URL syntax, with each query within this syntax serving as an argument in the designated function (Figure 1).

The functions in PubChemR are designed with flexibility in mind, allowing users to specify the type of information they need and the format in which they wish to receive it. For instance, data can be returned as R objects like data frames or lists, ready for analysis.

PubChemR pacakage can be installed from CRAN (The Comprehensive R Archive Network) and loaded as follow:

```
# install.packages("PubChemR", repos = "http://cran.us.r-project.org")
library("PubChemR")
```

The package is currently in a state of active development. The latest version in development can be accessed via GitHub (https://github.com/selcukorkmaz/PubChemR). This paper was composed utilizing **PubChemR** version 2.1.

Most functions in the package require three main arguments; domain, namespace, and identifier.

**1. Domain:** This represents the primary classification within the PubChem system that dictates the type of data being accessed. Examples of domains include "substance," "compound," "assay," "gene," and others. Each domain encapsulates specific types of scientific data, such as chemical compounds or genetic information.

**2. Namespace:** Within each domain, the namespace further specifies the method or criteria for querying the data. It acts as a sub-category within the domain that allows for more refined searches. For instance, in the compound domain, namespaces can be specific identifiers like "cid" for compound ID, or "name" for the compound's common name, among others.

**3. Identifier:** These are the actual data values used to perform the query. Identifiers can be vector of positive integers (e.g. cid, sid, aid) or strings (e.g. name, smiles, source, inchikey, formula). They are the key pieces of information that pinpoint the exact record or set of records to be retrieved from the database.

Additionally, the optional arguments "operation" and "searchtype" play crucial roles in refining the scope and focus of data queries. The "operation" argument specifies the type of data processing or retrieval task that should be performed on the identified records. For instance, operations can range from fetching complete data records to retrieving specific properties or summaries of compounds, genes, or assays. This flexibility allows users to access both broad overviews and detailed attributes of database entries according to their research needs. Meanwhile, the "searchtype" argument defines the method of search being employed. It could be a structured search, such as substructure or similarity search, which is essential for identifying compounds with particular chemical structures or features. These optional arguments enhance the API's versatility, enabling researchers to tailor queries more precisely and retrieve data that best fits their experimental and analytical requirements. For more detailed information, please refer to the official documentation at https://pubchem.ncbi.nlm.nih.gov/docs/pug-rest.

Table 1 provides detailed information about four key arguments: "domain," "searchtype," "namespace," and "operation." This table is designed to help users understand how to effectively utilize these components to customize queries within the PubChemR package.

| domain | searchtype | namespace | operation |
|---|---|---|---|
| substance | - | sid, sourceid/<source id>, sourceall/<source name>, name, <xref>, listkey | record, synonyms, sids, cids, aids, assaysummary, classification, <xrefs>, description |
| compound | <structure search> = {substructure, superstructure, similarity, identity}/{smiles, inchi, sdf, cid} <fast search> = {fastidentity, fastsimilarity_2d, fastsimilarity_3d, fastsubstructure, fastsuperstructure}/{smiles, smarts, inchi, sdf, cid}, fastformula | cid, name, smiles, inchi, sdf, inchikey, formula, <structure search>, <xref>, listkey, <fast search> | record, <compound property>, synonyms, sids, cids, aids, assaysummary, classification, <xrefs>, description, conformers |
| assay | - | aid, listkey, type/<assay type>, sourceall/<source name>, target/<assay target>, activity/<activity column name> | record, concise, aids, sids, cids, description, targets/<target type>, <doseresponse>, summary, classification |

| domain | searchtype | namespace | operation |
|---|---|---|---|
| gene | - | geneid, genesymbol, synonym | summary, aids, concise, pwaccs |
| protein | - | accession, gi, synonym | summary, aids, concise, pwaccs |
| pathway | - | pwacc | summary, cids, geneids, accessions |
| taxonomy | - | taxid, synonym | summary, aids |
| cell | - | cellacc, synonym | summary, aids |

Table 1: Overview of key arguments in the PubChemR package

In the following sections, we will explore each function in detail, examining the parameters they accept, the type of data they return, and providing examples to illustrate their use. These examples will serve as a guide for users to understand how to effectively utilize the **PubChemR** package to access and manipulate chemical and biological data for their specific needs.

First, we will concentrate on three primary functions, each focusing on a specific domain: `get_compounds` to retrieve compound data, `get_substances` to extract substance data, and `get_assays` to fetch assay data. These functions are capable of handling multiple queries simultaneously and return a *PubChemInstanceList* class. This is a specialized class specifically created for these functions to manage the complex PubChem data efficiently. After utilizing each of these functions, we will employ the `instance` function. This function is designed to retrieve detailed information about a compound from a *PubChemInstanceList*. It provides comprehensive details about the specific compound, including its instance details (i.e., slots). Finally, we will implement the `retrieve` function with the relevant slots to extract specific data elements from the compound data. This approach ensures that we can precisely access the required information from the vast amount of data available, thereby enhancing the efficiency and effectiveness of our data analysis process.

Next, we will fetch a variety of compound properties, such as molecular weight, chemical formula, isomeric SMILES, and more, using the `get_properties` function. Additionally, we will focus on two functions for downloading data from the PubChem database: `get_sdf` and `download`. The `get_sdf` function is specifically designed to download chemical structure data in the widely recognized Structure Data File (SDF) format. The `download` function streamlines the process of accessing and downloading content from the PubChem database.

Finally, we will introduce two new functions: `get_pug_rest` and `get_pug_view`. The `get_pug_rest` function provides a direct and efficient method for accessing a wide range of chemical data. In contrast, the `get_pug_view` function is designed to offer access to detailed summary reports and additional information that is not usually included in the primary PubChem Substance, Compound, or BioAssay records.

### 4.1 Retrieve Compund Information

The `get_compounds` function allows R users to retrieve compound information from the PubChem database. This function specifically targets retrieving compound-related information. This specialized focus is crucial for users who require direct and efficient access to detailed compound data, a common need in various fields of chemical research and analysis. Below, we will demonstrate how to retrieve compound data using different namespaces:

**a. Retrieving Compound Information by Name:** The `get_compounds` function simplifies the process of retrieving detailed compound information by using common compound names. This feature is particularly beneficial in scenarios where the specific CIDs of compounds are unknown or in educational contexts where common names are more frequently used. It simplifies the process of data retrieval for users who may not be familiar with the technical identifiers of compounds but are well-versed with their common or commercial names.

Consider the following example where the `get_compounds` function is employed to fetch data for compounds using their common names:

```
compounds <- get_compounds(identifier = c("aspirin", "caffeine", "glucose"), namespace = "name")
compounds
```

```
#>
#>  An object of class 'PubChemInstanceList'
#>
```

```
#>  Number of instances: 3
#>   - Domain: Compound
#>   - Namespace: Name
#>   - Identifier(s): aspirin, caffeine, ... and 1 more.
#>
#> * Run 'instance(...)' function to extract specific instances from the complete list, and
#>    'request_args(...)' to see all the requested instance identifiers.
#>  * See ?instance and ?request_args for details.
```

The request_args function can be executed to view all the requested instance identifiers:

```
request_args(object = compounds)
```

```
#> $namespace
#> [1] "name"
#>
#> $identifier
#> [1] "aspirin"  "caffeine" "glucose"
#>
#> $domain
#> [1] "compound"
#>
#> $operation
#> NULL
#>
#> $options
#> NULL
#>
#> $searchtype
#> NULL
```

To retrieve detailed information about a specific compound (e.g. aspirin), we can use the instance function on the result:

```
compound_aspirin <- instance(object = compounds, .which = "aspirin")
compound_aspirin
```

```
#>
#>  An object of class 'PubChemInstance'
#>
#>  Request Details:
#>   - Domain: Compound
#>   - Namespace: Name
#>   - Identifier: aspirin
#>
#>  Instance Details:
#>   - id (1): [<named list>] id
#>   - atoms (2): [<named list>] aid, element
#>   - bonds (3): [<named list>] aid1, aid2, order
#>   - coords (1): [<unnamed list>]
#>   - charge (1): [<unnamed numeric>]
#>   - props (23): [<unnamed list>]
#>  - count (10): [<named numeric>] heavy_atom, atom_chiral, atom_chiral_def, atom_chiral_undef, ...
#>
#> NOTE: Run getter function 'retrieve()' with element name above to extract data from corresponding list.
#>       See ?retrieve for details.
```

The instance function retrieves detailed information about the specific compound, including its various components (known as slots). In this example, the compound has seven slots: id, atoms, bonds, coords, charge, props, and count. To extract specific data elements from the compound data, we can use the retrieve function with the relevant slots. For example, using the *props* slot extracts detailed properties of the compound, including information such as label, name, data type, release, value, implementation, version, software, and source. This comprehensive information covers various physical, chemical, and structural properties of the compound.

```
retrieve(object = compound_aspirin, .slot = "props", .to.data.frame = TRUE)

#> # A tibble: 23 x 11
#>    Identifier label name  datatype release value implementation version software
#>    <chr>      <chr> <chr> <chr>    <chr>   <chr> <chr>          <chr>   <chr>
#>  1 aspirin    Comp~ Cano~ 5        2025.0~ 1     <NA>           <NA>    <NA>
#>  2 aspirin    Comp~ <NA>  7        2025.0~ 212   E_COMPLEXITY   3.4.8.~ Cactvs
#>  3 aspirin    Count Hydr~ 5        2025.0~ 4     E_NHACCEPTORS  3.4.8.~ Cactvs
#>  4 aspirin    Count Hydr~ 5        2025.0~ 1     E_NHDONORS     3.4.8.~ Cactvs
#>  5 aspirin    Count Rota~ 5        2025.0~ 3     E_NROTBONDS    3.4.8.~ Cactvs
#>  6 aspirin    Fing~ SubS~ 16       2025.0~ 0000~ E_SCREEN       3.4.8.~ Cactvs
#>  7 aspirin    IUPA~ Allo~ 1        2025.0~ 2-ac~ <NA>           2.7.0   Lexiche~
#>  8 aspirin    IUPA~ CAS-~ 1        2025.0~ 2-ac~ <NA>           2.7.0   Lexiche~
#>  9 aspirin    IUPA~ Mark~ 1        2025.0~ 2-ac~ <NA>           2.7.0   Lexiche~
#> 10 aspirin    IUPA~ Pref~ 1        2025.0~ 2-ac~ <NA>           2.7.0   Lexiche~
#> # i 13 more rows
#> # i 2 more variables: source <chr>, parameters <chr>
```

**b. Retrieving Compound Information by CID:** The `get_compounds` function can be used to extract detailed compound data utilizing CIDs. This feature is particularly advantageous for researchers who require precise and comprehensive data on specific compounds. By inputting a vector of CIDs, users can quickly access a vast amount of information for their research needs.

Here's an illustrative example demonstrating the use of `get_compounds` to obtain data for a set of compounds using their CIDs:

```
compounds <- get_compounds(identifier = c(2244, 305), namespace = "cid")
compounds

#>
#>  An object of class 'PubChemInstanceList'
#>
#>  Number of instances: 2
#>   - Domain: Compound
#>   - Namespace: CID
#>   - Identifier(s): 2244, 305
#>
#> * Run 'instance(...)' function to extract specific instances from the complete list, and
#>    'request_args(...)' to see all the requested instance identifiers.
#>  * See ?instance and ?request_args for details.
```

Similarly, we can use the `instance` function on the result to retrieve detailed information about CID 2244:

```
compound_2244 <- instance(object = compounds, .which = 2244)
compound_2244

#>
#>  An object of class 'PubChemInstance'
#>
#>  Request Details:
#>   - Domain: Compound
#>   - Namespace: CID
#>   - Identifier: 2244
#>
#>  Instance Details:
#>   - id (1): [<named list>] id
#>   - atoms (2): [<named list>] aid, element
#>   - bonds (3): [<named list>] aid1, aid2, order
#>   - coords (1): [<unnamed list>]
#>   - charge (1): [<unnamed numeric>]
#>   - props (23): [<unnamed list>]
#>  - count (10): [<named numeric>] heavy_atom, atom_chiral, atom_chiral_def, atom_chiral_undef, ...
#>
#> NOTE: Run getter function 'retrieve()' with element name above to extract data from corresponding list.
#>       See ?retrieve for details.
```

Similar to the previous section, we can use the `retrieve` function with the element names (i.e. slots) mentioned above to extract specific data sections.

**c. Advanced Search with SMILES:** In cheminformatics, the Simplified Molecular Input Line Entry System (SMILES) is a widely-used method for representing chemical structures. The `get_compounds` function package adeptly handles queries based on SMILES strings, enabling users to search for compounds by their structural characteristics. The main advantage of using SMILES is their precise representation of molecular structures, unlike CIDs or chemical names that can be ambiguous or inapplicable.

Consider the following example where the `get_compounds` function is used to search for compounds using their SMILES strings:

```
compounds_by_smiles <- get_compounds(identifier = c("CC(=O)OC1=CC=CC=C1C(=O)O",
                                                    "CN1C=NC2=C1C(=O)N(C(=O)N2C)C"),
                                     namespace = "smiles")
compounds_by_smiles
```

```
#>
#>  An object of class 'PubChemInstanceList'
#>
#>  Number of instances: 2
#>   - Domain: Compound
#>   - Namespace: SMILES
#>   - Identifier(s): CC(=O)OC1=CC=CC=C1C(=O)O, CN1C=NC2=C1C(=O)N(C(=O)N2C)C
#>
#> * Run 'instance(...)' function to extract specific instances from the complete list, and
#>    'request_args(...)' to see all the requested instance identifiers.
#>  * See ?instance and ?request_args for details.
```

In this example, compounds_by_smiles object is a *PubChemInstanceList* containing data for compounds that correspond to the provided SMILES strings. The strings "CC(=O)OC1=CC=CC=C1C(=O)O" and "CN1C=NC2=C1C(=O)N(C(=O)N2C)C" represent the molecular structures of aspirin and caffeine, respectively. The function returns a *PubChemInstanceList* where each element contains comprehensive information about these compounds.

Let's only access data for *CC(=O)OC1=CC=CC=C1C(=O)O* using the instance function.

```
compound_smiles <- instance(object = compounds_by_smiles, .which = "CC(=O)OC1=CC=CC=C1C(=O)O")
compound_smiles
```

```
#>
#>  An object of class 'PubChemInstance'
#>
#>  Request Details:
#>   - Domain: Compound
#>   - Namespace: SMILES
#>   - Identifier: CC(=O)OC1=CC=CC=C1C(=O)O
#>
#>  Instance Details:
#>   - id (1): [<named list>] id
#>   - atoms (2): [<named list>] aid, element
#>   - bonds (3): [<named list>] aid1, aid2, order
#>   - coords (1): [<unnamed list>]
#>   - charge (1): [<unnamed numeric>]
#>   - props (23): [<unnamed list>]
#>  - count (10): [<named numeric>] heavy_atom, atom_chiral, atom_chiral_def, atom_chiral_undef, ...
#>
#> NOTE: Run getter function 'retrieve()' with element name above to extract data from corresponding list.
#>       See ?retrieve for details.
```

Finally, we can utilize the `retrieve` function to access data in each slot of the *compound_smiles* object, as demonstrated earlier.

### 4.2 Retrieve Substance Information

The `get_substances` function is for researchers looking to explore the extensive substance records in the PubChem database. PubChem's definition of substances encompasses a broad spectrum of chemical entities, ranging from unique samples of individual chemical compounds to complex mixtures. These substances often manifest in diverse forms, including but not limited to different salts, isotopes, complexes, or combinations of various compounds.

The `get_substances` function is designed to meet the specific requirements of accessing a wide range of substance records. It uses unique substance identifiers (SIDs) for each substance in PubChem, allowing users to easily and accurately retrieve detailed information. This feature is especially important in situations where understanding the differences between various forms or versions of a compound is critical, such as in drug research, managing chemical databases, or meeting regulatory standards.

In the following example, we will retrieve substance data for three substances using their SIDs:

- Aspirin (SID: 103164874): Aspirin, also known as acetylsalicylic acid, is widely used as an analgesic to relieve pain, reduce fever, and act as an anti-inflammatory medication. As a substance, it encompasses various forms and preparations of aspirin available from different sources.
- Caffeine (SID: 403385742): Caffeine is a central nervous system stimulant commonly found in coffee, tea, and various energy drinks. As a substance, it includes different sources and formulations of caffeine beyond its pure chemical structure.
- Glucose (SID: 403435554): Glucose is a simple sugar that serves as a primary energy source for living organisms. As a substance, it includes various forms and sources of glucose, providing detailed information beyond the pure compound.

Using the `get_substances` function, we can access detailed records for these substances in the PubChem database by providing their SIDs.

```
substances <- get_substances(identifier = c(103164874, 403385742, 403435554),
                             namespace = "sid")
substances
```

```
#>
#>  An object of class 'PubChemInstanceList'
#>
#>  Number of instances: 3
#>   - Domain: Substance
#>   - Namespace: SID
#>   - Identifier(s): 103164874, 403385742, ... and 1 more.
#>
#> * Run 'instance(...)' function to extract specific instances from the complete list, and
#>    'request_args(...)' to see all the requested instance identifiers.
#>  * See ?instance and ?request_args for details.
```

Next, let's fetch detailed substance information for glucose (SID: 403435554). First, we need to run the instance function to see the slots that contain substance data:

```
instance(object = substances, .which = 403435554)
```

```
#>
#>  Substance Data from PubChem Database
#>
#>  Request Details:
#>   - Domain: Substance
#>   - Namespace: SID
#>   - Identifier: 403435554
#>
#>  Number of substances retrieved: 1
#>
#>  Substances contain data within following slots;
#>   - sid (2): [<named numeric>] id, version
#>   - source (1): [<named list>] db
```

```
#>   - xref (3): [<unnamed list>]
#>   - compound (2): [<unnamed list>]
#>
#> NOTE: Run getter function 'retrieve()' with element name above to extract data from corresponding list.
#>       See ?retrieve for details.
```

The output shows four slots that contain substance data for glucose. We can retrieve data from each slot using the `retrieve` function. For example, we can extract detailed compound information, such as the compound ID, atom IDs and elements, charges, bond details, and coordinates for molecular structure visualization:

```
retrieve(object = substances, .which = 403435554, .slot = "compound", .to.data.frame = FALSE)
```

```
#> $Identifier
#> [1] 403435554
#>
#> [[2]]
#> [[2]]$id
#> type
#>    0
#>
#> [[2]]$atoms
#> [[2]]$atoms$aid
#>  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16
#>
#> [[2]]$atoms$element
#>  [1] 8 8 8 8 8 6 6 8 6 6 6 6 1 1 1 1
#>
#>
#> [[2]]$bonds
#> [[2]]$bonds$aid1
#>  [1]  6  7  7  8  9  9 10 10 10 11 11 11 12 12 12 12
#>
#> [[2]]$bonds$aid2
#>  [1]  5  6 13  7  4  8  3  9 16  2 10 15  1  7 11 14
#>
#> [[2]]$bonds$order
#>  [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
#>
#>
#> [[2]]$coords
#> [[2]]$coords[[1]]
#> [[2]]$coords[[1]]$type
#> [1] 1 3
#>
#> [[2]]$coords[[1]]$aid
#>  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16
#>
#> [[2]]$coords[[1]]$conformers
#> [[2]]$coords[[1]]$conformers[[1]]
#> [[2]]$coords[[1]]$conformers[[1]]$x
#>  [1] -1.1936  0.3873  2.0000  2.0000 -2.0000 -1.1936 -0.3873  0.3873  1.1936
#> [10]  1.1936  0.3873 -0.3873 -0.9849 -0.9129 -0.1439  1.7192
#>
#> [[2]]$coords[[1]]$conformers[[1]]$y
#>  [1] -0.4492 -1.3794 -0.4492  1.3794  0.9143  1.3794  0.9143  1.3794  0.9143
#> [10]  0.0144 -0.4492  0.0144  0.8106  0.3182 -0.7429  0.3168
#>
#> [[2]]$coords[[1]]$conformers[[1]]$style
#> [[2]]$coords[[1]]$conformers[[1]]$style$annotation
#> [1] 6 5 6 5
#>
#> [[2]]$coords[[1]]$conformers[[1]]$style$aid1
#> [1]  7 10 11 12
```

```
#>
#> [[2]]$coords[[1]]$conformers[[1]]$style$aid2
#> [1] 13 16 15 14
#>
#>
#>
#>
#>
#>
#> [[2]]$charge
#> [1] 0
#>
#>
#> [[3]]
#> [[3]]$id
#> [[3]]$id$type
#> [1] 1
#>
#> [[3]]$id$id
#>  cid
#> 5793
```

Besides SIDs, the function supports additional namespaces such as sourceid, sourceall, name, xref, and listkey. This flexibility enables researchers to access substance records from a range of perspectives, depending on their available data or specific requirements.

### 4.3 Retrieve Assay Information

The `get_assays` function is another key function for researchers needing detailed biological assay information from the PubChem database. These assays, essential for drug discovery, toxicology, and pharmacology, measure biological activities of substances. The function simplifies accessing this data, crucial for bioinformatics and cheminformatics (Korkmaz, 2020; Yamasan and Korkmaz, 2024). It enables customized queries in the PubChem database, allowing users to explore a wide range of assay data, including drug efficacy and toxicity. Its versatility in handling different parameters facilitates a specific data retrieval approach.

The utility of the `get_assays` function extends across multiple research scenarios. In drug discovery, It enables researchers to explore assays related to potential drug compounds, aiding in the comprehension of their efficacy and safety profiles. In toxicological studies, the function is key in acquiring insights into the toxic effects of diverse substances.

In the field of scientific research, particularly in areas such as pharmacology, toxicology, and biochemistry, researchers frequently encounter specific assays that are of interest to their studies. These assays are often identified by their Assay IDs (AIDs), which are referenced in scientific literature or various databases. The get_assays function provides a direct and efficient means for researchers to access comprehensive information about these particular assays.

Utilizing the `get_assays` function, researchers can input a vector of AIDs to retrieve detailed data about each corresponding assay. This functionality is especially beneficial for those who need to analyze and interpret assay data as part of their research projects or for educational purposes.

Consider the following practical example where we retrieve assay data from PubChem using their AIDs.

First, we use the `get_assays` function to retrieve data for the specified assays. The identifier parameter is a vector of AIDs (485314, 485341, 504466, 624202, and 651820), and the namespace parameter specifies that the identifiers are AIDs.

```
assays <- get_assays(identifier = c(485314, 485341, 504466, 624202, 651820),
                     namespace = "aid")
assays
```

```
#>
#>  An object of class 'PubChemInstanceList'
#>
#>  Number of instances: 5
#>   - Domain: Assay
```

```
#>   - Namespace: AID
#>   - Identifier(s): 485314, 485341, ... and 3 more.
#>
#> * Run 'instance(...)' function to extract specific instances from the complete list, and
#>    'request_args(...)' to see all the requested instance identifiers.
#>  * See ?instance and ?request_args for details.
```

The output shows that the assays object is of class '*PubChemInstanceList*' and contains data for 5 assays identified by their AIDs. It includes information about the number of instances (5 in this case), the domain (Assay), the namespace (AID), and the specific identifiers. The output also provides hints on how to proceed further.

Now, we can use the `instance` function to extract specific instances from the complete list. For example, the following code extracts the detailed data for the assay with identifier 485314 from the list of assays:

```
assay_485314 <- instance(object = assays, .which = 485314)
assay_485314
```

```
#>
#>  An object of class 'PubChemInstance'
#>
#>  Request Details:
#>   - Domain: Assay
#>   - Namespace: AID
#>   - Identifier: 485314
#>
#>  Instance Details:
#>   - aid (2): [<named numeric>] id, version
#>   - aid_source (1): [<named list>] db
#>   - name (1): [<unnamed character>]
#>   - description (3): [<unnamed character>]
#>   - protocol (1): [<unnamed character>]
#>   - comment (4): [<unnamed character>]
#>   - xref (4): [<unnamed list>]
#>   - results (23): [<unnamed list>]
#>   - revision (1): [<unnamed numeric>]
#>   - target (1): [<unnamed list>]
#>   - activity_outcome_method (1): [<unnamed numeric>]
#>   - dr (1): [<unnamed list>]
#>   - grant_number (1): [<unnamed character>]
#>   - project_category (1): [<unnamed numeric>]
#>
#> NOTE: Run getter function 'retrieve()' with element name above to extract data from corresponding list.
#>       See ?retrieve for details.
```

The output provides a detailed view of the assay_485314 object, which is of class 'PubChemInstance'. It includes both request details and instance details

The output also notes that we can use the `retrieve` function with the element names above to extract data from the corresponding list. For example, we can retrieve the results of the assay, providing detailed data on the outcomes observed during the assay:

```
retrieve(object = assays, .which = 485314, .slot = "results")
```

```
#> # A tibble: 41 x 8
#>    Identifier tid   name                description     type  unit  ac    tc
#>         <dbl> <chr> <chr>               <chr>           <chr> <chr> <chr> <chr>
#>  1     485314 1     Phenotype           Indicates type~ 4     254   <NA>  <NA>
#>  2     485314 2     Potency             Concentration ~ 1     5     TRUE  <NA>
#>  3     485314 3     Efficacy            Maximal effica~ 1     15    <NA>  <NA>
#>  4     485314 4     Analysis Comment    Annotation/not~ 4     254   <NA>  <NA>
#>  5     485314 5     Curve_Description   A description ~ 4     254   <NA>  <NA>
#>  6     485314 6     Fit_LogAC50         The logarithm ~ 1     254   <NA>  <NA>
#>  7     485314 7     Fit_HillSlope       The Hill slope~ 1     254   <NA>  <NA>
```

```
#>  8     485314 8     Fit_R2              R^2 fit value ~ 1     254   <NA>  <NA>
#>  9     485314 9     Fit_InfiniteActivity The asymptotic~ 1     15    <NA>  <NA>
#> 10     485314 10    Fit_ZeroActivity    Efficacy at ze~ 1     15    <NA>  <NA>
#> # i 31 more rows
```

By following these steps and using the `retrieve` function, we can access detailed and specific data from the assay records in PubChem. This comprehensive approach allows researchers to gather all necessary information about assays, aiding in their analysis and research activities.

### 4.4 Retrieve Property Information

The `get_properties` is another important function for researchers who require access to specific chemical property data from the PubChem database. This function is designed with the aim of simplifying the process of querying PubChem for a variety of compound properties, such as molecular weight, chemical formula, isomeric SMILES, and more. It is particularly useful for those in need of detailed chemical information across a range of compounds.

The `get_properties` function allows users to specify a set of properties and the identifiers of the compounds for which these properties are required. The function then queries the PubChem database and retrieves the requested data.

Consider the following practical application of the `get_properties` function:

First, we use the `get_properties` function to retrieve specific properties for several compounds identified by their names:

```
props <- get_properties(
  properties = c("MolecularWeight", "MolecularFormula", "InChI"),
  identifier = c("aspirin", "caffeine", "glucose"),
  namespace = "name"
)
```

In this example, the *properties* parameter specifies that we want to retrieve the molecular weight, molecular formula, and InChI for each compound. The *identifier* parameter lists the common names of the compounds (aspirin, caffeine, and glucose), and the *namespace* parameter indicates that these identifiers are compound names. The function fetches the specified properties from the PubChem database.

To extract the properties for a specific compound, such as aspirin, we can use the `retrieve` function:

```
retrieve(instance(props, "aspirin"), .slot = NULL)
```

```
#> # A tibble: 1 x 6
#>   Identifier   CID MolecularFormula MolecularWeight InChI               InChIKey
#>   <chr>      <dbl> <chr>            <chr>           <chr>               <chr>
#> 1 aspirin     2244 C9H8O4           180.16          InChI=1S/C9H8O4/c1~ BSYNRYM~
```

This tibble shows the identifier (aspirin), CID (2244), molecular formula (C9H8O4), molecular weight (180.16), InChI, and InChIKey.

To combine the properties of all compounds into a single data frame, we set `.combine.all` as `TRUE`:

```
retrieve(props, .combine.all = TRUE)
```

```
#> # A tibble: 3 x 6
#>   Identifier   CID MolecularFormula MolecularWeight InChI               InChIKey
#>   <chr>      <dbl> <chr>            <chr>           <chr>               <chr>
#> 1 aspirin     2244 C9H8O4           180.16          InChI=1S/C9H8O4/c1~ BSYNRYM~
#> 2 caffeine    2519 C8H10N4O2        194.19          InChI=1S/C8H10N4O2~ RYYVLZV~
#> 3 glucose     5793 C6H12O6          180.16          InChI=1S/C6H12O6/c~ WQZGKKK~
```

This tibble shows the identifier, CID, molecular formula, molecular weight, InChI, and InChIKey for aspirin, caffeine, and glucose.

To return only the selected properties for all compounds, we can specify the properties using the `.slot` argument. The output will be a tibble with only the molecular weight and molecular formula for each compound:

```
retrieve(props, .combine.all = TRUE,
  .slot = c("MolecularWeight", "MolecularFormula"))

#>   Identifier MolecularWeight MolecularFormula
#> 1    aspirin          180.16           C9H8O4
#> 2   caffeine          194.19        C8H10N4O2
#> 3    glucose          180.16          C6H12O6
```

There are 40 compound properties that we can fetch from the PubChem:

```
property_map(type = "all")

#>  [1] "MolecularFormula"         "MolecularWeight"
#>  [3] "CanonicalSMILES"          "IsomericSMILES"
#>  [5] "InChI"                    "InChIKey"
#>  [7] "IUPACName"                "XLogP"
#>  [9] "ExactMass"                "MonoisotopicMass"
#> [11] "TPSA"                     "Complexity"
#> [13] "Charge"                   "HBondDonorCount"
#> [15] "HBondAcceptorCount"       "RotatableBondCount"
#> [17] "HeavyAtomCount"           "IsotopeAtomCount"
#> [19] "AtomStereoCount"          "DefinedAtomStereoCount"
#> [21] "UndefinedAtomStereoCount" "BondStereoCount"
#> [23] "DefinedBondStereoCount"   "UndefinedBondStereoCount"
#> [25] "CovalentUnitCount"        "Volume3D"
#> [27] "ConformerModelRMSD3D"     "ConformerModelRMSD3D"
#> [29] "XStericQuadrupole3D"      "YStericQuadrupole3D"
#> [31] "ZStericQuadrupole3D"      "FeatureCount3D"
#> [33] "FeatureAcceptorCount3D"   "FeatureDonorCount3D"
#> [35] "FeatureAnionCount3D"      "FeatureCationCount3D"
#> [37] "FeatureRingCount3D"       "FeatureHydrophobeCount3D"
#> [39] "EffectiveRotorCount3D"    "ConformerCount3D"
```

The `type` argument in the `property_map` function determines the method of searching within the available properties. Setting `type = "contain"` will retrieve all properties that include the strings specified in the properties argument. In the following example, we fetch properties for a range of CIDs from 2244 to 2260. The properties argument includes the keywords "mass" and "molecular", and the `propertyMatch` argument is set to `type = "contain"`. This setup ensures that the function retrieves any properties containing the specified keywords.

```
props <- get_properties(
  properties = c("mass", "molecular"),
  identifier = 2244:2260,
  namespace = "cid",
  propertyMatch = list(
    type = "contain"
  )
)
retrieve(props, .combine.all = TRUE, .to.data.frame = TRUE)

#> # A tibble: 17 x 6
#>    Identifier   CID MolecularFormula MolecularWeight ExactMass  MonoisotopicMass
#>         <int> <dbl> <chr>            <chr>           <chr>      <chr>
#>  1       2244  2244 C9H8O4           180.16          180.04225~ 180.04225873
#>  2       2245  2245 C21H27N5O7S      493.5           493.16311~ 493.16311939
#>  3       2246  2246 C40H52O4         596.8           596.38656~ 596.38656014
#>  4       2247  2247 C28H31FN4O       458.6           458.24818~ 458.24818979
#>  5       2248  2248 C17H35N5O6       405.5           405.25873~ 405.25873385
#>  6       2249  2249 C14H22N2O3       266.34          266.16304~ 266.16304257
#>  7       2250  2250 C33H35FN2O5      558.6           558.25300~ 558.25300038
#>  8       2251  2251 C10H13N5O13P3-3  504.16          503.97227~ 503.97227148
#>  9       2252  2252 C10H16N2O4       228.24          228.11100~ 228.11100700
#> 10       2253  2253 C10H16N2O4       228.24          228.11100~ 228.11100700
```

```
#> 11       2254  2254 C30H44O16S2-2    724.8           724.20707~ 724.20707766
#> 12       2255  2255 C30H46O16S2      726.8           726.22272~ 726.22272772
#> 13       2256  2256 C8H14ClN5        215.68          215.09377~ 215.0937732
#> 14       2257  2257 C22H20N3O9+3     470.4           470.11995~ 470.11995422
#> 15       2258  2258 C22H17N3O9       467.4           467.09647~ 467.09647913
#> 16       2259  2259 C22H14O9         422.3           422.06378~ 422.06378202
#> 17       2260  2260 C44H62N2O12      811.0           810.43027~ 810.43027542
```

Moreover, we can extract properties that start or end with specific strings. In the following example, we fetch properties that start with the word "molecular." Here, the `type` parameter is set to `"start"`, and the `.ignore.case` parameter is set to `TRUE` to make the search case-insensitive.

```
props <- get_properties(
  properties = "molecular",
  identifier = 2244:2260,
  namespace = "cid",
  propertyMatch = list(
    type = "start",
    .ignore.case = TRUE
  )
)
retrieve(props, .combine.all = TRUE, .to.data.frame = TRUE)
```

```
#> # A tibble: 17 x 4
#>    Identifier   CID MolecularFormula MolecularWeight
#>         <int> <dbl> <chr>            <chr>
#>  1       2244  2244 C9H8O4           180.16
#>  2       2245  2245 C21H27N5O7S      493.5
#>  3       2246  2246 C40H52O4         596.8
#>  4       2247  2247 C28H31FN4O       458.6
#>  5       2248  2248 C17H35N5O6       405.5
#>  6       2249  2249 C14H22N2O3       266.34
#>  7       2250  2250 C33H35FN2O5      558.6
#>  8       2251  2251 C10H13N5O13P3-3  504.16
#>  9       2252  2252 C10H16N2O4       228.24
#> 10       2253  2253 C10H16N2O4       228.24
#> 11       2254  2254 C30H44O16S2-2    724.8
#> 12       2255  2255 C30H46O16S2      726.8
#> 13       2256  2256 C8H14ClN5        215.68
#> 14       2257  2257 C22H20N3O9+3     470.4
#> 15       2258  2258 C22H17N3O9       467.4
#> 16       2259  2259 C22H14O9         422.3
#> 17       2260  2260 C44H62N2O12      811.0
```

This output indicates that "MolecularFormula" and "MolecularWeight" are the properties available in PubChem that start with the word "molecular."

Next, we extract properties that end with the word "mass." Here, the `type` parameter is set to `"end"`, and the `.ignore.case parameter` is set to `TRUE` to make the search case-insensitive.

```
props <- get_properties(
  properties = "mass",
  identifier = 2244:2260,
  namespace = "cid",
  propertyMatch = list(
    type = "end",
    .ignore.case = TRUE
  )
)
retrieve(props, .combine.all = TRUE, .to.data.frame = TRUE)
```

```
#> # A tibble: 17 x 4
#>    Identifier   CID ExactMass    MonoisotopicMass
#>         <int> <dbl> <chr>        <chr>
```

```
#>  1       2244  2244 180.04225873 180.04225873
#>  2       2245  2245 493.16311939 493.16311939
#>  3       2246  2246 596.38656014 596.38656014
#>  4       2247  2247 458.24818979 458.24818979
#>  5       2248  2248 405.25873385 405.25873385
#>  6       2249  2249 266.16304257 266.16304257
#>  7       2250  2250 558.25300038 558.25300038
#>  8       2251  2251 503.97227148 503.97227148
#>  9       2252  2252 228.11100700 228.11100700
#> 10       2253  2253 228.11100700 228.11100700
#> 11       2254  2254 724.20707766 724.20707766
#> 12       2255  2255 726.22272772 726.22272772
#> 13       2256  2256 215.0937732  215.0937732
#> 14       2257  2257 470.11995422 470.11995422
#> 15       2258  2258 467.09647913 467.09647913
#> 16       2259  2259 422.06378202 422.06378202
#> 17       2260  2260 810.43027542 810.43027542
```

This output indicates that "ExactMass" and "MonoisotopicMass" are the properties available in PubChem that end with the word "mass."

Finally, to return all available properties of the requested compounds, set properties = NULL and the propertyMatch argument to type = "all".

```
props <- get_properties(
  properties = NULL,
  identifier = 2244:2260,
  namespace = "cid",
  propertyMatch = list(
    type = "all"
  )
)
retrieve(props, .combine.all = TRUE)
```

```
#> # A tibble: 17 x 41
#>    Identifier   CID MolecularFormula MolecularWeight CanonicalSMILES
#>         <int> <dbl> <chr>            <chr>           <chr>
#>  1       2244  2244 C9H8O4           180.16          CC(=O)OC1=CC=CC=C1C(=O)O
#>  2       2245  2245 C21H27N5O7S      493.5           CC1(C(N2C(S1)C(C2=O)NC(=O)~
#>  3       2246  2246 C40H52O4         596.8           CC1=C(C(CC(C1=O)O)(C)C)C=C~
#>  4       2247  2247 C28H31FN4O       458.6           COC1=CC=C(C=C1)CCN2CCC(CC2~
#>  5       2248  2248 C17H35N5O6       405.5           CC(C1CCC(C(O1)OC2C(C(C(C(C~
#>  6       2249  2249 C14H22N2O3       266.34          CC(C)NCC(COC1=CC=C(C=C1)CC~
#>  7       2250  2250 C33H35FN2O5      558.6           CC(C)C1=C(C(=C(N1CCC(CC(CC~
#>  8       2251  2251 C10H13N5O13P3-3  504.16          C1=NC(=C2C(=N1)N(C=N2)C3C(~
#>  9       2252  2252 C10H16N2O4       228.24          CC(C)(C)C1=C(C(=O)NO1)CC(C~
#> 10       2253  2253 C10H16N2O4       228.24          CC(C)(C)C1=C(C(=O)NO1)CC(C~
#> 11       2254  2254 C30H44O16S2-2    724.8           CC(C)CC(=O)OC1C(C(C(OC1OC2~
#> 12       2255  2255 C30H46O16S2      726.8           CC(C)CC(=O)OC1C(C(C(OC1OC2~
#> 13       2256  2256 C8H14ClN5        215.68          CCNC1=NC(=NC(=N1)Cl)NC(C)C
#> 14       2257  2257 C22H20N3O9+3     470.4           C1=CC(=C(C=C1C(=C2C=CC(=O)~
#> 15       2258  2258 C22H17N3O9       467.4           C1=CC(=C(C=C1C(=C2C=CC(=O)~
#> 16       2259  2259 C22H14O9         422.3           C1=CC(=C(C=C1C(=C2C=CC(=O)~
#> 17       2260  2260 C44H62N2O12      811.0           CCC(C(=O)NCC=CC=C(C)C(C(C)~
#> # i 36 more variables: IsomericSMILES <chr>, InChI <chr>, InChIKey <chr>,
#> #   IUPACName <chr>, XLogP <dbl>, ExactMass <chr>, MonoisotopicMass <chr>,
#> #   TPSA <dbl>, Complexity <dbl>, Charge <dbl>, HBondDonorCount <dbl>,
#> #   HBondAcceptorCount <dbl>, RotatableBondCount <dbl>, HeavyAtomCount <dbl>,
#> #   IsotopeAtomCount <dbl>, AtomStereoCount <dbl>,
#> #   DefinedAtomStereoCount <dbl>, UndefinedAtomStereoCount <dbl>,
#> #   BondStereoCount <dbl>, DefinedBondStereoCount <dbl>, ...
```

### 4.5 Download SDF Data

The `get_sdf` function is designed specifically for downloading chemical structure data in the widely recognized Structure Data File (SDF) format. This format is essential in the exchange of chemical structure information, encompassing comprehensive details such as molecular structure, associated properties, and extended chemical data. The primary function of `get_sdf` is to facilitate the retrieval of SDF files for specific compounds using their unique CID from the PubChem database.

To obtain an SDF file for a particular compound, users can execute the `get_sdf` function with the compound's CID as the identifier. For instance, to download the SDF file for a compound with CID 2244, the following code is used:

```
get_sdf(identifier = 2244, namespace = "cid")
```

This code triggers the download of the SDF file for the specified compound, saving it in a temporary folder with a unique, time-stamped file name. Furthermore, users can define the `path` and `file_name` arguments to customize the download.

### 4.6 Download Data with Different Formats

The `download` function simplifies the process of accessing and downloading content from the PubChem database. This function is especially significant for researchers who need to retrieve various types of chemical data in different formats for their work. At its core, the `download` function serves to fetch data from the PubChem database using a specified identifier, and then save this data in a chosen format to a user-defined location on the local file system. The function supports a wide range of output formats, including JSON, XML, SDF and PNG.

For example, to `download` a JSON file for the compound "aspirin" and save it to a folder named "Compound" in the current directory, one would use the following code:

```
download( filename = "Aspirin", outformat = "json", path = tempdir(),
identifier = "aspirin", namespace = "name", domain = "compound", overwrite = TRUE)
```

This flexibility in specifying parameters makes the `download` function particularly useful for diverse research requirements, from simple data retrievals to complex queries.

### 4.7 Accessing and Exploring Chemical Data with PUG REST Service

The `get_pug_rest` function is designed to provide easy and efficient access to the extensive chemical data in PubChem. This function highlights the advanced capabilities of the Power User Gateway (PUG) REST service provided by PubChem (Kim et al., 2015, 2018). It stands out for users who require programmatic interaction with PubChem's extensive database, simplifying the otherwise complex process of data retrieval and analysis. By leveraging the PUG REST service, `get_pug_rest` provides a direct and efficient pathway for accessing a vast array of chemical data, making it an important resource for researchers in various fields who rely on accurate and extensive chemical information for their work. This function is essential for modern computational chemistry, providing access to big data and efficient data processing, which are crucial for advancing research and development in the chemical sciences.

PUG REST is a simple way to access PubChem's data and services, designed for use in scripts, web page JavaScript, and third-party applications. This interface is a simpler and more user-friendly alternative to the complex XML and SOAP envelopes used by other PUG versions. Its design is based on the PubChem identifier system, which includes SID for substances, CID for compounds, and AID for assays, making targeted data retrieval easier. The request architecture in PUG REST is logically segmented into three core components: input (identifiers), operation (actions on identifiers), and output (the format on the PubChem API side).

Overall, the `get_pug_rest` function meets various user needs with different input methods, operations, and outputs, allowing users to customize their queries to the PubChem database in numerous ways.

**1. Retrieving Chemical Structure Information:** Users can request detailed information about chemical structures using different identifiers like SIDs, CIDs, or common names. The `get_pug_rest` function provides a way to access this information efficiently through PubChem's PUG REST service. For instance, the following R code demonstrates how to retrieve chemical structure information using an SID:

```
chemical_structure <- get_pug_rest(identifier = "10000",
                                   namespace = "sid",
                                   domain = "substance",
                                   output = "JSON")
chemical_structure

#>
#>  An object of class 'PugRestInstance'
#>
#>  Request Details:
#>   - Domain: Substance
#>   - Namespace: SID
#>   - Operation: <NULL>
#>   - Identifier: 10000
#>
#> NOTE: Run getter function 'pubChemData(...)' to extract raw data retrieved from PubChem Database.
#>       See ?pubChemData for details.
```

This code requests detailed information about the chemical structure associated with the SID "10000", which corresponds to the compound dihydroergotamine. The `get_pug_rest` function sends a query to the PubChem database, specifying the identifier type (namespace), the domain of interest, and the output format for the response from the PubChem API. The result, stored in the chemical_structure object, contains comprehensive data about the substance, including its structural details, which can then be utilized for further analysis or research.

Upon execution, the chemical_structure_result object contains detailed information about the substance with SID 10000. Now, we use `pubChemData` function to access the chemical structure data.

```
chemical_structure_result <- pubChemData(chemical_structure)
```

The output is a nested list structure, and here is a breakdown of the key components:

**SID and Version:** The identifier (SID) and its version are provided:

```
chemical_structure_result$PC_Substances[[1]]$sid

#>      id version
#>   10000       7
```

This indicates that the substance has SID 10000 and is on version 7.

**Source Database:** Information about the source database is included:

```
chemical_structure_result$PC_Substances[[1]]$source$db$name

#> [1] "KEGG"
```

The substance is sourced from the KEGG database, and its KEGG ID is "C07798":

```
chemical_structure_result$PC_Substances[[1]]$source$db$source_id

#>      str
#> "C07798"
```

**Synonyms:** Various synonyms for the substance are listed:

```
chemical_structure_result$PC_Substances[[1]]$synonyms

#> [1] "511-12-6"          "C07798"            "Dihydroergotamine"
```

These include its CAS (Chemical Abstracts Service) number "511-12-6", its KEGG ID "C07798", and its common name "Dihydroergotamine".

**Comments:** Additional comments provide links to related substances:

```
chemical_structure_result$PC_Substances[[1]]$comment
```

```
#> [1] "Same as: <a href=\"http://pubchem.ncbi.nlm.nih.gov/summary/summary.cgi?sid=96024534\">D07837</a>"
```

This indicates that SID 10000 is the same as another substance with SID 96024534.

**Cross-references (Xrefs):** Cross-references to other identifiers and databases:

```
chemical_structure_result$PC_Substances[[1]]$xref
```

```
#> [[1]]
#>    regid
#> "C07798"
#>
#> [[2]]
#>         rn
#> "511-12-6"
#>
#> [[3]]
#>                        dburl
#> "http://www.genome.jp/kegg/"
#>
#> [[4]]
#>                                                 sburl
#> "http://www.genome.jp/dbget-bin/www_bget?cpd:C07798"
```

This includes references to the KEGG and its associated URLs.

**Compound Information:** The compound section details atomic and bonding information. The atoms and their elements:

```
chemical_structure_result$PC_Substances[[1]]$compound[[1]]$atoms$aid
```

```
#>  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
#> [26] 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46
```

```
chemical_structure_result$PC_Substances[[1]]$compound[[1]]$atoms$element
```

```
#>  [1] 8 8 8 8 8 7 7 7 7 7 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6
#> [39] 6 6 6 6 6 1 1 1
```

Atoms are identified by their atomic number, indicating elements like oxygen (8), nitrogen (7), carbon (6), and hydrogen (1).

Bonding information includes atom pairs and bond order:

```
chemical_structure_result$PC_Substances[[1]]$compound[[1]]$bonds$aid1
```

```
#>  [1]  1  1  2  3  4  5  6  6  6  7  7  7  8  8  9  9  9 10 10 11 12 12 13 13 15
#> [26] 15 17 17 17 17 18 18 19 20 20 21 22 22 23 23 23 26 28 28 31 33 35 35 36 39
#> [51] 40 41 42
```

```
chemical_structure_result$PC_Substances[[1]]$compound[[1]]$bonds$aid2
```

```
#>  [1] 11 13 11 14 16 27 11 14 15 12 16 21 13 27 18 32 38 31 34 12 19 44 14 30 16
#> [26] 26 18 20 25 45 29 46 24 22 33 24 28 31 25 27 32 35 29 34 36 37 39 40 37 41
#> [51] 42 43 43
```

```
chemical_structure_result$PC_Substances[[1]]$compound[[1]]$bonds$order
```

```
#>  [1] 1 1 1 2 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 2
#> [39] 1 1 1 1 1 2 1 1 2 1 2 1 2 2 1
```

**Coordinates:** The coordinates of the atoms provide spatial information about the structure:

```
chemical_structure_result$PC_Substances[[1]]$compound[[1]]$coords[[1]]$conformers

#> [[1]]
#> [[1]]$x
#>  [1] 27.2047 28.3501 25.1187 31.8384 21.7294 28.4024 30.6466 24.3766 24.0201
#> [10] 19.4272 28.3618 29.5069 26.0652 26.0652 29.5419 30.6873 21.7236 22.8806
#> [19] 29.5069 20.5784 31.7917 20.5784 22.8806 31.7977 21.7236 29.5361 22.8806
#> [28] 21.7178 22.8689 26.0595 19.4272 24.0201 19.4330 21.7178 30.6699 18.2878
#> [37] 18.2878 25.1712 30.6582 31.8150 31.7975 32.9487 32.9428 29.4778 20.5667
#> [46] 24.1194
#>
#> [[1]]$y
#>  [1] -12.4990 -11.8386 -15.4207 -15.1577 -12.4814 -14.4798 -13.1652 -12.4230
#>  [9] -16.4548 -20.4226 -13.1593 -12.4990 -13.1593 -14.4857 -15.1460 -14.4915
#> [17] -16.4548 -17.1210 -11.1785 -17.1095 -12.5107 -18.4418 -14.4740 -11.1843
#> [25] -15.1285 -16.4665 -13.1418 -19.1019 -18.4535 -11.7451 -19.1019 -15.1285
#> [33] -16.4492 -20.4226 -17.1327 -18.4418 -17.1095 -17.1210 -18.4535 -16.4782
#> [41] -19.1196 -17.1444 -18.4649 -13.8253 -15.7888 -17.9275
#>
#> [[1]]$style
#> [[1]]$style$annotation
#> [1] 6 5 5 6 6 5 5
#>
#> [[1]]$style$aid1
#> [1] 11 12 13 15 17 18 23
#>
#> [[1]]$style$aid2
#> [1]  2 44 30 26 45 46 27
```

These coordinates allow for visualization and further spatial analysis of the chemical structure.

**2. Performing Structure Searches:** The function facilitates searches like substructure or similarity searches and faster synchronous searches for identity, similarity, substructure, and superstructure. These faster searches typically return results in a single call, significantly improving efficiency for users who require quick access to chemical structure data.

For example, to perform a fast identity search, the following R code is used:

```
structure_search <- get_pug_rest(identifier = "5793",
                                 namespace = "cid",
                                 domain = "compound",
                                 operation = "cids",
                                 searchtype = "fastidentity",
                                 options = list(identity_type = "same_connectivity"),
                                 output = "JSON")
structure_search

#>
#>  An object of class 'PugRestInstance'
#>
#>  Request Details:
#>   - Domain: Compound
#>   - Namespace: CID
#>   - Operation: cids
#>   - Identifier: 5793
#>
#> NOTE: Run getter function 'pubChemData(...)' to extract raw data retrieved from PubChem Database.
#>       See ?pubChemData for details.
```

This code initiates a search using the CID "5793" to find all compounds with the same connectivity. The search results are returned in list object.

The output from this search provides a list of CIDs that match the criteria:

```
structure_search_result <- pubChemData(structure_search)
length(structure_search_result$IdentifierList$CID)
```

```
#> [1] 348

head(structure_search_result$IdentifierList$CID)

#> [1]  5793   206  6036 18950 64689 66371
```

The output contains a comprehensive list of CIDs that have the same connectivity as the original compound with CID 5793, referring to acetaminophen, a commonly used analgesic and antipyretic agent. This indicates that all these compounds have the same atomic connectivity but might differ in other aspects such as stereochemistry or charge states. This fast identity search is particularly useful for researchers looking to quickly find all compounds with the same basic structure, which can then be further analyzed for properties, activities, or potential as drug candidates.

**3. Accessing BioAssay Data:** The function provides a gateway to comprehensive BioAssay records, including detailed descriptions, datasets, concise readouts, and target information. This is particularly useful for researchers who need to analyze biological activities of compounds.

For example, to retrieve concise data such as the active concentration readout, the following R code can be used:

```
bioassay_data <- get_pug_rest(identifier = "504526",
                              namespace = "aid",
                              domain = "assay",
                              operation = "concise",
                              output = "CSV")
bioassay_data

#>
#>  An object of class 'PugRestInstance'
#>
#>  Request Details:
#>   - Domain: Assay
#>   - Namespace: AID
#>   - Operation: concise
#>   - Identifier: 504526
#>
#> NOTE: Run getter function 'pubChemData(...)' to extract raw data retrieved from PubChem Database.
#>        See ?pubChemData for details.
```

The output provides a detailed dataset which includes the Activity Outcome, Target Accession, and other relevant information for each assay result.

```
bioassay_data_result <- pubChemData(bioassay_data)
head(bioassay_data_result)

#>      AID       SID      CID Activity.Outcome Target.Accession Target.GeneID
#> 1 504526 103061373  6619281           Active               NA            NA
#> 2 504526 103904139  2971528           Active               NA            NA
#> 3 504526 103904144  4969604         Inactive               NA            NA
#> 4 504526 104169543 49842897           Active               NA            NA
#> 5 504526 104169544 49842896           Active               NA            NA
#> 6 504526 104169545  1077725         Inactive               NA            NA
#>   Activity.Value..uM. Activity.Name
#> 1                 3.6          IC50
#> 2                31.4          IC50
#> 3                50.0          IC50
#> 4                 5.1          IC50
#> 5                34.6          IC50
#> 6                50.0          IC50
#>                                                                                Assay.Name
#> 1 A Cell Based HTS Approach for the Discovery of New Inhibitors of Respiratory syncytial virus (RSV) using synth
#> 2 A Cell Based HTS Approach for the Discovery of New Inhibitors of Respiratory syncytial virus (RSV) using synth
#> 3 A Cell Based HTS Approach for the Discovery of New Inhibitors of Respiratory syncytial virus (RSV) using synth
#> 4 A Cell Based HTS Approach for the Discovery of New Inhibitors of Respiratory syncytial virus (RSV) using synth
#> 5 A Cell Based HTS Approach for the Discovery of New Inhibitors of Respiratory syncytial virus (RSV) using synth
```

```
#> 6 A Cell Based HTS Approach for the Discovery of New Inhibitors of Respiratory syncytial virus (RSV) using synth
#>     Assay.Type PubMed.ID RNAi
#> 1 Confirmatory        NA   NA
#> 2 Confirmatory        NA   NA
#> 3 Confirmatory        NA   NA
#> 4 Confirmatory        NA   NA
#> 5 Confirmatory        NA   NA
#> 6 Confirmatory        NA   NA
```

The bioassay_data dataframe contains several columns, each providing specific information about the assay results. For example:

- *AID (Assay Identifier):* This column contains the unique identifier for the assay, which in this case is "504526" for all rows, indicating that all data pertains to the same assay.
- *SID (Substance Identifier) and CID (Compound Identifier):* These columns list the identifiers for the substances and compounds tested in the assay.
- *Activity.Outcome:* This column indicates whether the substance or compound was found to be "Active" or "Inactive" in the assay.
- *Target.Accession:* This column would typically contain the accession numbers for the protein targets involved in the assay. An accession number is a unique identifier assigned to a protein sequence record. In the given dataset, all entries show "NA", indicating that specific target accession numbers are not provided for these assay results.
- *Target.GeneID:* This column would contain the GeneID, a unique identifier for genes provided by the NCBI Gene database. Similar to the Target.Accession column, all entries in this dataset show "NA", suggesting that no specific gene identifiers are associated with these assay results.
- *Activity.Value..uM.*: This column shows the concentration at which the activity was measured, usually given in micromolar (uM). For example, values range from 0.22 uM to 50.00 uM, with "IC50" indicating the concentration at which 50% inhibition is observed.
- *Activity.Name:* This column typically indicates the type of activity measured, in this case, "IC50" for inhibitory concentration.
- *Assay.Name:* This column provides a detailed description of the assay. For example, all entries describe a cell-based high-throughput screening (HTS) approach for discovering new inhibitors of Respiratory Syncytial Virus (RSV) using synthesized compounds.
- *Assay.Type:* This column indicates the type of assay, with all entries marked as "Confirmatory," suggesting these are follow-up tests to initial screenings.
- *PubMed.ID and RNAi:* These columns are included but contain "NA," indicating no specific PubMed reference or RNA interference information is provided for these entries.

The concise format of this dataset allows researchers to quickly assess the activity outcomes of various substances and compounds tested in the assay. For instance, the dataset shows that compounds with SIDs 103061373, 103904139, and others are active against RSV at specific concentrations, while others are inactive, providing valuable insights into potential therapeutic candidates.

**4. Gene and Protein Data Retrieval:** To retrieve gene and protein data, the `get_pug_rest` function can be employed to access detailed information about genes and proteins, which is crucial for genetic and molecular biology research. This can be done by retrieving concise bioactivity data for a specific gene or protein, using gene IDs or protein accession numbers.

Here is an example of how to retrieve concise bioactivity data for a specific gene:

```
geneData <- get_pug_rest(identifier = "13649",
                         namespace = "geneid",
                         domain = "gene",
                         operation = "concise",
                         output = "CSV")
geneData
```

```
#>
#>  An object of class 'PugRestInstance'
#>
#>  Request Details:
#>   - Domain: Gene
#>   - Namespace: DomainSpecific
```

```
#>   - Operation: concise
#>   - Identifier: 13649
#>
#> NOTE: Run getter function 'pubChemData(...)' to extract raw data retrieved from PubChem Database.
#>       See ?pubChemData for details.
```

The output for this function provides a dataframe with information on bioactivity data related to the gene ID "13649". This gene ID corresponds to the epidermal growth factor receptor (EGFR) gene in mice. This gene encodes a transmembrane glycoprotein that is a member of the protein kinase superfamily. The encoded protein is a receptor for members of the epidermal growth factor family. Mutations, amplifications, or misregulations of EGFR or family members are implicated in various cancers. (ATIF)

```
geneData_result <- pubChemData(geneData)
head(geneData_result)
```

```
#>     AID       SID       CID Activity.Outcome Target.Accession
#> 1 66438 103250953  25017867           Active           Q01279
#> 2 66438 103432098    454217           Active           Q01279
#> 3 69721 103358917 135512509      Unspecified           Q01279
#> 4 69721 103358918 135434086      Unspecified           Q01279
#> 5 69721 103358919 135455949      Unspecified           Q01279
#> 6 69722 103253186   5328592      Unspecified           Q01279
#>   Activity.Value..uM.          Activity.Name
#> 1                0.01 Effective concentration
#> 2                0.22 Effective concentration
#> 3               22.70                   IC50
#> 4               36.90                   IC50
#> 5               11.30                   IC50
#> 6              100.00                   IC50
#>                                                                                Assay.Name
#> 1                                Inhibition of epidermal growth factor binding in C3H10T1/2 cells
#> 2                                Inhibition of epidermal growth factor binding in C3H10T1/2 cells
#> 3                    Inhibition of Epidermal growth factor receptor mediated mitogenesis of NIH3T3 cells
#> 4                    Inhibition of Epidermal growth factor receptor mediated mitogenesis of NIH3T3 cells
#> 5                    Inhibition of Epidermal growth factor receptor mediated mitogenesis of NIH3T3 cells
#> 6 Inhibition of epidermal growth factor receptor (EGFR-mediated tyrosine autophosphorylation in mouse fibrobla
#>     Assay.Type PubMed.ID RNAi
#> 1 Confirmatory   1597853   NA
#> 2 Confirmatory   1597853   NA
#> 3 Confirmatory   9748366   NA
#> 4 Confirmatory   9748366   NA
#> 5 Confirmatory   9748366   NA
#> 6 Confirmatory   8027985   NA
```

This data provides researchers with detailed insights into the bioactivity of different substances tested in relation to a specific gene. The availability of target accession numbers and detailed assay descriptions makes it easier to understand the context and significance of each entry in the dataset. This information is crucial for advancing genetic and molecular biology research by understanding the effects of various substances on specific genes and proteins.

To retrieve concise bioactivity data for the specified protein with the accession number "Q01279" (which corresponds to the EGFR in mice), use the following R code:

```
protein_data <- get_pug_rest(identifier = "Q01279",
                             namespace = "accession",
                             domain = "protein",
                             operation = "concise",
                             output = "CSV")
protein_data
```

```
#>
#>  An object of class 'PugRestInstance'
#>
```

```
#>  Request Details:
#>   - Domain: Protein
#>   - Namespace: DomainSpecific
#>   - Operation: concise
#>   - Identifier: Q01279
#>
#> NOTE: Run getter function 'pubChemData(...)' to extract raw data retrieved from PubChem Database.
#>        See ?pubChemData for details.

protein_data_result <- pubChemData(protein_data)
head(protein_data_result)

#>       AID       SID       CID Activity.Outcome Target.GeneID
#> 1  415759 103294831   5289418           Active         13649
#> 2 1527521 312367127  71496458           Active         13649
#> 3 1527521 440227038 139593668     Inconclusive         13649
#> 4   69724 103399893  11349700           Active         13649
#> 5   69730 103379682  10500718           Active         13649
#> 6  106697 103166416    689033      Unspecified         13649
#>   Activity.Value..uM. Activity.Name
#> 1                  NA
#> 2               0.081          IC50
#> 3                  NA          IC50
#> 4               0.110          IC50
#> 5               1.000          IC50
#> 6              46.200          IC50
#>                                                                                                                                  Ass
#> 1                                    Inhibition of EGFR in mouse HER14 cells assessed as inhibition of EGF-stimu
#> 2 Inhibition of wild type EGFR in mouse BAF3 cells assessed as reduction in cell proliferation incubated for 72
#> 3 Inhibition of wild type EGFR in mouse BAF3 cells assessed as reduction in cell proliferation incubated for 72
#> 4                                                                        Inhibition of epidermal growth factor recepto
#> 5                                                  Inhibition of epidermal growth factor receptor phosphorylat
#> 6                                                                          Inhibition of EGF-dependent mouse keratino
#>     Assay.Type PubMed.ID
#> 1        Other   9139660
#> 2 Confirmatory  31689114
#> 3 Confirmatory  31689114
#> 4 Confirmatory  14640561
#> 5 Confirmatory  11462983
#> 6 Confirmatory  10090785
```

This data provides researchers with detailed insights into the bioactivity of different substances tested in relation to a specific protein. The availability of target gene IDs and assay information makes it easier to understand the context and significance of each entry in the dataset.

**5. Pathway Information:** The function offers access to detailed pathway information, essential for bioinformatics and molecular biology research. It provides a list of pathways involving a specific protein. For example, P00533 is the accession number for the human EGFR, which is a protein involved in the regulation of cell growth, survival, proliferation, and differentiation.

To retrieve pathway information for protein accession P00533:

```
pathway_data <- get_pug_rest(identifier = "P00533",
                             namespace = "accession",
                             domain = "protein",
                             operation = "pwaccs",
                             output = "JSON")
pathway_data

#>
#>  An object of class 'PugRestInstance'
#>
#>  Request Details:
#>   - Domain: Protein
#>   - Namespace: DomainSpecific
```

```
#>   - Operation: pwaccs
#>   - Identifier: P00533
#>
#> NOTE: Run getter function 'pubChemData(...)' to extract raw data retrieved from PubChem Database.
#>       See ?pubChemData for details.
```

The output provides a list of pathways:

```
pathway_data_result <- pubChemData(pathway_data)
head(pathway_data_result$InformationList$Information[[1]]$PathwayAccession)
```

```
#> [1] "PathBank:SMP0000472" "PathBank:SMP0000473" "PathBank:SMP0000474"
#> [4] "PathBank:SMP0000475" "PathBank:SMP0000476" "PathBank:SMP0063810"
```

This information provides a comprehensive list of pathways involving the specified protein, which is essential for understanding its biological roles and interactions. Each pathway entry includes the database source and the specific pathway identifier.

**6. Taxonomy Data:** The function enables researchers to access detailed taxonomy information, aiding in biological and environmental research. It provides summaries of taxonomy data for given taxonomic identifiers. For example, Taxonomy ID 2697049 corresponds to Severe acute respiratory syndrome coronavirus 2 (SARS-CoV-2):

```
taxonomy_data <- get_pug_rest(identifier = c("2697049"),
                              namespace = "taxid",
                              domain = "taxonomy",
                              operation = "summary",
                              output = "JSON")
taxonomy_data
```

```
#>
#>  An object of class 'PugRestInstance'
#>
#>  Request Details:
#>   - Domain: Taxonomy
#>   - Namespace: DomainSpecific
#>   - Operation: summary
#>   - Identifier: 2697049
#>
#> NOTE: Run getter function 'pubChemData(...)' to extract raw data retrieved from PubChem Database.
#>       See ?pubChemData for details.
```

The output provides detailed taxonomy summaries:

```
taxonomy_data_result <- pubChemData(taxonomy_data)
taxonomy_data_result$TaxonomySummaries$TaxonomySummary
```

```
#> [[1]]
#> [[1]]$TaxonomyID
#> [1] 2697049
#>
#> [[1]]$ScientificName
#> [1] "Severe acute respiratory syndrome coronavirus 2"
#>
#> [[1]]$CommonName
#> [1] ""
#>
#> [[1]]$Rank
#> [1] "no rank"
#>
#> [[1]]$RankedLineage
#>                                                 Species
#> "Severe acute respiratory syndrome-related coronavirus"
#>                                                   Genus
```

```
#>                                       "Betacoronavirus"
#>                                                  Family
#>                                     "Coronaviridae"
#>                                                   Order
#>                                        "Nidovirales"
#>                                                   Class
#>                                      "Pisoniviricetes"
#>                                                  Phylum
#>                                       "Pisuviricota"
#>                                                 Kingdom
#>                                     "Orthornavirae"
#>                                        Superkingdom
#>                                            "Viruses"
#>
#> [[1]]$Synonym
#> [1] "2019-nCoV"
#> [2] "COVID-19 virus"
#> [3] "HCoV-19"
#> [4] "Human coronavirus 2019"
#> [5] "SARS-2"
#> [6] "SARS2"
#> [7] "SARS-CoV2"
#> [8] "Severe acute respiratory syndrome coronavirus 2"
```

This information includes taxonomy ID, scientific name, common name, ranks, ranked lineages, and synonyms for the specified taxa. This detailed taxonomy data is essential for various research applications, including evolutionary studies and disease research.

**7. Cell Line Information:** The function is also useful in accessing detailed information about various cell lines, vital for cellular biology and pharmacology research. For example, CHEMBL3308376 corresponds to the HeLa cell line:

```
cell_line_data <- get_pug_rest(identifier = c("CHEMBL3308376"),
                               namespace = "cellacc",
                               domain = "cell",
                               operation = "summary",
                               output = "JSON")
cell_line_data
```

```
#>
#>  An object of class 'PugRestInstance'
#>
#>  Request Details:
#>   - Domain: Cell
#>   - Namespace: DomainSpecific
#>   - Operation: summary
#>   - Identifier: CHEMBL3308376
#>
#> NOTE: Run getter function 'pubChemData(...)' to extract raw data retrieved from PubChem Database.
#>       See ?pubChemData for details.
```

The output provides detailed cell line summaries:

```
cell_line_data_result <- pubChemData(cell_line_data)
cell_line_data_result$CellSummaries$CellSummary
```

```
#> [[1]]
#> [[1]]$CellAccession
#> [1] "CVCL_0030"
#>
#> [[1]]$Name
#> [1] "HeLa"
#>
#> [[1]]$Sex
```

```
#> [1] "Female"
#>
#> [[1]]$Category
#> [1] "Cancer cell line"
#>
#> [[1]]$SourceTissue
#> [1] "uterine cervix"
#>
#> [[1]]$SourceTaxonomyID
#> [1] 9606
#>
#> [[1]]$SourceOrganism
#> [1] "Homo sapiens (human)"
#>
#> [[1]]$Synonym
#> [1] "HELA"                  "Hela"                  "He La"
#> [4] "He-La"                 "HeLa-CCL2"             "Henrietta Lacks cells"
#> [7] "Helacyton gartleri"
```

This information includes cell accession number, name, sex, category, source tissue, source taxonomy ID, source organism, and synonyms for the specified cell lines.

### 4.8 Enhancing Chemical Data Access with PUG View Service

The `get_pug_view` function is designed to provide access to detailed summary reports and additional information not typically found in primary PubChem Substance, Compound, or BioAssay records. It utilizes the PUG View service, a REST-style web service of PubChem (Kim et al., 2019), to generate comprehensive reports for individual PubChem records (Figure 2). The primary aim of `get_pug_view` is to offer a different approach from the PUG REST service, focusing on delivering complete summary reports rather than smaller bits of information. This function supports various data formats and record types, making it a versatile tool for users needing comprehensive information from the PubChem database. Key aspects include:

**Flexible Data Retrieval:** Users can choose between obtaining an index (summary or table of contents) or full data retrieval, catering to both overview and detailed information requirements. This flexibility allows users to access just the right amount of data they need for their specific research purposes.

**Diverse Record Types:** The function is capable of accessing a wide range of records, including compounds, substances, bioassays, patents, genes, proteins, pathways, taxonomies, and cell lines. This broad capability ensures that users can retrieve comprehensive data across various scientific domains using their respective identifiers or names.

**Annotations and Detailed Information:** The `get_pug_view` function can retrieve specific types of information, such as experimental properties, safety and hazard labeling, and more. This feature is particularly valuable for users needing in-depth annotations and detailed descriptions across PubChem's extensive databases.

**Comprehensive Reports:** It provides detailed summaries that encompass chemical properties, biological activities, safety information, patents, and literature references. This comprehensive reporting is crucial for researchers who require a holistic view of PubChem records for their studies and analyses.

In summary, the `get_pug_view` function offers in-depth and comprehensive reports that facilitate advanced research and development activities. By leveraging the PUG View service, it enables efficient access to detailed and annotated data, enhancing the user's ability to make informed decisions based on extensive PubChem records.

The `get_pug_view` function finds its application in various scenarios, making it a crucial resource in chemical data analysis:

**1. Full Data Record:** For researchers requiring comprehensive data of compounds, substances, or bioassays, `get_pug_view` provides detailed reports including experimental properties, safety information, and more.

We will initialize the retrieval of comprehensive data for the compound with ID 2244 (Aspirin) from PubChem using the following code chunk:

```
full_record_2244 <- get_pug_view(annotation = "data",
```

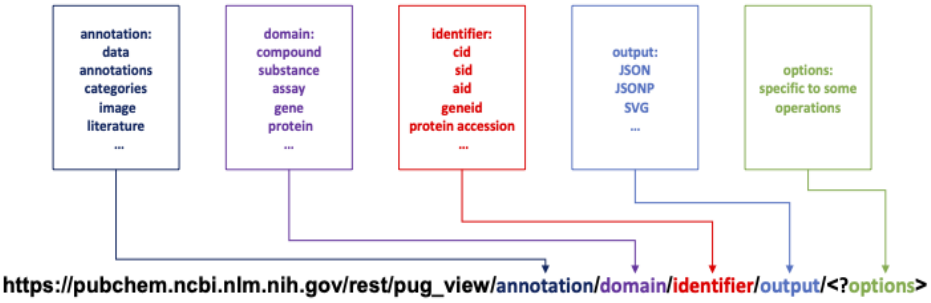


**Figure 2:** Using the PubChemR package to access PubChem's PUG-View database, with queries in URL syntax serving as function arguments

```
                                identifier = "2244",
                                domain = "compound",
                                output = "JSON")
full_record_2244

#>
#>  PUG View Data from PubChem Database
#>
#>  Request Details:
#>   - Domain: Compound
#>   - Annotation: data
#>   - Identifier: 2244
#>
#>  Pug View Details:
#>   - RecordType (1): [<unnamed character>]
#>   - RecordNumber (1): [<unnamed numeric>]
#>   - RecordTitle (1): [<unnamed character>]
#>   - Section (20): [<unnamed list>] Structures, Chemical Safety, ... and 18 more.
#>   - Reference (246): [<unnamed list>]
#>
#> NOTE: Run getter function 'retrieve()' with element name above to extract data from corresponding list.
#>       See ?retrieve for details.
```

The resulting object contains extensive information about the compound, including various sections and references.

We can extract the record type, record number, and record title to confirm that the record pertains to a chemical compound, to identify the specific compound ID for Aspirin, and to verify that Aspirin is the common name for the compound with ID 2244. Moreoveer, `retrieve` function can be used to show detailed information about the references, including reference number, source name, source ID, name, description, URL, and license URL. Each reference provides insights into various aspects related to the compound Aspirin, sourced from databases like the Australian Industrial Chemicals Introduction Scheme (AICIS), CAMEO Chemicals, CAS Common Chemistry, and others. These references include descriptions of the chemical properties, regulatory information, safety data, and links to the original sources for further details.

To access the specific sections within the retrieved data, we use the following code:

```
sections <- retrieve(object = full_record_2244, .slot = "Section")
sections

#>
#>  PUG View Data Sections
#>
#>  Request Details:
#>   - Record Type: CID
#>   - Record Number: 2244
#>   - Record Title: Aspirin
```

```
#>
#>  Section Details:
#>   - Number of available sections: 20
#>   - Section headings: Structures, Chemical Safety, ... and 18 more.
#>
#> NOTE: Run getter function 'section()' to extract section data. To list available sections, run 'sectionList()'
#>       See ?section and ?sectionList for details.
```

This code retrieves the sections available within the full record of Aspirin. The output indicates that there are 20 sections available, with headings such as Structures, Chemical Safety, and others.

Now, we can employ the `sectionList` function to display all sections available in the "sections" object:

```
sectionList(object = sections)
```

```
#> # A tibble: 20 x 2
#>    SectionID Headings
#>    <chr>     <chr>
#>  1 S1        Structures
#>  2 S2        Chemical Safety
#>  3 S3        Names and Identifiers
#>  4 S4        Chemical and Physical Properties
#>  5 S5        Spectral Information
#>  6 S6        Related Records
#>  7 S7        Chemical Vendors
#>  8 S8        Drug and Medication Information
#>  9 S9        Pharmacology and Biochemistry
#> 10 S10       Use and Manufacturing
#> 11 S11       Identification
#> 12 S12       Safety and Hazards
#> 13 S13       Toxicity
#> 14 S14       Associated Disorders and Diseases
#> 15 S15       Literature
#> 16 S16       Patents
#> 17 S17       Interactions and Pathways
#> 18 S18       Biological Test Results
#> 19 S19       Taxonomy
#> 20 S20       Classification
```

The "SectionID" column contains unique identifiers for each section, labeled S1 through S20. The "Headings" column provides descriptive titles for these sections, indicating the type of information contained within each.

The sections listed are:

**1. Structures:** Details on the structural information of Aspirin.

**2. Chemical Safety:** Information related to the safety measures and regulations for handling Aspirin.

**3. Names and Identifiers:** Various names and identifiers associated with Aspirin.

**4. Chemical and Physical Properties:** Data on the chemical and physical properties of Aspirin.

**5. Spectral Information:** Spectral data related to Aspirin.

**6. Related Records:** Records related to Aspirin.

**7. Chemical Vendors:** Information about vendors that supply Aspirin.

**8. Drug and Medication Information:** Details on the use of Aspirin as a drug or medication.

**9. Pharmacology and Biochemistry:** Information on the pharmacological and biochemical properties of Aspirin.

**10. Use and Manufacturing:** Data on the use and manufacturing processes of Aspirin.

**11. Identification:** Identification information for Aspirin.

**12. Safety and Hazards:** Safety hazards associated with Aspirin.

**13. Toxicity:** Toxicological information about Aspirin.

**14. Associated Disorders and Diseases:** Disorders and diseases associated with Aspirin.

**15. Literature:** References to literature involving Aspirin.

**16. Patents:** Patent information related to Aspirin.

**17. Interactions and Pathways:** Biological interactions and pathways involving Aspirin.

**18. Biological Test Results:** Results from biological tests conducted on Aspirin.

**19. Taxonomy:** Taxonomic information related to Aspirin.

**20. Classification:** Classification information for Aspirin.

These sections provide a comprehensive overview of various aspects of Aspirin. Now, we will focus on detailed data from the first section, "Structures," with the section ID "S1".

First, we assign the section to an object using the `section` function and then examine its contents:

```
s1 <- section(object = sections, .id = "S1")
```

The output provides an overview of the "Structures" section:

```
s1

#>
#>  PUG View Data Sections (Structures)
#>
#>  Request Details:
#>   - Record Type: CID
#>   - Record Number: 2244
#>   - Record Title: Aspirin
#>
#>  Section Details:
#>   - TOCHeading (1): [<unnamed character>]
#>   - Description (1): [<unnamed character>]
#>   - Section (3): [<unnamed list>] 2D Structure, 3D Conformer, ... and 1 more.
#>
#> NOTE: Run getter function 'retrieve()' with element name above to extract data from corresponding list.
#>       See ?retrieve for details.
#>
#> NOTE: Run getter function 'section()' to extract section data. To list available sections, run 'sectionList()'
#>       See ?section and ?sectionList for details.
```

Next, we list the sub-sections within the "Structures" section. The output shows the available sub-sections:

```
sectionList(object = s1)

#> # A tibble: 3 x 2
#>   SectionID Headings
#>   <chr>     <chr>
#> 1 S1        2D Structure
#> 2 S2        3D Conformer
#> 3 S3        Crystal Structures
```

This breakdown allows us to see that the "Structures" section contains detailed depictions of Aspirin, including 2D structures, 3D conformers, and crystal structures. Each sub-section can be further explored to gain more specific information about the molecular structure of Aspirin.

**2. Accessing Specific Headings:** Users can retrieve data under specific headings for targeted information, such as boiling points or viscosity measurements.

First, we initiate the retrieval of data for a specific heading using the `get_pug_view` function, focusing on the "Boiling Point" heading within the "heading" domain.

```
specific_headings <- get_pug_view(annotation = "annotations",
                                  identifier = "Boiling Point",
                                  domain = "heading",
                                  output = "JSON",
                                  headingType = "Compound")

specific_headings
```

```
#>
#>  PUG View Data from PubChem Database
#>
#>  Request Details:
#>   - Domain: DomainSpecific (heading)
#>   - Annotation: annotations
#>   - Identifier: Boiling%20Point
#>
#>  Pug View Details:
#>   - Annotation (1000): [<unnamed list>]
#>   - Page (1): [<unnamed numeric>]
#>   - TotalPages (1): [<unnamed numeric>]
#>
#> NOTE: Run getter function 'retrieve()' with element name above to extract data from corresponding list.
#>       See ?retrieve for details.
```

The function call returns data from the PubChem database, specifying details about the request, including the domain, annotation type, identifier, and the number of annotations retrieved. It highlights that 1000 annotations were fetched, spread across multiple pages.

Next, we utilize the `retrieve` function to extract the "Annotation" slot from the *specific_headings* object. By displaying the annotations, we gain insight into the structure and content of the data. Each annotation includes the source name, source ID, compound name, description, URLs, and detailed data related to the boiling point, complete with references and specific values.

```
annotation <- retrieve(specific_headings, .slot = "Annotation", .to.data.frame = FALSE)
annotation[[1]]
```

```
#> $SourceName
#> [1] "Hazardous Substances Data Bank (HSDB)"
#>
#> $SourceID
#> [1] "30"
#>
#> $Name
#> [1] "NITROGLYCERIN"
#>
#> $Description
#> [1] "The Hazardous Substances Data Bank (HSDB) is a toxicology database that focuses on the toxicology of potent
#>
#> $URL
#> [1] "https://pubchem.ncbi.nlm.nih.gov/source/hsdb/30"
#>
#> $LicenseURL
#> [1] "https://www.nlm.nih.gov/web_policies.html"
#>
#> $Data
#> $Data[[1]]
#> $Data[[1]]$TOCHeading
#>            type     #TOCHeading
#>      "Compound" "Boiling Point"
#>
#> $Data[[1]]$Description
#> [1] "PEER REVIEWED"
#>
#> $Data[[1]]$Reference
#> [1] "O'Neil, M.J. (ed.). The Merck Index - An Encyclopedia of  Chemicals, Drugs, and Biologicals. 13th Edition,
#>
#> $Data[[1]]$ExtendedReference
#> $Data[[1]]$ExtendedReference[[1]]
#> $Data[[1]]$ExtendedReference[[1]]$Citation
#> [1] "O'Neil, M.J. (ed.). The Merck Index - An Encyclopedia of  Chemicals, Drugs, and Biologicals. 13th Edition,
#>
#> $Data[[1]]$ExtendedReference[[1]]$Matched
#>     PCLID
```

```
#> 900133450
#>
#>
#>
#> $Data[[1]]$Value
#> $Data[[1]]$Value$StringWithMarkup
#> $Data[[1]]$Value$StringWithMarkup[[1]]
#>               String
#> "Explodes at 218 °C"
#>
#>
#>
#>
#> $Data[[2]]
#> $Data[[2]]$TOCHeading
#>            type     #TOCHeading
#>      "Compound" "Boiling Point"
#>
#> $Data[[2]]$Description
#> [1] "PEER REVIEWED"
#>
#> $Data[[2]]$Reference
#> [1] "Weast, R.C. (ed.) Handbook of Chemistry and Physics. 69th ed. Boca Raton, FL: CRC Press Inc., 1988-1989., p
#>
#> $Data[[2]]$ExtendedReference
#> $Data[[2]]$ExtendedReference[[1]]
#> $Data[[2]]$ExtendedReference[[1]]$Citation
#> [1] "Weast, R.C. (ed.) Handbook of Chemistry and Physics. 69th ed. Boca Raton, FL: CRC Press Inc., 1988-1989., p
#>
#> $Data[[2]]$ExtendedReference[[1]]$Matched
#>     PCLID
#> 900017161
#>
#>
#>
#> $Data[[2]]$Value
#> $Data[[2]]$Value$StringWithMarkup
#> $Data[[2]]$Value$StringWithMarkup[[1]]
#>                              String
#> "BOILING POINT: 125 °C @ 2 MM HG"
#>
#>
#>
#>
#>
#> $ANID
#> [1] 2
#>
#> $LinkedRecords
#> $LinkedRecords$CID
#> [1] 4510
```

This step reveals detailed information about various compounds, including their boiling points. For instance, it might display data such as "NITROGLYCERIN" from the "Hazardous Substances Data Bank (HSDB)" with specific boiling point details like "Explodes at 218 °C."

**3. Literature and Publication Data:** The function can be used to retrieve literature associated with a compound, aiding in academic research and publication review.

In the given example, we are querying the PubChem database for literature information related to the compound with identifier "1234" in the compound domain. The identifier "1234" corresponds to the compound Gallopamil. Gallopamil is a pharmaceutical compound used primarily as a calcium channel blocker, which is useful in treating cardiovascular conditions such as angina pectoris and hypertension.

By running the `get_pug_view` function with the identifier "1234", we retrieve various details about

Gallopamil, including references to related literature.

```
literature <- get_pug_view(annotation = "literature",
                           identifier = "1234",
                           domain = "compound",
                           output = "JSON")
literature
```

```
#>
#>  PUG View Data from PubChem Database
#>
#>  Request Details:
#>   - Domain: Compound
#>   - Annotation: literature
#>   - Identifier: 1234
#>
#>  Pug View Details:
#>   - RecordType (1): [<unnamed character>]
#>   - RecordNumber (1): [<unnamed numeric>]
#>   - AllURL (1): [<unnamed character>]
#>   - Subheadings (15): [<unnamed list>]
#>
#> NOTE: Run getter function 'retrieve()' with element name above to extract data from corresponding list.
#>       See ?retrieve for details.
```

The following code provides a URL, which directs to the PubMed search page for the specified compound. This link is valuable for researchers seeking detailed literature information related to the compound identified by CID 1234.

```
retrieve(literature, .slot = "AllURL")
```

```
#> # A tibble: 1 x 1
#>   value
#>   <chr>
#> 1 https://www.ncbi.nlm.nih.gov/sites/entrez?cmd=search&db=pubmed&term=%22Gallop~
```

**4. 3D Protein Structures:** The `get_pug_view` function allows access to detailed 3D protein structure information associated with specific compounds. These 3D structures provide critical insights into the molecular interactions, mechanisms of action, and potential binding sites of compounds, which are essential for understanding their biological effects. Access to such detailed structural data aids in drug design, understanding enzyme mechanisms, and studying protein-ligand interactions. The visual representation of these structures, along with associated metadata like MMDB (Molecular Modeling Database) and PDB (Protein Data Bank) IDs, URLs for accessing detailed pages, and descriptions of the structures, further enhances the utility of this function in scientific research and development.

The following code retrieves a list of 3D protein structures associated with Aspirin (ID:2244):

```
list_3d_proteins <- get_pug_view(annotation = "structure",
                                 identifier = "2244",
                                 domain = "compound",
                                 output = "JSON")
list_3d_proteins
```

```
#>
#>  PUG View Data from PubChem Database
#>
#>  Request Details:
#>   - Domain: Compound
#>   - Annotation: structure
#>   - Identifier: 2244
#>
#>  Pug View Details:
#>   - RecordType (1): [<unnamed character>]
#>   - RecordNumber (1): [<unnamed numeric>]
```

```
#>   - URL (1): [<unnamed character>]
#>   - NumberOfStructures (1): [<unnamed numeric>]
#>   - Structures (8): [<unnamed list>]
#>
#> NOTE: Run getter function 'retrieve()' with element name above to extract data from corresponding list.
#>       See ?retrieve for details.
```

This code fetches detailed information about the 3D protein structures related to Aspirin. The output includes several components.

First, the URL for the data is retrieved:

```
retrieve(list_3d_proteins, .slot = "URL")
```

```
#> # A tibble: 1 x 1
#>   value
#>   <chr>
#> 1 https://www.ncbi.nlm.nih.gov/sites/entrez?LinkName=pccompound_structure&db=pc~
```

This output provides the URL to the PubChem page that contains detailed information about the 3D structures of Aspirin. This URL can be visited to explore further details visually and interactively.

Next, the number of 3D structures available is retrieved:

```
retrieve(list_3d_proteins, .slot = "NumberOfStructures")
```

```
#> # A tibble: 1 x 1
#>   value
#>   <dbl>
#> 1     8
```

The output indicates that there are 8 different 3D structures available for Aspirin.

Finally, details of each structure are retrieved:

```
list_3d_proteins_structures <- retrieve(list_3d_proteins, .slot = "Structures", .to.data.frame = FALSE)
```

This code outputs a list of details for each 3D structure, including the MMDB ID, PDB ID, URLs for accessing and visualizing the structures, descriptions, and taxonomic information. For example, one of the structures is described as "Cryo-EM structure of aspirin-bound ABCC4," with the MMDB ID 230639 and PDB ID "8J3W." The structure is associated with Homo sapiens (human) as indicated by the taxonomy information.

```
list_3d_proteins_structures[[1]]
```

```
#> $MMDB_ID
#> [1] 230639
#>
#> $PDB_ID
#> [1] "8J3W"
#>
#> $URL
#> [1] "https://www.ncbi.nlm.nih.gov/Structure/mmdb/mmdbsrv.cgi?uid=230639"
#>
#> $ImageURL
#> [1] "https://www.ncbi.nlm.nih.gov/Structure/mmdb/mmdbimage.fcgi?small=t&id=230639"
#>
#> $Description
#> [1] "Cryo-EM structure of aspirin-bound ABCC4"
#>
#> $Taxonomy
#> $Taxonomy$ID
#> [1] 9606
#>
#> $Taxonomy$Name
#> [1] "Homo sapiens"
```

**5. NCBI LinkOut Records:** The `get_pug_view` function is capable of listing all LinkOut records for substances, compounds, or assays, which is beneficial for tracking external resources and databases linked to specific chemical entities. This functionality is especially useful for researchers who need to access a wide array of related data from different external sources.

Here is an example of how the LinkOut records can be retrieved for identifier "1234":

```
ncbi_linkouts <- get_pug_view(annotation = "linkout",
                              identifier = "1234",
                              domain = "compound",
                              output = "JSON")
```

Next, the retrieved data can then be accessed as follows:

```
retrieve(ncbi_linkouts, .slot = "ObjUrl", .to.data.frame = FALSE)
```

```
#> $Url
#> [1] "http://partnersolution.ingenuity.com/?cid=97ae3f91eab87a&p1=EntrezPubChem&p2=GV&s=&ipaUri=%2Fpa%2Fapi
#>
#> $SubjectType
#> [1] "molecular interactions"
#>
#> $Category
#> [1] "Chemical Information"
#>
#> $Attribute
#> [1] "subscription/membership/fee required"
#>
#> $Provider
#>                            Name                       NameAbbr
#> "Ingenuity Pathways Analysis"                     "Ingenuity"
#>                              Id                             Url
#>                          "5628"    "http://www.ingenuity.com"
```

The extracted data includes information such as the URL of the external resource, the subject type, the category of information, attributes indicating if a subscription or fee is required, and the provider's details. For instance, the URL "http://partnersolution.ingenuity.com/?cid=97ae3f91eab87a&p1=EntrezPubChem&p2=GV&s=&ipaUri=%2Fpa%2Fapi%2Fv2%2Fgeneview%3Fapplicationname%3DEntrezPubChem%26geneId%3DING:qkb%26geneidtype%3Dingenuity" points to a resource provided by Ingenuity Pathways Analysis, categorized under "Chemical Information" and related to "molecular interactions". This URL requires a subscription or membership for access.

Such detailed LinkOut records facilitate the exploration of interconnected data across various platforms, enabling researchers to efficiently gather comprehensive information related to their chemical entities of interest.

## 5 Discussion

### 5.1 Related Packages

Several R packages provide access to chemical data and tools for cheminformatics, each with its unique focus and capabilities:

- **ChemmineR:** A comprehensive cheminformatics toolkit for R, offering functionalities for compound data processing and analysis. It includes tools for compound classification, similarity searching, and structure-activity relationship modeling.
- **webchem:** Designed for retrieving chemical information from various web sources, this package facilitates automated queries and integrates data into R objects for further analysis, focusing on structured data retrieval and usage.
- **rcdk:** An R interface to the CDK, allowing users to manipulate and analyze chemical data. This package supports molecular structure parsing, descriptor calculation, and fingerprint generation, making it invaluable for computational chemistry, drug discovery, and bioinformatics research.

- **ChemmineOB:** An interface with OpenBabel for chemical format conversions and molecular property calculations, enhancing cheminformatics workflows by offering easy access to OpenBabel's powerful functions directly from R.
- **BridgeDbR:** Provides access to the BridgeDb framework, facilitating identifier mapping across different biological databases and supporting a wide range of identifier types, which is particularly useful in systems biology, genomics, and metabolomics studies.
- **RMassBank:** Tailored for the creation and handling of mass spectrometry databases, providing tools for building, querying, and managing MassBank records essential for compound identification and annotation in MS experiments.
- **rgoslin:** An R package for the systematic annotation of lipid species using the GOSlin format, supporting the conversion of lipid names to a structured format and ensuring consistency and accuracy in lipidomics studies.

**PubChemR**, in comparison, is designed specifically to interface with the PubChem database, providing a focused approach for accessing chemical data within this database through R. While **ChemmineR** and **webchem** offer broad functionalities and access to multiple sources, **PubChemR** specializes in efficient and targeted data interactions with PubChem, making it particularly suitable for users who predominantly rely on PubChem for their chemical data needs.

In the Python ecosystem, libraries such as **PubChempy** (Swain, 2017), **ChemSpiPy** (Swain, 2018), and **CIRpy** (Swain, 2016) offer functionalities similar to those described here. PubChemPy, like **PubChemR**, provides a direct interface with the PubChem database for accessing chemical molecules and their properties, supporting various chemical searches, standardization, format conversions, depiction, and property retrieval. ChemSpiPy offers easy access to the ChemSpider web service, enabling chemical searches, downloads, and property retrieval. CIRpy interfaces with the Chemical Identifier Resolver (CIR) by the Computer-Aided Drug Design (CADD) Group at the National Cancer Institute (NCI), simplifying the conversion of chemical identifiers and calculation of properties, along with supporting diverse file format downloads.

### 5.2 Usage Policy of PUG REST Service

Please note that PUG REST is not intended for handling very large volumes of requests, such as those numbering in the millions. To prevent overloading the PubChem servers, it is requested that any script or application limit the request rate to no more than five requests per second. This measure helps ensure the stability and reliability of the service for all users. For more information on request volume limitations and automated rate limiting (throttling), please refer to PubChem's dynamic request throttling documentation (`https://pubchem.ncbi.nlm.nih.gov/docs/dynamic-request-throttling`).

In some cases, a 503 HTTP status code may be returned when the server is temporarily unable to service the request due to maintenance downtime or capacity issues. If this occurs, it is advisable to try the request again later. This status code indicates that the server is currently overloaded or undergoing maintenance, and retrying the request after some time should allow it to be processed successfully.

### 5.3 Furter Research

The primary contribution of **PubChemR** is its facilitation of direct access to PubChem's chemical data through the R programming environment. This functionality addresses a specific need for researchers who rely on R for data analysis and representation, allowing for more straightforward integration of chemical data into their workflows However, the effectiveness of **PubChemR** is closely tied to the quality and comprehensiveness of the PubChem database. Inaccuracies or gaps in the PubChem data directly impact the outputs of **PubChemR**, which is an important consideration for users relying on this tool for research or data analysis. As PubChem's database is dynamic and continually updated, keeping **PubChemR** synchronized with these updates is critical for maintaining its accuracy and relevance.

While **PubChemR** currently focuses on accessing PubChem data and does not include data analysis functions, future enhancements could potentially explore these areas. For instance, more sophisticated data analysis capabilities within the package itself could be developed, enabling users to perform preliminary analysis within the same framework. Another possible direction could be the integration with other chemical databases, expanding the range of accessible data. Additionally, improving the user interface for greater ease of use and better data visualization capabilities could make the tool more accessible to a wider range of users with varying levels of expertise in R programming. These are potential directions for future research and development, although there are no immediate plans to implement these features.

## 6 Summary

**PubChemR** represents a significant advancement in accessing chemical data through the R programming environment. It offers a straightforward, effective, and easy-to-use way to access information from the PubChem database, improving how users can get chemical data. **PubChemR** combines practical features with user-friendly design, making it a useful tool for researchers in various scientific areas. As it continues to receive updates and enhancements, **PubChemR** will keep up with changes in chemical data and computational technology.

## References


Y. Cao, A. Charisi, L.-C. Cheng, T. Jiang, and T. Girke. Chemminer: a compound mining framework for r. *Bioinformatics*, 24(15):1733–1734, 2008. [p150]

B. Chen, D. Wild, and R. Guha. Pubchem as a source of polypharmacology. *Journal of chemical information and modeling*, 49(9):2044–2055, 2009. [p150]

R. Guha. Chemical informatics functionality in r. *Journal of Statistical Software*, 18:1–16, 2007. [p150]

K. Horan and T. Girke. *ChemmineOB: R interface to a subset of OpenBabel functionalities*, 2024. URL https://www.bioconductor.org/packages/release/bioc/html/ChemmineOB.html. R package version 1.42.0. [p150]

S. Kim, P. A. Thiessen, E. E. Bolton, and S. H. Bryant. Pug-soap and pug-rest: web services for programmatic access to chemical information in pubchem. *Nucleic acids research*, 43(W1):W605–W611, 2015. [p165]

S. Kim, P. A. Thiessen, E. E. Bolton, J. Chen, G. Fu, A. Gindulyte, L. Han, J. He, S. He, B. A. Shoemaker, et al. Pubchem substance and compound databases. *Nucleic acids research*, 44(D1):D1202–D1213, 2016. [p150]

S. Kim, P. A. Thiessen, T. Cheng, B. Yu, and E. E. Bolton. An update on pug-rest: Restful interface for programmatic access to pubchem. *Nucleic Acids Research*, 46(W1):W563–W570, 2018. [p165]

S. Kim, P. A. Thiessen, T. Cheng, J. Zhang, A. Gindulyte, and E. E. Bolton. Pug-view: programmatic access to chemical annotations integrated in pubchem. *Journal of cheminformatics*, 11(1):1–11, 2019. [p175]

D. Kopczynski, N. Hoffmann, B. Peng, and R. Ahrends. Goslin: a grammar of succinct lipid nomenclature. *Analytical Chemistry*, 92(16):10957–10960, 2020. [p150]

S. Korkmaz. Deep learning-based imbalanced data classification for drug discovery. *Journal of chemical information and modeling*, 60(9):4180–4190, 2020. [p159]

S. Korkmaz, B. E. Yamasan, and D. Goksuluk. *PubChemR: Interface to the 'PubChem' Database for Chemical Data Retrieval*, 2024. URL https://CRAN.R-project.org/package=PubChemR. R package version 2.0. [p150]

C. Leemans, E. Willighagen, A. Bohler, and L. Eijssen. *BridgeDbR: Code for Using BridgeDb Identifier Mapping Framework From Within R*, 2024. URL https://www.bioconductor.org/packages/release/bioc/html/BridgeDbR.html. R package version 2.14.0. [p150]

Q. Li, T. Cheng, Y. Wang, and S. H. Bryant. Pubchem as a public resource for drug discovery. *Drug discovery today*, 15(23-24):1052–1057, 2010. [p150]

M. A. Stravs, E. L. Schymanski, H. P. Singer, and J. Hollender. Automatic recalibration and processing of tandem mass spectra using formula annotation. *Journal of Mass Spectrometry*, 48(1):89–99, 2013. [p150]

M. Swain. *CIRpy: Python wrapper for the NCI Chemical Identifier Resolver*, 2016. URL https://github.com/mcs07/CIRpy. [p184]

M. Swain. *PubChemPy: Python wrapper for the PubChem PUG REST API*, 2017. URL https://github.com/mcs07/PubChemPy. [p184]

M. Swain. *ChemSpiPy-A Python wrapper for the ChemSpider API*, 2018. URL https://github.com/mcs07/ChemSpiPy. [p184]

E. Szöcs, T. Stirling, E. R. Scott, A. Scharmüller, and R. B. Schäfer. webchem: an r package to retrieve chemical information from the web. *Journal of Statistical Software*, 93:1–17, 2020. [p150]

Y. Wang, J. Xiao, T. O. Suzek, J. Zhang, J. Wang, and S. H. Bryant. Pubchem: a public information system for analyzing bioactivities of small molecules. *Nucleic acids research*, 37(suppl_2):W623–W633, 2009. [p150]

Y. Wang, J. Xiao, T. O. Suzek, J. Zhang, J. Wang, Z. Zhou, L. Han, K. Karapetyan, S. Dracheva, B. A. Shoemaker, et al. Pubchem's bioassay database. *Nucleic acids research*, 40(D1):D400–D412, 2012. [p150]

B. E. Yamasan and S. Korkmaz. Binding activity classification of anti-sars-cov-2 molecules using deep learning across multiple assays. *Balkan Medical Journal*, 41(3):186, 2024. [p159]

*Selcuk Korkmaz*
*Trakya University*
*Department Biostatistics*
*Edirne, Türkiye*
*ORCiD: 0000-0003-4632-6850*
selcukkorkmaz@trakya.edu.tr

*Bilge Eren Yamasan*
*Trakya University*
*Department Biophysics*
*Edirne, Türkiye*
*ORCiD: 0000-0002-6525-2503*
berenyamasan@trakya.edu.tr

*Dincer Goksuluk*
*Erciyes University*
*Department Biostatistics*
*Kayseri, Türkiye*
*ORCiD: 0000-0002-2752-7668*
dincergoksuluk@erciyes.edu.tr

# boiwsa: An R Package for Seasonal Adjustment of Weekly Data

*by Tim Ginker*

**Abstract** This article introduces the R package boiwsa for the seasonal adjustment of weekly data based on the discounted least squares method. It provides a user-friendly interface for computing seasonally adjusted estimates of weekly data and includes functions for creation of country-specific prior adjustment variables, as well as diagnostic tools to assess the quality of the adjustments. The utility of the package is demonstrated through two case studies: one based on US data of gasoline production characterized by a strong trend-cycle and dominant intra-yearly seasonality, and the other based on Israeli data of initial unemployment claims with two seasonal cycles (intra-yearly and intra-monthly) and the impact of two moving holidays.

## 1 Introduction

Policymakers and industry practitioners have long utilized weekly and other high-frequency data for timely updates on the state of various sectors of economic activity. Seasonal adjustment is a crucial component of this analysis as it facilitates the economic interpretation of data variations by allowing researchers to analyze these changes separately from the periodic fluctuations introduced by seasonal factors.

This article presents the R package **boiwsa** for seasonal adjustment of weekly data based on the discounted least squares regression (DWR) introduced by Harrison and Johnston (1984). It provides a user-friendly interface for computing seasonally adjusted estimates of weekly data and includes functions for the creation of country-specific prior adjustment variables, as well as diagnostic tools to assess the quality of the adjustments. This equips practitioners with the ability to perform seasonal adjustments on their weekly data, thereby facilitating informed decision-making across a diverse array of applications.

Although attempts to develop procedures for the seasonal adjustment of weekly data date back to the early 20th century (Crum, 1927), the volume of literature addressing this challenge remains relatively limited (for the latest review of the available methodology, see Proietti and Pedregal, 2023). Moreover, while there has been notable growth in the number of high-frequency indicators requiring seasonal adjustment, the associated open-source software is less advanced and more intricate than the easily accessible and user-friendly tools commonly available for monthly and quarterly data (see Evans et al., 2021; Hyndman and Killick, 2024).

The seasonal adjustment of high-frequency data presents multiple challenges due to its unique characteristics, which limit the straightforward application of conventional statistical methodologies. Most seasonal adjustment approaches generally encompass two primary steps: detrending and smoothing the cycle sub-series to compute the seasonal components. For instance, with monthly data, an initial application of a wide filter calculates and eliminates the trend. Subsequently, the detrended values are independently smoothed for each month, starting from January and progressing sequentially, to derive the seasonal components.

However, implementing a similar procedure on high-frequency data is not always feasible due to its potential for varying periodicity, presence of multiple seasonal cycles, and the moving window problem. For example, the number of weeks in a year varies between 52 and 53. Additionally, weekly data may exhibit multiple cycles, such as intra-monthly and intra-yearly patterns. As a result, commonly used methods like `X-13 ARIMA-SEATS`, which work well for data with a single and fixed seasonal period, cannot be directly applied on weekly data.

To illustrate the challenge associated with the moving window problem, we constructed an artificial unobserved daily series underlying the observed weekly data. This synthetic daily data exhibits a simple monthly seasonal pattern characterized by a linear decrease from 100 to 0 over the course of a month. Consider the first six months of 2023, as depicted in Figure 1. The black line represents the daily data, while, without loss of generality, each blue window corresponds to the first week of a month. Additionally, the green triangles denote the observed weekly averages derived from the black line. It becomes evident that each week aligns with a different portion of the intra-monthly cycle. As a result, even though the underlying daily seasonal pattern is constant, weekly aggregation produces a complex and evolving seasonality. Consequently, in this case, any method assuming a constant number of periods in a seasonal cycle and estimating a seasonal index for each would yield biased estimates.

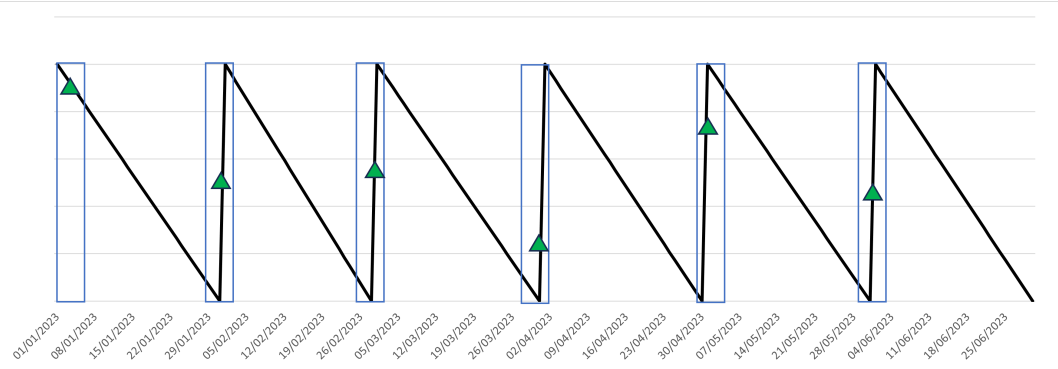


**Figure 1:** Illustration of the moving window challenge.

Currently, a number of software tools for the seasonal adjustment of weekly data are in various stages of development. One such program is the `MoveReg` weekly seasonal adjustment method developed by the U.S. Bureau of Labor. This program can be executed using EViews or SAS, and its methodology is well documented and straightforward to implement. However, it is not available as open-source code. There are also some promising R packages currently under development, with the most advanced being `Ecce Signum` (McElroy and Livsey, 2022)[1]. This package offers a method (in the Business Formation Statistics illustration) for the seasonal adjustment of weekly data that is based on the fractional airline model (FAM). However, it is worth mentioning a number of limitations associated with the aforementioned implementation of FAM. First, it implicitly assumes that the data must be differenced to eliminate a trend, which may not necessarily be present in the data. Second, similar to `MoveReg`, it is tailored to handle solely the intra-yearly cycle, a characteristic that could potentially restrict its applicability to data characterized by a substantial intra-monthly cycle, as presented in the application to the weekly unemployment claims in Israel.

Other software programs, occasionally employed for the seasonal adjustment of weekly data, rely on the Seasonal-Trend decomposition using Loess (STL), developed by Cleveland et al. (1990). A recent extension by Bandara et al. (2024) introduced the concept of Multiple Seasonal-Trend decomposition using Loess (MSTL), thereby extending its application to time series featuring multiple seasonal cycles. It's important to note that both STL and MSTL decompositions are fundamentally based on the cycle subseries smoothing. They assume a constant number of seasonal factors in each cycle, and thus are unable to address the moving window problem discussed earlier. Additionally, these methods lack the capability to effectively account for trading day and moving holiday effects.

Finally, several studies suggest the utilization of forecasting models for the decomposition of time series, such as Prophet (Taylor and Letham, 2018) or TBATS (De Livera et al., 2011). However, these methods have been observed to yield less accurate decompositions (Bandara et al., 2024).

The paper is organized as follows. Section 2 outlines our methodology. In Section 3, we provide examples of how to use the **boiwsa** package, which is illustrated through two cases. The first example is based on US gasoline production data, which exhibits a strong trend-cycle and dominant intra-yearly seasonality. The second example is based on Israeli data of initial unemployment claims, where the level is relatively stable, but there are two seasonal cycles - intra-yearly and intra-monthly - and a pronounced effect of two moving holidays. 4 concludes.

## 2 Methodology

In this section, we describe our methodology. Our approach aligns closely with the locally-weighted least squares procedure introduced by Cleveland et al. (2014) that is implemented in the `MoveReg` program, albeit with several adjustments. First, in order to simplify the adjustment process, and facilitate automation, instead of first-differencing, we opt to extract the trend component using Friedman's SuperSmoother, implemented through the `stats::supsmu()` function (R Core Team, 2024). Second, we incorporate a variation of DWR to enable the seasonal component to evolve dynamically over time. Unlike the weight structure proposed by Cleveland et al. (2014), our DWR implementation involves a single hyperparameter controlling the weight decay rate. In addition, DWR can be modified so that a different discount factor is applied to each model parameter. Although it is reasonable for weekly data to use the same discount factor for all parameters, future applications on higher frequency data could benefit from allowing each seasonal component to evolve at a different speed.

[1]The package is available on GitHub. See https://github.com/tuckermcelroy/sigex

We consider the following decomposition model of the observed series $y_t$:

$$y_t = T_t + S_t + H_t + O_t + I_t, \tag{1}$$

where $T_t$ represents the trend component, $S_t$ the seasonal component, $H_t$ the holiday and trading-day effects, $O_t$ and $I_t$ the outlier and irregular components respectively, and $t$ denotes the date of the last day within a given week.

The seasonal component is specified using trigonometric variables as follows

$$\begin{aligned} S_t &= \sum_{k=1}^{K} \left( \alpha_k^y \sin(\frac{2\pi k D_t^y}{n_t^y}) + \beta_k^y \cos(\frac{2\pi k D_t^y}{n_t^y}) \right) + \\ &\quad \sum_{l=1}^{L} \left( \alpha_l^m \sin(\frac{2\pi l D_t^m}{n_t^m}) + \beta_l^m \cos(\frac{2\pi l D_t^m}{n_t^m}) \right), \end{aligned} \tag{2}$$

where $K$ and $L$ define the number of yearly and monthly pairs of trigonometric variables, $D_t^y$ and $D_t^m$ are the day of the year and the day of the month, and $n_t^y$ and $n_t^m$ are the number of days in the given year or month[2] (Pierce et al., 1984). Thus, the seasonal adjustment procedure takes into account the existence of two cycles, namely intra-yearly and intra-monthly.

Similarly to the `X-11` method (Ladiray and Quenneville, 2001), our procedure employs an iterative approach to estimate the different components. The seasonal adjustment algorithm comprises eight steps, which are detailed below:

- Step 1: Estimation of trend ($T_t^{(1)}$) using `stats::supsmu()`.
- Step 2: Estimation of the Seasonal-Irregular component:

$$y_t - T_t^{(1)} = S_t + H_t + O_t + I_t.$$

- Step 2* (Optional): Searching for additive outliers using the method proposed by Findley et al. (1998).
- Step 2** (Optional): Identifying the optimal number of trigonometric variables in (2).
- Step 3: Calculation of seasonal factors, along with other potential factors such as $H_t$ or $O_t$, is done through DWR on the seasonal-irregular component extracted in Step 2. In this application, the discounting rate decays over the years. For each year $t$ and the observed year $\tau$, a geometrically decaying weight function is represented as: $w_t = r^{|t-\tau|}$, where $r \in (0, 1]$. Several important points are worth mentioning. First, when $r = 1$, the method simplifies to ordinary least squares regression with constant seasonality. On the contrary, smaller values of $r$ permit a more rapid rate of change in the seasonal component. However, it is advised against setting it below 0.5 to prevent overfitting. In addition, the choice of $r$ affects the strength of revisions in the seasonally adjusted data, with higher values of $r$ leading to potentially stronger revisions. Second, our methodology differs from the conventional one-way discounting, enabling the inclusion of future observations in the computation of seasonal factors. This approach circumvents the limitations of the forecasting methods discussed in Bandara et al. (2024). Finally, the choice of year-based discounting is driven by the fact that in traditional discount-weighted regression, even with a conservative choice of $r = 0.95$, in weekly data, observations separated by more than 2 years would carry nearly negligible weight. Therefore, the use of year-based discounting prevents an overly rapid decay which may potentially lead to unstable estimates of the seasonal component.
- Step 4: Estimation of the trend ($T_t^{(2)}$) from the seasonally and outlier adjusted series using `stats::supsmu()`.
- Step 5: Estimation of the Seasonal-Irregular component:

$$y_t - T_t^{(2)} = S_t + H_t + O_t + I_t$$

- Step 6: Computing the final seasonal factors (and possibly other factors such as $H_t$ or $O_t$) using DWR, as in step 3.
- Step 7: Estimation of the final seasonally adjusted series:

$$y_t - S_t - H_t$$

[2]Note that $n_t^y$ is either 365 or 366, depending on leap year, and that $n_t^m$ is either 28, 29, 30, or 31.

- Step 8: Computing the final trend ($T_t^{(3)}$) estimate from the seasonally and outlier adjusted series using `stats::supsmu()`.

## 3 Use of boiwsa

In this section, we illustrate the use of our package and its key functionalities through two examples[3]. The package is available on GitHub[4], and via the Comprehensive R Archive Network (CRAN)[5]. The first example is based on the data with a strong trend-cycle and dominant intra-yearly seasonality. The second example is based on the data where the level is relatively stable, but there are two seasonal cycles - intra-yearly and intra-monthly - and a pronounced effect of two moving holidays.

### 3.1 Example 1: Gasoline production in the US

Our first example uses the data on US finished motor gasoline product copied from the **fpp2** package (Hyndman, 2023), which is presented in Figure 2 below. For convenience, it is stored as a `data.frame` with two columns: one with the series to be adjusted, stored as a numeric vector, and the second with the dates stored as a vector of class `Date`.

```
library(dplyr)
library(ggplot2)
library(boiwsa)

ggplot() +
  geom_line(
    aes(
      x = gasoline.data$date,
      y = gasoline.data$y
    ),
    color = "royalblue"
  ) +
  theme_bw() +
  ylab("Barrels per day (millions)") +
  xlab("Year")
```

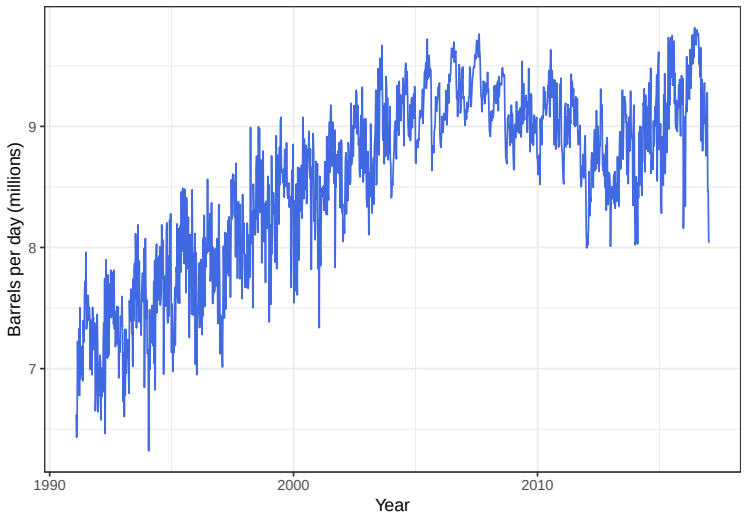


**Figure 2:** Weekly gasoline production in the US, February 2, 1991 to January 20, 2017.

Once users have their data loaded, they can use the `boiwsa` function to perform weekly seasonal adjustment. The following code shows the application of the package using automatic model selection.

[3]The example code utilizes **dplyr** (Wickham et al., 2023), **ggplot2** (Wickham et al., 2024), and **gridExtra** (Auguie, 2017)

[4]See https://github.com/timginker/boiwsa

[5]See https://cran.r-project.org/package=boiwsa

```
res <- boiwsa(
  x = gasoline.data$y,
  dates = gasoline.data$date
)
```

In general, the procedure can be applied with minimum interventions and requires only the series to be adjusted (`x` argument) and the associated dates (`dates` argument) provided in a date format. Unless specified otherwise (i.e., `my.k_l = NULL`), the procedure automatically identifies the best number of trigonometric variables in (2) to model the intra-yearly ($K$) and intra-monthly ($L$) seasonal cycles based on the Akaike Information Criterion corrected for small sample sizes (AICc). The information criterion can be adjusted through the `ic` option. Like other software, there are three options: "aic", "aicc", and "bic". The weighting decay rate is specified by `r`. By default $r = 0.8$ which is similar to what is customary in the literature (Harvey, 1990). For the full list of arguments and their description, see Table 1 below.

**Table 1:** Input arguments for boiwsa

| Input argument | Description |
|---|---|
| x | Numeric vector with series to be seasonally adjusted |
| dates | Vector of class "Date", containing the data dates |
| r | Defines the rate of decay of the weights. Should be between zero and one. By default is set to 0.8 |
| auto.ao.search | Boolean. Search for additive outliers |
| out.threshold | t-statistic threshold in outlier search. By default is set to 3.8 as suggested by Findley (1998) |
| ao.list | Vector with user-specified additive outliers in a date format |
| my.k_l | Numeric vector defining the number of yearly and monthly trigonometric variables. If NULL, is found automatically using the information criteria (AICc) |
| H | Matrix with holiday/working day factors or other user-defined preadjustments |
| ic | Information criterion used in the automatic search for the number of trigonometric regressors. There are three options: aic, aicc, and bic. aicc is set as default |
| method | Decomposition type: additive or multiplicative (log transformation) |

In addition, the procedure automatically searches for additive outliers (AO) using the method described in Appendix C of Findley et al. (1998). To disable the automatic AO search, set `auto.ao.search = F`. To add user-defined AOs, use the `ao.list` option. As suggested by Findley et al. (1998), the t-statistic threshold for outlier search is by default set to 3.8. However, since high-frequency data are generally more noisy (Proietti and Pedregal, 2023), it could be advantageous to consider setting a higher threshold by adjusting the `out.threshold` argument.

The `boiwsa` function returns a list object containing the results. The seasonally adjusted series is stored in a vector called `sa`. The estimated seasonal factors are stored as `sf`. In addition, the user can see the number of trigonometric terms chosen in automatic search (`my.k_l`) and the position of additive outliers (`ao.list`) found by the automatic routine. For the full list of outputs and their description, see Table 2.

**Table 2:** Output values for boiwsa

| Value | Description |
|---|---|
| sa | Numeric vector with seasonally adjusted series |
| x | Numeric vector with series to be seasonally adjusted |
| my.k_l | Number of trigonometric variables used to model the seasonal pattern |
| sf | Estimated seasonal effects. Note that these contain the seasonal effects represented by the trigonometric variables as well as the user-defined preadjustments in H |
| hol.factors | Estimated holiday effects or other user-defined variables supplied in H |
| out.factors | Estimated outlier effects |
| beta | DWR coefficients for the last year |
| trend | Final trend estimate (T_t^(3)) |
| ao.list | Additive outlier dates |
| m | lm object. Unweighted OLS regression on the full sample |

After the seasonal adjustment, we can plot the adjusted data to visualize the seasonal pattern:

```
plot(res)
```

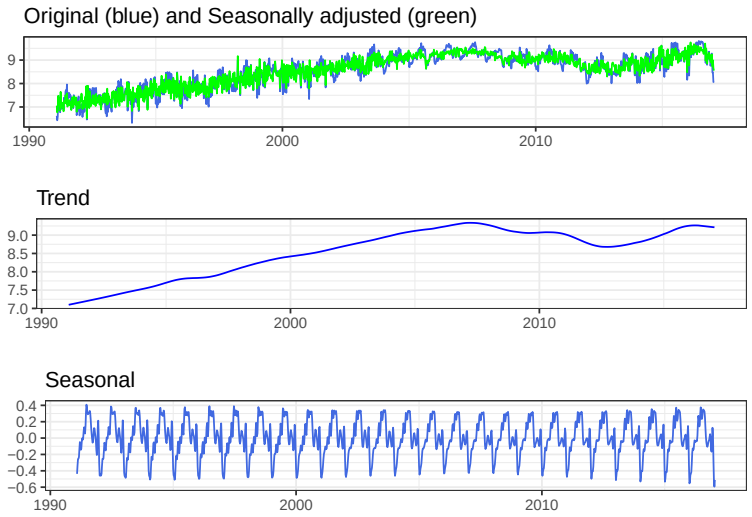


**Figure 3:** Weekly gasoline production in the US, February 2, 1991 to January 20, 2017.

To assess the quality of the adjustment, we can plot the autoregressive spectrum of the original and seasonally adjusted data, as illustrated in the code below:

```
# spectrum of the original series (after detrending)
spec0 <- spec.ar((res$x - res$trend), order = 60, plot = F)
# spectrum of the seasonally adjusted series (after detrending)
spec1 <- spec.ar((res$sa - res$trend), order = 60, plot = F)

# plot
ggplot() +
  geom_line(aes(x = spec0$freq, y = spec0$spec, color = "orig")) +
  geom_line(aes(x = spec0$freq, y = spec1$spec, color = "sa")) +
  geom_vline(xintercept = 1:2 / 4.34, linetype = "dashed") +
  geom_text(aes(x = 1:2 / 4.34, label = "
 Intra-monthly cycle peaks", y = 0.5 * max(spec0$spec)), colour = "black", angle = 90) +
  geom_vline(xintercept = (1:3) / 52.1775, linetype = "dashed") +
  geom_text(aes(x = 3 / 52.1775, label = "
First three intra-yearly cycle peaks", y = 0.5 * max(spec0$spec)), colour = "black", angle = 90) +
  scale_color_manual(
    name = "",
    values = c("orig" = "#31a354", "sa" = "#3182bd"),
    labels = c("Original", "Seasonally adjusted")
  ) +
  theme_bw() +
  theme(legend.position = "bottom") +
  theme(legend.text = element_text(size = 11)) +
  ylab(" ") +
  xlab("Frequency")
```

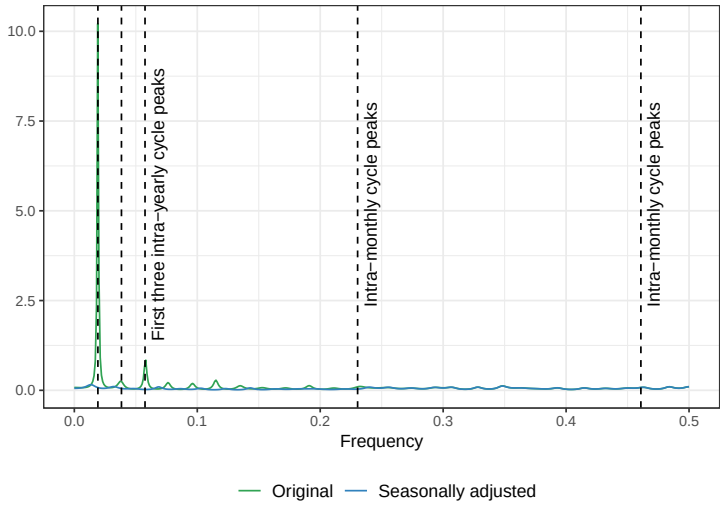


**Figure 4:** Autoregressive spectrum of weekly gasoline production in the US.

In spectral analysis, it is customary to remove the trend component prior to conducting the analysis. This step prevents the trend from obscuring the peaks associated with the seasonal component. If this pre-processing step is omitted, particularly in a time series with a strong trend, the resulting spectrum will predominantly display peaks at the beginning of the axes, thereby complicating the identification of seasonal peaks. Additionally, to compute the spectrum, the number of lags is set to 60. Finally, the first three yearly cycle peaks and the two monthly peaks are marked using vertical lines. Figure 4, which can also be generated using `boiwsa::plot_spec`, illustrates that the series originally had a single intra-yearly seasonal cycle, but this component was completely removed by the procedure.

We can also inspect the output to check if the number of trigonometric terms chosen by the automatic procedure matches our visual findings:

```
print(res)

#>
#>  number of yearly cycle variables:  12
#>  number of monthly cycle variables:  0
#>  list of additive outliers:  1998-03-28
```

As can be seen, the number of yearly terms, $K$, is 12 and the number of monthly terms is zero, which is consistent with the observed spectrum.

### 3.2 Example 2: Initial unemployment claims in Israel

In this subsection, we present our second example that is based on Israeli data. The data has no obvious trend but has two pronounced seasonal cycles, intra-yearly and intra-monthly. Moreover, there is a strong impact of two moving holidays and a working day effect.

The series under consideration is the weekly number of initial registrations at the Israeli Employment Service. Due to a prolonged period of structural change associated with the COVID-19 crisis, we limit our sample to January 11, 2014, to January 4, 2020. It is worth noting that addressing such events in the context of high-frequency data presents a significant challenge. Furthermore, given the duration of the COVID-19 crisis, the standard methods of incorporating outlier variables, which are also available in our package, appear to be incapable of providing a satisfactory solution in this context.

Registration and reporting at the Employment Service are mandatory prerequisites for individuals seeking to receive an unemployment benefit. Therefore, applicants are expected to register promptly after their employment has been terminated. Given that most employment contracts conclude toward the end of the month, an increased number of applications is anticipated at the beginning of the month, leading to an intra-monthly seasonal pattern. Additionally, as can be seen in Figure 5 below, on an annual basis, three distinct peaks are observed, with the final one occurring in August. This peak is closely tied to seasonal workers, leading to the creation of an intra-yearly cycle.

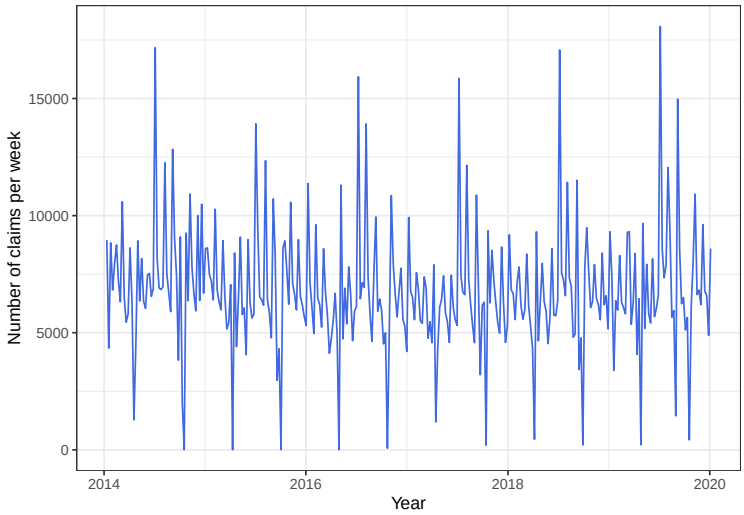


**Figure 5:** Weekly number of initial unemployment claims in Israel, January 11, 2014 to January 4, 2020.

Furthermore, each year, there are two weeks in which the activity plunges to nearly zero due to the existence of two moving holidays associated with Rosh Hashanah and Pesach. Moreover, a working day effect is expected, which leads to a reduced number of applications in weeks with fewer working days. These effects are captured and modeled through additional variables generated by the dedicated functions in **boiwsa**.

To generate a working day variable, we use the `boiwsa::simple_td` function, designed to aggregate the count of full working days within a week and normalize it. This function requires two parameters: the data dates and a `data.frame` object containing information about working days. The `data.frame` should be in a daily frequency and contain two columns: "date" and "WORKING_DAY_PART". For a complete working day, the "WORKING_DAY_PART" column should be assigned a value of 1, for a half working day 0.5, and for a holiday, the value should be set to 0.

Moving holiday variables can be created using the `boiwsa::genhol` function. These variables are computed using the Easter formula in Table 2 of Findley et al. (1998), with the calendar centering to avoid bias, as indicated in the documentation. In the present example, the impact of each holiday is concentrated within a single week, resulting in a noticeable drop and subsequent increase in the number of registrations during the following week. To account for this effect, we employ dummy variables that are globally centered. These dummy variables are created using a custom function - `boiwsa::my_rosh`, which is created for this illustrative scenario.

The code below illustrates the entire process based on the `boiwsa::lbm dataset`: creation of working day adjustment variables using the `boiwsa::simple_td` function; creation of moving holiday variables using the dedicated functions, and adding the combined input into the `boiwsa::boiwsa` function.

```
# creating an input for simple_td
dates_il %>%
  select(DATE_VALUE, ISR_WORKING_DAY_PART) %>%
  `colnames<-`(c("date", "WORKING_DAY_PART")) %>%
  mutate(date = as.Date(date)) -> df.td
# creating a matrix with a working day variable
td <- simple_td(dates = lbm$date, df.td = df.td)

# generating the Rosh Hashanah and Pesach moving holiday variables
rosh <- my_rosh(
  dates = lbm$date,
  holiday.dates = holiday_dates_il$rosh
)
# renaming (make sure that all the variables in H have distinct names)
colnames(rosh) <- paste0("rosh", colnames(rosh))

pesach <- my_rosh(
  dates = lbm$date,
  holiday.dates = holiday_dates_il$pesah,
  start = 3, end = -1
```

```
)
colnames(pesach) <- paste0("pesach", colnames(pesach))


# combining the prior adjustment variables in a single matrix
H <- as.matrix(cbind(rosh[, -1], pesach[, -1], td[, -1]))
# running seasonal adjustment routine
res <- boiwsa(
  x = lbm$IES_IN_W_ADJ,
  dates = lbm$date,
  H = H,
  out.threshold = 3.8
)
```

Subsequently, we can visually examine the results of the procedure presented in Figure 6 below:

```
plot(res)
```

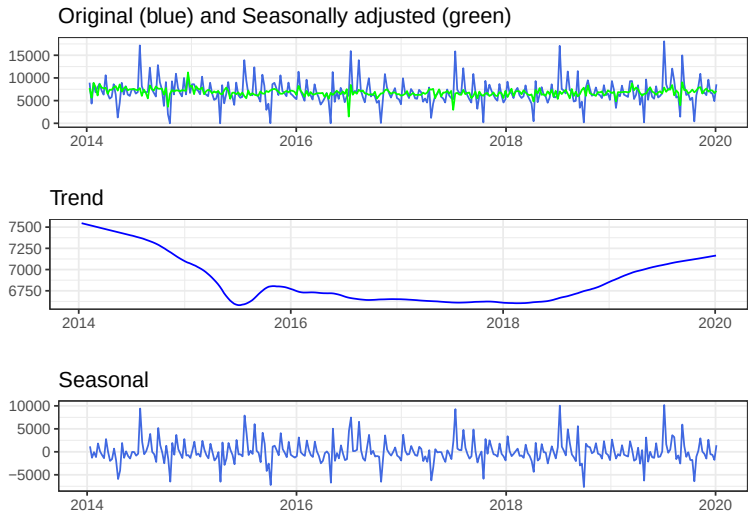


**Figure 6:** Weekly number of initial unemployment claims in Israel: Original, Seasonally adjusted, and the decomposition with 'out.threshold=3.8'.

As we can see in the plot, the procedure has successfully eliminated the annual and monthly seasonal cycles, along with the influences of moving holidays. Nevertheless, it's notable that a few pronounced declines emerge in the seasonally adjusted series from 2016 onward. As previously mentioned, weekly data often contain more noise, potentially prompting the inclusion of outlier variables where they might not be necessary. This inclusion could introduce bias to the seasonal component, consequently leading to the observed distortions.

While the suitability of the approach depends on the specific application, it appears that the recommended outlier threshold of 3.8, typically suggested for monthly data, might be insufficiently conservative for the weekly series. Consequently, a careful examination of the identified outliers is strongly advised. To tackle this issue, one possible solution involves raising the `out.threshold`, as demonstrated in the code and Figure summarizing the results below.

```
res <- boiwsa(
  x = lbm$IES_IN_W_ADJ,
  dates = lbm$date,
  H = H,
  out.threshold = 5
)

plot(res)
```

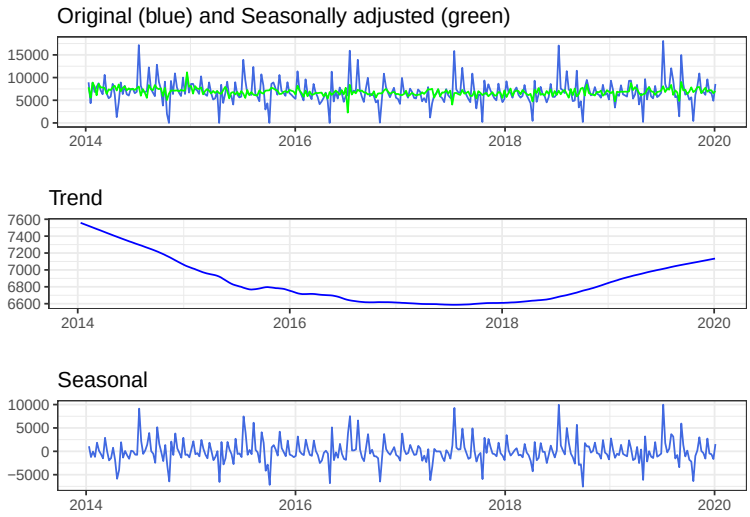


**Figure 7:** Weekly number of initial unemployment claims in Israel: Original, Seasonally adjusted, and the decomposition with 'out.threshold=5'.

Following a thorough visual examination of the seasonally adjusted data, we can now move forward with the spectrum diagnostics. As illustrated in Figure 8 below, corroborating our initial analysis of potential underlying seasonal patterns, it becomes evident that the data has two distinct seasonal cycles. Additionally, it is noteworthy that our procedure successfully removed the corresponding peaks, thereby highlighting its effectiveness.

```
plot_spec(res)
```

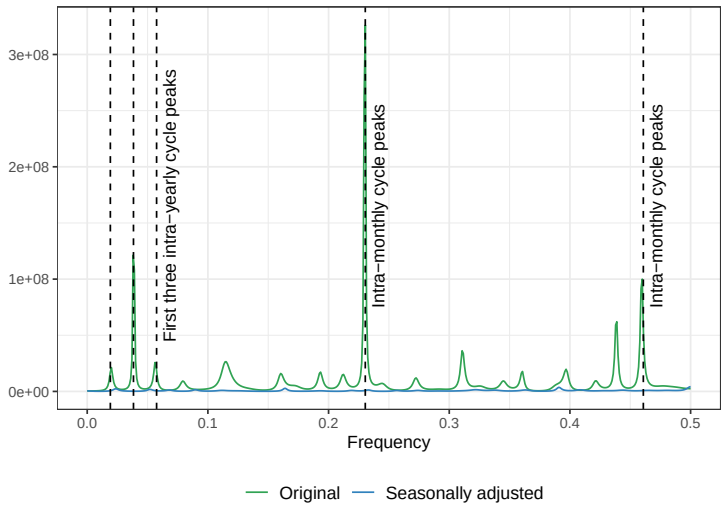


**Figure 8:** Autoregressive spectrum of a weekly number of initial unemployment claims in Israel.

## 4 Summary

The package **boiwsa** is developed to equip practitioners with the ability to perform seasonal adjustments on the weekly data, thereby facilitating informed decision-making across a diverse array of applications. It provides a user-friendly interface for computing seasonally adjusted estimates of weekly data and includes functions for the creation of country-specific prior adjustment variables, as well as diagnostic tools to assess the quality of the adjustments. The empirical applications presented in this paper demonstrate it's functionality and ability to perform under various practical scenarios.

## Acknowledgments

I would like to thank the editor and two anonymous referees for their remarks and suggestions that helped improve the paper and the package. I'm grateful to Karsten Webel for his review and advice. I would also like to thank Ariel Mantzura, Eyal Argov, Daniel Rosenman, and Ramsis Gara for their valuable suggestions and discussions. The views expressed in this paper are those of the author and do not necessarily reflect the views of the Bank of Israel.

## References

B. Auguie. *gridExtra: Miscellaneous Functions for "Grid" Graphics*, 2017. URL https://CRAN.R-project.org/package=gridExtra. R package version 2.3. [p189]

K. Bandara, R. J. Hyndman, and C. Bergmeir. MSTL: A seasonal-trend decomposition algorithm for time series with multiple seasonal patterns. *International Journal of Operational Research*, 2024. In press. [p187, 188]

R. B. Cleveland, W. S. Cleveland, J. E. McRae, and I. Terpenning. STL: A seasonal-trend decomposition. *Journal of Official Statistics*, 6(1):3–73, 1990. [p187]

W. P. Cleveland, T. D. Evans, and S. Scott. Weekly Seasonal Adjustment - A Locally-weighted Regression Approach. Economic working papers, Bureau of Labor Statistics, 2014. [p187]

W. L. Crum. Weekly fluctuations in outside bank debits. *The Review of Economic Statistics*, 9(1):30–36, 1927. [p186]

A. M. De Livera, R. J. Hyndman, and R. D. Snyder. Forecasting time series with complex seasonal patterns using exponential smoothing. *Journal of the American Statistical Association*, 106(496):1513–1527, 2011. URL https://doi.org/10.1198/jasa.2011.tm09771. [p187]

T. D. Evans, B. C. Monsell, and M. Sverchkov. Review of Available Programs for Seasonal Adjustment of Weekly Data. In *Proceedings of the Joint Statistical Meetings*. American Statistical Association, 2021. [p186]

D. F. Findley, B. C. Monsell, W. R. Bell, M. C. Otto, and B.-C. Chen. New capabilities and methods of the X-12-arima seasonal-adjustment program. *Journal of Business & Economic Statistics*, 16(2):127–152, 1998. ISSN 07350015. URL https://doi.org/10.2307/1392565. [p188, 190, 193]

P. J. Harrison and F. R. Johnston. Discount weighted regression. *The Journal of the Operational Research Society*, 35(10):923–932, 1984. ISSN 01605682, 14769360. URL https://doi.org/10.2307/2582135. [p186]

A. C. Harvey. *Forecasting, structural time series models and the Kalman filter*. Cambridge University Press, 1990. [p190]

R. Hyndman. *fpp2: Data for "Forecasting: Principles and Practice" (2nd Edition)*, 2023. URL https://CRAN.R-project.org/package=fpp2. R package version 2.5. [p189]

R. J. Hyndman and R. Killick. *CRAN Task View: Time Series Analysis*, 2024. URL https://CRAN.R-project.org/view=TimeSeries. Version 2024-10-02. [p186]

D. Ladiray and B. Quenneville. *Seasonal adjustment with the X-11 method*. Springer New York, NY, 2001. [p188]

T. S. McElroy and J. A. Livsey. Ecce Signum: An R Package for multivariate signal extraction and time series analysis. *arXiv preprint arXiv:2201.02148*, 2022. URL https://doi.org/10.48550/arXiv.2201.02148. [p187]

D. A. Pierce, M. R. Grupe, and W. P. Cleveland. Seasonal adjustment of the weekly monetary aggregates: A model-based approach. *Journal of Business & Economic Statistics*, 2(3):260–270, 1984. ISSN 07350015. URL https://doi.org/10.2307/1391708. [p188]

T. Proietti and D. J. Pedregal. Seasonality in High Frequency Time Series. *Econometrics and Statistics*, 27:62–82, 2023. URL https://doi.org/10.1016/j.ecosta.2022.02.001. [p186, 190]

R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2024. URL http://www.R-project.org/. ISBN 3-900051-07-0. [p187]

S. J. Taylor and B. Letham. Forecasting at scale. *The American Statistician*, 72(1):37–45, 2018. URL https://doi.org/10.7287/peerj.preprints.3190v2. [p187]

H. Wickham, R. François, L. Henry, K. Müller, and D. Vaughan. *dplyr: A Grammar of Data Manipulation*, 2023. URL https://CRAN.R-project.org/package=dplyr. R package version 1.1.4. [p189]

H. Wickham, W. Chang, L. Henry, T. L. Pedersen, K. Takahashi, C. Wilke, K. Woo, H. Yutani, D. Dunnington, T. van den Brand, and P. Posit. *ggplot2: Create Elegant Data Visualisations Using the Grammar of Graphics*, 2024. URL https://cran.r-project.org/package=ggplot2. R package version 3.5.1. [p189]

*Tim Ginker*
*Bank of Israel*
*Bank of Israel. POB 780, 91007, Jerusalem, Israel*
*ORCiD: 0000-0002-7138-5417*
timginker@gmail.com

# Bioconductor Notes, September 2024

*by Maria Doyle, Bioconductor Community Manager, and Bioconductor Core Developer Team*

**Abstract** We discuss general project news.

## 1 Introduction

Bioconductor provides tools for the analysis and comprehension of high-throughput genomic data. The project has entered its twentieth year, with funding for core development and infrastructure maintenance secured through 2025 (NIH NHGRI 2U24HG004059). Additional support is provided by NIH NCI, Chan-Zuckerberg Initiative, National Science Foundation, Microsoft, and Amazon. In this news report, we give some updates on core team and project activities.

## 2 Software

In May 2024, Bioconductor 3.19 was released*. It is compatible with R 4.4 and includes 2300 software packages, 430 experiment data packages, 926 up-to-date annotation packages, 30 workflows, and 5 books. Books are built regularly from source, ensuring full reproducibility; an example is the community-developed Orchestrating Single-Cell Analysis with Bioconductor.

**Note: Bioconductor 3.20 was subsequently released in October 2024. For details on the latest release, visit the Bioconductor website.*

## 3 Community and Impact

### 3.1 Outreachy Internships

We participated in the May-August 2024 Outreachy Internship program, during which intern Scholastica Urua contributed to the Microbiome Study Curation project. Scholastica reflected on her experience in a blog post, available here. Her work was recognized with the award for best Microbiome Virtual International Forum MicroTalk. The recording of her talk can be viewed on YouTube.

### 3.2 Bioconductor Athena Award

In July 2024, Beatriz Calvo-Serra was honoured as the inaugural recipient of the Bioconductor Athena Award. Beatriz was a passionate contributor to computational biology and an active member of the Bioconductor community. For more details, see this blog post.

## 4 Conferences

### 4.1 BioC2024 Recap

The annual BioC conference was held July 24-26 2024 at the Van Andel Institute in Grand Rapids, Michigan. Over 350 participants took part, with 116 attending in person and 240 joining virtually, allowing participants from regions as far away as Latin America, Africa, and Asia to be part of the event. This year's conference also marked our first time in the Mid-West US, highlighting the expanding reach and diversity of our community. See recap blog post here

### 4.2 EuroBioC2024 Recap

The European Bioconductor Conference (EuroBioC2024), held in Oxford in September 2024, welcomed over 100 in-person attendees. Highlights included keynote talks and workshops. Community-driven events like EuroBioC continue to foster collaboration and innovation. See recap blog post here.

### 4.3 BioCAsia 2024 Registration Open

Registration for BioCAsia 2024 is now open! The conference will take place on November 7–8, 2024, in Sydney, Australia, fostering collaboration among the bioinformatics community in the Asia-Pacific region. A Conference Access Award is available to assist presenters and participants with registration fees.

## 5 Boards and Working Groups Updates

### 5.1 Annual Call for CAB and TAB Nominations

In July 2024, Bioconductor opened its annual call for nominations to the Community Advisory Board (CAB) and Technical Advisory Board (TAB). These boards play a vital role in guiding Bioconductor's technical development, community outreach, and long-term viability. The nomination period closed on August 31, 2024, and we thank everyone who applied or shared the call within their networks. For more information on the CAB and TAB, visit:

- Community Advisory Board (CAB)
- Technical Advisory Board (TAB)

If you are interested in becoming involved with any Bioconductor working group please contact the group leader(s).

## 6 Using Bioconductor

Start using Bioconductor by installing the most recent version of R and evaluating the commands

```
if (!requireNamespace("BiocManager", quietly = TRUE))
    install.packages("BiocManager")
BiocManager::install()
```

Install additional packages and dependencies, e.g., SingleCellExperiment, with

```
BiocManager::install("SingleCellExperiment")
```

Docker images provides a very effective on-ramp for power users to rapidly obtain access to standardized and scalable computing environments. Key resources include:

- bioconductor.org to install, learn, use, and develop Bioconductor packages.
- A list of available software linking to pages describing each package.
- A question-and-answer style user support site and developer-oriented mailing list.
- A community slack workspace (sign up) for extended technical discussion.
- The F1000Research Bioconductor gateway for peer-reviewed Bioconductor workflows as well as conference contributions.
- The Bioconductor YouTube channel includes recordings of keynote and talks from recent conferences, in addition to video recordings of training courses.
- Our package submission repository for open technical review of new packages.

Upcoming and recently completed events are browsable at our events page.

The Technical and and Community Advisory Boards provide guidance to ensure that the project addresses leading-edge biological problems with advanced technical approaches, and adopts practices (such as a project-wide Code of Conduct) that encourages all to participate. We look forward to welcoming you!

We welcome your feedback on these updates and invite you to connect with us through the Bioconductor Slack workspace or by emailing `community@bioconductor.org`.

*Maria Doyle, Bioconductor Community Manager*
*University of Limerick*

*Bioconductor Core Developer Team*
*Dana-Farber Cancer Institute, Roswell Park Comprehensive Cancer Center, City University of New York, Fred Hutchinson Cancer Research Center, Mass General Brigham*

# R Foundation News

*by Torsten Hothorn*

## 1 Donations and members

Membership fees and donations received between 2024-12-06 and 2025-03-26.

#Donations

Gilberto Camara (Brazil) James Curran (New Zealand) Joseph Luchman (United States) Rudolph Martin (United States) Riccardo Martinelli (Italy) The University of Auckland, Statistics Department (New Zealand) Kem Phillips (United States) Fergus Reig Gracia (Spain) Thomas Stadler (Switzerland) Sebastiano Trevisani (Italy) iOMEDICO, Freiburg (Germany)

#Supporting benefactors

b-data GmbH, Winterthur (Switzerland) #Supporting institutions

Institute of Botany of the Czech Academy of Sciences, Pruhonice (Czechia) oikostat GmbH, Ettiswil (Switzerland)

#Supporting members

Richard Abdill (United States) Michael Blanks (United States) Pina Brinker (Germany) Ivan Maria Castellani (Italy) Keith Chamberlain (United States) Cédric Chambru (Switzerland) John Chandler (United States) Michael Chirico (United States) Gerard Conaghan (United Kingdom) Brandon Dahl (United States) Dan Dediu (Spain) Kevin DeMaio (United States) Anna Doizy (Réunion) Fraser Edwards (United Kingdom) Martin Elff (Germany) Isaac Florence (United Kingdom) Neil Frazer (United States) Bernd Fröhlich (Germany) Sven Garbade (Germany) Jan Marvin Garbuszus (Germany) Eduardo García Galea (Spain) Brian Gramberg (Netherlands) Spencer Graves (United States) Krushi Gurudu (United States) Frank Hafner (United States) Joe Harwood (United Kingdom) Philippe Heymans Smith (Costa Rica) Adam Hill (United States) Alexander Huelle (Germany) ken ikeda (Japan) Heidi Imker (United States) Sebastian Jentschke (Norway) JUNE KEE KIM (Korea, Republic of) Ziyad Knio (United States) Chris Kuty (United States) Caleb Lareau (United States) Thomas Levine (United States) Baoxiao Liu (Netherlands) Philippe MICHEL (France) Ernst Molitor (Germany) David Monterde (Spain) Stefan Moog (Germany) Keon-Woong Moon (Korea, Republic of) Steffen Moritz (Germany) yoshinobu nakahashi (Japan) Tsubasa Narihiro (Japan) Sermet Pekin (Turkey) PierGianLuca Porta Mana (Norway) Ingo Ruczinski (United States) Choonghyun Ryu (Korea, Republic of) Dejan Schuster (Germany) Harald Sterly (Germany) Kai Streicher (Switzerland) Robert van den Berg (Austria) Petr Waldauf (Czechia) Fredrik Wartenberg (Sweden) Nan Xiao (United States) Yihui Xie (China) Agnieszka Zawiejska (Poland) Vaidotas Zemlys-Balevičius (Lithuania) Guangyu Zeng (China)

*Torsten Hothorn*
*Universität Zürich*
*Switzerland*
*ORCiD: 0000-0001-8301-0471*
Torsten.Hothorn@R-project.org