# SIMAT: GC-SIM-MS Analayis Tool

Mo R. Nezami Ranjbar

April 20, 2026

## 1 Selected Ion Monitoring

Gas chromatography coupled with mass spectrometry (GC-MS) is one of the promising technologies for qualitative and quantitative analysis of small biomolecules. Because of the existence of spectral libraries, GC-MS instruments can be set up efficiently for targeted analysis. Also, to increase sensitivity, samples can be analyzed in selected ion monitoring (SIM) mode. While many software have been provided for analysis of untargeted GC-MS data, no specific tool does exist for processing of GC-MS data acquired with SIM.

## 2 SIMAT package

SIMAT is a tool for analysis of GC-MS data acquired in SIM mode. The tool provides several functions to import raw GC-SIM-MS data and standard format mass spectral libraries. It also provides guidance for fragment selection before running the targeted experiment in SIM mode by using optimization. This is done by considering overlapping peaks from a library provided by the user. Other functionalities include retention index calibration to improve target identification and plotting EICs of individual peaks in specific runs which can be used for visual assessment. In summary, the package has several capabilities, including:

- Processing gas chromatography coupled with mass spectrometry data acquired in selected ion monitoring (SIM) mode.
- Peak detection and identification.
- Similarity score calculation.
- Retention index (RI) calibration.
- Reading NIST mass spectral library (MSL) format.
- Importing netCDF raw files.
- EIC and TIC visualization

- Providing guidance in choosing appropriate fragments for the targets of interest by using an optimization algorithm.

# 3 Examples

Here, we provide some examples of the usage of different functions in the SIMAT package. After installation, we start by loading the package and example data sets included in the SIMAT library.

```
> # load the package
> library(SIMAT)
> # load the extracted data from a CDF file of a SIM run
> data(Run)
> # load the target table information
> data(target.table)
> # load the background library to be used with fragment selection
> data(Library)
> # load retention index table from RI standards
> data(RItable)
```

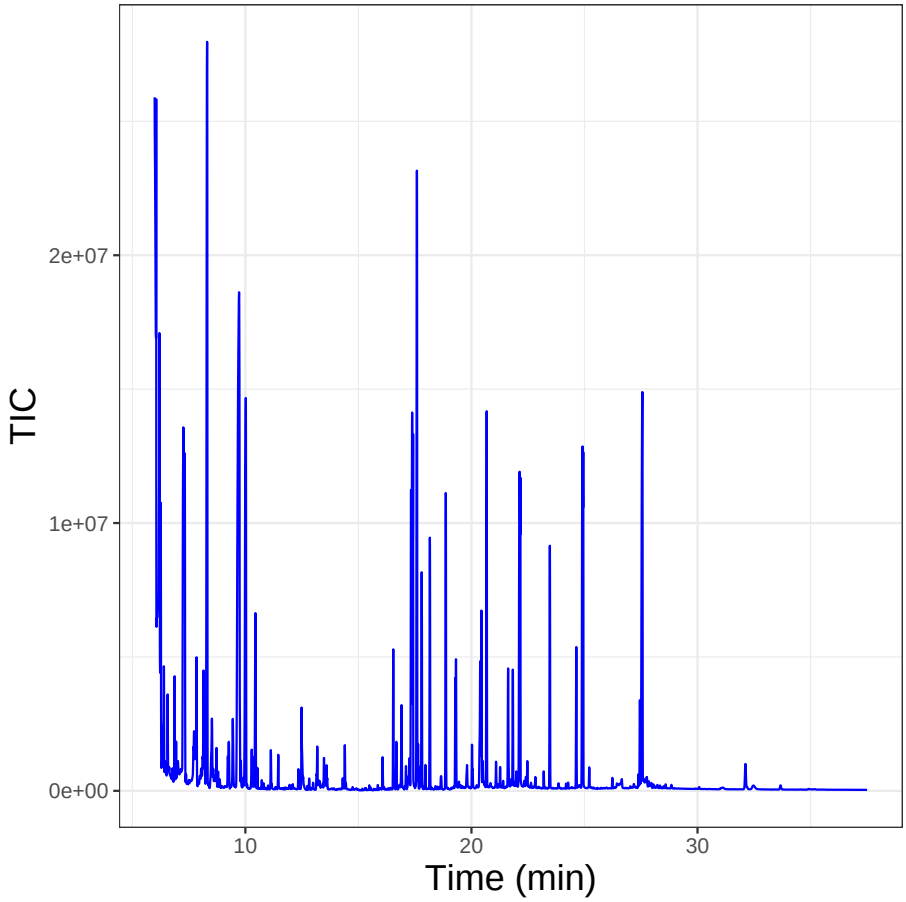
First let us examine the contents of the loaded data sets. Please note that you can find more details by checking the manual page for each data set. Starting with `Run` we can see that this object is a list, including four items, retention time, scans, scan information (in the `pk` filed), and the TIC data. We can also check some values for each field:

```
> # check the names of different fields in Run
> names(Run)

[1] "rt"  "sc"  "tic" "pk"

> # show some values for the the first three fields
> head(as.data.frame(Run[c("rt", "sc", "tic")]))

       rt sc      tic
1 359.233  1 25898079
2 359.491  2 24634398
3 359.749  3 24210034
4 360.008  4 23703436
5 360.266  5 23286408
6 360.524  6 22603404

> # see what is included in the scan information for the first scan
> Run$pk[[1]]

       mz intensity
 [1,]  55    108216
```

```
 [2,]  56    171072
 [3,]  66     10519
 [4,]  72   1361408
 [5,]  73   5509632
 [6,]  74    633600
 [7,]  75    763776
 [8,]  98     24528
 [9,] 117    609280
[10,] 118    169408
[11,] 120    132160
[12,] 121     68560
[13,] 122      5632
[14,] 146   8388096
[15,] 147   2875904
[16,] 148   1116672
[17,] 156      4714
[18,] 170      2638
[19,] 171     27576
[20,] 190     69376
[21,] 191   3803648
[22,] 194     41664
```

We can also plot the total ion chromatogram (TIC) of the run:

```
> # plot the TIC of the selected Run
> plotTIC(Run = Run)
```




For the target table information, which is a list object, we have three fileds, i.e. `compound`, `ms`, and `numFrag`:

```
> # check the name of included fields
> names(target.table)

[1] "compound" "ms"       "numFrag"

> # check the first lines of the target.table
> head(as.data.frame(target.table[c("compound", "numFrag")]))

     compound numFrag
1   Analyte 6       4
2  Analyte 33       4
3  Analyte 91       4
4 Analyte 104       4
5 Analyte 120       4
6 Analyte 135       4
```

```
> # check the contents of the ms field
> target.table$ms[[1]]

[1]  56  98 170 171
```

Similarly, for the library:

```
> # check the name of included fields
> names(Library)

[1] "compound" "rt"       "ri"       "ms"       "sp"

> # check the first lines of Library
> head(as.data.frame(Library[c("rt", "ri", "compound")]))

     rt    ri   compound
1 6.332 696.9  Analyte 4
2 6.361 699.0  Analyte 5
3 6.388 700.8  Analyte 6
4 6.420 703.1  Analyte 7
5 6.458 705.7  Analyte 8
6 7.504 778.6 Analyte 29

> # check the contents of the ms and sp fields related to the mass and intensity
> # of the fragments, i.e. spectral information
> Spectrum <- data.frame(ms = Library$ms[[1]], sp = Library$sp[[1]])
> head(Spectrum)

  ms  sp
1 52   7
2 53  32
3 54  32
4 55  89
5 56 294
6 59 993

> # plot the spectrum
> plot(x = Spectrum$ms, y = Spectrum$sp, type = "h", lwd = 2, col = "blue",
+      xlab = "mass", ylab = "intensity", main = Library$compound[1])
```

At last, let us find out what is included in `RItable`:

```
> # check the name of included fields
> names(RItable)

[1] "rt" "ri"

> # check the first lines of RItable
> head(RItable)
```

```
     rt   ri
1  7.82  800
2  9.26  900
3 10.65 1000
4 13.26 1200
5 15.61 1400
6 17.74 1600
```

Now we need to get the Targets from the provided target table and the library:

```
> # get targets info using target table and provided library
> Targets <- getTarget(Method = "library", Library = Library,
+                      target.table = target.table)
> # check the fields of Targets
> names(Targets)
```

```
[1] "rt"         "ri"         "compound"   "ms"         "sp"
[6] "quantFrag"  "sortedFrag"
```

```
> # check the first lines of some fields
> head(as.data.frame(Targets[c("compound", "rt", "ri")]))
```

```
     compound     rt    ri
1   Analyte 6  6.388 700.8
2  Analyte 33  7.582 784.0
3  Analyte 91  9.151 893.1
4 Analyte 104  9.599 919.5
5 Analyte 120  9.966 947.7
6 Analyte 135 10.321 975.0
```

To find the corresponding peaks in the run, we can call `getPeak` function:

```
> # get the peaks for this run corresponding to Targets
> runPeaks <- getPeak(Run = Run, Targets = Targets)
> # check the length of runPeaks (number of targets)
> length(runPeaks)
```

```
[1] 1
```

```
> # check the fields for each peak
> names(runPeaks[[1]][[1]])
```

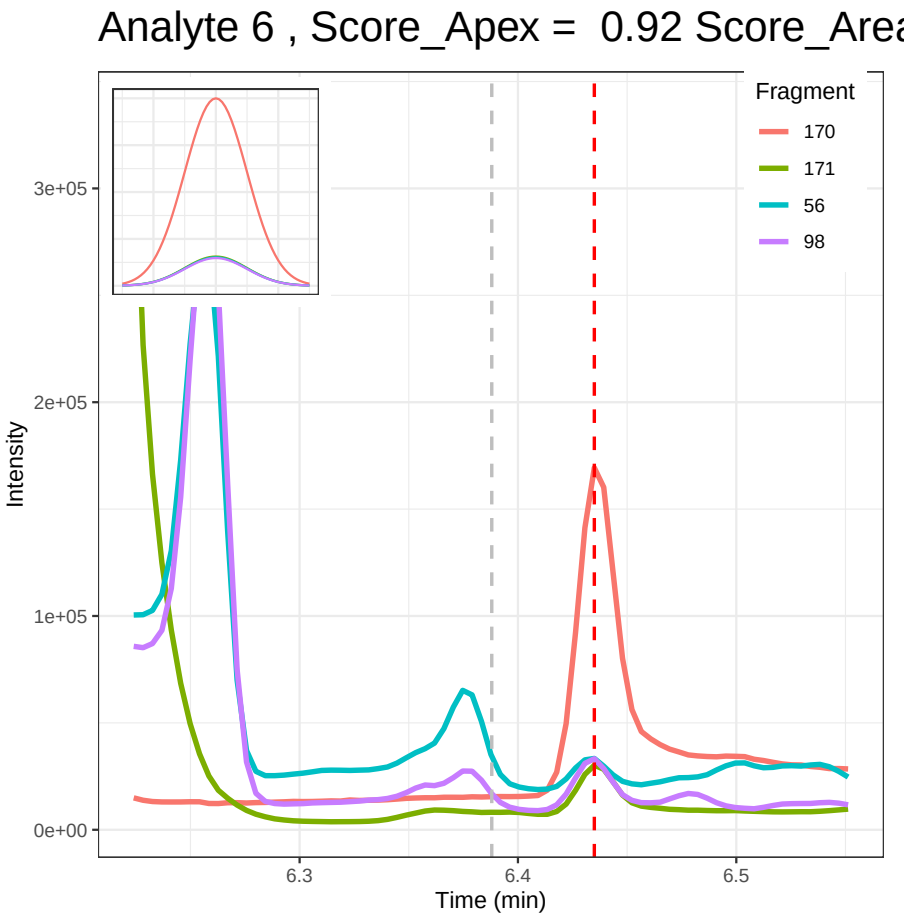
```
 [1] "rtApex"    "intApex"   "RI"        "scoreApex" "scoreArea" "area"
 [7] "EIC"       "RT"        "ms"        "sp"        "rt0"       "ri0"
[13] "compound"
```

```
> # area of the EIC of the first target
> runPeaks[[1]][[1]]$area
```

```
[1]  78504 129442 890934 172192
```

Following that, the extracted ion chromatogram (EIC) of the retrieved peaks can be visualized using `plotEIC` function:

```
> # plot the EIC of the first peak (target) on the list
> plotEIC(peakEIC = runPeaks[[1]][[1]])
```




However, the above is done without retention time calibration. To adjust for RI, first we call `getRI` to create a function which can be used to calculate the RI given the retention time:

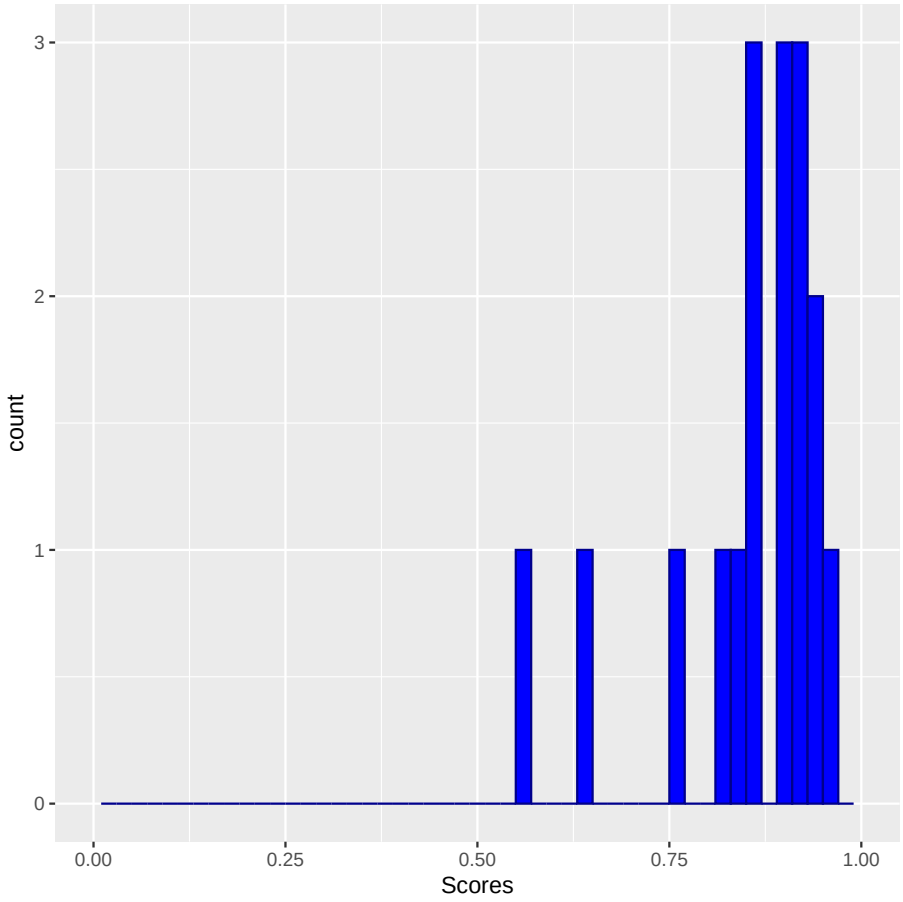
```
> # create the RI calibration function
> calibRI <- getRI(RItable)
> # calculate the RI of an RT = 12.32min
> calibRI(12.32)
```

```
[1] 1127.969
```

```
> # get the peaks for this run corresponding to Targets using RI calibration
> runPeaksRI <- getPeak(Run = Run, Targets = Targets, calibRI = calibRI)
```

It is informative to check the scores for the detected targets. This can be done by using a specific target in an individual run, or by finding the scores of all targets at once and looking at the histogram of the scores:

```
> # find the similarity score of the found targets
> Scores <- getPeakScore(runPeaks = runPeaks, plot = TRUE)
> # check the value of scores
> print(Scores)

           [,1]
 [1,] 0.8916002
 [2,] 0.7696184
 [3,] 0.9144938
 [4,] 0.9337329
 [5,] 0.8696336
 [6,] 0.9027617
 [7,] 0.9229788
 [8,] 0.5645287
 [9,] 0.8589739
[10,] 0.9150041
[11,] 0.8148486
[12,] 0.8911238
[13,] 0.6301923
[14,] 0.8460670
[15,] 0.9582448
[16,] 0.8516795
[17,] 0.9453319
```




To use the fragment selection function, i.e. `optFrag`, we can use the example background library `bgLib`. This is recommended to be done before the experiment, as after running the experiment, it may not be possible to find an optimum choice among the set of monitored fragments. We can also check to see what is the difference between the default set of the fragments, and the ones selected by `optFrag` function:

```
> # get the optimized version of the target list
> optTargets <- optFrag(Library = Library, target.table = target.table,
+                       forceOpt = TRUE)
> # check the fragments of the first target
> # the mass of fragments
> Targets$ms[[1]]

[1]  56  98 170 171

> # the intensity of fragments
> Targets$sp[[1]]
```

```
[1]  152  148 1000  156

> # check them after optimization
> # the mass of fragments
> optTargets$ms[[1]]

[1] 170 171 185 143

> # the intensity of fragments
> optTargets$sp[[1]]

[1] 1000  156  141   65
```

In the example above, the `optFrag` function is used directly. However, it is usually used within the `getTarget` function, where the user can set if optimization is desired.

# 4 Future Work

Improved peak detection and more options for data visualization are the main aspects of the next version. A GUI, where users can import and export data, is also considered for in future versions. Finally, it is planned to add support for importing other types of raw data such as mzML and mzXML together with other mass spectral library formats rather than NIST MSL.