# Package: PMScanR (via r-universe)

November 10, 2026

**Title** Protein motifs analysis and visualisation

**Version** 1.2.0

**Description** Provides tools for large-scale protein motif analysis and visualization in R. PMScanR facilitates the identification of motifs using external tools like PROSITE's ps_scan (handling necessary file downloads and execution) and enables downstream analysis of results. Key features include parsing scan outputs, converting formats (e.g., to GFF-like structures), generating motif occurrence matrices, and creating informative visualizations such as heatmaps, sequence logos (via seqLogo/ggseqlogo). The package also offers an optional Shiny-based graphical user interface for interactive analysis, aiming to streamline the process of exploring motif patterns across multiple protein sequences.

**URL** https://github.com/prodakt/PMScanR

**BugReports** https://github.com/prodakt/PMScanR/issues

**License** GPL-3

**Encoding** UTF-8

**LazyData** false

**RoxygenNote** 7.3.2

**VignetteBuilder** knitr

**biocViews** MotifDiscovery, Visualization

**Imports** dplyr (>= 1.1.0), shiny, bslib, shinyFiles, plotly, rtracklayer, reshape2, ggseqlogo, ggplot2, seqinr, magrittr, rlang, utils, stringr, BiocFileCache

**Suggests** BiocStyle, knitr, seqLogo, rmarkdown, testthat (>= 3.0.0)

**SystemRequirements** Perl

**Config/testthat/edition** 3

**Config/pak/sysreqs** cmake make libbz2-dev libicu-dev liblzma-dev libuv1-dev libxml2-dev libssl-dev perl xz-utils zlib1g-dev

**Repository** Bioconductor 3.23

**Date/Publication** 2026-04-28 13:05:36 UTC

**RemoteUrl** https://github.com/bioc/PMScanR

**RemoteRef** RELEASE_3_23

**RemoteSha** dd111f5b170c9a656e714e426be047af4aa997a1

# Contents

extractProteinMotifs . . . 2
extractSegments . . . 3
freqPie . . . 4
gff2matrix . . . 5
matrix2hm . . . 5
matrixToSquareHeatmap . . . 6
readProsite . . . 7
readPsa . . . 7
runPMScanRShiny . . . 8
runPsScan . . . 9

**Index** **11**

---

`extractProteinMotifs` *Extract protein motifs form a file in PSA format*

---

## Description

This function reads a file in PSA format containing protein sequences and extracts motifs based on specific patterns.

## Usage

```
extractProteinMotifs(file_path)
```

## Arguments

| | |
|---|---|
| `file_path` | A PSA file which specifying the path to the input file. The file should contain protein motifs. |

## Value

A list with a motif identifier (e.g. PSXXXXX) and the coresponding to it motif sequence which is associated wuth that identifier.

## Examples

```
file_path <- system.file("extdata/out_Hb_psa.txt", package = "PMScanR")
if (file_path != "") {
  protein_motifs <- extractProteinMotifs(file_path)
}
```

---

`extractSegments` *Extract sequence fragments from a list of sequences*

---

## Description

This function iterates over a list of sequences and extracts a sub-sequence from each based on a specified start and end position.

## Usage

```
extractSegments(sequences, from, to)
```

## Arguments

| | |
|---|---|
| `sequences` | A list of sequences, where each element is a vector of single characters. This is typically the output of ‘seqinr::read.fasta‘. |
| `from` | An integer specifying the starting position for the extraction. |
| `to` | An integer specifying the ending position for the extraction. |

## Value

A list representing the extracted sub-sequences. Sequences that were too short to have a fragment extracted are omitted from the list.

## Examples

```
# Get the path to the example FASTA file
fasta_file <- system.file("extdata", "hemoglobins.fasta", package = "PMScanR")

if (nzchar(fasta_file)) {
  sequences <- seqinr::read.fasta(fasta_file, seqtype = "AA")
  segments <- extractSegments(sequences, from = 10, to = 20)
}
```

`freqPie` *Create a pie chart showing protein motif distribution*

## Description

This function calculates the occurrences and percentages for each protein motif in the 'Name' column of a GFF-like data frame. It then creates a pie chart using 'ggplot2' to visualize the distribution.

## Usage

```
freqPie(data)
```

## Arguments

| | |
|---|---|
| `data` | A data frame in GFF format containing a column named 'Name' with the names of each protein motif. |

## Value

A ggplot object representing the pie chart.

## Examples

```
# Create sample data frame similar to parsed GFF output
sample_data <- data.frame(
  seqid = rep(c("Seq1", "Seq2"), each = 5),
  source = rep("PROSITE", 10),
  type = rep("MOTIF", 10),
  start = sample(1:100, 10),
  end = sample(101:200, 10),
  score = runif(10),
  strand = sample(c("+", "-"), 10, replace = TRUE),
  phase = sample(0:2, 10, replace = TRUE),
  Name = sample(c("Zinc_finger", "EGF_domain", "Kinase_domain"), 10, replace = TRUE)
)

# Generate the pie chart
motif_pie_chart <- freqPie(sample_data)
# print(motif_pie_chart)
```

## gff2matrix *Convert GFF to a binary occurrence matrix*

### Description

This function takes a GFF data frame and converts it into a binary matrix, indicating the presence (1) or absence (0) of a feature in a sequence.

### Usage

```
gff2matrix(input)
```

### Arguments

| | |
|---|---|
| input | A data frame containing GFF data, typically generated by ‘rtracklayer::import.gff‘ and converted to a data frame. It must have ’type’, ’start’, ’end’, and ’seqnames’ columns. |

### Value

A matrix, where values are binary: ‘1‘ indicates the presence of a feature, and ‘0‘ indicates its absence.

### Examples

```
gff_file <- system.file("extdata/out_Hb_gff.txt", package = "PMScanR")
if (nzchar(gff_file)) {
  gff_data <- as.data.frame(rtracklayer::import.gff(gff_file))
  motif_matrix <- gff2matrix(gff_data)
  # print(head(motif_matrix))
}
```

## matrix2hm *Generate a heatmap from a matrix*

### Description

This function generates a heatmap using the ‘plotly‘ package. The heatmap highlights specific rows and columns provided by the user, while the rest of the matrix is dimmed. The function also adds grid lines to the heatmap for better readability.

### Usage

```
matrix2hm(input, x = NULL, y = NULL)
```

### Arguments

| | |
|---|---|
| input | A matrix containing the data to be visualized in the heatmap |
| x | A character vector specifying the columns to highlight in the heatmap |
| y | A character vector specifying the rows to highlight in the heatmap |

### Value

A heatmap with highlighted specified rows and columns

### Examples

```
# Create a sample matrix with row and column names
mat <- matrix(c(1, 0, 1, 0), 2, 2)
colnames(mat) <- c("Col1", "Col2")
rownames(mat) <- c("Row1", "Row2")
heatmap <- matrix2hm(input = mat, x = "Col1", y = "Row1")
heatmap
```

---

## matrixToSquareHeatmap *Generate a square heatmap from a matrix*

---

### Description

This function generates a heatmap using 'plotly', ensuring the plot has a square aspect ratio. It highlights user-specified rows and columns.

### Usage

```
matrixToSquareHeatmap(input, x = NULL, y = NULL)
```

### Arguments

| | |
|---|---|
| input | A matrix containing the data to be visualized. |
| x | A character vector specifying the columns to highlight. |
| y | A character vector specifying the rows to highlight. |

### Value

A plotly heatmap object with a square layout.

### Examples

```
# Create a sample matrix
mat <- matrix(c(1, 0, 1, 0), 2, 2)
colnames(mat) <- c("Col1", "Col2")
rownames(mat) <- c("Row1", "Row2")
sq_heatmap <- matrixToSquareHeatmap(input = mat, x = "Col1", y = "Row1")
# To display in an interactive session:
# sq_heatmap
```

readProsite *Convert PROSITE format to a GFF-like Data Frame*

## Description

This function parses a file from a PROSITE scan into a data frame, extracting information about motif occurrences into a GFF-like structure.

## Usage

```
readProsite(prosite_input)
```

## Arguments

| | |
|---|---|
| prosite_input | Path to the PROSITE scan output file. |

## Value

A data frame with columns approximating GFF fields plus additional PROSITE-specific information.

## Examples

```
prosite_file <- system.file("extdata", "PROSITEoutput.txt", package = "PMScanR")

# Check that the example file exists before running
if (nzchar(prosite_file)) {
  gff_like_data <- readProsite(prosite_file)
  # You can view the output with:
  # head(gff_like_data)
}
```

readPsa *Parse a PSA (PROSITE Scan ASCII) File*

## Description

This function reads a file in PSA format and converts it into a standardized, GFF-like data frame for downstream analysis. It is robust to files that may contain extraneous, non-PSA formatted data at the end.

## Usage

```
readPsa(psa_file)
```

### Arguments

| | |
|---|---|
| psa_file | A character string specifying the path to the input PSA file. |

### Value

A data frame with a GFF-like structure, including all original placeholder columns.

### Examples

```
# Get the path to the example PSA file included with the package
psa_file_path <- system.file("extdata", "out_Hb_psa.txt", package = "PMScanR")

# Check that the file exists before running the example
if (nzchar(psa_file_path)) {
  gff_like_data <- readPsa(psa_file_path)
  # You can view the output with:
  # head(gff_like_data)
}
```

---

## runPMScanRShiny *Launch the PMScanR Shiny Application*

---

### Description

Calling this function will launch the interactive graphical user interface for the PMScanR package.

### Usage

```
runPMScanRShiny()
```

### Details

This function sets a higher file upload size limit for Shiny and then launches the application, which is built using an internal UI function (‘buildUi‘) and server function (‘buildServer‘).

### Value

This function is called for its side effect of launching the Shiny application and does not return a value.

### Examples

```
if (interactive()) {
  # To run the app, simply call the function
  runPMScanRShiny()
}
```

runPsScan *Run PS-Scan with Caching and Improved Execution*

## Description

This function runs the PROSITE ps_scan tool. It handles the downloading and caching of required executables and databases using BiocFileCache, detects the operating system, and executes the scan in a robust manner.

## Usage

```
runPsScan(in_file, out_file, out_format, os = NULL)
```

## Arguments

| | |
|---|---|
| in_file | Path to the input file containing protein sequences. |
| out_file | Path for the output file where results will be saved. |
| out_format | The output format for ps_scan (e.g., 'gff', 'psa'). |
| os | The operating system ('WIN', 'LINUX', 'MAC'). If NULL, it is detected automatically. |

## Value

Invisibly returns the exit status of the ps_scan command. The primary output is the result file created at 'out_file'.

## Examples

```
# This example is resource-intensive and requires an internet connection
# on first run to cache necessary files.
if (interactive()) {
  # Create a dummy input file for the example
  fasta_content <- c(">sp|P02025|HEMA_MESAU Hemoglobin subunit alpha",
                     "MVLSAADKGNVKAAWGKVGGHAAEYGAEALERMFLSFPTTKTYFPHFDLSHGSAQVKGHG",
                     "KKVADALTNAVAHVDDMPNALSALSDLHAHKLRVDPVNFKLLSHCLLVTLAAHLPAEFT",
                     "PAVHASLDKFLASVSTVLTSKYR")
  in_file <- tempfile(fileext = ".fasta")
  writeLines(fasta_content, in_file)

  out_file <- tempfile(fileext = ".gff")

  # The first run will download and cache ~100MB of data.
  # Subsequent runs will use the cached files.
  runPsScan(in_file = in_file, out_format = 'gff', out_file = out_file)

  # Clean up temporary files
  unlink(in_file)
  unlink(out_file)
```

```
}
```

# Index

extractProteinMotifs, 2
extractSegments, 3

freqPie, 4

gff2matrix, 5

matrix2hm, 5
matrixToSquareHeatmap, 6

readProsite, 7
readPsa, 7
runPMScanRShiny, 8
runPsScan, 9