

# Package ‘PeakSegDisk’

January 20, 2025

**Type** Package

**Title** Disk-Based Constrained Change-Point Detection

**Version** 2024.10.1

**Maintainer** Toby Dylan Hocking <toby.hocking@r-project.org>

**Description** Disk-based implementation of  
Functional Pruning Optimal Partitioning with up-down constraints  
<[doi:10.18637/jss.v101.i10](https://doi.org/10.18637/jss.v101.i10)> for single-sample peak calling  
(independently for each sample and genomic problem),  
can handle huge data sets ( $10^7$  or more).

**License** GPL-3

**Depends** R ( $\geq 2.10$ )

**Imports** data.table ( $\geq 1.9.8$ )

**Suggests** testthat, ggplot2, future.apply, future, knitr, markdown

**VignetteBuilder** knitr

**URL** <https://github.com/tdhock/PeakSegDisk>

**BugReports** <https://github.com/tdhock/PeakSegDisk/issues>

**NeedsCompilation** yes

**Author** Toby Dylan Hocking [aut, cre]

**Repository** CRAN

**Date/Publication** 2024-10-02 04:30:07 UTC

## Contents

ChIPreads . . . . .	2
coef.PeakSegFPOP_df . . . . .	3
coef.PeakSegFPOP_dir . . . . .	3
col.name.list . . . . .	4
fread.first . . . . .	5
fread.last . . . . .	6
Mono27ac . . . . .	7

PeakSegFPOP_df . . . . .	8
PeakSegFPOP_dir . . . . .	10
PeakSegFPOP_file . . . . .	13
PeakSegFPOP_vec . . . . .	14
plot.PeakSegFPOP_df . . . . .	16
plot.PeakSegFPOP_dir . . . . .	16
sequentialSearch_dir . . . . .	17
summary.PeakSegFPOP_dir . . . . .	18
wc2int . . . . .	19
writeBedGraph . . . . .	20

<b>Index</b>	<b>21</b>
--------------	-----------

---

ChIPreads	<i>Reads aligned to hg19 from two ChIP-seq experiments</i>
-----------	------------------------------------------------------------

---

## Description

These data are raw aligned reads which have been mapped to the human genome, hg19. One is sample ID McGill0004, experiment H3K36me3, chr9, chunk H3K36me3\_AM\_immune/8. The other is sample ID McGill0002, experiment H3K4me3, chr2, chunk H3K4me3\_PGP\_immune/7. The chunk ID numbers refer to parts of the McGill benchmark data set, <https://rcdata.nau.edu/genomic-ml/chip-seq-chunk-db/>

## Usage

```
data("ChIPreads")
```

## Format

A data frame with 40396 observations on the following 4 variables.

experiment either H3K36me3 or H3K4me3

chrom either chr9 or chr2

chromStart 0-based start position of read

chromEnd 1-based end position of read

count number of times a read occurred with the given chromStart/end in this sample/experiment

## Details

Peak detection algorithms are typically run on a sequence of non-negative integer count data, one data point for each genomic position. These data are useful for proving that peak detection methods are robust to different sequences: (1) spatially correlated, non-independent aligned read coverage; (2) un-correlated, independent representations such as first or last read.

---

coef.PeakSegFPOP\_df    *coef PeakSegFPOP df*

---

**Description**

Create a list of data tables describing PeakSegFPOP model and data.

**Usage**

```
## S3 method for class 'PeakSegFPOP_df'  
coef(object,  
      ...)
```

**Arguments**

object	object
...	...

**Value**

list of data tables with named elements segments, loss, data, changes, peaks.

**Author(s)**

Toby Dylan Hocking <toby.hocking@r-project.org> [aut, cre]

---

coef.PeakSegFPOP\_dir    *coef PeakSegFPOP dir*

---

**Description**

Compute changes and peaks to display/plot.

**Usage**

```
## S3 method for class 'PeakSegFPOP_dir'  
coef(object,  
      ...)
```

**Arguments**

object	object
...	...

**Value**

model list with additional named elements peaks and changes.

**Author(s)**

Toby Dylan Hocking <toby.hocking@r-project.org> [aut, cre]

---

col.name.list	<i>col name list</i>
---------------	----------------------

---

**Description**

Named list of character vectors (column names of bed/bedGraph/tsv files), used to read data files, which do not contain a header / column names. Each name corresponds to a data/file type, and each value is a character vector of column names expected in that file. loss is for the coverage.bedGraph\_penalty=VALUE\_loss.tsv file generated by [PeakSegFPOP\\_file](#); segments is for the coverage.bedGraph\_penalty=VALUE\_segments.bed generated by [PeakSegFPOP\\_file](#); coverage is for the coverage.bedGraph file which is used as input to [PeakSegFPOP\\_file](#).

**Usage**

```
"col.name.list"
```

**Examples**

```
library(PeakSegDisk)
r <- function(chrom, chromStart, chromEnd, coverage){
  data.frame(chrom, chromStart, chromEnd, coverage)
}
four <- rbind(
  r("chr1", 0, 10, 2),
  r("chr1", 10, 20, 10),
  r("chr1", 20, 30, 14),
  r("chr1", 30, 40, 13))
write.table(
  four, tmp <- tempfile(),
  sep="\t", row.names=FALSE, col.names=FALSE)
read.table(
  tmp, col.names=col.name.list$coverage)

pstr <- "10.5"
PeakSegFPOP_file(tmp, pstr)
outf <- function(suffix){
  paste0(tmp, "_penalty=", pstr, "_", suffix)
}
fread.first(outf("segments.bed"), col.name.list$segments)
fread.first(outf("loss.tsv"), col.name.list$loss)
```

---

fread.first	<i>Quickly read first line</i>
-------------	--------------------------------

---

## Description

Read the first line of a text file. Useful for quickly checking if the coverage.bedGraph\_penalty=VALUE\_segments.bed file is consistent with the coverage.bedGraph\_penalty=VALUE\_loss.tsv file. (used by the [PeakSegFPOP\\_dir](#) caching mechanism)

## Usage

```
fread.first(file.name,  
            col.name.vec)
```

## Arguments

file.name	Name of file to read.
col.name.vec	Character vector of column names.

## Value

Data table with one row.

## Author(s)

Toby Dylan Hocking <toby.hocking@r-project.org> [aut, cre]

## Examples

```
library(PeakSegDisk)
r <- function(chrom, chromStart, chromEnd, coverage){
  data.frame(chrom, chromStart, chromEnd, coverage)
}
four <- rbind(
  r("chr1", 0, 10, 2),
  r("chr1", 10, 20, 10),
  r("chr1", 20, 30, 14),
  r("chr1", 30, 40, 13))
write.table(
  four, tmp <- tempfile(),
  sep="\t", row.names=FALSE, col.names=FALSE)
pstr <- "10.5"
PeakSegFPOP_file(tmp, pstr)

outf <- function(suffix){
  paste0(tmp, "_penalty=", pstr, "_", suffix)
}
segments.bed <- outf("segments.bed")
first.seg.line <- fread.first(segments.bed, col.name.list$segments)
```

```
last.seg.line <- fread.last(segments.bed, col.name.list$segments)

loss.tsv <- outf("loss.tsv")
loss.row <- fread.first(loss.tsv, col.name.list$loss)

seg.bases <- first.seg.line$chromEnd - last.seg.line$chromStart
loss.row$bases == seg.bases
```

---

fread.last

*Quickly read last line*

---

## Description

Read the last line of a text file. Useful for quickly checking if the coverage.bedGraph\_penalty=VALUE\_segments.bed file is consistent with the coverage.bedGraph\_penalty=VALUE\_loss.tsv file. (used by the [PeakSegFPOP\\_dir](#) caching mechanism)

## Usage

```
fread.last(file.name,
           col.name.vec)
```

## Arguments

file.name	Name of file to read.
col.name.vec	Character vector of column names.

## Value

Data table with one row.

## Author(s)

Toby Dylan Hocking <toby.hocking@r-project.org> [aut, cre]

## Examples

```
library(PeakSegDisk)
r <- function(chrom, chromStart, chromEnd, coverage){
  data.frame(chrom, chromStart, chromEnd, coverage)
}
four <- rbind(
  r("chr1", 0, 10, 2),
  r("chr1", 10, 20, 10),
  r("chr1", 20, 30, 14),
  r("chr1", 30, 40, 13))
write.table(
  four, tmp <- tempfile(),
```

```

    sep="\t", row.names=FALSE, col.names=FALSE)
pstr <- "10.5"
PeakSegFPOP_file(tmp, pstr)

outf <- function(suffix){
  paste0(tmp, "_penalty=", pstr, "_", suffix)
}
segments.bed <- outf("segments.bed")
first.seg.line <- fread.first(segments.bed, col.name.list$segments)
last.seg.line <- fread.last(segments.bed, col.name.list$segments)

loss.tsv <- outf("loss.tsv")
loss.row <- fread.first(loss.tsv, col.name.list$loss)

seg.bases <- first.seg.line$chromEnd - last.seg.line$chromStart
loss.row$bases == seg.bases

```

---

Mono27ac

*A small ChIP-seq data set in which peaks can be found using Peak-SegFPOP*

---

## Description

The data come from an H3K27ac ChIP-seq experiment which was aligned to the human reference genome (hg19), aligned read counts were used to produce the coverage data; looking at these data in a genome browser was used to produce the labels. ChIP-seq means Chromatin Immunoprecipitation followed by high-throughput DNA sequencing; it is an assay used to characterize genome-wide DNA-protein interactions. In this experiment the protein of interest is histone H3, with the specific modification K27ac (hence the name H3K27ac). Large counts (peaks) therefore indicate regions of the reference genome with high likelihood of interaction between DNA and that specific protein, in the specific Monocyte sample tested.

## Usage

```
data("Mono27ac")
```

## Format

A list of 2 data.tables: coverage has 4 columns (chrom, chromStart, chromEnd, count=number of aligned reads at each position on chrom:chromStart-chromEnd); labels has 4 columns (chrom, chromStart, chromEnd, annotation=label at chrom:chromStart-chromEnd). chrom refers to the chromosome on which the data were gathered (chr11), chromStart is the 0-based position before the first base of the data/label, chromEnd is the 1-based position which is the last base of the data/label. Therefore, each chromEnd on each row should be equal to the chromStart of the next row.

**Source**

UCI Machine Learning Repository, chipseq data set, problem directory H3K27ac-H3K4me3\_TDHAM\_BP/samples/Mono1\_580000 Links: <https://archive.ics.uci.edu/ml/datasets/chipseq> for the UCI web page; <https://github.com/tdhock/feature-learning-benchmark> for a more detailed explanation.

---

 PeakSegFPOP\_df

*PeakSeg penalized solver for data.frame*


---

**Description**

Write data frame to disk then run [PeakSegFPOP\\_dir](#) solver.

**Usage**

```
PeakSegFPOP_df(count.df,
               pen.num, base.dir = tempdir())
```

**Arguments**

count.df	data.frame with columns count, chromStart, chromEnd. These data will be saved via <a href="#">writeBedGraph</a> , creating a plain text file with the following four columns: chrom (character chromosome name), chromStart (integer start position), chromEnd (integer end position), count (integer aligned read count on chrom from chromStart+1 to chromEnd); see also <a href="https://genome.ucsc.edu/goldenPath/help/bedgraph.htm">https://genome.ucsc.edu/goldenPath/help/bedgraph.htm</a>
pen.num	Non-negative numeric scalar.
base.dir	base.dir/chrXX-start-end/coverage.bedGraph will be written, where chrXX is the chrom column, start is the first chromStart position, and end is the last chromEnd position.

**Value**

List of solver results, same as [PeakSegFPOP\\_dir](#).

**Author(s)**

Toby Dylan Hocking <[toby.hocking@r-project.org](mailto:toby.hocking@r-project.org)> [aut, cre]

**Examples**

```
## Simulate a sequence of Poisson count data.
sim(seg <- function(seg.mean, size.mean=15){
  seg.size <- rpois(1, size.mean)
  rpois(seg.size, seg.mean)
})
set.seed(1)
seg.mean.vec <- c(1.5, 3.5, 0.5, 4.5, 2.5)
z.list <- lapply(seg.mean.vec, sim(seg))
```



```

z.rep.vec <- unlist(z.list)

## Plot the simulated data sequence.
if(require(ggplot2)){
  count.df <- data.frame(
    position=seq_along(z.rep.vec),
    count=z.rep.vec)
  gg.count <- ggplot()+
    geom_point(aes(
      position, count),
      shape=1,
      data=count.df)
  gg.count
}

## Plot the true changes.
n.segs <- length(seg.mean.vec)
seg.size.vec <- sapply(z.list, length)
seg.end.vec <- cumsum(seg.size.vec)
change.vec <- seg.end.vec[-n.segs]+0.5
change.df <- data.frame(
  changepoint=change.vec)
gg.change <- gg.count+
  geom_vline(aes(
    xintercept=changepoint),
    data=change.df)
gg.change

## Plot the run-length encoding of the same data.
z.rle.vec <- rle(z.rep.vec)
chromEnd <- cumsum(z.rle.vec$lengths)
coverage.df <- data.frame(
  chrom="chrUnknown",
  chromStart=c(0L, chromEnd[-length(chromEnd)]),
  chromEnd,
  count=z.rle.vec$values)
gg.rle <- gg.change+
  geom_segment(aes(
    chromStart+0.5, count, xend=chromEnd+0.5, yend=count),
  data=coverage.df)
gg.rle

## Fit a peak model and plot the segment means.
fit <- PeakSegDisk::PeakSegFPOP_df(coverage.df, 10.5)
gg.rle+
  geom_segment(aes(
    chromStart+0.5, mean, xend=chromEnd+0.5, yend=mean),
    color="green",
    data=fit$segments)

## Default plot method shows data as geom_step.
(gg <- plot(fit))

```

```
## Plot data as points to verify the step representation.
gg+
  geom_point(aes(
    position, count),
    color="grey",
    shape=1,
    data=count.df)
```

---

PeakSegFPOP\_dir

*PeakSeg penalized solver with caching*


---

## Description

Main function/interface for the PeakSegDisk package. Run the low-level solver, [PeakSegFPOP\\_file](#), on one genomic segmentation problem directory, and read the result files into R. Actually, this function will first check if the result files are already present (and consistent), and if so, it will simply read them into R (without running [PeakSegFPOP\\_file](#)) – this is a caching mechanism that can save a lot of time. To run the algo on an integer vector, use [PeakSegFPOP\\_vec](#); for a data.frame, use [PeakSegFPOP\\_df](#). To compute the optimal model for a given number of peaks, use [sequentialSearch\\_dir](#).

## Usage

```
PeakSegFPOP_dir(problem.dir,
  penalty.param, db.file = NULL)
```

## Arguments

- |               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
|---------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| problem.dir   | Path to a directory like sampleID/problems/chrXX-start-end which contains a coverage.bedGraph file with the aligned read counts for one genomic segmentation problem. This must be a plain text file with the following four columns: chrom (character chromosome name), chromStart (integer start position), chromEnd (integer end position), count (integer aligned read count on chrom from chromStart+1 to chromEnd); see also <a href="https://genome.ucsc.edu/goldenPath/help/bedgraph.html">https://genome.ucsc.edu/goldenPath/help/bedgraph.html</a> . Note that the standard coverage.bedGraph file name is required; for full flexibility the user can run the algo on an arbitrarily named file via <a href="#">PeakSegFPOP_file</a> (see that man page for an explanation of how storage on disk happens). |
| penalty.param | non-negative numeric penalty parameter (larger values for fewer peaks), or character scalar which can be interpreted as such. 0 means max peaks, Inf means no peaks.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| db.file       | character scalar: file for writing temporary cost function database – there will be a lot of disk writing to this file. Default NULL means to write the same disk where the input bedGraph file is stored; another option is tempfile() which may result in speedups if the input bedGraph file is on a slow network disk and the temporary storage is a fast local disk.                                                                                                                                                                                                                                                                                                                                                                                                                                              |

## Details

Finds the optimal change-points using the Poisson loss and the PeakSeg constraint (changes in mean alternate between non-decreasing and non-increasing). For  $N$  data points, the functional pruning algorithm is  $O(\log N)$  memory. It is  $O(N \log N)$  time and disk space. It computes the exact solution to the optimization problem in vignette("Examples", package="PeakSegDisk").

## Value

Named list of two data.tables:

**segments** has one row for every segment in the optimal model,  
**loss** has one row and contains the following columns:

**penalty** same as input parameter

**segments** number of segments in optimal model

**peaks** number of peaks in optimal model

**bases** number of positions described in bedGraph file

**bedGraph.lines** number of lines in bedGraph file

**total.loss** total Poisson loss =  $\sum_i m_i - z_i * \log(m_i) = \text{mean.pen.cost} * \text{bases} - \text{penalty} * \text{peaks}$

**mean.pen.cost** mean penalized cost =  $(\text{total.loss} + \text{penalty} * \text{peaks}) / \text{bases}$

**equality.constraints** number of adjacent segment means that have equal values in the optimal solution

**mean.intervals** mean number of intervals/candidate changepoints stored in optimal cost functions  
 – useful for characterizing the computational complexity of the algorithm

**max.intervals** maximum number of intervals

**megabytes** disk usage of \*.db file

**seconds** timing of PeakSegFPOP\_file

## Author(s)

Toby Dylan Hocking <toby.hocking@r-project.org> [aut, cre]

## Examples

```
data(Mono27ac, package="PeakSegDisk", envir=environment())
data.dir <- file.path(
  tempfile(),
  "H3K27ac-H3K4me3_TDHAM_BP",
  "samples",
  "Mono1_H3K27ac",
  "S001YW_NCMLS",
  "problems",
  "chr11-60000-580000")
dir.create(data.dir, recursive=TRUE, showWarnings=FALSE)
write.table(
  Mono27ac$coverage, file.path(data.dir, "coverage.bedGraph"),
```

```

col.names=FALSE, row.names=FALSE, quote=FALSE, sep="\t")

## Compute one model with penalty=1952.6
(fit <- PeakSegDisk::PeakSegFPOP_dir(data.dir, 1952.6))
summary(fit)#same as fit$loss

## Visualize that model.
ann.colors <- c(
  noPeaks="#f6f4bf",
  peakStart="#ffafaf",
  peakEnd="#ff4c4c",
  peaks="#a445ee")
if(require(ggplot2)){
  lab.min <- Mono27ac$labels[1, chromStart]
  lab.max <- Mono27ac$labels[.N, chromEnd]
  plist <- coef(fit)
  gg <- ggplot()+
    theme_bw()+
    geom_rect(aes(
      xmin=chromStart/1e3, xmax=chromEnd/1e3,
      ymin=-Inf, ymax=Inf,
      fill=annotation),
      color="grey",
      alpha=0.5,
      data=Mono27ac$labels)+
    scale_fill_manual("label", values=ann.colors)+
    geom_step(aes(
      chromStart/1e3, count),
      color="grey50",
      data=Mono27ac$coverage)+
    geom_segment(aes(
      chromStart/1e3, mean,
      xend=chromEnd/1e3, yend=mean),
      color="green",
      size=1,
      data=plist$segments)+
    geom_vline(aes(
      xintercept=chromEnd/1e3, linetype=constraint),
      color="green",
      data=plist$changes)+
    scale_linetype_manual(
      values=c(
        inequality="dotted",
        equality="solid"))
  print(gg)
  print(gg+coord_cartesian(xlim=c(lab.min, lab.max)/1e3, ylim=c(0, 10)))
  ## Default plotting method only shows model.
  print(gg <- plot(fit))
  ## Data can be added on top of model.
  print(
    gg+
    geom_step(aes(
      chromStart, count),

```

```
        color="grey50",
        data=mono27ac$coverage)
    )
}
```

---

PeakSegFPOP\_file      *PeakSegFPOP using disk storage*

---

### Description

Run the PeakSeg Functional Pruning Optimal Partitioning algorithm, using a file on disk to store the  $O(N)$  function piece lists, each of size  $O(\log N)$ . This is a low-level function that just runs the algo and produces the result files (without reading them into R), so normal users are recommended to instead use [PeakSegFPOP\\_dir](#), which calls this function then reads the result files into R.

### Usage

```
PeakSegFPOP_file.bedGraph.file,
pen.str, db.file = NULL)
```

### Arguments

`bedGraph.file` character scalar: tab-delimited tabular text file with four columns: chrom, chromStart, chromEnd, coverage. The algorithm creates a large temporary file in the same directory, so make sure that there is disk space available on that device.

`pen.str` character scalar that can be converted to a numeric scalar via `as.numeric`: non-negative penalty. More penalty means fewer peaks. "0" and "Inf" are OK. Character is required rather than numeric, so that the user can reliably find the results in the output files, which are in the same directory as `bedGraph.file`, and named using the penalty value, e.g. `coverage.bedGraph_penalty=136500650856.439_loss.tsv`

`db.file` character scalar: file for writing temporary cost function database – there will be a lot of disk writing to this file. Default NULL means to write the same disk where the input `bedGraph` file is stored; another option is `tempfile()` which may result in speedups if the input `bedGraph` file is on a slow network disk and the temporary storage is a fast local disk.

### Value

A named list of input parameters, and the temporary cost function database file size in megabytes.

### Author(s)

Toby Dylan Hocking <toby.hocking@r-project.org> [aut, cre]

**Examples**

```

r <- function(chrom, chromStart, chromEnd, coverage){
  data.frame(chrom, chromStart, chromEnd, coverage)
}
four <- rbind(
  r("chr1", 0, 10, 2),
  r("chr1", 10, 20, 10),
  r("chr1", 20, 30, 14),
  r("chr1", 30, 40, 13))
dir.create(prob.dir <- tempfile())
coverage.bedGraph <- file.path(prob.dir, "coverage.bedGraph")
write.table(
  four, coverage.bedGraph,
  sep="\t", row.names=FALSE, col.names=FALSE)
pstr <- "10.5"
result.list <- PeakSegDisk::PeakSegFPOP_file(coverage.bedGraph, pstr)
dir(prob.dir)

## segments file can be read to see optimal segment means.
outf <- function(suffix){
  paste0(coverage.bedGraph, "_penalty=", pstr, suffix)
}
segments.bed <- outf("_segments.bed")
seg.df <- read.table(segments.bed)
names(seg.df) <- col.name.list$segments
seg.df

## loss file can be read to see optimal Poisson loss, etc.
loss.tsv <- outf("_loss.tsv")
loss.df <- read.table(loss.tsv)
names(loss.df) <- col.name.list$loss
loss.df

```

---

PeakSegFPOP\_vec

*PeakSeg penalized solver for integer vector*


---

**Description**

Convert integer data vector to run-length encoding, then run [PeakSegFPOP\\_df](#).

**Usage**

```
PeakSegFPOP_vec(count.vec,
               pen.num)
```

**Arguments**

count.vec	integer vector, noisy non-negative count data to segment.
pen.num	Non-negative numeric scalar.

**Value**

List of solver results, same as [PeakSegFPOP\\_dir](#).

**Author(s)**

Toby Dylan Hocking <toby.hocking@r-project.org> [aut, cre]

**Examples**

```
## Simulate a sequence of Poisson data.
sim(seg <- function(seg.mean, size.mean=15){
  seg.size <- rpois(1, size.mean)
  rpois(seg.size, seg.mean)
})
set.seed(1)
seg.mean.vec <- c(1.5, 3.5, 0.5, 4.5, 2.5)
z.list <- lapply(seg.mean.vec, sim(seg))
z.rep.vec <- unlist(z.list)

## Plot the simulated data.
if(require(ggplot2)){
  count.df <- data.frame(
    position=seq_along(z.rep.vec),
    count=z.rep.vec)
  gg.count <- ggplot()+
    geom_point(aes(
      position, count),
      shape=1,
      data=count.df)
  print(gg.count)
  ## Plot the true changepoints.
  n.segs <- length(seg.mean.vec)
  seg.size.vec <- sapply(z.list, length)
  seg.end.vec <- cumsum(seg.size.vec)
  change.vec <- seg.end.vec[-n.segs]+0.5
  change.df <- data.frame(
    changepoint=change.vec)
  gg.change <- gg.count+
    geom_vline(aes(
      xintercept=changepoint),
      data=change.df)
  print(gg.change)
  ## Fit a peak model and plot it.
  fit <- PeakSegDisk::PeakSegFPOP_vec(z.rep.vec, 10.5)
  print(
    gg.change+
      geom_segment(aes(
        chromStart+0.5, mean, xend=chromEnd+0.5, yend=mean),
        color="green",
        data=fit$segments)
  )
  ## A pathological data set.
```

```
z.slow.vec <- 1:length(z.rep.vec)
fit.slow <- PeakSegDisk::PeakSegFPOP_vec(z.slow.vec, 10.5)
rbind(fit.slow$loss, fit$loss)
}
```

---

`plot.PeakSegFPOP_df`    *plot PeakSegFPOP df*

---

### **Description**

Plot a PeakSeg model with attached data.

### **Usage**

```
## S3 method for class 'PeakSegFPOP_df'
plot(x,
     ...)
```

### **Arguments**

x	x
...	...

### **Value**

a ggplot.

### **Author(s)**

Toby Dylan Hocking <toby.hocking@r-project.org> [aut, cre]

---

`plot.PeakSegFPOP_dir`    *plot PeakSegFPOP dir*

---

### **Description**

Plot a PeakSeg model with attached data.

### **Usage**

```
## S3 method for class 'PeakSegFPOP_dir'
plot(x,
     ...)
```



**Arguments**

x	x
...	...

**Value**

a ggplot.

**Author(s)**

Toby Dylan Hocking <toby.hocking@r-project.org> [aut, cre]

---

sequentialSearch\_dir *Compute PeakSeg model with given number of peaks*

---

**Description**

Compute the most likely peak model with at most the number of peaks given by `peaks.int`. This function repeated calls `PeakSegFPOP_dir` with different penalty values, until either (1) it finds the `peaks.int` model, or (2) it concludes that there is no `peaks.int` model, in which case it returns the next simplest model (with fewer peaks than `peaks.int`). The first pair of penalty values (0, Inf) is run in parallel via the user-specified future plan, if the `future.apply` package is available.

**Usage**

```
sequentialSearch_dir(problem.dir,
  peaks.int, verbose = 0)
```

**Arguments**

<code>problem.dir</code>	problemID directory in which <code>coverage.bedGraph</code> has already been computed. If there is a <code>labels.bed</code> file then the number of incorrect labels will be computed in order to find the target interval of minimal error penalty values.
<code>peaks.int</code>	int: target number of peaks.
<code>verbose</code>	numeric verbosity: if >0 then <code>cat</code> is used to print a message for each penalty.

**Value**

Same result list from `PeakSegFPOP_dir`, with an additional component "others" describing the other models that were computed before finding the optimal model with `peaks.int` (or fewer) peaks. Additional loss columns are as follows: `under`=number of peaks in smaller model during binary search; `over`=number of peaks in larger model during binary search; `iteration`=number of times `PeakSegFPOP` has been run.

**Author(s)**

Toby Dylan Hocking <toby.hocking@r-project.org> [aut, cre]

**Examples**

```

## Create simple 6 point data set discussed in supplementary
## materials. GFPOP/GPDPA computes up-down model with 2 peaks, but
## neither CDPA (PeakSegDP::cDPA) nor PDPA (jointseg)
r <- function(chrom, chromStart, chromEnd, coverage){
  data.frame(chrom, chromStart, chromEnd, coverage)
}
supp <- rbind(
  r("chr1", 0, 1, 3),
  r("chr1", 1, 2, 9),
  r("chr1", 2, 3, 18),
  r("chr1", 3, 4, 15),
  r("chr1", 4, 5, 20),
  r("chr1", 5, 6, 2)
)
data.dir <- file.path(tempfile(), "chr1-0-6")
dir.create(data.dir, recursive=TRUE)
write.table(
  supp, file.path(data.dir, "coverage.bedGraph"),
  sep="\t", row.names=FALSE, col.names=FALSE)

## register a parallel future plan to compute the first two
## penalties in parallel during the sequential search.
if(interactive() && requireNamespace("future"))future::plan("multisession")

## Compute optimal up-down model with 2 peaks via sequential search.
fit <- PeakSegDisk::sequentialSearch_dir(data.dir, 2L)

if(require(ggplot2)){
  ggplot()+
    theme_bw()+
    geom_point(aes(
      chromEnd, coverage),
      data=supp)+
    geom_segment(aes(
      chromStart+0.5, mean,
      xend=chromEnd+0.5, yend=mean),
      data=fit$segments,
      color="green")
}

```

---

summary.PeakSegFPOP\_dir

*summary PeakSegFPOP dir*


---

**Description**

Summary of [PeakSegFPOP\\_dir](#) object.

**Usage**

```
## S3 method for class 'PeakSegFPOP_dir'  
summary(object,  
  ...)
```

**Arguments**

object	object
...	...

**Value**

Data table with one row and columns describing model summary.

**Author(s)**

Toby Dylan Hocking <toby.hocking@r-project.org> [aut, cre]

---

wc2int

*wc2int*

---

**Description**

Convert wc output to integer number of lines.

**Usage**

```
wc2int(wc.output)
```

**Arguments**

wc.output	Character scalar: output from wc.
-----------	-----------------------------------

**Value**

integer

**Author(s)**

Toby Dylan Hocking <toby.hocking@r-project.org> [aut, cre]

---

writeBedGraph	<i>Write bedGraph file</i>
---------------	----------------------------

---

### Description

Write a data.frame in R to a bedGraph file on disk. This must be a plain text file with the following four columns: chrom (character chromosome name), chromStart (integer start position), chromEnd (integer end position), count (integer aligned read count on chrom from chromStart+1 to chromEnd); see also <https://genome.ucsc.edu/goldenPath/help/bedgraph.html>

### Usage

```
writeBedGraph(count.df,  
              coverage.bedGraph)
```

### Arguments

count.df            data.frame with four columns: chrom, chromStart, chromEnd, count.  
coverage.bedGraph    file path where data will be saved in plain text / bedGraph format.

### Value

NULL (same as write.table).

### Author(s)

Toby Dylan Hocking <toby.hocking@r-project.org> [aut, cre]

### Examples

```
library(PeakSegDisk)  
data(Mono27ac, envir=environment())  
coverage.bedGraph <- file.path(  
  tempfile(),  
  "H3K27ac-H3K4me3_TDHAM_BP",  
  "samples",  
  "Mono1_H3K27ac",  
  "S001YW_NCMLS",  
  "problems",  
  "chr11-60000-580000",  
  "coverage.bedGraph")  
dir.create(  
  dirname(coverage.bedGraph),  
  recursive=TRUE, showWarnings=FALSE)  
writeBedGraph(Mono27ac$coverage, coverage.bedGraph)  
fread.first(coverage.bedGraph, col.name.list$coverage)  
fread.last(coverage.bedGraph, col.name.list$coverage)
```

# Index

## \* datasets

ChIPreads, [2](#)

Mono27ac, [7](#)

cat, [17](#)

ChIPreads, [2](#)

coef.PeakSegFPOP\_df, [3](#)

coef.PeakSegFPOP\_dir, [3](#)

col.name.list, [4](#)

fread.first, [5](#)

fread.last, [6](#)

Mono27ac, [7](#)

PeakSegDisk (PeakSegFPOP\_dir), [10](#)

PeakSegFPOP\_df, [8](#), [10](#), [14](#)

PeakSegFPOP\_dir, [5](#), [6](#), [8](#), [10](#), [13](#), [15](#), [17](#), [18](#)

PeakSegFPOP\_file, [4](#), [10](#), [13](#)

PeakSegFPOP\_vec, [10](#), [14](#)

plot.PeakSegFPOP\_df, [16](#)

plot.PeakSegFPOP\_dir, [16](#)

sequentialSearch\_dir, [10](#), [17](#)

summary.PeakSegFPOP\_dir, [18](#)

wc2int, [19](#)

writeBedGraph, [8](#), [20](#)