```
## Warning:  replacing previous import 'ggplot2::Position' by 'BiocGenerics::Position'
when loading 'soGGi'
```

Visualisation, transformations and arithmetic operations for grouped genomic intervals

Thomas Carroll[1*]

[1] Bioinformatics Facility, MRC Clincal Sciences Centre;

*`thomas.carroll (at)imperial.ac.uk`

April 24, 2017

Abstract

The soGGi package provides tools to summarise sequence data, genomic signal and motif occurrence over grouped genomic intervals as well to perform complex subsetting, arithmetic operations and transformations on these genomic summaries prior to visualisation.

As with other Bioconductor packages such as CoverageView and seqPlots. soGGi plots average signal across groups of genomic regions. soGGI provides flexibity in both its data aquisition and visualisation. Single or paired-end BAM files, bigWigs, rleLists and PWM matrices can be provided as input alongside a GRanges object or BED file location. soGGi can plot summarises across actual size and normalised features such as genomic intervals of differing length (as with genes). The use of normalised size plots for genes can however obsure high resolution events around the TSS. To address this, combination plots can be created within soGGI allowing for fine detail at the edges of normalised regions.

Arithmetic operation and transformations can be easily performed on soGGi objects allowing for rapid operations between profiles such as the substraction of input signal or quantile normalisation of replicates within a group.

soGGi integrates the ggplot2 package and add functionality to rapidly subset, facet and colour profiles by their overlaps with GenomicRanges objects or grouping by metadata column IDs.

The plotting, arithmetic and tranformation functions within soGGi allow for rapid evaluation of groups of genomic intervals and provide a toolset for user-defined analysis of these summaries over groups.

Contents

1 Standard workflow

1.1 The soggi function

The `regionPlot()` function is used to summarise signal, reads or PWM occurrence over grouped genomic intervals. Input can be BAM, bigWig or a PWM matrix and the regions to summarise over a character string of path to BED file or a GRanges object. The full set of soggi function arguments can be found in help pages.

In this example, signal coverage is summarised from a BAM file using the defaults. The default style of region plot is to produce a normalised by size region plot.

```
library(soGGi)
chipExample <- regionPlot("pathToBAM/mybam.bam",myGRangesObject,format="bam")
```

A pre-computed data set is included in the package containing averages profiles created with command above for DNAse, Pol2, H3k9ac and H3k3me3. The object itself cantains all counts along interval region windows in assays slots and information of the samples in metadata slot accessible by `assays()` and `metadata()` respectively.

```
library(soGGi)
data(chipExampleBig)
chipExampleBig

## class: ChIPprofile
## dim: 201 300
## metadata(2): names AlignedReadsInBam
## assays(10): '' '' ... '' ''
## rownames: NULL
## rowData names(5): name biotype Feature giID giID.1
## colnames(300): Start-1 Start-2 ... End+99 End+100
## colData names(0):
```

This object contains 10 sets of profiles for 200 genes. The object can be subset using [[to select samples of interest.

```
chipExampleBig[[1]]

## class: ChIPprofile
## dim: 201 300
## metadata(2): names AlignedReadsInBam
## assays(1): ''
## rownames: NULL
## rowData names(5): name biotype Feature giID giID.1
## colnames(300): Start-1 Start-2 ... End+99 End+100
## colData names(0):

chipExampleBig$highdnase

## class: ChIPprofile
## dim: 201 300
## metadata(2): names AlignedReadsInBam
## assays(1): ''
## rownames: NULL
## rowData names(5): name biotype Feature giID giID.1
## colnames(300): Start-1 Start-2 ... End+99 End+100
## colData names(0):
```

Similarly profile objects can be concatenated or bound together using c and rbind.

```
c(chipExampleBig[[1]],chipExampleBig[[2]])

## class: ChIPprofile
## dim: 201 300
## metadata(2): names AlignedReadsInBam
## assays(2): '' ''
## rownames: NULL
## rowData names(5): name biotype Feature giID giID.1
## colnames(300): Start-1 Start-2 ... End+99 End+100
## colData names(0):

rbind(chipExampleBig[[1]],chipExampleBig[[2]])

## class: ChIPprofile
## dim: 402 300
## metadata(2): names AlignedReadsInBam
## assays(1): ''
## rownames: NULL
## rowData names(5): name biotype Feature giID giID.1
## colnames(300): Start-1 Start-2 ... End+99 End+100
## colData names(0):
```
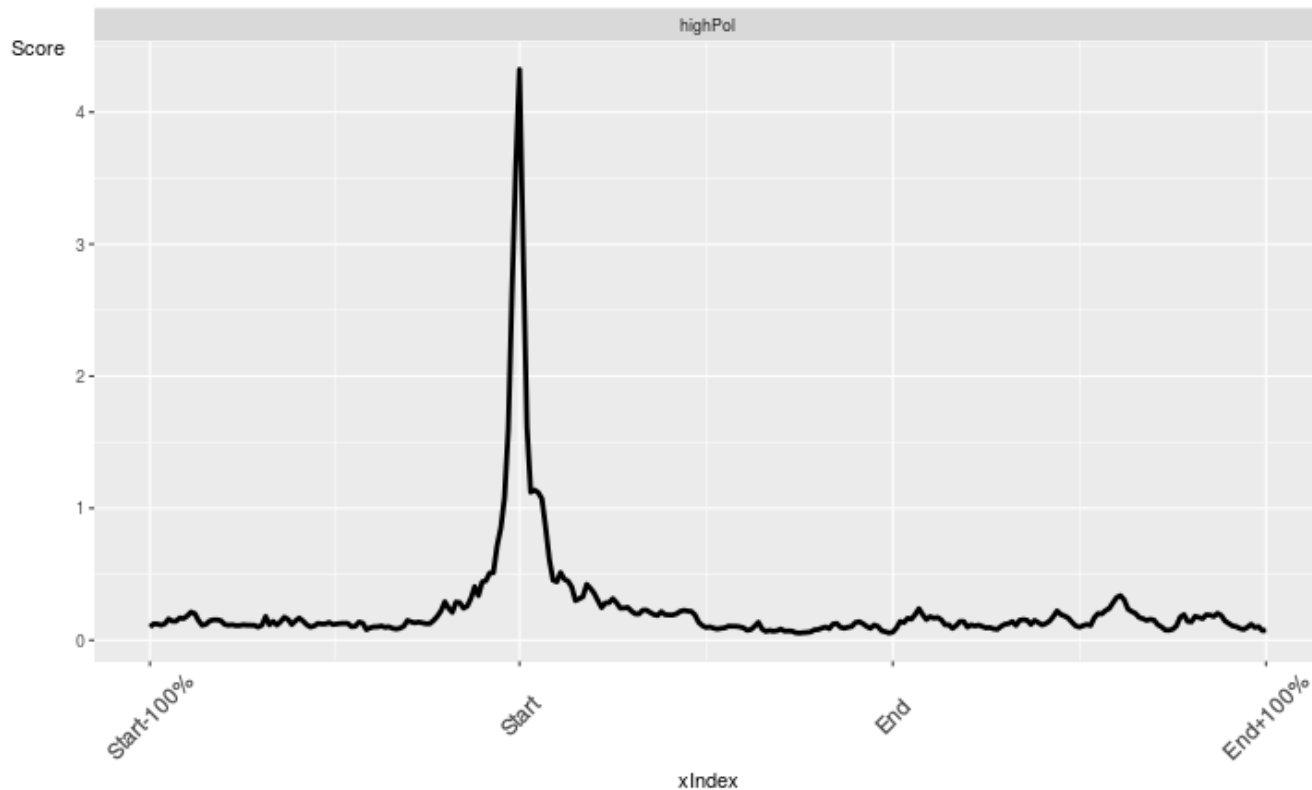
Figure 1: **Example profile plot.** The plot generated by plotRegion() function on a single sample soGGi object. The x-axis shows the normalised length and the y-axis shows the coverage in windows. A Clear peak around the TSS can be observed for this Pol2 ChIP-seq profile.

1.2 Plotting profiles

The `plotRegion()` function is used to produce profile plots. `plotRegion()` uses ggplot2 to generate plots and so returned object can be highly customisable using ggplot2 methods.

```
plotRegion(chipExampleBig[[3]])
```

When dealing with objects with multiple samples, the arguments groupBy and colourBy specify whether to facet or colour by Sample/Group respectively.

```
library(ggplot2)
plotRegion(chipExampleBig,colourBy="Sample", groupBy="Sample", freeScale=TRUE)
```

Here some samples can be seen to be noisy. Windsorisation can be applied when plotting using the outliers argument.

```
library(ggplot2)
plotRegion(chipExampleBig,colourBy="Sample", outliers=0.01, groupBy="Sample",freeScale=TRUE
```

The `plotRegion()` can also be used to group genomic intervals while plotting using the `gts` argument. The `gts` argument either takes a GRanges object or a list of character vectors and the `summariseBy`
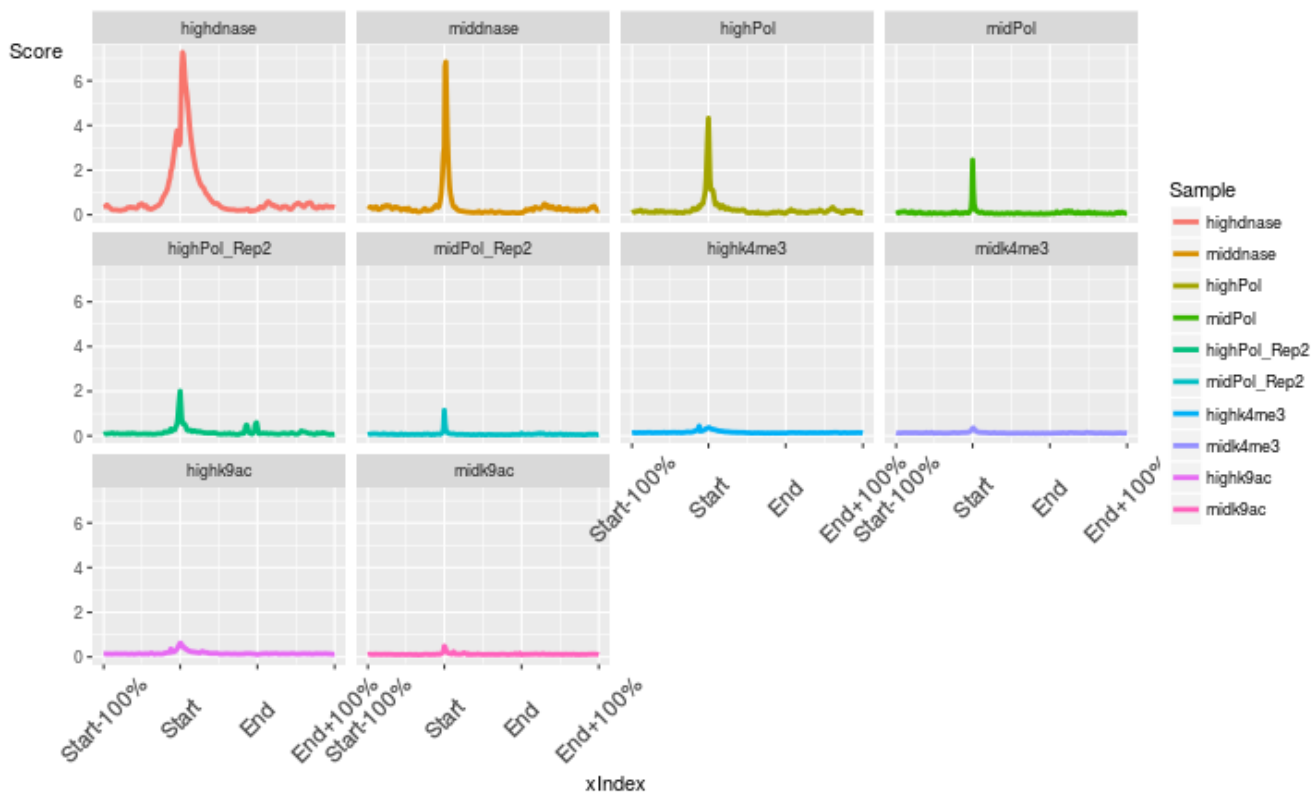
Figure 2: **Multi-Sample profile plot.** Here multiple samples are plotted simultaneously. Enrichment around TSS can be seen for all profiles with H3k9ac showing more enrichment in the gene body.

argument to specify metadata to use.

```
library(GenomicRanges)
subsetsCharacter <- list(first25 = (as.vector(rowRanges(chipExampleBig[[1]])$name[1:25])),

subsetsGRanges <- GRangesList(low=(rowRanges(chipExampleBig[[1]])[1:25]), high=rowRanges(ch

plotRegion(chipExampleBig[[1]],gts=subsetsCharacter,summariseBy = "name")
plotRegion(chipExampleBig[[1]],gts=subsetsGRanges)
```
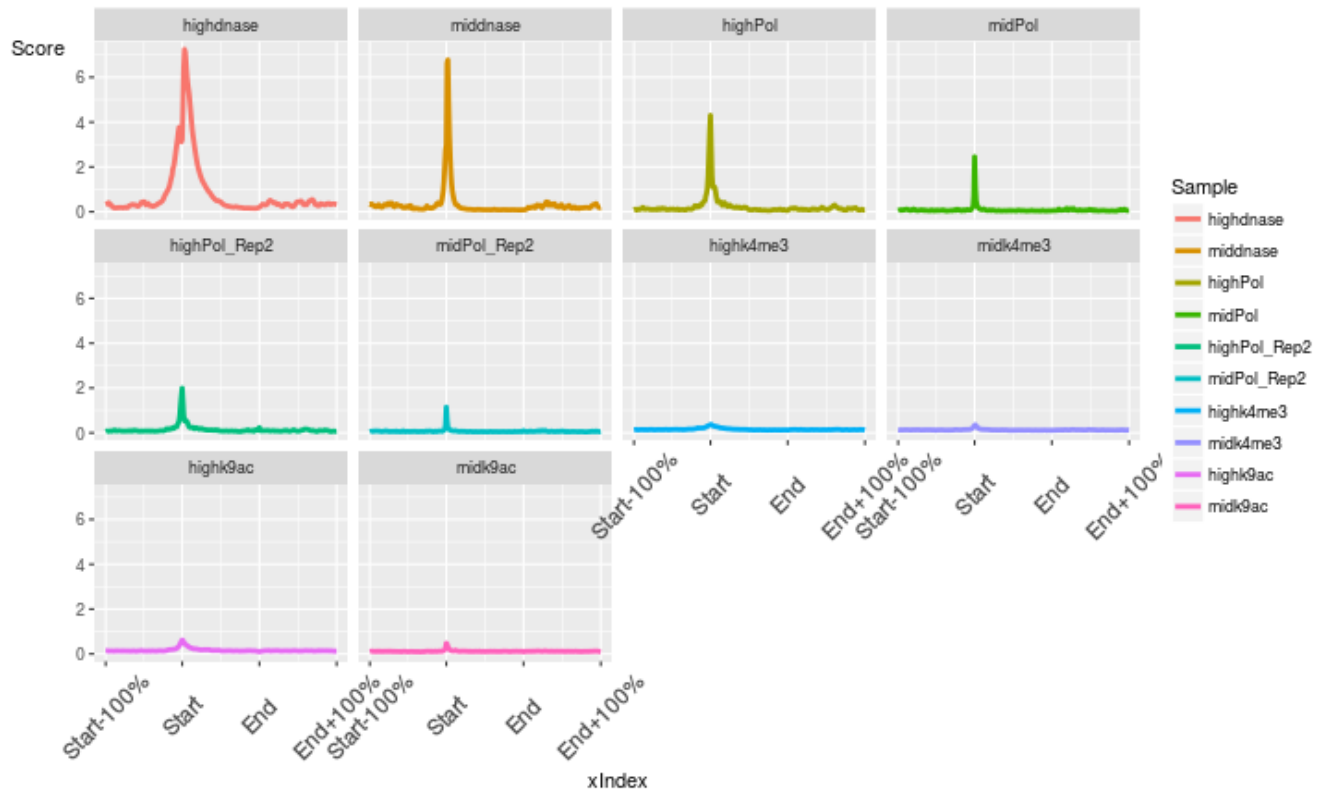
Figure 3: **Multi-Sample profile plot with windsorisation.** This multi-sample plot has windsorisation applied to outliers. The resulting plot is smoother than that seen in figure 2.

2 Transformations and arithmetic operations

2.1 Simple arithmetic operations on grouped profiles

Common arithmetic operations and tranformations can be used with soGGi profile objects allowing for further analysis post summarisation and iteratively over visualisations.

Here we summarise RNApol2 high and low and compare between replicates.

```
pol_Profiles <- c((chipExampleBig$highPol+chipExampleBig$midPol)
, (chipExampleBig$highPol_Rep2+chipExampleBig$midPol_Rep2))
plotRegion(pol_Profiles,colourBy="Sample",outliers=0.01, groupBy="Sample", freeScale=TRUE)
```

Common normalisations, log transformations and other mathematical functions such as mean() are also implemented to allow for the comparison within and between profiles.

In this example the profiles are log2 transformed with zeros being assigned the minimum value for that region.

```
log2Profiles <- log2(chipExampleBig)
```
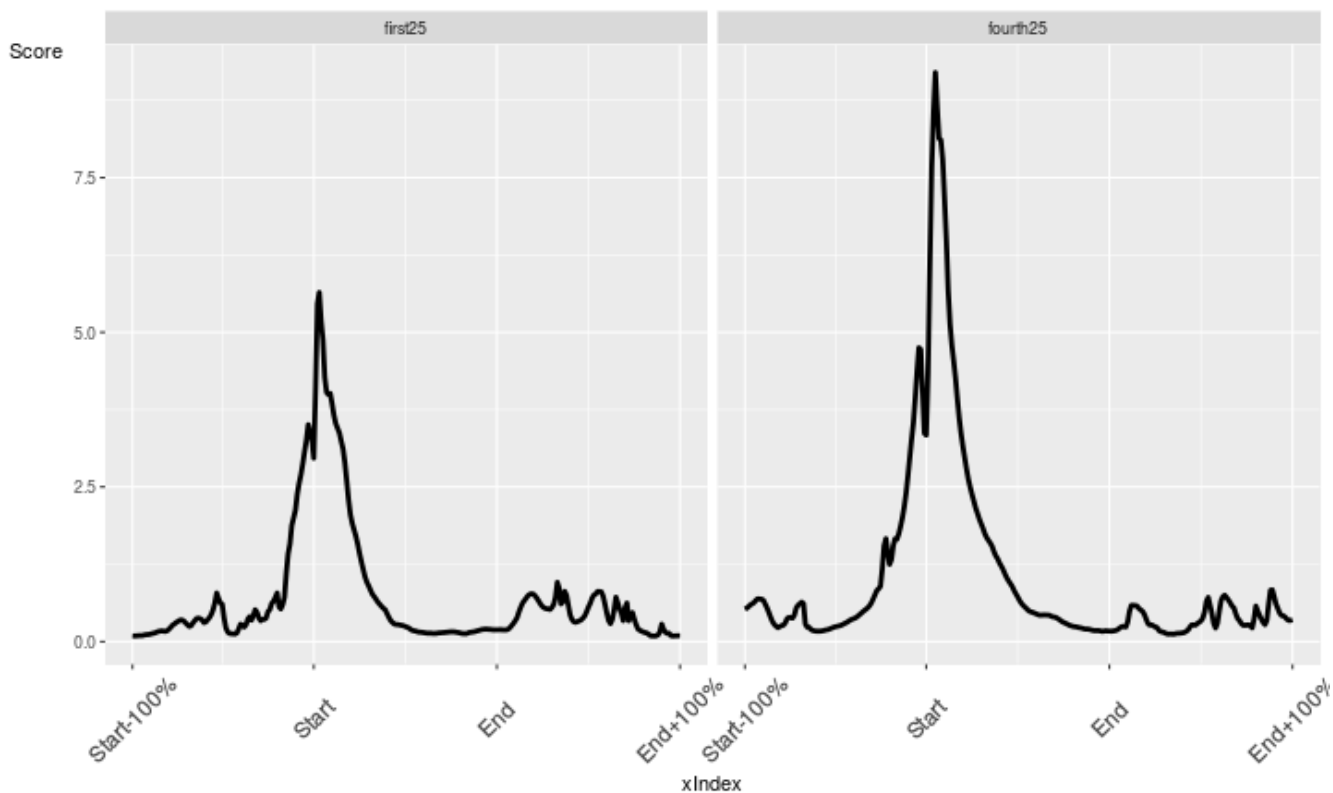
Figure 4: **DNAse grouped genomic intervals plot.** This plots shows the plot for DNAse signal across normalised regions with separate profiles for each group of genomic intervals defined in gts. .

```
plotRegion(log2Profiles,colourBy="Sample",outliers=0.01, groupBy="Sample",freeScale=TRUE)
```

From this log2 transformed data we can look at the difference between Pol2 profiles

```
log2Polhigh <- mean(log2Profiles$highPol, log2Profiles$highPol_Rep2)
log2Polmid <- mean(log2Profiles$midPol, log2Profiles$midPol_Rep2)
diffPol <- log2Polhigh-log2Polmid


diffh3k9ac <- log2Profiles$highk9ac-log2Profiles$midk9ac


plotRegion(c(diffPol,diffh3k9ac),colourBy="Sample",outliers=0.01, groupBy="Sample", freeSca
```

Quantile normalisation of allow windows in regions between samples can allow for better better visual comparison of changes between conditions when dealing with larger numbers of replicates. Here for demonstration we apply it two samples but with real data higher sample numbers would be recommended.

```
normHighPol <- normalise(c(chipExampleBig$highPol, chipExampleBig$highPol_Rep2), method="qu
normMidPol <- normalise(c(chipExampleBig$midPol, chipExampleBig$midPol_Rep2), method="quant
```
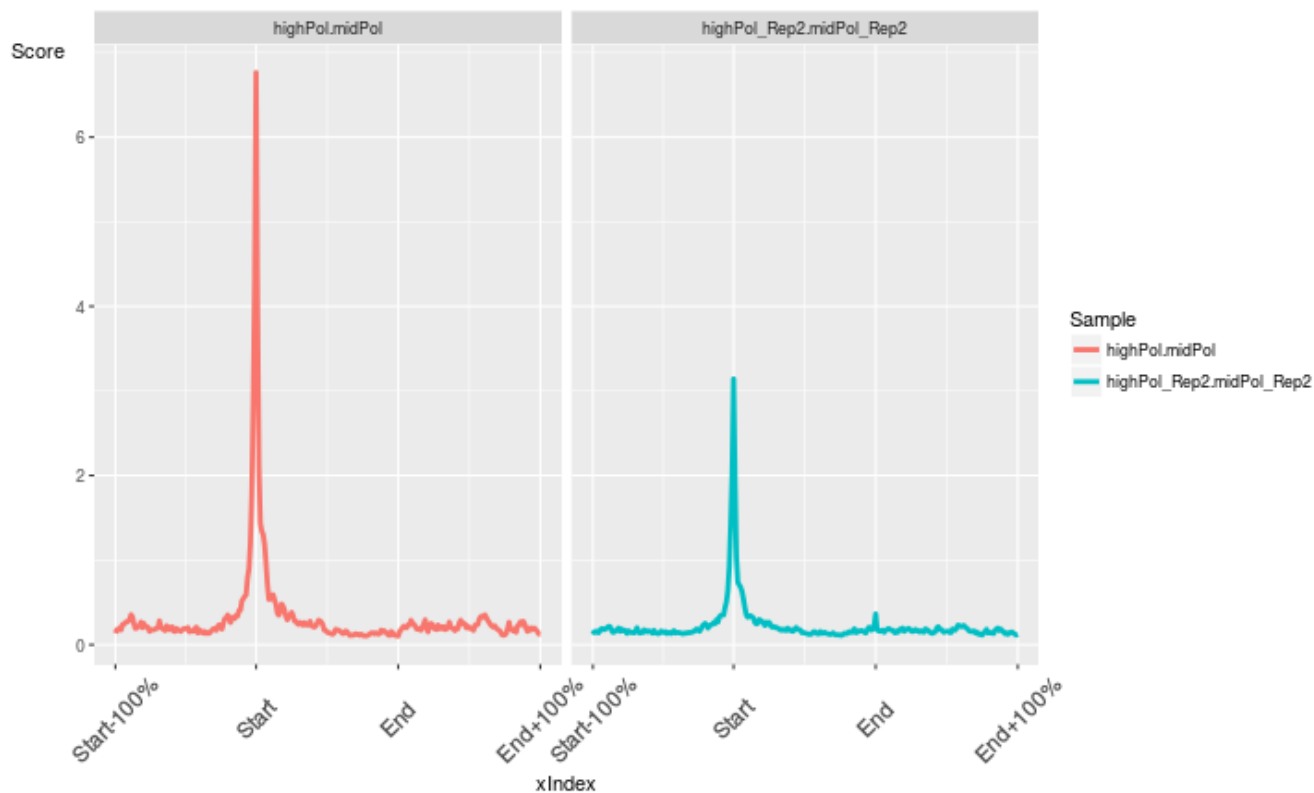
Figure 5: **Plotting arithmetic results.** This plot demonstrates the ability to plot the profiles generated by the results of arithmetic operations. Here data is combined within replicate number and results plotted.

```
normPol <-c(normHighPol$highPol, normHighPol$highPol_Rep2, normMidPol$midPol, normMidPol$m:
plotRegion(normPol,colourBy="Sample",outliers=0.01, groupBy="Sample", freeScale=TRUE)
```

3   Creating GRanges combinations for plotting

A common operation in the analysis of summaries over genomic intervals is to compare between different sets of grouped genomic intervals. soGGi includes helper functions to deal with grouped genomic intervals.

The `groupByOverlaps()` function creates all combinations of grouped genomic intervals from GRanges-Lists and so is useful to evaluate how summaries change over subclasses of genomic intervals (such as over co-occuring peak sets)

The `findconsensusRegions()` and `summitPipeline()` functions identifies consensus regions between GRanges objects and re-summits consensus region sets respectively. This approach has been previously implemented to identify reproducible peak sets between biological replciates.
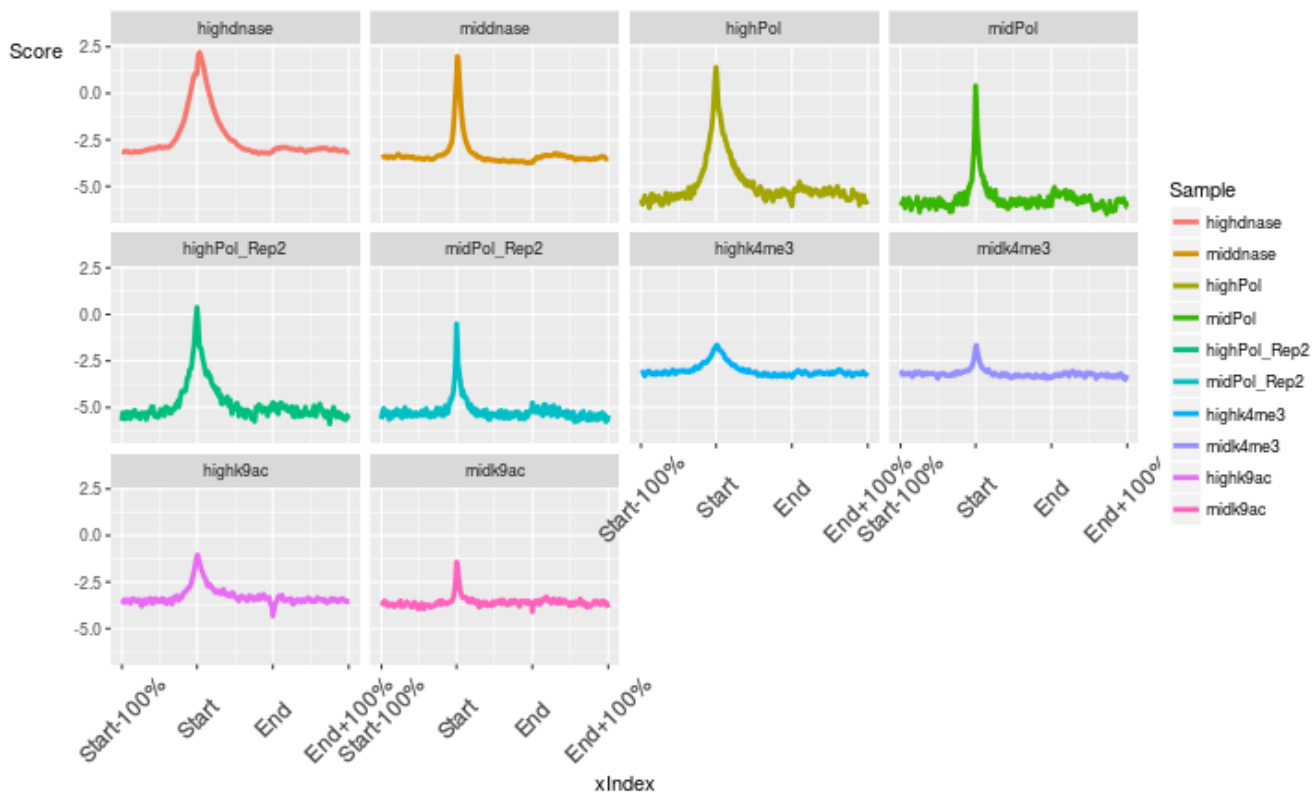
Figure 6: **Plotting log2 transformed profiles.** This plots shows the resulting profiles after log2 transformation of ChIP-data across antibodies.

3.1 Grouping genomic interval sets and plotting results

In this example, two antibodies for the transcription factor Ikaros are used to plot the signal over common and unique peaks for each antibody. First the peaks sets are defined using `groupByOverlaps()`

```
data(ik_Example)
ik_Example

## $ha
## GRanges object with 200 ranges and 2 metadata columns:
##          seqnames               ranges strand |          ID     Score
##             <Rle>            <IRanges>  <Rle> |    <factor> <factor>
##     [1]      chr1 [5218713, 5219618]       * |   MACS_peak_4    60.84
##     [2]      chr1 [6372427, 6373665]       * |   MACS_peak_7   438.68
##     [3]      chr1 [6456792, 6457633]       * |   MACS_peak_9    96.87
##     [4]      chr1 [7080191, 7080946]       * |  MACS_peak_15    57.82
##     [5]      chr1 [9225736, 9227873]       * |  MACS_peak_17   166.57
##     ...       ...                  ...     ... .          ...      ...
##   [196]      chr1 [36765816, 36767310]     * | MACS_peak_130    127.4
##   [197]      chr1 [36822798, 36824995]     * | MACS_peak_131  1387.29
##   [198]      chr1 [36995290, 36996196]     * | MACS_peak_132    73.84
```
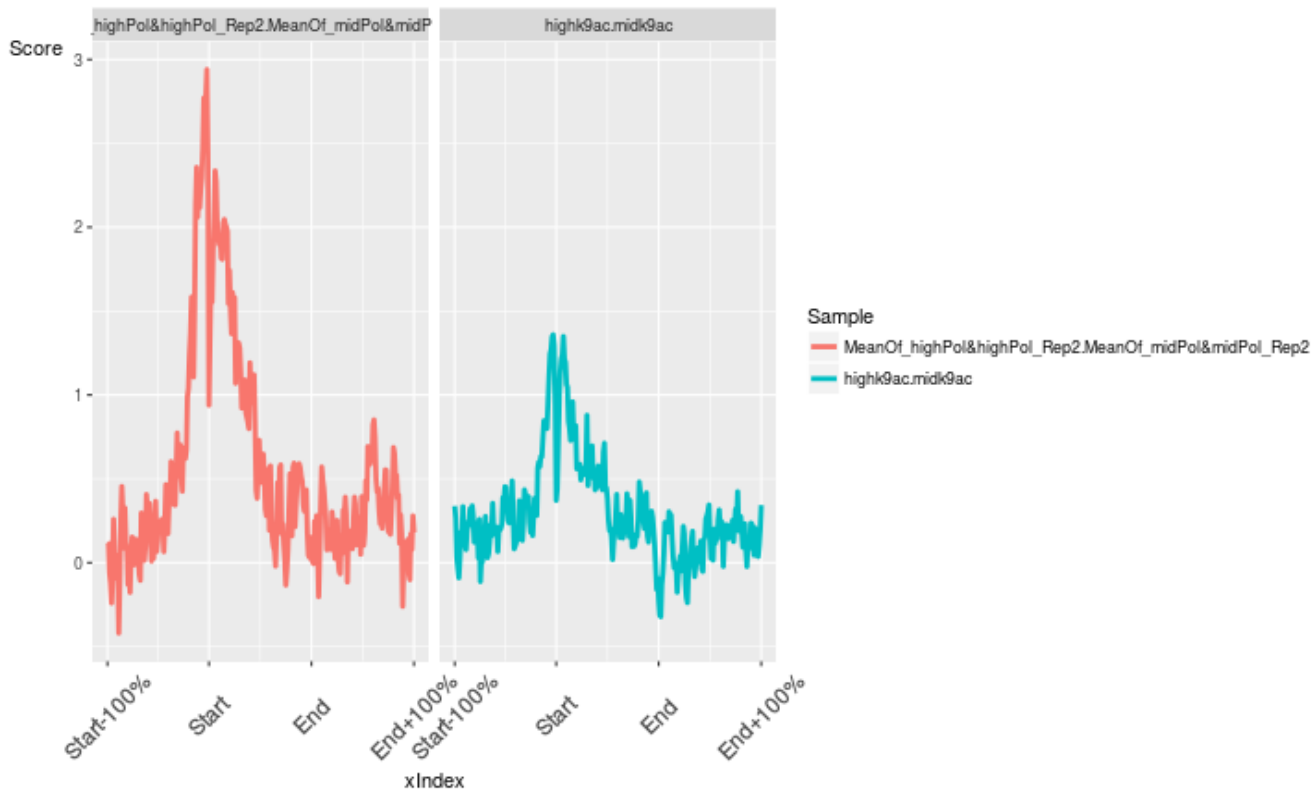
Figure 7: **Plotting differentials.** In this plot the log2 difference between high and low samples for H3k9ac and Pol2 replicates is shown.

```
##    [199]     chr1 [37044729, 37045925]     * | MACS_peak_135    193.63
##    [200]     chr1 [37356201, 37357028]     * | MACS_peak_138     66.39
##    -------
##   seqinfo: 21 sequences from an unspecified genome; no seqlengths
##
## $endo
## GRanges object with 200 ranges and 2 metadata columns:
##         seqnames                 ranges strand |            ID     Score
##            <Rle>              <IRanges>  <Rle> |      <factor> <factor>
##     [1]     chr1  [4695718, 4695917]        * |    MACS_peak_2    98.76
##     [2]     chr1  [4775045, 4775288]        * |    MACS_peak_3   100.91
##     [3]     chr1  [4775403, 4775658]        * |    MACS_peak_4     50.4
##     [4]     chr1  [4775664, 4776101]        * |    MACS_peak_5    150.6
##     [5]     chr1  [4797674, 4798084]        * |    MACS_peak_6   651.89
##     ...      ...                    ...    ... .            ...      ...
##   [196]     chr1 [24740900, 24741101]       * | MACS_peak_211   301.48
##   [197]     chr1 [30929613, 30929871]       * | MACS_peak_220   106.95
##   [198]     chr1 [30930510, 30931164]       * | MACS_peak_221  1604.65
##   [199]     chr1 [31005552, 31005763]       * | MACS_peak_222   174.71
```
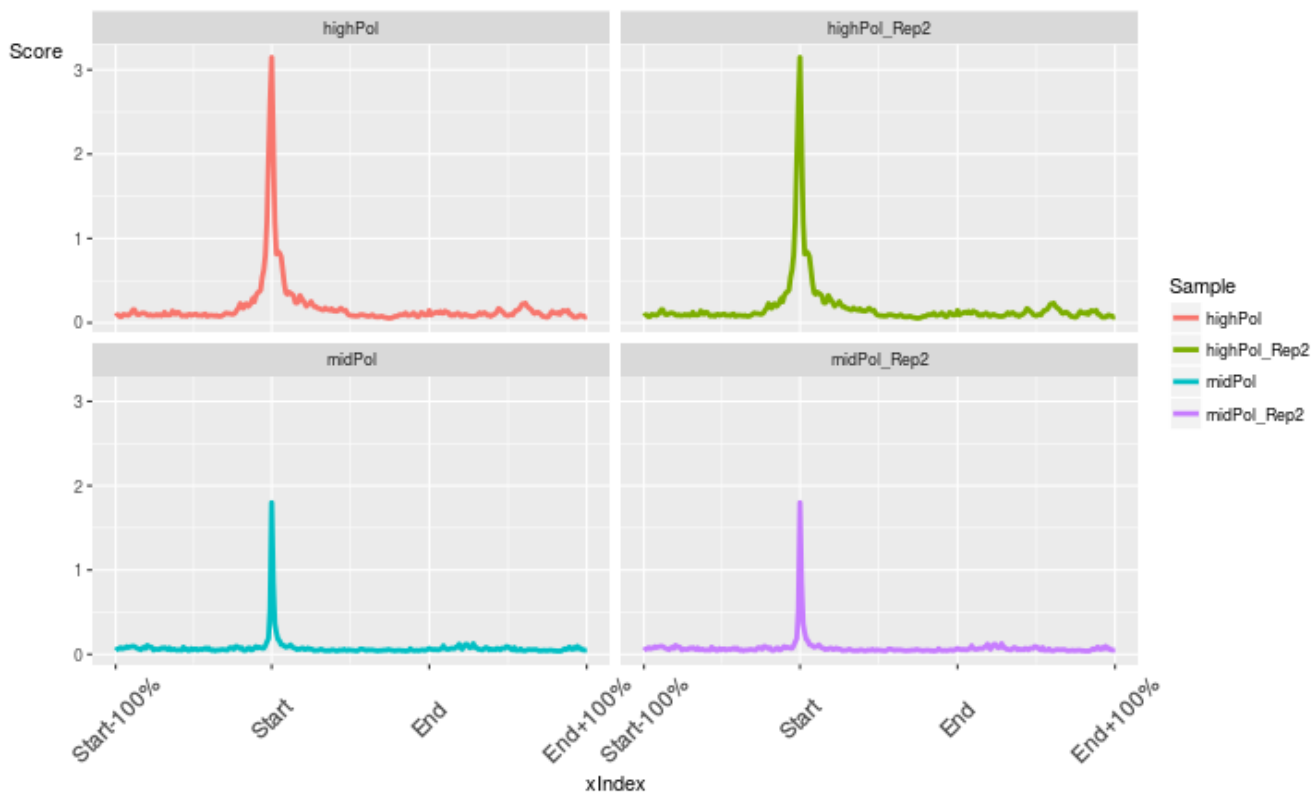
Figure 8: **Quantile normalisation.** In this toy example, data has been quantiled normalised within groups and the results plotted. This demonstrates the uniformity in data following quantile normalisation.

```
##     [200]     chr1 [31006685, 31007192]        * | MACS_peak_223    529.42
##     -------
##     seqinfo: 21 sequences from an unspecified genome; no seqlengths

peakSetCombinations <- groupByOverlaps(ik_Example)
peakSetCombinations

## $endo
## GRanges object with 100 ranges and 1 metadata column:
##           seqnames                 ranges strand | grangesGroups
##              <Rle>              <IRanges>  <Rle> |      <factor>
##     [1]      chr1   [4695718, 4695917]        * |          endo
##     [2]      chr1   [4775045, 4775288]        * |          endo
##     [3]      chr1   [4775403, 4775658]        * |          endo
##     [4]      chr1   [4775664, 4776101]        * |          endo
##     [5]      chr1   [4797674, 4798084]        * |          endo
##     ...       ...                  ...      ... .           ...
##    [96]      chr1 [21534723, 21534898]        * |          endo
##    [97]      chr1 [21635845, 21636030]        * |          endo
```

```
##     [98]      chr1 [21705049, 21705198]       * |           endo
##     [99]      chr1 [21739213, 21739804]       * |           endo
##    [100]      chr1 [21757663, 21757886]       * |           endo
##    -------
##   seqinfo: 21 sequences from an unspecified genome; no seqlengths
##
## $ha
## GRanges object with 137 ranges and 1 metadata column:
##          seqnames              ranges strand | grangesGroups
##             <Rle>           <IRanges>  <Rle> |      <factor>
##     [1]     chr1   [5218713, 5219618]      * |            ha
##     [2]     chr1   [6372427, 6373665]      * |            ha
##     [3]     chr1   [6456792, 6457633]      * |            ha
##     [4]     chr1   [7080191, 7080946]      * |            ha
##     [5]     chr1   [9225736, 9227873]      * |            ha
##     ...      ...                 ...    ... .           ...
##   [133]     chr1 [89527991, 89530046]      * |            ha
##   [134]     chr1 [89768074, 89769026]      * |            ha
##   [135]     chr1 [89837445, 89838176]      * |            ha
##   [136]     chr1 [90598333, 90599360]      * |            ha
##   [137]     chr1 [91540499, 91541383]      * |            ha
##    -------
##   seqinfo: 21 sequences from an unspecified genome; no seqlengths
##
## $`ha-endo`
## GRanges object with 63 ranges and 1 metadata column:
##          seqnames              ranges strand | grangesGroups
##             <Rle>           <IRanges>  <Rle> |      <factor>
##     [1]     chr1   [4847117, 4848878]      * |       ha-endo
##     [2]     chr1   [5072908, 5073864]      * |       ha-endo
##     [3]     chr1   [6204023, 6205748]      * |       ha-endo
##     [4]     chr1   [6252357, 6253568]      * |       ha-endo
##     [5]     chr1   [6395852, 6396924]      * |       ha-endo
##     ...      ...                 ...    ... .           ...
##    [59]     chr1 [24012021, 24013019]      * |       ha-endo
##    [60]     chr1 [24684410, 24686139]      * |       ha-endo
##    [61]     chr1 [24740575, 24741381]      * |       ha-endo
##    [62]     chr1 [30929409, 30931590]      * |       ha-endo
##    [63]     chr1 [31005112, 31007518]      * |       ha-endo
##    -------
##   seqinfo: 21 sequences from an unspecified genome; no seqlengths
```

The output from `groupByOverlaps()` can then be used to subset precomputed profiles of HA and Endogenous ChIP signal. Here we apply a log2 transformation to the data before plotting and cleaning
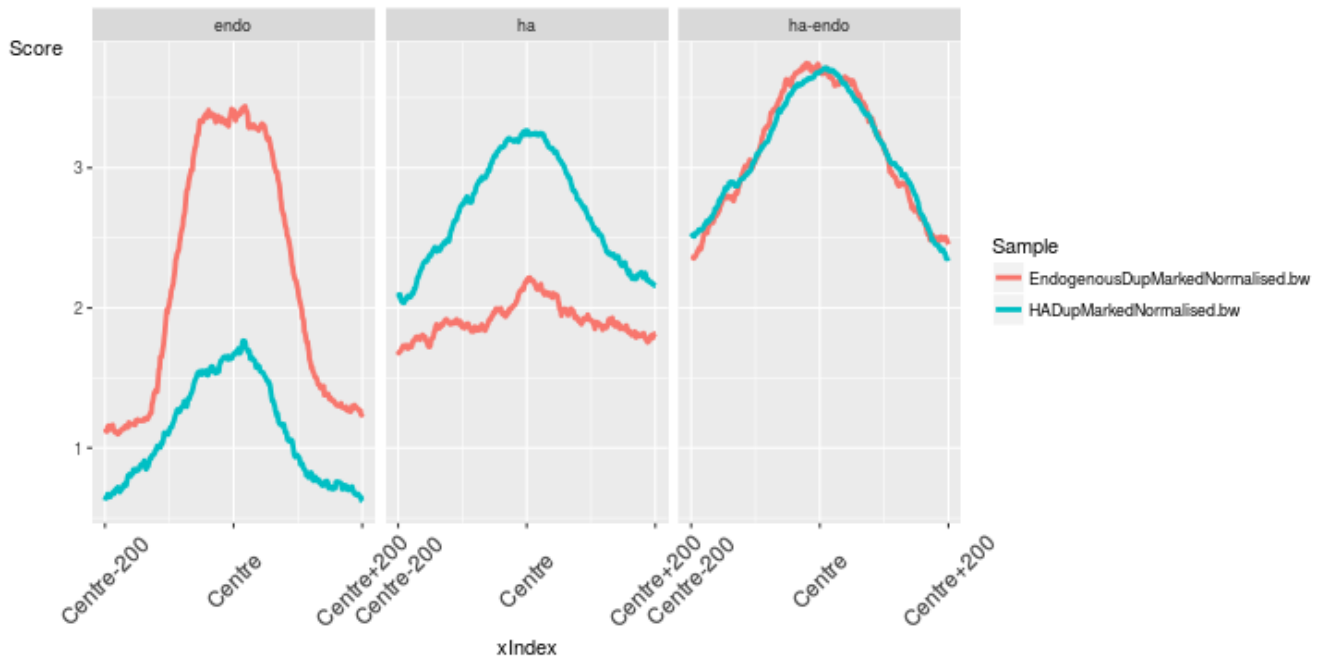
Figure 9: **Signal over common and unique Ikaros peaks.** This plot shows that, as expected, common and unique peaks show different profiles for the Ikaros antibodies.

up profile with windsorisation

```
data(ik_Profiles)
ik_Profiles

## class: ChIPprofile
## dim: 4800 401
## metadata(2): names AlignedReadsInBam
## assays(2): '' ''
## rownames: NULL
## rowData names(3): ID Score giID
## colnames(401): Point_Centre-200 Point_Centre-199 ... Point_Centre199
##   Point_Centre200
## colData names(0):

log2Ik_Profiles <- log2(ik_Profiles)
plotRegion(log2Ik_Profiles,outliers=0.01,gts=peakSetCombinations, groupBy="Group",colourBy=
plotRegion(log2Ik_Profiles[[1]] - log2Ik_Profiles[[2]] ,outliers=0.01,gts=peakSetCombinatio
```

This confirms that common and unique peaksets have different levels of the separate antibody signals. This can be better demonstrated by subtracting to signal sets from each other and re-plotting over groups as seen in the final example.
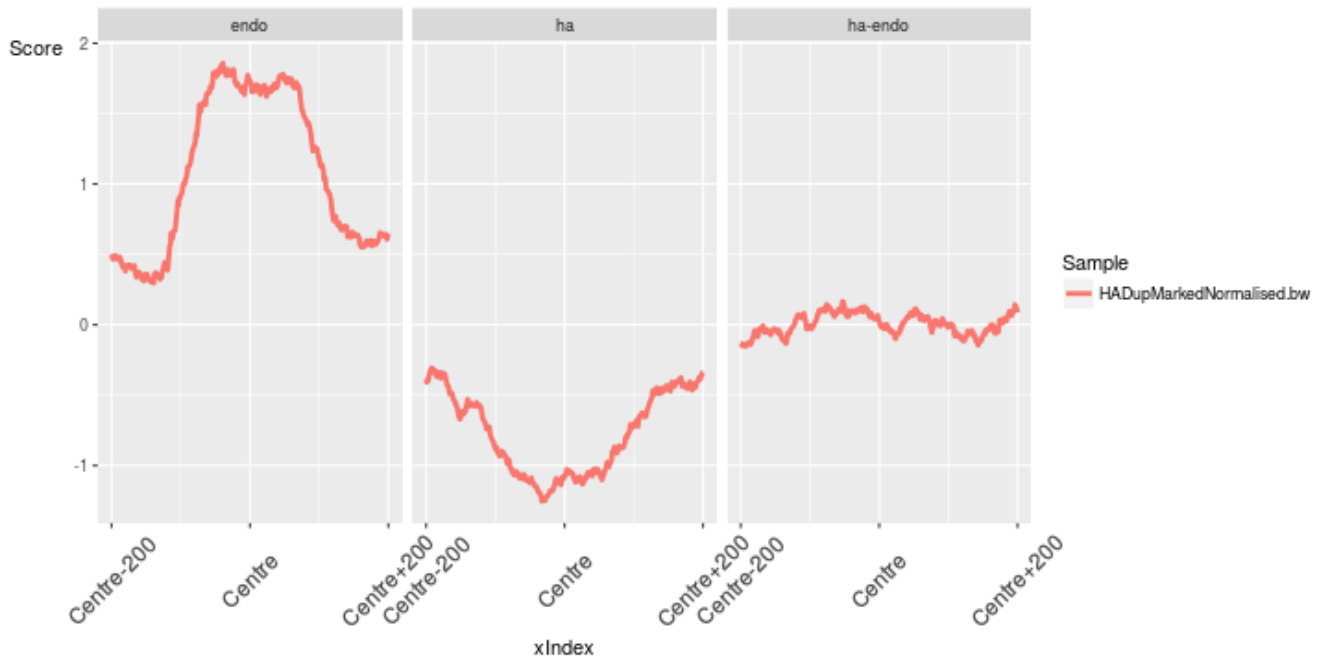
Figure 10: **Differential Ikaros profiles over common and unique sets.** The plot of difference in log2 HA and Endogenous Ikaros signal over peaks shows the expected difference in Ikaros antibody signaland uniformity of signal of common peaks.

```
toLatex(sessionInfo())
```

- R version 3.4.0 (2017-04-21), `x86_64-pc-linux-gnu`
- Locale: `LC_CTYPE=en_US.UTF-8`, `LC_NUMERIC=C`, `LC_TIME=en_US.UTF-8`, `LC_COLLATE=C`, `LC_MONETARY=en_US.UTF-8`, `LC_MESSAGES=en_US.UTF-8`, `LC_PAPER=en_US.UTF-8`, `LC_NAME=C`, `LC_ADDRESS=C`, `LC_TELEPHONE=C`, `LC_MEASUREMENT=en_US.UTF-8`, `LC_IDENTIFICATION=C`
- Running under: `Ubuntu 16.04.2 LTS`
- Matrix products: default
- BLAS: `/home/biocbuild/bbs-3.5-bioc/R/lib/libRblas.so`
- LAPACK: `/home/biocbuild/bbs-3.5-bioc/R/lib/libRlapack.so`
- Base packages: base, datasets, grDevices, graphics, methods, parallel, stats, stats4, utils
- Other packages: Biobase 2.36.0, BiocGenerics 0.22.0, DelayedArray 0.2.0, GenomeInfoDb 1.12.0, GenomicRanges 1.28.0, IRanges 2.10.0, S4Vectors 0.14.0, SummarizedExperiment 1.6.0, ggplot2 2.2.1, knitr 1.15.1, matrixStats 0.52.2, soGGi 1.8.0
- Loaded via a namespace (and not attached): BiocParallel 1.10.0, BiocStyle 2.4.0, Biostrings 2.44.0, GenomeInfoDbData 0.99.0, GenomicAlignments 1.12.0, Matrix 1.2-9, RColorBrewer 1.1-2, RCurl 1.95-4.8, Rcpp 0.12.10, Rsamtools 1.28.0, ShortRead 1.34.0, XML 3.98-1.6, XVector 0.16.0, backports 1.0.5, bitops 1.0-6, chipseq 1.26.0, colorspace 1.3-2, compiler 3.4.0, digest 0.6.12, evaluate 0.10, grid 3.4.0, gtable 0.2.0, highr 0.6, htmltools 0.3.5, hwriter 1.3.2, labeling 0.3, lattice 0.20-35, latticeExtra 0.6-28, lazyeval 0.2.0, magrittr 1.5, munsell 0.4.3, plyr 1.8.4, preprocessCore 1.38.0, reshape2 1.4.2, rmarkdown 1.4, rprojroot 1.2,

rtracklayer 1.36.0, scales 0.4.1, stringi 1.1.5, stringr 1.2.0, tibble 1.3.0, tools 3.4.0, yaml 2.1.14, zlibbioc 1.22.0