Processing and Visualisation of High-Throughput Sequencing Data with ShortRead and HilbertVis

Simon Anders

European Bioinformatics Institute,
Hinxton, Cambridge, UK

sanders@fs.tum.de

version 2: 2009-06-30

Abstract

This document serves a double purpose: (i) It explains the use of the Bioconductor packages *HilbertVis* and *HilbertVisGUI*. This pair of packages offers a tool to visualise very long one-dimensional data vectors (with up to billions of entries) in an efficient fashion that allows to get a quick impression of the spatial distribution and rough shape of the features present in the data. This is especially useful in the initial exploration of high-resolution position-dependent genomic data, such as tiling array or ChIP-Seq data. (ii) It provides a specific example by walking the reader through the task of processing ChIP-Seq data using the stand-alone alignment tool Maq and the Bioconductor packages Biostrings, ShortRead and HilbertVis/HilbertVisGUI.

Note: If you are only interested in the use of the `HilbertVis`/`HilbertVisGUI` packages, you can skip the first section and start reading at Section 6.

Note: If you have trouble installing the package `HilbertVisGUI`, read the file `INSTALL` in the package.

1 Introduction

Bioconductor offers substantial support for genomic experiments, which, for the case of microarray platforms, including tiling arrays, has reached maturity already a while ago. For data from high-throughput sequencing experiments, development of new tools is currently (mid 2008) ongoing. In this document, I would like to show what can already been done by re-doing step for step the analysis of an already published Solexa ChIP-Seq experiment.

I use this to demonstrate some aspects of the ShortRead package (by M. Morgan, [Mor]) and the use of my packages "HilbertVis" and "HilbertVisGUI". ShortRead introduces data structures to represent aligned short sequence reads and offers functions to read in such data from files output by the SolexaPipeline (the software that Illumina provides with its GenomeAnalyzer machine) or by Maq (a stand-alone alignment program, [LRD08]). ShortRead's data structures are based on the infrastructure provided by Biostrings.

As an example, we use data from a published study, Ref. [BCC+07], on histone methylations in the human genome. Although this was not the main focus of that study, we re-analyse the data for histone methylation patterns H3K4me1 and H3K4me3, as these are data sets of manageable size. We first re-do the alignment with Maq, then use ShortRead to read the result into R and then visualise the data with HilbertVis.

2 The example data

The authors of our example have deposited their raw data in the NCBI's Provisional Short Read Archive (SRA, `http://www.ncbi.nlm.nih.gov/Traces/sra/sra.cgi`) under accession number SRA000206. Use the "Submissions" tab in the archive's "Download Facility" to find the submission (under "SRA000"). You will get to a directory that contains all the data as output by Bustard, the base-calling program in the SolexaPipeline, as well as a number of fairly self-explanatory XML files with meta-data.[1] There are 3 lanes for H3K4me1 and 7 lanes for H3K4me3.

I have used Maq to align the reads from these lanes against the human reference genome as provided on Ensembl. Doing so requires converting the `_seq.txt` and `_prb.txt` files for the lanes to the Sanger Institute's FASTQ format and on to Maq's BFQ (binary FASTQ format). Likewise, the reference genome is converted to one large BFA (binary FASTA) file. Then, the `maq map` command may be used to perform the alignment. As these steps are described in the documentation on the Maq web site, I do not go into detail here.[2] In the end, we have, as output from Maq, a mapping file for each lane. I have put these files onto my web page. So, if you want to try out the following steps for yourself, please download them from `http://www.ebi.ac.uk/~anders/ShortReadExampleData/`. Note, however, that you should use a machine with at least 4 GB of RAM to perform the examples.

Of course, Maq is not the only choice to align the reads to the genome. You may as well use Eland (the alignment program that comes with the SolexaPipeline), which can be read in as well by ShortRead, so that the following steps apply to this case as well. Within certain limits, the matching functionality of the Biostrings package allows you to even do everything within R. Finally, there are other alignment tools specialised for high-throughput sequences. Recently, the `ShortRead` package's `readAligned` function was extended and it can now parse the output formats of several popular tools, including Eland, Maq, SOAP, Bowtie, and the SAM format used e.g. by BWA.

3 Reading in the alignment

Assume that the current working directory contains two sub-directories, names `H3K4me1` and `H3K4me3`. Then we can read in all the files of pattern `run`*xx*`lane`*x*`.map` with the following commands:

```
> library("ShortRead")
> maps.me1 <- sapply( list.files( "H3K4me1", "run.*lane.\\.map" ),
+    function(filename) readAligned( "H3K4me1", filename, type="MAQMapShort" ) )
> maps.me3 <- sapply( list.files( "H3K4me3", "run.*lane.\\.map" ),
+    function(filename) readAligned( "H3K4me3", filename, type="MAQMapShort" ) )
```

[1] When I first wrote this vignette in June 2008, the SRA was still in a provisional state, and the presentation of the data has changed since then. You can still find the old files in the subdirectory "provisional".

[2] However, feel free to contact me if you want to know details.

Here, readAligned takes three arguments: the directory that contains the map file, the name of the map file, and the type of data to be read, for Maq alignment data MAQMap. (Our example data has been aligned with an older version of Maq, prior to the recent change in binary format in Maq version 0.7. Using the type MAQMapShort allows to read the old format.) You may also use the type SolexaExport to read in mappings produced by Eland (see help page for readAligned for details on the supported formats). In any case, the function readAligned returns an S4 object of class AlignedRead.

4 The class AlignedRead

An AlignedRead object is conceptionally quite similar to a data frame. It contains as many "rows" as there are mapped reads:

```
> length( maps.me1$run4_lane8.map )

[1] 3465080
```

For each read, all the data parsed from the map file are stored. Think of these types of data as of columns of a data frame, even though you do not access them with the $ operator but with accessor functions. The "columns" chromosome, position and width show where in the genome the reads were mapped:

```
> head( chromosome( maps.me1$run4_lane8.map ) )

[1] 10 10 10 10 10 10
113 Levels: 10 11 12 13 14 15 16 17 18 19 1 20 21 22 2 3 4 5 6 7 8 9 MT X ... NT_113898

> head( position( maps.me1$run4_lane8.map ) )

[1] 50547 53424 58681 61890 64043 66900

> head( width( maps.me1$run4_lane8.map ) )

[1] 25 25 25 25 25 25
```

As we see, the first 6 of the 3.4 mio reads in lane 8 of run 4 were all mapped to chromosome 10, to the given positions, and extending from there all by 25 bp.[3]

The actual reads are stored as well,

```
> head( sread( maps.me1$run4_lane8.map ) )

  A DNAStringSet instance of length 6
    width seq
[1]    25 CCAGGAGAATATGCAATGATGACAA
[2]    25 TATAGAGCATTAAACCACCAAAGCT
[3]    25 TAACCAACTCAAGTGCCCATCAGTG
[4]    25 TGATTGTGCCACTGCACTCTAGCAA
[5]    25 TTGCTGGCACCAGGGACCAGGAGGA
[6]    25 TACCATCTCACACCACTTAGAATGG
```

[3]Note that Maq stores the aligned reads in order of their alignment. Hence, we start with very low base-pair indices, which then increase. Maq has also started with chromosome 10, as that one happened to be the first one in the BFA file.

as are the reads' identifiers (which here encode their position on the lane):

```
> head( id( maps.me1$run4_lane8.map ) )

  A BStringSet instance of length 6
    width seq
[1]    14 R:8:34:810:204
[2]    14 R:8:82:891:530
[3]    15 R:8:136:501:225
[4]    13 R:8:68:11:564
[5]    15 R:8:101:523:977
[6]    14 R:8:15:487:873
```

These last two objects are not ordinary R `character` vectors but `DNAStringSet` and `BStringSet` objects. These are specialised data structures provided by the `Biostrings` package designed to handle large amounts of character (or sequence) data. They are not elementary-type vectors but S4 objects. (See the Biostrings vignette for details.) As they only mimic a vector they cannot be columns of a data frame. This is the reason why `AlignedRead` is not a data frame although its structure is reminiscent of one.

Other information stored in the `AlignedRead` object is the base-call quality as reported by Bustard, here given in FASTQ quality string representation. (See the Maq web site for an explanation of the format.)

```
> head( quality( maps.me1$run4_lane8.map ) )

class: FastqQuality
quality:
  A BStringSet instance of length 6
    width seq
[1]    25 +//59>@II1ACIIFIIII:IIIII
[2]    25 IIIIIIIIIIIII7II5CI<.7+&
[3]    25 III1III4$@IIFI%3/DI$%A8*+
[4]    25 +&%,;+@;*7%6I+;*I>+%,@I@I
[5]    25 +IIIIII5HB3IHIE1&+++8,7+)
[6]    25 II%5I9.&II<I%8'%#7*,0%@9$
```

Each of the letters codes for for the quality of a base call, i.e., which stands for the probability that the base call is incorrect. To see the actual quality scores, coerce the quality BStringSet to a matrix:

```
> quals <- as( head( quality( maps.me1$run4_lane8.map ) ), "matrix" )
> quals

     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12] [,13] [,14]
[1,]   10   14   14   20   24   29   31   40   40    16    32    34    40    40
[2,]   40   40   40   40   40   40   40   40   40    40    40    40    40    40
[3,]   40   40   40   16   40   40   40   19    3    31    40    40    37    40
[4,]   10    5    4   11   26   10   31   26    9    22     4    21    40    10
[5,]   10   40   40   40   40   40   40   20   39    33    18    40    39    40
[6,]   40   40    4   20   40   24   13    5   40    40    27    40     4    23
     [,15] [,16] [,17] [,18] [,19] [,20] [,21] [,22] [,23] [,24] [,25]
```

```
[1,]    37    40    40    40    40    25    40    40    40    40    40
[2,]    22    40    40    20    34    40    27    13    22    10     5
[3,]     4    18    14    35    40     3     4    32    23     9    10
[4,]    26     9    40    29    10     4    11    31    40    31    40
[5,]    36    16     5    10    10    10    23    11    22    10     8
[6,]     6     4     2    22     9    11    15     4    31    24     3
```

For an explanation how the letters are converted to scores, look up the FASTQ standard. (Wikipedia has a good explanation.) The numbers are Phred scores, i.e. the probability for a base being wrong is given by $10^{-Q/10}$:

```
> 10 ^ ( -quals / 10 )
```

```
      [,1]       [,2]       [,3]       [,4]        [,5]        [,6]
[1,] 1e-01 0.03981072 0.03981072 0.01000000 0.003981072 0.001258925
[2,] 1e-04 0.00010000 0.00010000 0.00010000 0.000100000 0.000100000
[3,] 1e-04 0.00010000 0.00010000 0.02511886 0.000100000 0.000100000
[4,] 1e-01 0.31622777 0.39810717 0.07943282 0.002511886 0.100000000
[5,] 1e-01 0.00010000 0.00010000 0.00010000 0.000100000 0.000100000
[6,] 1e-04 0.00010000 0.39810717 0.01000000 0.000100000 0.003981072
             [,7]        [,8]         [,9]        [,10]        [,11]
[1,] 0.0007943282 0.000100000 0.0001000000 0.0251188643 0.0006309573
[2,] 0.0001000000 0.000100000 0.0001000000 0.0001000000 0.0001000000
[3,] 0.0001000000 0.012589254 0.5011872336 0.0007943282 0.0001000000
[4,] 0.0007943282 0.002511886 0.1258925412 0.0063095734 0.3981071706
[5,] 0.0001000000 0.010000000 0.0001258925 0.0005011872 0.0158489319
[6,] 0.0501187234 0.316227766 0.0001000000 0.0001000000 0.0019952623
            [,12]        [,13]       [,14]        [,15]      [,16]      [,17]
[1,] 0.0003981072 0.0001000000 0.000100000 0.0001995262 0.00010000 0.00010000
[2,] 0.0001000000 0.0001000000 0.000100000 0.0063095734 0.00010000 0.00010000
[3,] 0.0001000000 0.0001995262 0.000100000 0.3981071706 0.01584893 0.03981072
[4,] 0.0079432823 0.0001000000 0.100000000 0.0025118864 0.12589254 0.00010000
[5,] 0.0001000000 0.0001258925 0.000100000 0.0002511886 0.02511886 0.31622777
[6,] 0.0001000000 0.3981071706 0.005011872 0.2511886432 0.39810717 0.63095734
            [,18]        [,19]       [,20]       [,21]        [,22]
[1,] 0.0001000000 0.0001000000 0.003162278 0.000100000 0.0001000000
[2,] 0.0100000000 0.0003981072 0.000100000 0.001995262 0.0501187234
[3,] 0.0003162278 0.0001000000 0.501187234 0.398107171 0.0006309573
[4,] 0.0012589254 0.1000000000 0.398107171 0.079432823 0.0007943282
[5,] 0.1000000000 0.1000000000 0.100000000 0.005011872 0.0794328235
[6,] 0.0063095734 0.1258925412 0.079432823 0.031622777 0.3981071706
            [,23]        [,24]     [,25]
[1,] 0.0001000000 0.0001000000 0.0001000
[2,] 0.0063095734 0.1000000000 0.3162278
[3,] 0.0050118723 0.1258925412 0.1000000
[4,] 0.0001000000 0.0007943282 0.0001000
[5,] 0.0063095734 0.1000000000 0.1584893
[6,] 0.0007943282 0.0039810717 0.5011872
```

Maq calculates from this information and from the uniqueness and perfectness of the alignment an alignment score, which is stored in an `alignQuality` object:

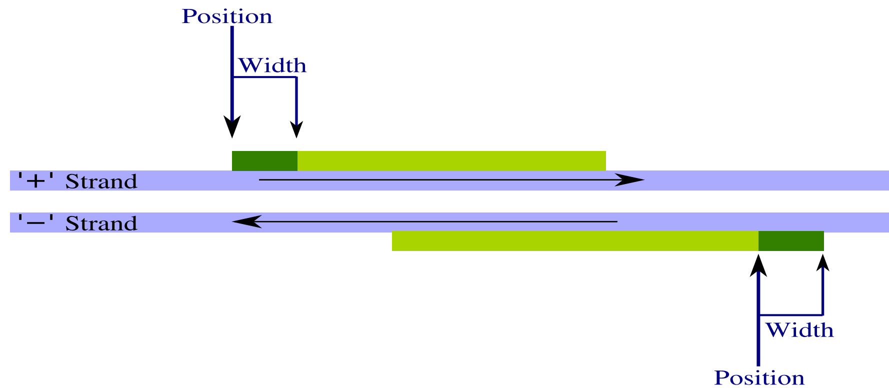


Figure 1: The `strand` information shows how the read is aligned against the genome. If `strand` is `"+"`, the `position` indicates the start of the DNA read (dark green) as well as the start of the whole fragment. The part of the fragment that was not read (light green) extends to the right (i.e. towards larger chromosome coordinates). If `strand` is `"-"`, then the fragment extends to the left. As `position` always indicates the left edge of the read (but not necessarily an edge of the whole fragment) it now points to a position within the whole fragment.

```
> alignQuality( maps.me1$run4_lane8.map )

class: IntegerQuality
quality: 0 0 ... 9 9 (3465080 total)
```

The actual integer vector of qualities (one number per read) can be obtained with the accessor function `quality`

```
> head( quality( alignQuality( maps.me1$run4_lane8.map ) ) )

[1] 0 0 0 0 0 0
```

An alignment quality score of 0 may mean that the read could not be uniquely aligned and has been put by Maq at one of the possible positions, chosen at random.

As before, the probability for the alignment being wrong is $10^{-Q/10}$, where Q is the quality score. Bear in mind that all these probabilities are estimates derived partly by heuristics. See the SolexaPipeline manual and the Maq paper for details before relying on them.

The accessor function `strand` reports whether the read was mapped to the "+" or to the "–" strand of the chromosome. It returns a factor with three levels:

```
> head( strand( maps.me1$run4_lane8.map ) )

[1] - + + - + +
Levels: - + *
```

You should never see the level "*" in an AlignedData object. It is used in other contexts to indicate that a strand information is not just unavailable (this would be an `NA`) but does not have any meaning.

Remember that Solexa sequencing is not strand-specific (unless you use one of the new strand-specific RNA-Seq protocols). Hence, it is better to think of this factor not as information on the strand but rather on the *direction* of the fragment. Have a look at Fig. 1 for an illustration.

The fields described so far are available for all `AlignedRead` objects. Depending on the alignment software that was used additional information may be available. The slot `alignData` is meant to hold such information. The fields that you can see here are explained in the manual to Maq.

```
> alignData( maps.me1$run4_lane8.map )

An object of class "AlignedDataFrame"
  readName: 1, 2, ..., 3465080  (3465080 total)
  varLabels and varMetadata description:
    nMismatchBestHit: Number of mismatches of the best hit
    mismatchQuality: Sum of mismatched base qualities of the best hit
    nExactMatch24: Number of 0-mismatch hits of the first 24 bases
    nOneMismatch24: Number of 1-mismatch hits of the first 24 bases
```

`AlignedDataFrame` is a subclass of `AnnotatedDataFrame`. Hence, we can see the meaning of the columns from the meta information displayed above and access the underlying data frame with `pData`:

```
> head( pData( alignData( maps.me1$run4_lane8.map ) ) )

  nMismatchBestHit mismatchQuality nExactMatch24 nOneMismatch24
1                0               0             3              0
2                0               0             2             85
3                2              17             0              1
4                1               4             1             34
5                1              30             0              4
6                1               4            85             85
```

5 Coverage

In ChIP-Seq, one is usually interested in the number of precipitated DNA fragments in the sample that were mapped to each genomic locus. This is best represented by what is often called a "coverage vector" (or sometimes a "pile-up vector"). This is a very long `integer` vector with as many elements as there are base pairs in the chromosome under consideration. Each vector element counts the number of fragments that were mapped such that they cover this base pair. The function `coverage` in the `ShortRead` package calculates such a vector from alignment information.[4]

In order to allocate a vector of the right size, `pileup` needs to know the length of the chromosome. `readBfaToc` obtains the lengths of all sequences in a BFA file (binary FASTA, the compressed FASTA format used by Maq). As a BFA file has a table of content at the beginning, `readBfaToc` only has to read the header of the BFA file and is hence quite fast.

```
> seqlens <- readBfaToc( "Homo_sapiens.NCBI36.48.dna.all.bfa" )
> seqlens
       10        11        12        13        14        15        16        17
135374737 134452384 132349534 114142980 106368585 100338915  88827254  78774742
```

[4]The first version of this vignette used the `pileup` function instead. Both functions do essentially the same job but `pileup` returns an ordinary vector while `coverage` (which was not yet available then) returns an `Rle` vector, as explained in the following.

```
       18         19          1         20         21         22          2          3 
 76117153   63811651  247249719   62435964   46944323   49691432  242951149  199501827 
        4          5          6          7          8          9         MT          X 
191273063  180857866  170899992  158821424  146274826  140273252      16571  154913754 
        Y  NT_113887  NT_113947  NT_113903  NT_113908  NT_113940  NT_113917  NT_113963 
 57772954       3994       4262      12854      13036      19187      19840      24360 
NT_113876  NT_113950  NT_113946  NT_113920  NT_113911  NT_113907  NT_113937  NT_113941 
    25994      28709      31181      35155      36148      37175      37443      37498 
NT_113909  NT_113921  NT_113919  NT_113960  NT_113945  NT_113879  NT_113938  NT_113928 
    38914      39615      40524      40752      41001      42503      44580      44888 
NT_113906  NT_113904  NT_113873  NT_113966  NT_113943  NT_113914  NT_113948  NT_113886 
    46082      50950      51825      68003      81310      90085      92689      96249 
NT_113932  NT_113929  NT_113878  NT_113927  NT_113900  NT_113918  NT_113875  NT_113942 
   104388     105485     106433     111864     112804     113275     114056     117663 
NT_113926  NT_113934  NT_113954  NT_113953  NT_113874  NT_113883  NT_113924  NT_113933 
   119514     120350     129889     131056     136815     137703     139260     142595 
NT_113884  NT_113890  NT_113870  NT_113881  NT_113939  NT_113956  NT_113951  NT_113902 
   143068     143687     145186     146010     147354     150002     152296     153959 
NT_113913  NT_113958  NT_113949  NT_113889  NT_113936  NT_113957  NT_113961  NT_113925 
   154740     158069     159169     161147     163628     166452     166566     168820 
NT_113882  NT_113916  NT_113930  NT_113955  NT_113944  NT_113901  NT_113905  NT_113872 
   172475     173443     174588     178865     182567     182896     183161     183763 
NT_113952  NT_113912  NT_113935  NT_113880  NT_113931  NT_113923  NT_113915  NT_113885 
   184355     185143     185449     185571     186078     186858     187035     189789 
NT_113888  NT_113871  NT_113964  NT_113877  NT_113910  NT_113962  NT_113899  NT_113965 
   191469     197748     204131     208942     211638     217385     520332    1005289 
NT_113898 
  1305230 
```

If you try to reproduce this example, you may not have the BFA file[5]. So, you can obtain the object `seqlens` manually with the following command (which omits the `NT_xxxxx` contigs):

```
> seqlens <- c( `10`=135374737, `11`=134452384, `12`=132349534, `13`=114142980,
+ `14`=106368585, `15`=100338915, `16`=88827254, `17`=78774742, `18`=76117153,
+ `19`=63811651, `1`=247249719, `20`=62435964, `21`=46944323, `22`=49691432,
+ `2`=242951149, `3`=199501827, `4`=191273063, `5`=180857866, `6`=170899992,
+ `7`=158821424, `8`=146274826, `9`=140273252, MT=16571, X=154913754, Y=57772954 )
```

In order to get coverage vectors for all chromosomes, using only mappings in `maps.me3$run13_lane4.map` with a mapping quality of at least 10, we first create a new `AlignedRead` object containing only these reads (note that we also filter out reads that map to chromosomes or contigs for which we do not have sequence lengths)

```
> filteredReads <- maps.me3$run13_lane4.map[
+    chromosome( maps.me3$run13_lane4.map ) %in% names(seqlens) &
+    quality(alignQuality( maps.me3$run13_lane4.map )) >= 10 ]
```

and then run the `coverage` function[6] on these:

[5] I have not put it on my web page as it is very big and easily created from the Ensembl files.

[6] texttt coverage is a generic method defined in the `IRanges` object. Here, we use its specialization for `ReadAligned` objects, defined in the `ShortRead` package. For the help pages, see both `?coverage` and `class?AlignedRead`.

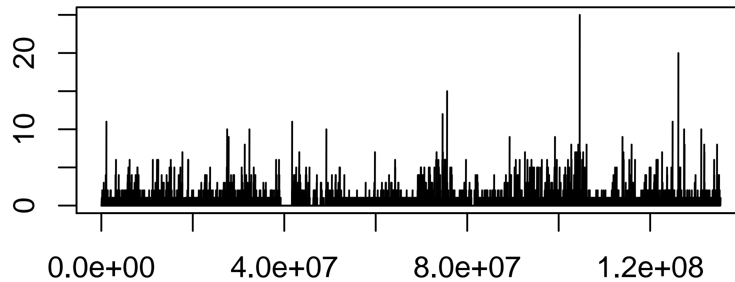


Figure 2: Output of `plotLongVector( me3.p10 )`.

```
> coverage.me3.lane4 <- coverage( filteredReads, width=seqlens )
> coverage.me3.lane4

A GenomeData instance
chromosomes(25): 10 11 12 13 14 15 16 17 18 19 1 20 21 22 2 3 4 5 6 7 8 9 MT X Y
```

The object `coverage.me3` is a `GenomeData` object, essentially a list of coverage vectors, one for each chromosome.

Here is the coverage vector for chromsome 10:

```
> coverage.me3.lane4$`10`

  'integer' Rle instance of length 135374737 with 56705 runs
  Lengths:  82039 24 2417 24 20699 24 4617 24 2284 24 ...
  Values :  0 1 0 1 0 1 0 1 0 1 ...
```

This is a vector with 135 mio elements, i.e., one number for each base pair on chromosome 10. If we stored this as an ordinary vector in RAM, it would need 135 MB. However, it contains long stretches of constant values, and hence, `coverage` returns its result as run-length encoded (`Rle`) vectors. As you can see, the coverage vector contains a few ten thousands of "runs", i.e., of repeats of the same value, and stores this information in the form of the lengths and the values of these runs.

In principle, we could now plot this vector by converting it to an ordinary vector and using the standard "plot" function:

```
> plot( as.vector( coverage.me3.lane4$`10` ), type='h' )
```

However, this command takes very long, as it plots one needle for each vector element, spending most of its time plotting over and over at the same spot. The function `plotLongVector` (in `HilbertVis`) produces the same plot with a decent speed:

```
> library("HilbertVis")
> plotLongVector( coverage.me3.lane4$‘10‘ )
```

[Output: See Fig. 2.]

It does so by first partitioning the vector in 4,000 segments of equal length and the gets the maxima and minima of each segments (with the `shrinkVector` funcion). It the draws vertcial lines from the minima to the maxima. In case you want to write your own plotting function (because `plotLongVector` is rather rudimentary), you can use the function `shrinkVector` (in `HilbertVis`) to accomplish this.

In the form used above, the function `pileup` counts only which base pairs the actual read covers. Typically, the read length (here: 25 nt) is much shorter than the length of the DNA fragments. In the present data, the length of the fragments after sonication, adaptor ligation and gel-electrophoretic size selection was about 220 bp including adaptors, i. e., approx. 185 bp without adaptors. Given that the immuno-precipitated histone can be anywhere on the fragment, not necessarily within the part at the end that is actually sequenced (the "read") we get a less biased picture by incorporating this information into the calculation of the pile-up vector. The `coverage` can be called with an `extend` argument to extend each fragment by a certain size. It uses the strand information to know which direction to extend to (see Fig. 1).

```
> coverage.me3.lane4.ext <- coverage( filteredReads,
+    width=seqlens, extend=185L-width(filteredReads) )
```

Our coverage vector incorporated only the information from one lane. We get better count statistics by getting such a vector for each lane from the H3K4me3 sample and then simply summing them all up:

```
> sumUpCoverage <- function( lanes, seqLens, minAQual, fragmentLength )
+ {
+    res <- NULL
+    for( i in 1:length(lanes) ) {
+       filteredLane <- lanes[[i]][
+          quality(alignQuality( lanes[[i]] )) >= minAQual &
+          chromosome(lanes[[i]]) %in% names(seqlens) ]
+       cvg <- coverage( filteredLane, width = seqLens,
+          extend = as.integer(fragmentLength) - width(filteredLane) )
+       if( is.null( res ) )
+          res <- cvg
+       else {
+          stopifnot( all( names(res) == names(cvg) ) )
+          for( seq in names(res) )
+             res[[seq]] <- res[[seq]] + cvg[[seq]]
+       }
+    }
+    res
+ }
> coverage.me3 <- sumUpCoverage( maps.me3, seqlens, 10, 185 )
```

Note that `for` loops are used here instead of `sapply`. The latter may look more natural in R but it builds up a two-dimensional array of all the intermediate coverage vectors, which is wasteful. Even with the `for` loop the operation takes a while. Let's do the same for "me1":

```
> coverage.me1 <- sumUpCoverage( maps.me1, seqlens, 10, 185 )
```

As “me1” has only 3 lanes as opposed to “me3”’s 7 lanes, we cannot compare them directly. A simple way of normalizing is to divide by the “library size”, i.e., the total number of reads.

```
> nreads.me1 <- sum( sapply( maps.me1, length ) )
> coverage.me1.n <- GenomeData( lapply(
+    coverage.me1, function(r) r / nreads.me1 ) )
> nreads.me3 <- sum( sapply( maps.me3, length ) )
> coverage.me3.n <- GenomeData( lapply(
+    coverage.me3, function(r) r / nreads.me3 ) )
```

6 Visualisation with Hilbert curve plots

6.1 The Hilbert curve

Note: If you have skipped the previous sections as you only want to read about `HilbertVis`, here is what you need to know in order to start reading here: We have re-analysed part of the ChIP-Seq experiments done in Ref. [BCC+07], namely the data regarding histone methylation patterns H3K4me1 and H3K4me3. We have constructed two sets of very long “coverage” vectors in `IRanges`’s `Rle` form, `coverage.me1` and `coverage.me3`, which represent the human chromosomes and have a length corresponding to the number of base pairs of each chromosome. Each element corresponds to a base pair and counts how many precipitated and sequenced DNA fragments within the respective sample (H3K4me1 or H3K4me3) cover this position. The vectors `coverage.me1.n` and `coverage.me3.n` have been normalized by dividing by the total number of reads. To do your own experiments with these vectors, you can download these vectors (truncated to only contain chromosome 10, to save space) as R data file from `http://www.ebi.ac.uk/~anders/ShortReadExampleData/meX.chr10.rda`.

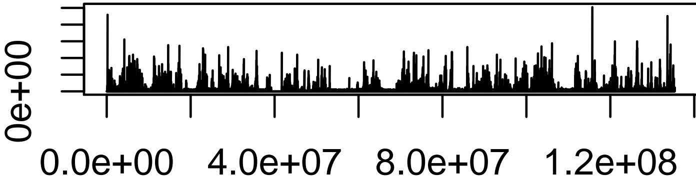
Note 2: Since I have written this vignette, I have restructured the package and split it into two parts, called “HilbertVis” and “HilbertVisGUI”. This text focuses on the functionality of “HilbertVisGUI”, which provides an interactive tool to explore data using teh visualisation technique desribed in the following. “HilbertVis” contains further functions to produce the same kind of images but without interactive tools, i.e. solely from the R command line. If you want to know more about these functions, which are not mentioned in the present text, see the vignette “Visualising very long data vectors with the Hilbert curve”, which is included in the “HilbertVis” package.

A first approach to visualising the two vectors is plotting them with the `plotLongVector` function described above:

```
> library( ShortRead )
> library( HilbertVis )
> library( HilbertVisGUI )
> par( mfrow = c(2,1) )
> plotLongVector( coverage.me1.n$`10`, main="Chr 10, H3K3me1" )
> plotLongVector( coverage.me3.n$`10`, main="Chr 10, H3K3me3" )
```

[Output: Fig. 3.]

The two vectors do look different but it is hard to make out what gives rise to the difference. Is the number of peaks different, or their distribution, or their typical width? Given that each pixel on the x axis corresponds to more than 100 kp, each of the needle can as well be a small peak, only a few fragment lengths wide, a wide peak with a base of tens of kb, or even a cluster of several peaks. We might zoom in somewhere but this is not too illuminating:



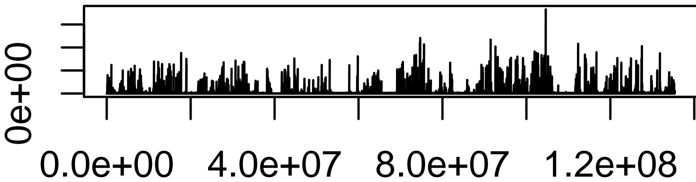
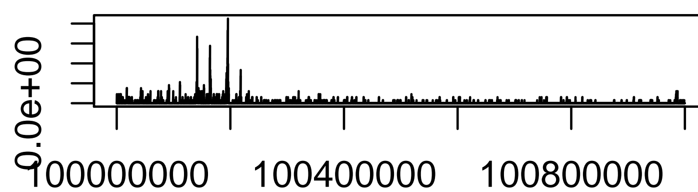


Figure 3: Pile-up representation of the ChIP-Seq data for H3K4me1 and H3K4me3, depicting the whole of chromosome 10.

Chr 10, H3K3me1



Chr 10, H3K3me3

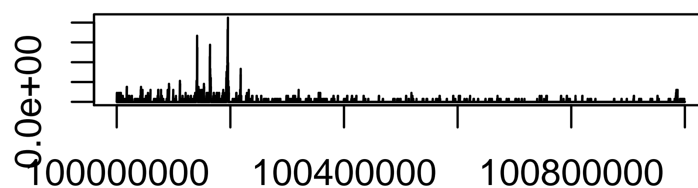


Figure 4: Zoom into a small portion of Fig. 3.

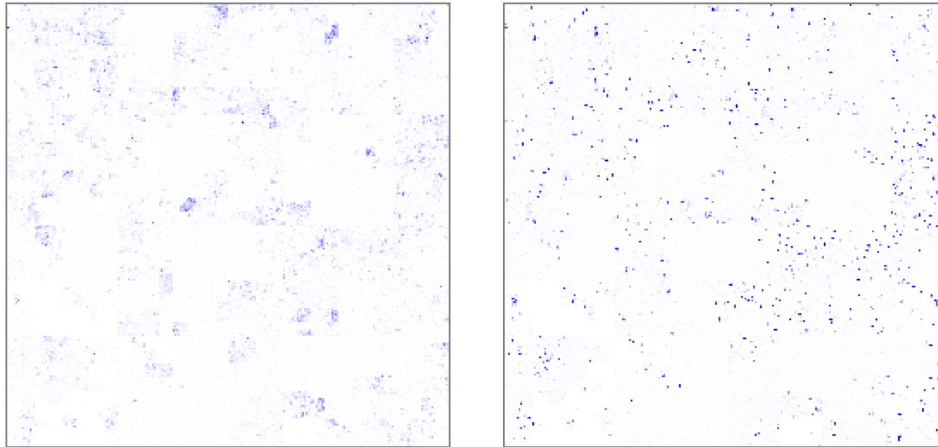


Figure 5: Hilbert curve plot of pile-ups for H3K4me1 (left) and H3K4me3 (right) on chromosome 10.

```
> par( mfrow = c(2,1) )
> plotLongVector( coverage.me1.n$`10`[100000000:101000000],
+    main="Chr 10, H3K3me1", offset=100000000 )
> plotLongVector( coverage.me1.n$`10`[100000000:101000000],
+    main="Chr 10, H3K3me3", offset=100000000 )
```

[Output: Fig. 4.]

The standard approach would be to export the pile-up vectors into a genome track format such as BED[7] and then use a genome browser such as those on the UCSC or Ensembl web sites, or IGB, to zoom in at many places to get a feeling for the data.

The Hilbert curve plot is an approach to display an as detailed picture of the whole chromosome as possible by letting each pixel of a large square represent a quite short part of the chromosome, coding with its colour for the maximum count in this short stretch, where the pixels are arranged such that neighbouring parts of the chromosome appear next to each other in the square. Furthermore, parts which are not directly neighbouring but are ion close distance should not be separated much in the square either. Fig. 5 shows the two pile-up vectors in this so-called Hilbert curve plot.

In order to understand this plot you need to know how the pixels are arranged to fulfil the requirements just outlined as well as possible. To my knowledge, the first to study this problem in detail and to come up with the solution also used here was D. A. Keim in Ref. [Kei96] (where he used the data to visualise long time-series data of stock-market prices). He went back to an old idea of Peano [Pea90] and Hilbert [Hil91], space-filling curves. Peano astonished the mathematics community at the end of the 19th century by presenting a continuous mapping of a line to a square, i.e., showed that a line can be folded up such that it passes through every point within a square, thus blurring the seemingly clear-cut distinction between one- and two-dimensional objects. Such a space-filling curve is a fractal, i.e., it has infinitely many corners and repeats its overall form in all levels of its details. Fig. 6 shows the first six level of the construction of Hilbert's variant of Peano's curve. Observe how at level k a line of length 2^{2k} passes through each "pixel" of a square of dimension $2^k \times 2^k$, and how this curve is produced connecting four copies (in different orientations) of the curve at the previous level, $k - 1$.

[7] A function to do that might be added soon to `ShortRead`.

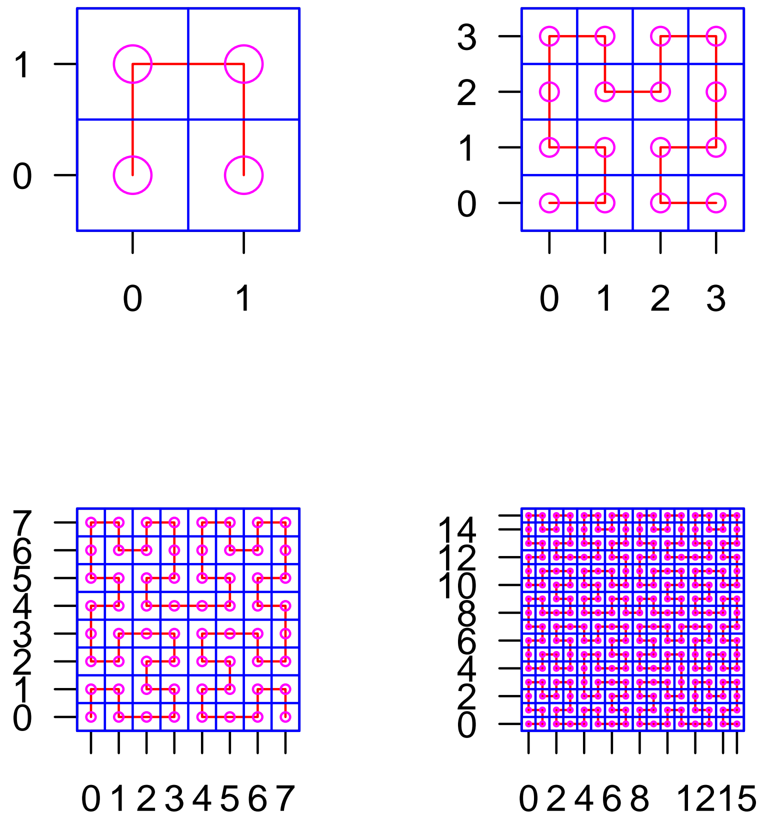


Figure 6: The first four levels of the Hilbert curve fractal.

Figure 6 has been produced with the function `plotHilbertCurve` which is provided just for demonstration purposes.

```
> library( grid )
> pushViewport( viewport( layout=grid.layout( 2, 2 ) ) )
> for( i in 1:4 ) {
+    pushViewport( viewport(
+        layout.pos.row=1+(i-1)%/%2, layout.pos.col=1+(i-1)%%2 ) )
+    plotHilbertCurve( i, new.page=FALSE )
+    popViewport( )
+ }
```

Going back to Fig. 5, we can now see clear difference between the two samples. The following observations my be made just from comparing these two plots: The peaks of H3K4me3 are tall, narrow, and well defined, while those for H3K4me1 are rather washed out. In both cases the peaks spread out over the whole chromosome, but some areas have nearly no signal. These empty parts are the same in both cases. These points were not clear only from Fig. . Exploring the plot interactively as described in the following allows to get considerable more insights.

6.2 The `HilbertVis` GUI

In order to study the pile-up vectors, you can now simply call

```
hilbertDisplay( coverage.me1.n$‘10‘, coverage.me3.n$‘10‘ )
```

A GUI, as depicted in Fig. 7 will pop up that allows you to interactively explore your data in the Hilbert curve plot representation. First, press the “Darker” button two or three times to get better contrast. Then, move the mouse over the coloured square and observe how the small red line in the right-hand gauge (labelled “Displayed part of sequence”) indicates where within the chromosome you are pointing. Playing with this feature allows you to quickly orient yourself on how the chromosome is folded into the square. You can also read off the exact position from the field “Bin under mouse cursor”[8]

Use the left mouse button to zoom in by clicking on one of the four quarters of the image. You can only zoom into a quarter, not into any part of the image, because this ensures that the displayed part is always a single consecutive stretch of the chromosome. The left-hand gauge (labelled “Full sequence”) indicates which part is displayed: the full width of the gauge represents the whole chromosome, the portion highlighted in red the part that is currently displayed in the square. The coordinates of the first and last displayed base are printed in the edges of the right-hand gauge. With the radio buttons labelled “Effect of left mouse button” you my switch from zooming into a quarter to zooming into a 1/64 part, i.e. into one of the small squares in a though 8×8 grid. Use the buttons at the bottom to zoom out.

If you have passed several vectors when calling `hilbertDisplay`, you may switch back and forth between them with the buttons “Next” and “Previous” (or by pressing Alt-N and Alt-P) in order to compare the displayed parts.

The two buttons “Coarser” and “Finer” allow to adjust the pixel size. Initially, each bin is represented by one pixel at your monitor’s resolution, and there are 512×512 pixels in the image. Pressing “Coarser” once blows up each image pixel to a 2×2 square of monitor pixels, which allows for easier viewing but reduces the number of displayed image pixels to 256×256, i.e., each pixel now represents four times as many base pairs.

[8]The display of the bin’s value is not yet functional.

Figure 7: The graphical user interface (GUI) provided by `HilbertVisGUI`.

There are a number of optional parameters to `hilbertDisplay` that come in handy, e.g., if you have vectors of differing length, if you want to customise colours or change a few other points. Refer to the help page (displayed with `?hilbertDisplay`) for details.

6.3 The callback interface

If you select the mode "Linear plot" as "Effect of left mouse button" and click somewhere in the plot, a windows pops up with a linear plot that displays the part of the chromosome represented by 256 pixels around the pixel on which you have clicked. (256 pixels correspond quite roughly to the size of the cross-hair mouse cursor). This is useful to get a detailed view of the shape of peaks.

To do the linear plot, HilbertDisplay calls the R function `simpleLinPlot` defined in the `HilbertVisGUI` package. This is a simple wrapper around the function `plotLongVector` discussed earlier. Here is the definition of simpleLinPlot:

```
> simpleLinPlot

function (data, info)
{
    hw <- (info$dispHi - info$dispLo)/1024
    left <- max(1, info$bin - hw)
    right <- min(info$bin + hw, length(data))
    plotLongVector(data[left:right], offset = left, main = info$seqName,
        shrinkLength = min(2 * hw + 1, 4000))
}
<environment: namespace:HilbertVisGUI>
```

You can replace this function by supplying your own plotting function as the argument `plotFun` to `hilbertDisplay`. Your function must take two arguments that should be called `data` and `info`, as above, and will be filled in by `hilbertDisplay` with the displayed vector and information about where the user clicked and which part of the vector is being displayed. Try the following example to see the format of this data:

```
dumpDataInsteadOfPlotting <- function( data, info ) {
   str( data )
   print( info )
}
hilbertDisplay( me1.p10, me3.p10, plotFunc=dumpDataInsteadOfPlotting )
```

Zoom in a bit, then switch to "linear plot" and click somewhere. `dumpDataInsteadOfPlotting` will be called and output such as the following appears on your R console:

```
num [1:135374737] 0 0 0 0 0 0 0 0 0 0 ...
$binLo
[1] 22950198

$bin
[1] 22950262

$binHi
[1] 22950327
```

```
$dispLo
[1] 16921843

$dispHi
[1] 25382764

$seqIdx
[1] 1

$seqName
[1] "me1.p10"
```

See ?hilbertDisplay for an explanation of those fields that are not self-explanatory.

This feature is meant to allow for customised linear plots (maybe using the GenomeGraph package to add annotation) but can also be used for other things than plotting, e.g., calculating some statistics about a peak clicked on.

6.4 Three-channel display

In order to look for spatial correlations in different data vectors, it may be useful to overlay the corresponding Hilbert curve plots in different colours. The function hilbertDisplayThreeChannel allows to display three data vectors simultaneously, using the red, green, and blue channel of the displayed image for the first, second, and third, vector. We may want to see whether the areas with strong H3K4me1 occurance are at the same chromosome regions as the majority of the H3K4me3 peaks. Furthermore, we may use the third channel to indicate the presence of exons.

We first obtain a list of all exons on chromosome 10 from EnsEMBL via BioMart:

```
> library( biomaRt )
> ensembl <- useMart("ensembl", dataset = "hsapiens_gene_ensembl")
> exons <- getBM( attributes=c( "exon_chrom_start", "exon_chrom_end" ),
+    filters="chromosome_name", values="10", mart=ensembl )
```

This is a set of intervals and hence best represented as an IRanges object:

```
> exon.chr10.ranges <- IRanges( start=exons$exon_chrom_start, end=exons$exon_chrom_end )
```

Them, we construct a vector that indicates for each base pair on chromosome 10, whether it is exonic or not (bty means of the values 1 and 0).

```
> exons.chr10 <- coverage( exon.chr10.ranges, width = seqlens[["10"]] )
```

With the following command, we get a 3-color representation of the three vectors in the HilbertDisplay GUI:

```
> hilbertDisplayThreeChannel(
+    coverage.me1.n$`10` * 5e5,
+    coverage.me3.n$`10` * 5e5,
+    exons.chr10 * .5 )
```

See Fig. refthreeColor for the image that the GUI shows. While the function hilbertDisplay adjusts to the value range of the data (or can be manually adjusted with optional the paletteSteps

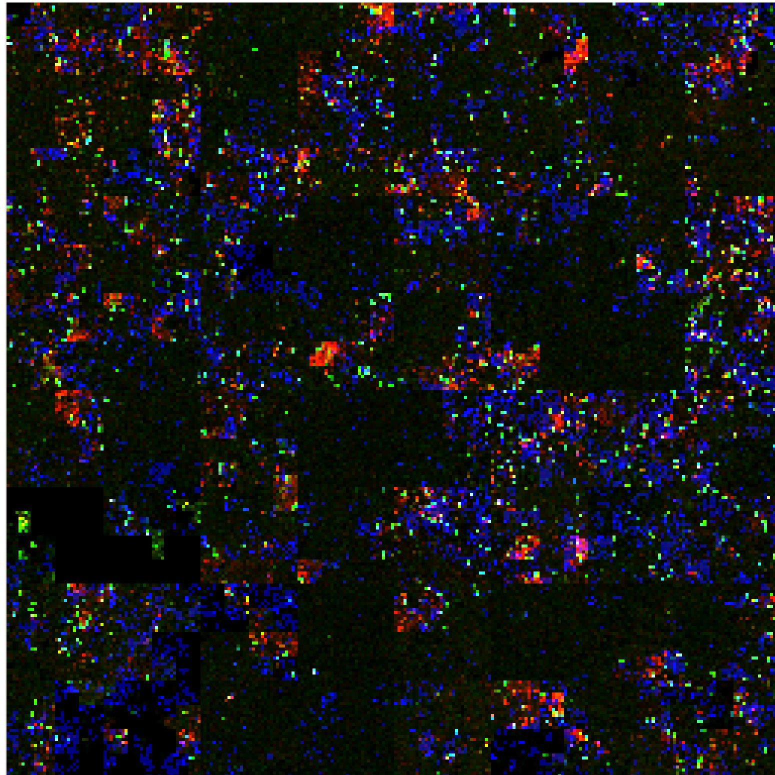


Figure 8: A three-color overlay (obtained with the function `hilbertDisplayThreeChannel`) of Hilbert curves for H3K4me1 (red), H3K4me3 (green) and an exon indication (blue). The image shows a zoom into the first quarter of chromosome 10 (i.e., the top left quarter of the images in Fig. 5).

argument), the function `hilbertDisplayThreeChannel` expects all three vectors to be in the value range between 0 and 1. This range is transformed to colours from black to a saturated red, green, and blue. Values below 0 or above 1 are cut and displayed as if they were 0 or 1. To get the pile-up vectors down to this range, an obvious step would be to divide by their maximum value. However, this gives a too dark value, and hence, I have chosen for Fig. 8 larger scaling factors, allowing extremely high peaks to become saturated.

7 Correlation with transcription start

A common plot to do with histone modification ChIP-Seq data is to see how the pile-up correlates with transcription start sites (TSS). This is done quite easily.

First, we get a list of known TSSs on chromosome 10 from EnsEMBL (again via BioMart).

```
> tss <- getBM( attributes=c( "transcript_start", "transcript_end", "strand" ),
+    filters="chromosome_name", values="10", mart=ensembl )
```

```
> head(tss)
```

```
  transcript_start transcript_end strand
1        104275728      104275989      1
2         21893406       21893512     -1
3        122104417      122104685     -1
4        121517754      121518044      1
5        120810487      120810613     -1
6        120535391      120535465     -1
```

Note that `transcript_start` is always smaller than `transcript_end`, even when the transcript is on the "–" strand. Hence, we have to use either the start or the end coordinate of the transcript, depending on the strand, to get the actual transcription start sites, i.e., the 5' ends of the transcripts. Then, we go through all TSS, cutting out a window from 2000 bp upstreams to 2000 bp downstreams of the TSS and sum these up these vectors of length 4001 (reversing them whenever they are from the "–" strand):

```
> tme1 <- rep( 0, 4001 )
> tme3 <- rep( 0, 4001 )
> for( i in 1:nrow(tss) ) {
+    if( tss$strand[i] == 1 ) {
+       tme1 <- tme1 + as.vector( seqextract( coverage.me1.n$`10`,
+          tss$transcript_start[i] - 2000, tss$transcript_start[i] + 2000 ) )
+       tme3 <- tme3 + as.vector( seqextract( coverage.me3.n$`10`,
+          tss$transcript_start[i] - 2000, tss$transcript_start[i] + 2000 ) )
+    } else {
+       tme1 <- tme1 + rev( as.vector( seqextract( coverage.me1.n$`10`,
+          tss$transcript_end[i] - 2000, tss$transcript_end[i] + 2000 ) ) )
+       tme3 <- tme3 + rev( as.vector( seqextract( coverage.me3.n$`10`,
+          tss$transcript_end[i] - 2000, tss$transcript_end[i] + 2000 ) ) )
+    }
+ }
```

Note the use of `as.vector`, which transforms the `Rle` vector into an ordinary one. Without it, we would sum up many short `Rle` vectors which is very slow.

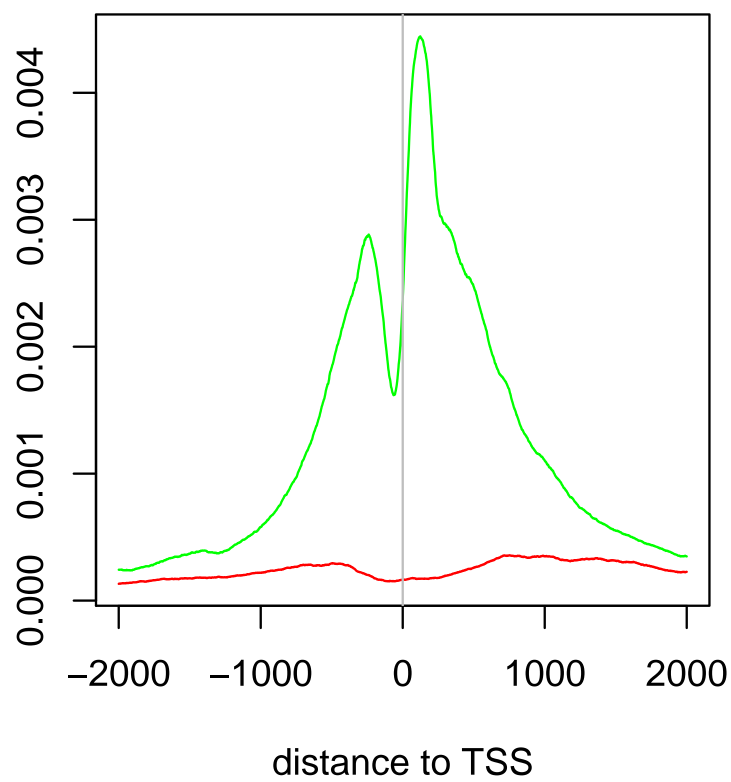


Figure 9: Correlation against transcription start sites for H3K4me1 (red) and H3K4me3 (green).

Normally, one would add all the other chromosomes, as well. For this vignette, we simply plot what we have so far:

```
> matplot( -2000:2000, cbind( tme1, tme3 ),
+    type="l", col=c("red","green"), lty="solid",
+    xlab="distance to TSS", ylab="" )
> abline( v=0, col="gray" )
```

[Output: Fig. 9.]

Session info

```
> sessionInfo()

R version 2.10.0 Under development (unstable) (2009-06-26 r48838)
x86_64-unknown-linux-gnu

locale:
[1] en_GB.UTF-8

attached base packages:
[1] grid      stats     graphics  grDevices utils     datasets  methods
[8] base

other attached packages:
[1] biomaRt_2.1.0       HilbertVisGUI_1.3.1 HilbertVis_1.3.2
[4] ShortRead_1.3.16    lattice_0.17-25     BSgenome_1.13.6
[7] Biostrings_2.13.22  IRanges_1.3.28

loaded via a namespace (and not attached):
[1] Biobase_2.5.4 hwriter_1.1   RCurl_0.98-1  XML_2.5-3
```

Version history

- v1: 2008-07-21
- v2: 2009-06-30

References

[BCC+07] A. Barski, S. Cuddapah, K. Cui, T.-Y. Roh, D. E. Schones, Z. Wang, G. Wei, I. Chepelev, K. Zhao. *High-resolution profiling of histone methylations in the human genome.* Cell **129** (2007), 823.

[Hil91] D. Hilbert. *Über stetige Abbildungen einer Linie auf ein Flächenstück.* Mathematische Annalen **38** (1891), 459.

[Kei96] D. A. Keim. *Pixel-oriented visualization techniques for exploring very large data bases.* J. Comp. Graph. Stat. **5** (1996), 58.

[LRD08] H. Li, J. Ruan, R. Durbin. *Mapping short DNA sequencing reads and calling variants using mapping quality scores.* Genome Res. **18** (2008), 1851.

[Mor] M. Morgan. *ShortRead: Base classes and methods for high-throughput short-read sequencing data.* R package version 0.1.23.

[Pea90] G. Peano. *Sur une courbe qui rempli toute une aire plaine.* Mathematische Annalen **36** (1890), 157.