genomeview Documentation

Noah Spies

Aug 11, 2023

Contents:

1 GenomeView Details 1

2 Convenience Methods for Displaying Genomic Data 3

3 Adding a coordinate axis 5

4 Visualizing read information in BAM files 7

5 Plotting continuous genomic data 9

6 Rendering shapes and text 11

7 Advanced Usage 13

8 What is GenomeView? 15

9 Installation 17

10 Quick Start 19

Index 21

CHAPTER 1

GenomeView Details

In the previous section, we saw how the `genomeview.visualize_data()` quickly assembled a document to visualize read data. Here, we'll discuss the behind-the-scenes work of setting up a document, views and adding tracks. These steps are useful if you wish to customize any aspect of the visualization process.

- *Step 1: creating the document*
- *Step 2: creating the genome views*
- *Step 3: adding the tracks to the genome view*
- *Step 4: exporting the visualization*

1.1 Step 1: creating the document

First we'll need a document. The only argument is to define the width of the view (think of this as its pixel width):

```
doc = genomeview.Document(900)
```

The document is where all the genome views will end up.

1.2 Step 2: creating the genome views

We're starting to get into the action here – a genome view defines a set of coordinates to visualize, and allows the addition of a number of tracks displaying different types of data for those coordinates.

To create a genome view, you'll optionally first create a genome "source" (basically a link to the reference genome sequence). The genome source is only required if rendering mismatches at the nucleotide level. Note that the reference file can be streamed over the internet if the index file is also present; for example, this version of hs37d5 (aka hg19/hg37).

Then, derive a view with the coordinates you'd like to visualize:

```
source = genomeview.FastaGenomeSource("/path/to/hg19.fasta")
view = genomeview.GenomeView("chr1", 219158937, 219169063, "+", source)
doc.add_view(view)
```

You can add as many genome views as you'd like to a single document, allowing you to visualize multiple genomic loci in the same document. Use `genomeview.ViewRow` to render multiple views in a horizontal row.

1.3 Step 3: adding the tracks to the genome view

The next step is to create tracks visualizing the actual data and add them to the genome view. Tracks are visualized in the order that they're added, so if you'd like to put the axis at the top, add it first, and if you'd like it at the bottom, add it last.

For example:

```
bam_track_hg002 = genomeview.SingleEndBAMTrack("/path/to/hg002.sorted.bam", name=
↪"HG002")
view.add_track(bam_track_hg002)

axis_track = genomeview.Axis()
view.add_track(axis_track)
```

1.4 Step 4: exporting the visualization

As mentioned in the previous section, the document can easily be visualized in-line in jupyter simply by placing the name of the document variable by itself at the end of a cell.

In addition, documents can be saved to SVG, PDF or PNG files using the *`genomeview.save()`* (format is inferred from the provided file-name extension).

Note that conversion to PDF/PNG requires inkscape, libRsvg or (PDF only) webkitToPDF to be installed.

CHAPTER 2

Convenience Methods for Displaying Genomic Data

`genomeview.visualize_data`(*file_paths*, *chrom*, *start*, *end*, *reference_path=None*, *width=900*, *axis_on_top=False*)

Creates a GenomeView document to display the data in the specified files (eg bam, bed, etc).

Parameters

- **file_paths** – this specifies the file paths to be rendered. It must be either a list/tuple of the paths, or a dictionary mapping {track_name:path}. (If you are using a python version prior to 3.6, use collections.ordereddict to ensure the order remains the same.) Currently supports files ending in .bam, .cram, .bed, .bed.gz, .bigbed, or .bigwig (or .bw). Most of these file types require a separate index file to be present (eg a .bam.bai or a .bed.gz.tbi file must exist).
- **chrom** – chromosome (or contig) to be rendered
- **start** – start coordinate of region to be rendered
- **end** – end coordinate of region to be rendered
- **reference_path** – path to fasta file specifying reference genomic sequence. This is required in order to display mismatches in bam tracks.
- **width** – the pixel width of the document
- **axis_on_top** – specifies whether the axis should be added at the bottom (default) or at the top

`genomeview.save`(*doc*, *outpath*, *outformat=None*)

Saves document *doc* to a file at *outpath*. By default, this file will be in SVG format; if it ends with .pdf or .png, or if outformat is specified, the document will be converted to PDF or PNG if possible.

Conversion to PDF and PNG require rsvg-convert (provided by librsvg), inkscape or webkitToPDF (PDF conversion only).

`genomeview.doc`

the `genomeview.Document` to be saved

`genomeview.outpath`

a string specifying the file to save to; file extensions of .pdf or .png will change the default output format

`genomeview.`**`outformat`**

override the file format; must be one of "pdf", "png", or (the default) "svg"

CHAPTER 3

Adding a coordinate axis

- *Axis*

3.1 Axis

class genomeview.**Axis**(*name=None*)

Displays coordinates for the current genomic view

genomeview.**get_ticks**(*start*, *end*, *target_n_labels=10*)

Tries to put an appropriate number of ticks at nice round coordinates between the genomic positions *start* and *end*. Tries but doesn't guarantee to create *target_n_labels* number of ticks / labels.

Returns a list of tuples (coordinate, label), where "label" is a nicely formatted string describing the coordinate

CHAPTER 4

Visualizing read information in BAM files

- *Single-ended read view*
- *Paired-ended read view*
- *Grouping reads by attribute*

4.1 Single-ended read view

class genomeview.**SingleEndBAMTrack**(*bam_path*, *name=None*)

Displays bam as single-ended reads

nuc_colors

defines the SVG colors used to display mismatched nucleotides

Type dict

insertion_color, deletion_color, clipping_color

SVG colors for insertions, deletions and soft/hard clipping

Type str

quick_consensus

specify whether the quick consensus mode should be used. When activated, mismatches wrt the reference genome are only shown when at least several reads support a variant at that position (useful when displaying high-error rate data types eg pacbio). Only relevant if draw_mismatches is also True. (default: True)

Type bool

draw_mismatches

whether to show mismatches with respect to the reference genome. (default: True).

Type bool

include_secondary
whether to draw alignments specified as "secondary" in the BAM flags (default: True).

Type bool

include_read_fn
callback function used to specify which reads should be included in the display. The function takes as its only argument a read (pysam.AlignedSegment) and returns True (yes, display the read) or False (no, don't display). If this function is not specified, by default all reads are shown.

draw_interval(*renderer*, *interval*)
Draw a read and then, if `self.draw_mismatches` is True, draw mismatches/indels on top.

fetch()
Iterator over reads from the bam file

Overload this method in subclasses to feed this track reads from a different source (for example, reads that are already in memory, rather than being read from a file).

match_chrom_format(*chrom*)
Ensures that the input argument *chrom* matches the chromosome name formatting in the bam file being visualized (ie "chr14" vs "14").

4.2 Paired-ended read view

class genomeview.**PairedEndBAMTrack**(*bam_path*, *name=None*)
Displays paired-end reads together (otherwise, same as `genomeview.SingleEndBAMTrack`).

overlap_color
color used to highlight portions of read pairs that are overlapping one another

4.3 Grouping reads by attribute

class genomeview.**GroupedBAMTrack**(*bam_path*, *keyfn*, *bam_track_class*, *name=None*)
Displays reads from a BAM, separated out into groups based on a feature of the reads. For example, group reads based on the value of tag.

keyfn
the function used to specify the groupings of reads. Takes as input a read (`pysam.AlignedSegment`).

bam_track_class
the class used to display each group of reads, should probably be either `genomeview.bamtrack.PairedEndBAMTrack` or `genomeview.bamtrack.SingleEndBAMTrack`

space_between
the amount of space (pixels) between groups. (Default: 10)

Type float

category_label_fn
a function that nicely formats the category labels. Takes as argument the result of the keyfn and should return a string. (Default: render as string)

CHAPTER 5

Plotting continuous genomic data

- *GraphTrack*

5.1 GraphTrack

class genomeview.**GraphTrack**(*name=None*, *x=None*, *y=None*)

Visualizes quantitative data as a line across coordinates within the current genomic view.

One or more datasets can be visualized (with different colors) on the same track using the `add_series()` method.

add_series(*x*, *y*, *color=None*, *label=None*)

Add a dataset corresponding to a single line in the track (ie, a "series"). Note that while a single GraphTrack can visualize multiple datasets, they are all plotted on the same y-axis and so should share the same units.

Parameters

- **x** – a list of genomic coordinates
- **y** – a list of data values; each y-value must correspond to a single x-value
- **color** – an SVG color for the line being plotted
- **label** – an optional text label for the graph being plotted (currently unused)

class genomeview.**BigWigTrack**(*path*, *nbins=1000*, *name=None*)

Visualizes continuous-valued data from a bigwig file. Requires pyBigWig module to be installed. Supports using web URLs as well as file paths.

CHAPTER 6

Rendering shapes and text

6.1 Drawing shapes and text

Each of the functions below takes as arguments information about the screen coordinates, and yields a series of SVG tags specifying lines, shapes and text to be drawn. Each function also accepts optional `kwdargs` which can include additional SVG attributes such as `fill` and `stroke` colors.

class genomeview.svg.**Renderer**(*backend*, *x*, *y*, *width*, *height*)

text(*self*, *x*, *y*, *text*, *size=10*, *anchor="middle"*, *family="Helvetica"*, ***kwdargs*)
Draws text. Anchor specifies the horizontal positioning of the text with respect to the point (x,y) and must be one of `{start, middle, end}`. Font `size` and `family` can also be specified.

text_with_background(*self*, *x*, *y*, *text*, *size=10*, *anchor="middle"*, *family="Helvetica"*, *text_color="black"*, *bg="white"*, *bg_opacity=0.8*, ***kwdargs*)
Draws text on an opaque background. `bg` specifies the background color, and `bg_opacity` ranges from 0 (completely transparent) to 1 (completely opaque).

rect(*x*, *y*, *width*, *height*, ***kwdargs*)
Draws a rect whose upper left point is (x,y) with the specified width and height.

line(*x1*, *y1*, *x2*, *y2*, ***kwdargs*)
Draws a line from (x1,y1) to (x2,y2).

line_with_arrows(*x1*, *y1*, *x2*, *y2*, *n=5*, *direction="right"*, *color="black"*, ***kwdargs*)
Draws a line from (x1,y1) to (x2,y2) with `n` arrows spaced between. The line and arrows will be drawn in the specified `color`.

6.2 Converting genomic coordinates to screen coordinates

class genomeview.**Scale**(*chrom*, *start*, *end*, *strand*, *source*)
Maintains information about a projection of a specific genomic interval into screen coordinates.

That is, we're interested in visualizing an interval (chrom:start-end) on a canvas of a specified pixel width. The scale enables converting genomic coordinates into the display coordinates.

get_seq(*start=None*, *end=None*, *strand='+'*)
Gets the nucleotide sequence of an interval. By default, returns the sequence for the current genomic interval.

relpixels(*genomic_size*)
Takes a genomic length (ie, a number of basepairs) and converts it to a relative screen length in pixels.

topixels(*genomic_position*)
Converts a genomic position to a pixel location in the current coordinate system.

CHAPTER 7

Advanced Usage

GenomeView is designed to be easily extended. The source code is a good place to start, but this document gives some insights into the design philosophy and the components involved in visualizing genomic data.

7.1 Graphics model

As explained in the tutorial, GenomeView lays out visual elements by nesting documents, views and tracks together, for example:

```
doc
-> genomeview 1
----> bam track 1
----> bam track 2
----> axis
-> genomeview 2
----> bam track 3
----> axis
```

Visualization is accomplished by traversing that hierarchy and asking each element to display itself as SVG code. In practice, each visual element has a `render()` method which yields lines of SVG code which specify the lines, shapes, text, etc used to display genomic data.

To simplify this process, an *`SVG Renderer`* object is passed along, providing a series of SVG primitive commands to enable drawing lines, shapes and text. This renderer takes care of compositing different visual elements such that x,y-coordinates can be specified relative to a local coordinate system. Shapes extending out of the region allocated for a specific visual element are clipped (eg reads extending past the last coordinate of the current window).

In addition, each `Track` maintains a *`Scale`* object which can be used to convert genomic coordinates to screen coordinates.

7.2 Adding visual elements to existing tracks

Each track object can include one or more pre-renderer or post-renderer functions which are used to draw items under or above the track. For example, the following post-renderer adds some text to the middle of an existing track:

```
def draw_label(renderer, element):
    x_middle = element.scale.pixel_width / 2
    y_middle = element.height / 2
    yield from renderer.text_with_background(x_middle, y_middle, "hello", anchor=
↪"middle")

track.prerenderers = [draw_label]
```

See the bams.ipynb jupyter notebook for more examples.

7.3 Custom tracks

While the tracks included with GenomeView contain numerous configurable options, sometimes it is necessary to either create a subclass providing new functionality.

Tracks should subclass from `genomeview.Track` or one of its subclasses. The `layout` method is called once prior to rendering in order to determine the height of the element (variable height tracks allow visualizing all reads in a window even when they are stacked high). Then the `render(self, renderer)` method is called, taking as argument a *`genomeview.svg.Renderer`*. Note that the `scale` object can be accessed as `self.scale` in order to convert genomic coordinates to screen positions.

Renderers should use the python 3.3+ `yield from renderer.shape()` command to yield complex multi-line SVG shapes created by the renderer.

CHAPTER 8

What is GenomeView?

GenomeView visualizes genomic data straight from python. Features include:

- Easily extensible
- Integrates with jupyter notebook / jupyterlab
- High-quality vector output to standard SVG format
- Includes built-in tracks to visualize:
  - BAMs (short and long reads)
    - Both single-ended and paired-ended views available
    - Includes a cython-optimized quick consensus module to visualize error-prone long-read data
    - Group BAM reads by tag or other features using python callbacks
  - Graphical data such as coverage tracks, wiggle files, etc

The output is suitable for static visualization in screen or print formats. GenomeView is not designed to produce interactive visualizations, although the python interface, through jupyter, provides an easy interface to quickly create new visualizations.

CHAPTER 9

Installation

GenomeView requires python 3.3 or greater. The following shell command should typically suffice for installing the latest release:

```
pip install genomeview
```

Or to install the bleeding edge from github:

```
pip install -U git+https://github.com/nspies/genomeview.git
```

To display bigWig graphical tracks, the pyBigWig python package must also be installed, eg `pip install pyBigWig`.

CHAPTER 10

Quick Start

To produce the visualization above, a single line of code suffices (in addition to information about the locations of the data and coordinates to be visualized):

```
dataset_paths = ["/path/to/pacbio_single_end_dataset.bam",
                 "/path/to/illumina_paired_end_dataset.bam",
                 "/path/to/genes.bed.gz"]
reference = "/path/to/reference.fa"

chrom = "chr1"
start = 224368899
end =   224398899

doc = genomeview.visualize_data(dataset_paths, chrom, start, end, reference)
```

If you are using jupyter notebook or jupyterlab, documents can be displayed simply by placing the name of the document on the last line of a cell by itself and running the cell.

To render the document to file, use the simple `genomeview.save()` command:

```
genomeview.save(doc, "/path/to/output.svg")    # or .png/.pdf
```

For more details on setting up your own document with fine-grained control over how the tracks are created and visualized, see the *next section*.

Index

A

add_series() (*genomeview.GraphTrack method*), 9
Axis (*class in genomeview*), 5

B

bam_track_class (*genomeview.GroupedBAMTrack attribute*), 8
BigWigTrack (*class in genomeview*), 9

C

category_label_fn (*genomeview.GroupedBAMTrack attribute*), 8

D

doc (*in module genomeview*), 3
draw_interval() (*genomeview.SingleEndBAMTrack method*), 8
draw_mismatches (*genomeview.SingleEndBAMTrack attribute*), 7

F

fetch() (*genomeview.SingleEndBAMTrack method*), 8

G

get_seq() (*genomeview.Scale method*), 12
get_ticks() (*in module genomeview*), 5
GraphTrack (*class in genomeview*), 9
GroupedBAMTrack (*class in genomeview*), 8

I

include_read_fn (*genomeview.SingleEndBAMTrack attribute*), 8
include_secondary (*genomeview.SingleEndBAMTrack attribute*), 7

K

keyfn (*genomeview.GroupedBAMTrack attribute*), 8

L

line() (*genomeview.svg.Renderer method*), 11
line_with_arrows() (*genomeview.svg.Renderer method*), 11

M

match_chrom_format() (*genomeview.SingleEndBAMTrack method*), 8

N

nuc_colors (*genomeview.SingleEndBAMTrack attribute*), 7

O

outformat (*in module genomeview*), 3
outpath (*in module genomeview*), 3
overlap_color (*genomeview.PairedEndBAMTrack attribute*), 8

P

PairedEndBAMTrack (*class in genomeview*), 8

Q

quick_consensus (*genomeview.SingleEndBAMTrack attribute*), 7

R

rect() (*genomeview.svg.Renderer method*), 11
relpixels() (*genomeview.Scale method*), 12
Renderer (*class in genomeview.svg*), 11

S

save() (*in module genomeview*), 3
Scale (*class in genomeview*), 11
SingleEndBAMTrack (*class in genomeview*), 7
space_between (*genomeview.GroupedBAMTrack attribute*), 8

T

`text()` (*genomeview.svg.Renderer method*), 11
`text_with_background()` (*genomeview.svg.Renderer method*), 11
`topixels()` (*genomeview.Scale method*), 12

V

`visualize_data()` (*in module genomeview*), 3