BiocPy: Porting Bioconductor representations to Python

Jayaram Kancherla

1/23/23

Table of contents

Welcome

BiocPy is an effort to bring core data structures and representations from Bioconductor to Python.

Packages in `BiocPy`

Currently, the following **core** packages are available

- `BiocFrame` (GitHub, Docs): A lite version of dataframes. It is not equivalent to `Pandas` but provides many similar operations.
- `GenomicRanges` (GitHub, Docs, BioC): Container class to represent genomic locations and support genomic analysis. Similar to Bioconductor's GenomicRanges.
- `SummarizedExperiment` (GitHub, Docs, BioC): Container class to represent genomic experiments, following Bioconductor's SummarizedExperiment.
- `SingleCellExperiment` (GitHub, Docs, BioC): Container class to represent single-cell experiments; follows Bioconductor's SingleCellExperiment.
- `MultiAssayExperiment` (GitHub, Docs, BioC): Container class to represent multiple experiments and assays performed over a set of samples. follows Bioconductor's MAE R/Bioc Package.

Utility packages

- `rds2py` (GitHub, Docs): Parse, extract and create Python representations for datasets stored in RDS files. Currently supports Bioconductor's `SummarizedExperiment` and `SingleCellExperiment` objects.
- `mopsy` (GitHub, Docs): Convenience library to perform row/column operations over numpy and scipy matrices. Provides an interface similar to base R matrix methods/MatrixStats methods.
- `pyBiocFileCache` (GitHub, Docs, BioC): File system based cache for resources & metadata.

This book will focus on end user tutorials for core Python packages we develop.

####
Notes
format:
html:
code-
fold:
false
exe-
cute:
en-
abled:
true
cache:
true

```
for nicer prints in this document

::: {.cell execution_count=1}
``` {.python .cell-code}
from rich import print
```
:::

Construct a `GenomicRanges` object

`GenomicRanges` holds genomic intervals and annotation about those intervals. Is it similar t

Import UCSC annotation or GTF file

A common way of accessing genome annotations for various organisms is from UCSC.

```python
import genomicranges
gr = genomicranges.readUCSC(genome="hg19")
print(gr)
```
```

````
Similarly methods are available to read a gtf file from disk as `GenomicRanges` object

```python
gr = genomicranges.readGTF(<PATH TO GTF>)
```

## from Pandas `DataFrame`

:::{.callout-note}
The `DataFrame` ***must*** contain columns ***`seqnames`, `starts` and `ends`*** to represent
:::

Similarly one can construct a `GenomicRanges` object from an existing Pandas `DataFrame`.

::: {.cell execution_count=2}
``` {.python .cell-code}
import genomicranges
import pandas as pd
from random import random

df = pd.DataFrame(
    {
        "seqnames": ["chr1", "chr2", "chr1", "chr3", "chr2"],
        "starts": [101, 102, 103, 104, 109],
        "ends": [112, 103, 128, 134, 111],
        "strand": ["*", "-", "*", "+", "-"],
        "score": range(0, 5),
        "GC": [random() for _ in range(5)],
    }
)

gr = genomicranges.fromPandas(df)
print(gr)
```

::: {.cell-output .cell-output-display}
```

        Class GenomicRanges with 5 intervals and 3 metadata columns
          columnNames: ['seqnames', 'starts', 'ends', 'strand', 'score', 'GC']

```
:::
````

```
:::


## from a dictionary

:::{.callout-note}
The object ***must*** contain keys ***`seqnames`, `starts` and `ends`*** to represent genomic
:::

::: {.cell execution_count=3}
``` {.python .cell-code}
from genomicranges import GenomicRanges
from random import random

obj = {
    "seqnames": [
        "chr1",
        "chr2",
        "chr2",
        "chr2",
        "chr1",
        "chr1",
        "chr3",
        "chr3",
        "chr3",
        "chr3",
    ],
    "starts": range(100, 110),
    "ends": range(110, 120),
    "strand": ["-", "+", "+", "*", "*", "+", "+", "+", "-", "-"],
    "score": range(0, 10),
    "GC": [random() for _ in range(10)],
}

gr = GenomicRanges(obj)
print(gr)
```

::: {.cell-output .cell-output-display}
```
        Class GenomicRanges with 10 intervals and 3 metadata columns
          columnNames: ['seqnames', 'starts', 'ends', 'strand', 'score', 'GC']
```
```

```
```
:::
:::

Set sequence information

::: {.cell execution_count=4}
``` {.python .cell-code}
from genomicranges import SeqInfo

seq_obj = {
    "seqnames": ["chr1", "chr2", "chr3",],
    "seqlengths": range(125, 128),
    "isCircular": [random() < 0.5 for _ in range(3)],
    "genome": "hg19",
}

seq = SeqInfo(seq_obj)

gr.seqInfo = seq
print(gr)
```

::: {.cell-output .cell-output-display}
```

        Class GenomicRanges with 10 intervals and 3 metadata columns
          columnNames: ['seqnames', 'starts', 'ends', 'strand', 'score', 'GC']

```
:::
:::

Getters and Setters

Accessors are available to access various properties of a `GenomicRanges` object.

::: {.cell execution_count=5}
``` {.python .cell-code}
# To access seqnames in the object
```

```
print(gr.seqnames)
```

::: {.cell-output .cell-output-display}
```
['chr1', 'chr2', 'chr2', 'chr2', 'chr1', 'chr1', 'chr3', 'chr3', 'chr3', 'chr3']
```
:::
:::

Access ***widths*** of each interval in the object

::: {.cell execution_count=6}
``` {.python .cell-code}
print(gr.width)
```

::: {.cell-output .cell-output-display}
```
[10, 10, 10, 10, 10, 10, 10, 10, 10, 10]
```
:::
:::

:::{.callout-note}
refer to the documentation on [Class:GenomicRanges](https://biocpy.github.io/GenomicRanges/a
:::

Following a *pythonic syntax*, you can also set or update the properties of the class.

To update the ***scores*** in the object,

::: {.cell execution_count=7}
``` {.python .cell-code}
gr.score = [round(random(), 2) for _ in range(10)]
print(gr)
print(f"scores: {gr.score}")
```

::: {.cell-output .cell-output-display}
```
```

```
        Class GenomicRanges with 10 intervals and 3 metadata columns
          columnNames: ['seqnames', 'starts', 'ends', 'strand', 'score', 'GC']

```
:::

::: {.cell-output .cell-output-display}
```
scores: [0.39, 0.68, 0.27, 0.35, 0.92, 0.55, 0.79, 0.16, 0.41, 0.41]
```
:::
:::

Add new metadata columns

::: {.cell execution_count=8}
``` {.python .cell-code}
gr["new_col"] = [round(random(), 3) for _ in range(10)]
print(gr)
```

::: {.cell-output .cell-output-display}
```
        Class GenomicRanges with 10 intervals and 4 metadata columns
          columnNames: ['seqnames', 'starts', 'ends', 'strand', 'score', 'GC', 'new_col']

```
:::
:::

Column method
Use the `column()` method to quickly access any column in the object. Useful for non-standar

::: {.cell execution_count=9}
``` {.python .cell-code}
print(gr.column("new_col"))
```

::: {.cell-output .cell-output-display}

```
[0.059, 0.27, 0.534, 0.967, 0.836, 0.051, 0.172, 0.69, 0.721, 0.821]
```
:::
:::

Access Ranges

`ranges()` is a generic method to access only the genomic intervals as dictionary, pandas `Da

::: {.cell execution_count=10}
``` {.python .cell-code}
# default to dict
print(gr.ranges())
```

::: {.cell-output .cell-output-display}
```
{
    'seqnames': ['chr1', 'chr2', 'chr2', 'chr2', 'chr1', 'chr1', 'chr3', 'chr3', 'chr3', 'ch
    'starts': range(100, 110),
    'ends': range(110, 120),
    'strand': ['-', '+', '+', '*', '*', '+', '+', '+', '-', '-']
}
```
:::
:::

:::{.callout-tip}
you can pass in any class that takes a dictionary as an input for `returnType`.
:::

::: {.cell execution_count=11}
``` {.python .cell-code}
# as pandas DataFrame
gr.ranges(returnType=pd.DataFrame)
```

::: {.cell-output .cell-output-stderr}
```
/opt/hostedtoolcache/Python/3.10.9/x64/lib/python3.10/site-packages/IPython/core/formatters.p
```

```
  return method()
```
:::

::: {.cell-output .cell-output-display execution_count=11}

```{=tex}
\begin{tabular}{llrrl}
\toprule
{} & seqnames &  starts &  ends & strand \\
\midrule
0 &     chr1 &     100 &   110 &      - \\
1 &     chr2 &     101 &   111 &      + \\
2 &     chr2 &     102 &   112 &      + \\
3 &     chr2 &     103 &   113 &      * \\
4 &     chr1 &     104 &   114 &      * \\
5 &     chr1 &     105 &   115 &      + \\
6 &     chr3 &     106 &   116 &      + \\
7 &     chr3 &     107 &   117 &      + \\
8 &     chr3 &     108 &   118 &      - \\
9 &     chr3 &     109 &   119 &      - \\
\bottomrule
\end{tabular}
```

:::
:::

`granges()` method returns a new `GenomicRanges` object of just the genomic locations

::: {.cell execution_count=12}
``` {.python .cell-code}
print(gr.granges())
```

::: {.cell-output .cell-output-display}
```
        Class GenomicRanges with 10 intervals and 1 metadata columns
          columnNames: ['seqnames', 'starts', 'ends', 'strand']

```

````
:::
:::


## Access metadata columns

This will access non-interval columns from the object.

::: {.cell execution_count=13}
``` {.python .cell-code}
print(gr.mcols())
```

::: {.cell-output .cell-output-display}
```
OrderedDict([('score', [0.39, 0.68, 0.27, 0.35, 0.92, 0.55, 0.79, 0.16, 0.41, 0.41]), ('GC',
0.6929802769659172, 0.6104783289252426, 0.9197865893490663, 0.25561287568532676, 0.319673537
0.5690516797462629, 0.5806161039109258, 0.4116865668330606, 0.2319194965797422]), ('new_col'
0.967, 0.836, 0.051, 0.172, 0.69, 0.721, 0.821])])
```
:::
:::




`<!-- quarto-file-metadata: eyJyZXNvdXJjZURpciI6ImNoYXB0ZXJzL2dyYW5nZXMifQ== -->`{=html}

```{=html}
<!-- quarto-file-metadata: eyJyZXNvdXJjZURpciI6ImNoYXB0ZXJzL2dyYW5nZXMiLCJib29rSXRlbVR5cGUiOi
```

# Slice and Iterate Operations {.unnumbered}

```````{.quarto-title-block template='/opt/quarto/share/projects/book/pandoc/title-block.md'
---
format:
  html:
    code-fold: false
execute:
  enabled: true
  cache: true
````
````

---

for nicer prints in this document

```
from rich import print
```

Lets resue the same `GenomicRanges` object from the previous section.

```
from genomicranges import GenomicRanges
from random import random

obj = {
    "seqnames": [
        "chr1",
        "chr2",
        "chr2",
        "chr2",
        "chr1",
        "chr1",
        "chr3",
        "chr3",
        "chr3",
        "chr3",
    ],
    "starts": range(100, 110),
    "ends": range(110, 120),
    "strand": ["-", "+", "+", "*", "*", "+", "+", "+", "-", "-"],
    "score": range(0, 10),
    "GC": [random() for _ in range(10)],
}

index = [f"idx_{i}" for i in range(10)]

gr = GenomicRanges(obj, rowNames=index)
print(gr)
```

```
Class GenomicRanges with 10 intervals and 3 metadata columns
  columnNames: ['seqnames', 'starts', 'ends', 'strand', 'score', 'GC']
```

Slice methods

slice by index

You can slice a `GenomicRange` object using the subset (`[]`) operator.

```
# slice the first 5 rows
print(gr[:5,])
```

```
Class GenomicRanges with 5 intervals and 3 metadata columns
  columnNames: ['seqnames', 'starts', 'ends', 'strand', 'score', 'GC']
```

slice by index names

you can also provide a list of index names to subset the object

```
index_to_subset = ["idx_8", "idx_7"]

print(gr[index_to_subset,])
```

```
Class GenomicRanges with 2 intervals and 3 metadata columns
  columnNames: ['seqnames', 'starts', 'ends', 'strand', 'score', 'GC']
```

Iterate over rows

To iterate over the rows of the object,

```
for index, row in gr:
    print(f"index: {index}, row: {row}")
```

```
index: idx_0, row: OrderedDict([('seqnames', 'chr1'), ('starts', 100), ('ends', 110), ('stra
0), ('GC', 0.15699855097599047)])

index: idx_1, row: OrderedDict([('seqnames', 'chr2'), ('starts', 101), ('ends', 111), ('stra
1), ('GC', 0.9432433256834035)])

index: idx_2, row: OrderedDict([('seqnames', 'chr2'), ('starts', 102), ('ends', 112), ('stra
2), ('GC', 0.2523324394339238)])

index: idx_3, row: OrderedDict([('seqnames', 'chr2'), ('starts', 103), ('ends', 113), ('stra
3), ('GC', 0.930675133630442)])

index: idx_4, row: OrderedDict([('seqnames', 'chr1'), ('starts', 104), ('ends', 114), ('stra
4), ('GC', 0.09263538061623999)])

index: idx_5, row: OrderedDict([('seqnames', 'chr1'), ('starts', 105), ('ends', 115), ('stra
5), ('GC', 0.3188121917847977)])

index: idx_6, row: OrderedDict([('seqnames', 'chr3'), ('starts', 106), ('ends', 116), ('stra
6), ('GC', 0.5518213406637695)])

index: idx_7, row: OrderedDict([('seqnames', 'chr3'), ('starts', 107), ('ends', 117), ('stra
7), ('GC', 0.5441774832655979)])

index: idx_8, row: OrderedDict([('seqnames', 'chr3'), ('starts', 108), ('ends', 118), ('stra
8), ('GC', 0.4082882562555442)])

index: idx_9, row: OrderedDict([('seqnames', 'chr3'), ('starts', 109), ('ends', 119), ('stra
9), ('GC', 0.46425062034000164)])
```

Interval based operations

> **Note**
>
> For detailed description, checkout Bioc GenomicRanges documentation

for nicer prints,

```
from rich import print
```

Lets resue the same `GenomicRanges` object from the previous section.

```
from genomicranges import GenomicRanges
from random import random

obj = {
    "seqnames": [
        "chr1",
        "chr2",
        "chr2",
        "chr2",
        "chr1",
        "chr1",
        "chr3",
        "chr3",
        "chr3",
        "chr3",
    ],
    "starts": [i for i in range(100, 110)],
    "ends": [i for i in range(110, 120)],
    "strand": ["-", "+", "+", "*", "*", "+", "+", "+", "-", "-"],
    "score": [i for i in range(0, 10)],
    "GC": [random() for _ in range(10)],
}

index = [f"idx_{i}" for i in range(10)]
```

```
gr = GenomicRanges(obj, rowNames=index)
print(gr)
```

```
      Class GenomicRanges with 10 intervals and 3 metadata columns
        columnNames: ['seqnames', 'starts', 'ends', 'strand', 'score', 'GC']
```

Intra-range transformations

- `flank()`: flank the intervals based on start or end or both.
- `shift()`: shifts all the ranges specified by the shift argument.
- `resize()`: resizes the ranges to the specified width where either the start, end, or center is used as an anchor
- `narrow()`: narrows the ranges
- `promoters()`: promoters generates promoter ranges for each range relative to the TSS.The promoter range is expanded around the TSS according to the upstream and downstream parameters.
- `restrict()`: restricts the ranges to the interval(s) specified by the start and end arguments
- `trim()`: trims out-of-bound ranges located on non-circular sequences whose length is not NA.

a few examples on how to use these methods,

```
# flank
flanked_gr = gr.flank(width=10, start=False, both=True)
print(flanked_gr)
```

```
Class GenomicRanges with 10 intervals and 3 metadata columns
  columnNames: ['seqnames', 'starts', 'ends', 'strand', 'score', 'GC']
```

```
# resize
resized_gr = gr.resize(width=10, fix="end", ignoreStrand=True)
print(resized_gr)
```

```
Class GenomicRanges with 10 intervals and 3 metadata columns
  columnNames: ['seqnames', 'starts', 'ends', 'strand', 'score', 'GC']
```

```
# narrow
narrow_gr = gr.narrow(end=4, width=3)
print(narrow_gr)
```

```
Class GenomicRanges with 10 intervals and 3 metadata columns
  columnNames: ['seqnames', 'starts', 'ends', 'strand', 'score', 'GC']
```

```
# promoters
prom_gr = gr.promoters()
print(prom_gr)
```

```
Class GenomicRanges with 10 intervals and 3 metadata columns
  columnNames: ['seqnames', 'starts', 'ends', 'strand', 'score', 'GC']
```

Inter-range methods

- `range()`: returns a new GenomicRanges object containing range bounds for each distinct (seqname, strand) pairing.
- `reduce()`: returns a new GenomicRanges object containing reduced bounds for each distinct (seqname, strand) pairing.
- `gaps()`: Finds gaps in the GenomicRanges object for each distinct (seqname, strand) pairing
- `disjoin()`: Finds disjoint intervals across all locations for each distinct (seqname, strand) pairing.
- `isDisjoint()`: Is the object contain disjoint intervals for each distinct (seqname, strand) pairing?

```
# range
range_gr = gr.range()
print(range_gr)
```

```
Class GenomicRanges with 7 intervals and 1 metadata columns
  columnNames: ['seqnames', 'strand', 'starts', 'ends']
```

```
# reduce
reduced_gr = gr.reduce(minGapwidth=10, withRevMap=True)
print(reduced_gr)
```

```
Class GenomicRanges with 7 intervals and 2 metadata columns
  columnNames: ['seqnames', 'strand', 'starts', 'ends', 'revmap']
```

```
# gaps
gapped_gr = gr.gaps(start=103) # OR
# gapped_gr = gr.gaps(end={"chr1": 120, "chr2": 120, "chr3": 120})
print(gapped_gr)
```

```
Class GenomicRanges with 4 intervals and 1 metadata columns
  columnNames: ['seqnames', 'strand', 'starts', 'ends']
```

```
# disjoin
disjoin_gr = gr.disjoin()
print(disjoin_gr)
```

```
Class GenomicRanges with 13 intervals and 1 metadata columns
  columnNames: ['seqnames', 'strand', 'starts', 'ends']
```

1 Summary

In summary, this book has no content whatsoever.

References