mutalyzer-crossmapper

Release 2.0.1

Jeroen F.J. Laros

Nov 21, 2021

CONTENTS:

This library provides an interface to convert (cross map) between different HGVS numbering systems.

Converting between the transcript oriented c. or n. and the genomic oriented g. numbering systems can be difficult, especially when the transcript in question resides on the complement strand.

Features:

- Support for genomic positions to standard coordinates and vice versa.
- Support for noncoding positions to standard coordinates and vice versa.
- Support for coding positions to standard coordinates and vice versa.
- Support for protein positions to standard coordinates and vice versa.
- Basic classes for loci that can be used for genomic loci other than genes.

Please see ReadTheDocs for the latest documentation.

CHAPTER

ONE

QUICK START

The `Genomic` class provides an interface to conversions between genomic positions and coordinates.

```
>>> from mutalyzer_crossmapper import Genomic
>>> crossmap = Genomic()
>>> crossmap.coordinate_to_genomic(0)
1
>>> crossmap.genomic_to_coordinate(1)
0
```

On top of the functionality provided by the `Genomic` class, the `NonCoding` class provides an interface to conversions between noncoding positions and coordinates.

```
>>> from mutalyzer_crossmapper import NonCoding
>>> exons = [(5, 8), (14, 20), (30, 35), (40, 44), (50, 52), (70, 72)]
>>> crossmap = NonCoding(exons)
>>> crossmap.coordinate_to_noncoding(35)
(14, 1, 0)
>>> crossmap.noncoding_to_coordinate((14, 1))
35
```

Add the flag `inverted=True` to the constructor when the transcript resides on the reverse complement strand.

On top of the functionality provided by the `NonCoding` class, the `Coding` class provides an interface to conversions between coding positions and coordinates as well as conversions between protein positions and coordinates.

```
>>> from mutalyzer_crossmapper import Coding
>>> cds = (32, 43)
>>> crossmap = Coding(exons, cds)
>>> crossmap.coordinate_to_coding(31)
(-1, 0, -1, 0)
>>> crossmap.coding_to_coordinate((-1, 0, -1))
31
```

Again, the flag `inverted=True` can be used for transcripts that reside on the reverse complement strand.

Conversions between protein positions and coordinates are done as follows.

```
>>> crossmap.coordinate_to_protein(41)
(2, 2, 0, 0, 0)
>>> crossmap.protein_to_coordinate((2, 2, 0, 0))
41
```

1.1 Introduction

Converting between the transcript oriented c. or n. and the genomic oriented g. numbering systems can be difficult, especially when the transcript in question resides on the reverse complement strand.

There is no zero (0) in any of the HGVS numbering systems. While this is not a major problem in itself, when combined with signed integers in the c. and n. numbering systems (e.g., position -1 and 1 are adjacent), it gives rise to various off by one errors when the conversion is not done properly. The use of *offsets* for intronic positions (e.g., position 12 and 12+1 can be adjacent) introduce yet an other type of discontinuity which makes doing arithmetical operations in these numbering systems extremely tedious and error prone. Finally, for transcripts that reside on the reverse complement strand, the direction of the numbering system is opposite to that of the genomic one (e.g., if c.1 equals g.10, then c.2 equals g.9), which introduces yet an other level of complexity.

This library aims to solve all aforementioned problems by providing an interface that is able to convert from any HGVS numbering system to a conventional *coordinate* system and back.

1.2 Installation

The software is distributed via PyPI, it can be installed with `pip`:

```
pip install mutalyzer-crossmapper
```

1.2.1 From source

The source is hosted on GitHub, to install the latest development version, use the following commands.

```
git clone https://github.com/mutalyzer/crossmapper.git
cd crossmapper
pip install .
```

1.3 Library

The library provides a number of classes to perform various conversions.

1.3.1 The `Genomic` class

The `Genomic` class provides an interface to conversions between genomic positions and coordinates.

```
>>> from mutalyzer_crossmapper import Genomic
>>> crossmap = Genomic()
```

The functions `coordinate_to_genomic()` and `genomic_to_coordinate` can be used to convert to and from genomic positions.

```
>>> crossmap.coordinate_to_genomic(0)
1
>>> crossmap.genomic_to_coordinate(1)
0
```

See section *Crossmapper* for a detailed description.

1.3.2 The NonCoding class

On top of the functionality provided by the `Genomic` class, the `NonCoding` class provides an interface to conversions between noncoding positions and coordinates. Conversions between positioning systems should be done via a coordinate.

```
>>> from mutalyzer_crossmapper import NonCoding
>>> exons = [(5, 8), (14, 20), (30, 35), (40, 44), (50, 52), (70, 72)]
>>> crossmap = NonCoding(exons)
```

Now the functions `coordinate_to_noncoding()` and `noncoding_to_coordinate()` can be used. These functions use a 3-tuple to represent a noncoding position.

Table 1: Noncoding positions.

index	description
0	Transcript position.
1	Offset.
2	Upstream or downstream offset.

In our example, the HGVS position "g.36" (coordinate `35`) is equivalent to position "n.14+1". We can convert between these two as follows.

```
>>> crossmap.coordinate_to_noncoding(35)
(14, 1, 0)
```

When the coordinate is upstream or downstream of the transcript, the last element of the tuple denotes the offset with respect to the transcript. This makes it possible to distinguish between intronic positions and those outside of the transcript.

```
>>> crossmap.coordinate_to_noncoding(2)
(1, -3, -3)
>>> crossmap.coordinate_to_noncoding(73)
(22, 2, 2)
```

Note that this last element is optional (and ignored) when a conversion to a coordinate is requested.

```
>>> crossmap.noncoding_to_coordinate((14, 1))
35
```

For transcripts that reside on the reverse complement strand, the `inverted` parameter should be set to `True`. In our example, HGVS position "g.36" (coordinate `35`) is now equivalent to position "n.9-1".

```
>>> crossmap = NonCoding(exons, inverted=True)
>>> crossmap.coordinate_to_noncoding(35)
(9, -1, 0)
>>> crossmap.noncoding_to_coordinate((9, -1))
35
```

See section *Crossmapper* for a detailed description.

1.3.3 The Coding class

The Coding class provides an interface to all conversions between positioning systems and coordinates. Conversions between positioning systems should be done via a coordinate.

```
>>> from mutalyzer_crossmapper import Coding
>>> exons = [(5, 8), (14, 20), (30, 35), (40, 44), (50, 52), (70, 72)]
>>> cds = (32, 43)
>>> crossmap = Coding(exons, cds)
```

On top of the functionality provided by the NonCoding class, the functions coordinate_to_coding() and coding_to_coordinate() can be used. These functions use a 4-tuple to represent a coding position.

Table 2: Coding positions.

index	description
0	Transcript position.
1	Offset.
2	Region.
3	Upstream or downstream offset.

The region denotes the location of the position with respect to the CDS. This is needed in order to work with the HGVS "-" and "*" positions.

Table 3: Coding position regions.

value	description	HGVS example
-1	Upstream of the CDS.	"c.-10"
0	In the CDS.	"c.1"
1	Downstream of the CDS.	"c.*10"

In our example, the HGVS position "g.32" (coordinate 31) is equivalent to position "c.-1". We can convert between these two as follows.

```
>>> crossmap.coordinate_to_coding(31)
(-1, 0, -1, 0)
>>> crossmap.coding_to_coordinate((-1, 0, -1))
31
```

The coordinate_to_coding() function accepts an optional degenerate argument. When set to True, positions outside of the transcript are no longer described using the offset notation.

```
>>> crossmap.coordinate_to_coding(4)
(-11, -1, -1, -1)
>>> crossmap.coordinate_to_coding(4, True)
(-12, 0, -1, -1)
```

Additionally, the functions coordinate_to_protein() and protein_to_coordinate() can be used. These functions use a 5-tuple to represent a protein position.

Table 4: Protein positions.

index	description
0	Protein position.
1	Codon position.
2	Offset.
3	Region.
4	Upstream or downstream offset.

In our example the HGVS position "g.42" (coordinate `41`) corresponds with position "p.2". We can convert between these to as follows.

```
>>> crossmap.coordinate_to_protein(41)
(2, 2, 0, 0, 0)
>>> crossmap.protein_to_coordinate((2, 2, 0, 0))
41
```

Note that the protein position only corresponds with the HGVS "p." notation when the offset equals `0` and the region equals `1`. In the following table, we show a number of annotated examples.

Table 5: Protein positions examples.

coordinate	protein position	description	HGVS position
`4`	`(-4, 2, -1, -1, -1)`	Upstream position.	invalid
`31`	`(-1, 3, 0, -1, 0)`	5' UTR position.	invalid
`36`	`(1, 3, 2, 0, 0)`	Intronic position.	invalid
`40`	`(2, 1, 0, 0, 0)`	Second amino acid, first nucleotide.	"p.2"
`41`	`(2, 2, 0, 0, 0)`	Second amino acid, second nucleotide.	"p.2"
`43`	`(1, 1, 0, 1, 0)`	3' UTR position.	invalid
`43`	`(2, 2, 2, 1, 2)`	Downstream position.	invalid

See section *Crossmapper* for a detailed description.

1.3.4 Locations

In many cases we need to know the nearest location with respect to a coordinate. For example, we need to know where the nearest exon is when we want to describe a position in an intron. The `nearest_location()` can be used to do exactly this.

```
>>> from mutalyzer_crossmapper import nearest_location
>>> nearest_location(exons, 37)
2
>>> nearest_location(exons, 38)
3
```

Notice that coordinate `37` is in the center of intron 2. By default `nearest_location()` will return the left location in case of a draw. This behaviour can be altered by setting the optional argument `p` to `1`.

```
>>> nearest_location(exons, 37, 1)
3
```

See section *Location* for a detailed description.

1.3.5 Basic classes

The `Coding` class makes use of a number of basic classes described in this section.

The `Locus` class

The `Locus` class is used to deal with offsets with respect to a single locus.

```
>>> from mutalyzer_crossmapper import Locus
>>> locus = Locus((10, 20))
```

This class provides the functions `to_position()` and `to_coordinate()` for converting from a locus position to a coordinate and vice versa. These functions work with a 2-tuple, see the section about *The NonCoding class* for the semantics.

```
>>> locus.to_position(9)
(1, -1)
```

For loci that reside on the reverse complement strand, the optional `inverted` constructor parameter should be set to `True`.

See section *Locus* for a detailed description.

The `MultiLocus` class

The `MultiLocus` class is used to deal with offsets with respect to multiple loci.

```
>>> from mutalyzer_crossmapper import MultiLocus
>>> multilocus = MultiLocus([(10, 20), (40, 50)])
```

The interface to this class is similar to that of the `Locus` class.

```
>>> multilocus.to_position(22)
(10, 3)
>>> multilocus.to_position(38)
(11, -2)
```

See section *MultiLocus* for a detailed description.

1.4 API documentation

1.4.1 Crossmapper

class mutalyzer_crossmapper.crossmapper.**Genomic**

Genomic crossmap object.

coordinate_to_genomic(*coordinate*)

Convert a coordinate to a genomic position (g./m./o.).

Parameters **coordinate** (*int*) – Coordinate.

Returns **int** Genomic position.

genomic_to_coordinate(*position*)
Convert a genomic position (g./m./o.) to a coordinate.

Parameters **position** (*int*) – Genomic position.

Returns int Coordinate.

class mutalyzer_crossmapper.crossmapper.**NonCoding**(*locations*, *inverted=False*)
NonCoding crossmap object.

Parameters

- **locations** (*list*) – List of locus locations.
- **inverted** (*bool*) – Orientation.

coordinate_to_genomic(*coordinate*)
Convert a coordinate to a genomic position (g./m./o.).

Parameters **coordinate** (*int*) – Coordinate.

Returns int Genomic position.

coordinate_to_noncoding(*coordinate*)
Convert a coordinate to a noncoding position (n./r.).

Parameters **coordinate** (*int*) – Coordinate.

Returns tuple Noncoding position.

genomic_to_coordinate(*position*)
Convert a genomic position (g./m./o.) to a coordinate.

Parameters **position** (*int*) – Genomic position.

Returns int Coordinate.

noncoding_to_coordinate(*position*)
Convert a noncoding position (n./r.) to a coordinate.

Parameters **position** (*tuple*) – Noncoding position.

Returns int Coordinate.

class mutalyzer_crossmapper.crossmapper.**Coding**(*locations*, *cds*, *inverted=False*)
Coding crossmap object.

Parameters

- **locations** (*list*) – List of locus locations.
- **cds** (*tuple*) – Locus location.
- **inverted** (*bool*) – Orientation.

coding_to_coordinate(*position*)
Convert a coding position (c./r.) to a coordinate.

Parameters **position** (*tuple*) – Coding position (c./r.).

Returns int Coordinate.

coordinate_to_coding(*coordinate*, *degenerate=False*)
Convert a coordinate to a coding position (c./r.).

Parameters

- **coordinate** (*int*) – Coordinate.

- **degenerate** (*bool*) – Return a degenerate position.

Returns tuple Coding position (c./r.).

coordinate_to_genomic(*coordinate*)
Convert a coordinate to a genomic position (g./m./o.).

Parameters **coordinate** (*int*) – Coordinate.

Returns int Genomic position.

coordinate_to_noncoding(*coordinate*)
Convert a coordinate to a noncoding position (n./r.).

Parameters **coordinate** (*int*) – Coordinate.

Returns tuple Noncoding position.

coordinate_to_protein(*coordinate*)
Convert a coordinate to a protein position (p.).

Parameters **coordinate** (*int*) – Coordinate.

Returns tuple Protein position (p.).

genomic_to_coordinate(*position*)
Convert a genomic position (g./m./o.) to a coordinate.

Parameters **position** (*int*) – Genomic position.

Returns int Coordinate.

noncoding_to_coordinate(*position*)
Convert a noncoding position (n./r.) to a coordinate.

Parameters **position** (*tuple*) – Noncoding position.

Returns int Coordinate.

protein_to_coordinate(*position*)
Convert a protein position (p.) to a coordinate.

Parameters **position** (*tuple*) – Protein position (p.).

Returns int Coordinate.

1.4.2 Location

mutalyzer_crossmapper.location.**nearest_location**(*ls*, *c*, *p=0*)
Find the location nearest to *c*. In case of a draw, the parameter *p* decides which index is chosen.

Parameters

- **ls** (*list*) – List of locations.
- **c** (*int*) – Coordinate.
- **p** (*int*) – Preference in case of a draw: 0: left, 1: right.

Returns int Nearest location.

1.4.3 Locus

class mutalyzer_crossmapper.locus.**Locus**(*location*, *inverted=False*)
Locus object.

Parameters

- **location** (*tuple*) – Locus location.
- **inverted** (*bool*) – Orientation.

to_coordinate(*position*)
Convert a position to a coordinate.

Parameters **position** (*int*) – Position.

Returns int Coordinate.

to_position(*coordinate*)
Convert a coordinate to a proper position.

Parameters **coordinate** (*int*) – Coordinate.

Returns tuple Position.

1.4.4 MultiLocus

class mutalyzer_crossmapper.multi_locus.**MultiLocus**(*locations*, *inverted=False*)
MultiLocus object.

Parameters

- **locations** (*list*) – List of locus locations.
- **inverted** (*bool*) – Orientation.

outside(*coordinate*)
Calculate the offset relative to this MultiLocus.

Parameters **coordinate** (*int*) – Coordinate.

Returns int Negative: upstream, 0: inside, positive: downstream.

to_coordinate(*position*)
Convert a position to a coordinate.

Parameters **position** (*int*) – Position.

Returns int Coordinate.

to_position(*coordinate*)
Convert a coordinate to a position.

Parameters **coordinate** (*int*) – Coordinate.

Returns tuple Position.

1.5 Contributors

Main developers:

- Jeroen F.J. Laros <J.F.J.Laros@lumc.nl> (Original author, maintainer)
- Martijn Vermaat <martijn@vermaat.name> (Restructuring, bug fixes, maintenance)

Other contributions by:

- Jonathan K. Vis <J.K.Vis@lumc.nl> (Packaging issues)
- Gerard C.P. Schaafsma (Bug fix)
- Gerben R. Stouten (Layout)

Find out who contributed:

```
git shortlog -s -e
```

PYTHON MODULE INDEX

m

INDEX