monoseq Documentation

Release 1.2.1

Martijn Vermaat <martijn@vermaat.name>

July 16, 2015

Contents

`monoseq` is a Python library for pretty-printing DNA and protein sequences using a monospace font. It also provides a simple command line interface.

Sequences are pretty-printed in the traditional way using blocks of letters where each line is prefixed with the sequence position. User-specified regions are highlighted and the output format can be HTML or plaintext with optional styling using ANSI escape codes for use in a terminal.

A simple example:

```
>>> from monoseq import pprint_sequence
>>> sequence = 'MIMANQPLWLDSEVEMNHYQQSHIKSKSPYFPEDKHICWIKIFKAFGT' * 4
>>> print pprint_sequence(sequence)
  1  MIMANQPLWL DSEVEMNHYQ QSHIKSKSPY FPEDKHICWI KIFKAFGTMI MANQPLWLDS
 61  EVEMNHYQQS HIKSKSPYFP EDKHICWIKI FKAFGTMIMA NQPLWLDSEV EMNHYQQSHI
121  KSKSPYFPED KHICWIKIFK AFGTMIMANQ PLWLDSEVEM NHYQQSHIKS KSPYFPEDKH
181  ICWIKIFKAF GT
```

An example, admittedly contrived, with annotations:

This IPython Notebook shows how to pretty-print sequences in an IPython Notebook.

CHAPTER 1

User documentation

New users should probably start here.

1.1 Installation

The `monoseq` source code is hosted on GitHub. Supported Python versions for running `monoseq` are 2.7, 3.2, 3.3 and PyPy (unit tests are run automatically on these platforms using the Travis CI service). `monoseq` can be installed either via the Python Package Index (PyPI) or from the source code.

1.1.1 Latest release via PyPI

To install the latest release via PyPI using pip:

```
pip install monoseq
```

1.1.2 Development version

You can also clone and use the latest development version directly from the GitHub repository:

```
git clone https://github.com/martijnvermaat/monoseq.git
cd monoseq
python setup.py install
```

1.2 User guide

So let's say we have this DNA sequence we want to pretty-print:

```
>>> sequence = ('cgcactcaaaacaaaggaagaccgtcctcgactgcagaggaagcaggaagctgtc'
...             'ggcccagctctgagcccagctgctggagccccgagcagcggcatggagtccgtgg'
...             'ccctgtacagctttcaggctacagagagcgacgagctggccttcaacaagggaga'
...             'cacactcaagatcctgaacatggaggatgaccagaactggtacaaggccgagctc'
...             'cggggtgtcgagggatttattcccaagaactacatccgcgtcaag')
```

We can do it with *`pprint_sequence()`*:

```
>>> from monoseq import pprint_sequence
>>> print pprint_sequence(sequence)
  1  cgcactcaaa acaaaggaag accgtcctcg actgcagagg aagcaggaag ctgtcggccc
 61  agctctgagc ccagctgctg gagccccgag cagcggcatg gagtccgtgg ccctgtacag
121  ctttcaggct acagagagcg acgagctggc cttcaacaag ggagacacac tcaagatcct
181  gaacatggag gatgaccaga actggtacaa ggccgagctc cggggtgtcg agggatttat
241  tcccaagaac tacatccgcg tcaag
```

(This also works if *sequence* is a Biopython `Bio.Seq.Seq` object.)

1.2.1 Controlling block and line lengths

By default, sequences are printed in blocks of 10 letters, 6 blocks per line. This can be customized with the *block_length* and *blocks_per_line* arguments:

```
>>> print pprint_sequence(sequence, block_length=8, blocks_per_line=7)
  1  cgcactca aaacaaag gaagaccg tcctcgac tgcagagg aagcagga agctgtcg
 57  gcccagct ctgagccc agctgctg gagccccg agcagcgg catggagt ccgtggcc
113  ctgtacag ctttcagg ctacagag agcgacga gctggcct tcaacaag ggagacac
169  actcaaga tcctgaac atggagga tgaccaga actggtac aaggccga gctccggg
225  gtgtcgag ggatttat tcccaaga actacatc cgcgtcaa g
```

```
>>> print pprint_sequence(sequence, block_length=20, blocks_per_line=2)
  1  cgcactcaaaacaaaggaag accgtcctcgactgcagagg
 41  aagcaggaagctgtcggccc agctctgagcccagctgctg
 81  gagccccgagcagcggcatg gagtccgtggccctgtacag
121  ctttcaggctacagagagcg acgagctggccttcaacaag
161  ggagacacactcaagatcct gaacatggaggatgaccaga
201  actggtacaaggccgagctc cggggtgtcgagggatttat
241  tcccaagaactacatccgcg tcaag
```

1.2.2 Output formats

As we'll see in the next section, certain parts of the sequence can be annotated for highlighting. For this to work, we need to specify another output format than the default `PlaintextFormat`.

`monoseq` includes three built-in output formats:

- `PlaintextFormat` for, well, generating plaintext.
- `AnsiFormat` adds ANSI escape codes for use in a terminal.
- `HtmlFormat` adds HTML tags for inclusion in an HTML document.

(And if this doesn't satisfy our needs, we can define custom output formats by implementing `Format`.)

Formats are specified with the *format* argument of the `pprint_sequence()` function. For example, this is the same example pretty-printed with `HtmlFormat`:

```
>>> from monoseq import HtmlFormat
>>> print pprint_sequence(sequence, blocks_per_line=3, format=HtmlFormat)
  <span class="monoseq-margin">1</span>  cgcactcaaa acaaaggaag accgtcctcg
 <span class="monoseq-margin">31</span>  actgcagagg aagcaggaag ctgtcggccc
 <span class="monoseq-margin">61</span>  agctctgagc ccagctgctg gagccccgag
 <span class="monoseq-margin">91</span>  cagcggcatg gagtccgtgg ccctgtacag
<span class="monoseq-margin">121</span>  ctttcaggct acagagagcg acgagctggc
<span class="monoseq-margin">151</span>  cttcaacaag ggagacacac tcaagatcct
<span class="monoseq-margin">181</span>  gaacatggag gatgaccaga actggtacaa
```

```
<span class="monoseq-margin">211</span>  ggccgagctc cggggtgtcg agggatttat
<span class="monoseq-margin">241</span>  tcccaagaac tacatccgcg tcaag
```

As you can see, `HtmlFormat` wraps the sequence positions in `<span>` tags with a `class` attribute value of `monoseq-margin`. This allows us to add custom styling to these numbers with a CSS stylesheet.

Note: In an HTML document, include the pretty-printed sequence within `<pre>` and `</pre>`. This preserves all whitespace and automatically selects a monospace font.

1.2.3 Sequence annotations

Subsequences can be highlighted in the pretty-printed sequence by specifying their positions. Such a specification is called an annotation. Several annotations can be provided and each of them will be highlighted in a distinct style (e.g., the first annotation is colored red and the second is printed in bold).

Let's assume our analysis shows positions 12 through 37 and 223 through 247 to be highly conserved between species. Of course, we want to annotate our sequence with this knowledge:

Note: Regions are defined as in slicing notation, so zero-based and open-ended.

Just for lack of imagination, we also want to make it clear where every 12th nucleotide is in our sequence. We can do this by defining a second annotation, which is printed in bold by `AnsiFormat`:

`AnsiFormat` supports up to three annotation levels and the third one is printed underlined. So if the middle third of the sequence would be our primary concern, we could underline it as follows:

1.2.4 Styling `HtmlFormat` output

The `HtmlFormat` output format supports up to 10 annotation levels, but how to style them is up to the user. All `monoseq` does is add `<span>` tags around annotations with `class` attribute values of `monoseq-annotation-{i}`, where `{i}` is the annotation level starting from 0.

Here are some example CSS rules for styling 4 annotation levels:

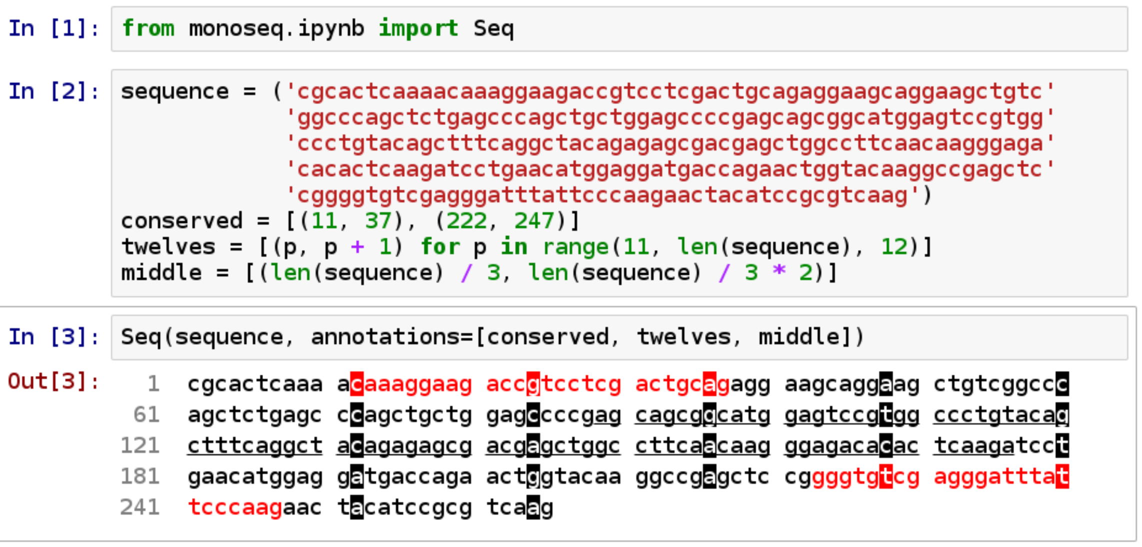
```css
pre {
    background: lightYellow;
    color: black;
}
.monoseq-margin {
    color: grey;
}
.monoseq-annotation-0 {
    color: red;
}
.monoseq-annotation-1 {
    background: black;
    color: lightYellow;
}
.monoseq-annotation-1 .monoseq-annotation-0 {
    background: red;
    color: lightYellow;
}
.monoseq-annotation-2 {
    text-decoration: underline;
}
```

```
.monoseq-annotation-3 {
    font-weight: bold;
}
```

Using these rules, a pretty-printed protein sequence will look something like this:

1.2.5 Using `monoseq` from the IPython Notebook

For pretty-printing sequences directly in the IPython Notebook, `monoseq.ipynb.Seq` is provided as a convenience wrapper around *`pprint_sequence()`*:



```
In [1]: from monoseq.ipynb import Seq

In [2]: sequence = ('cgcactcaaaacaaaggaagaccgtcctcgactgcagaggaagcaggaagctgtc'
                    'ggcccagctctgagcccagctgctggagccccgagcagcggcatggagtccgtgg'
                    'ccctgtacagctttcaggctacagagagcgacgagctggccttcaacaagggaga'
                    'cacactcaagatcctgaacatggaggatgaccagaactggtacaaggccgagctc'
                    'cggggtgtcgagggatttattcccaagaactacatccgcgtcaag')
        conserved = [(11, 37), (222, 247)]
        twelves = [(p, p + 1) for p in range(11, len(sequence), 12)]
        middle = [(len(sequence) / 3, len(sequence) / 3 * 2)]

In [3]: Seq(sequence, annotations=[conserved, twelves, middle])

Out[3]:   1  cgcactcaaa acaaaggaag accgtcctcg actgcagagg aagcaggaag ctgtcggccc
         61  agctctgagc ccagctgctg gagccccgag cagcggcatg gagtccgtgg ccctgtacag
        121  ctttcaggct acagagagcg acgagctggc cttcaacaag ggagacacac tcaagatcct
        181  gaacatggag gatgaccaga actggtacaa ggccgagctc cggggtgtcg agggatttat
        241  tcccaagaac tacatccgcg tcaag
```

This supports up to four levels of annotation, displayed as red, inverted, underlined, and bold.

See this IPython Notebook for some examples.

1.3 Command line interface

Simple pretty-printing of sequences on the command line is done with the `monoseq` command. It reads one or more sequences from a file or from standard input and pretty-prints them to standard output. The input can be a raw sequence, or any number of sequences in FASTA format.

Example:

```
martijn@hue:~$ S=MIMANQPLWLDSEVEMNHYQQSHIKSKSPYFPEDKHICWIKIFKAFGT
martijn@hue:~$ echo $S$S$S$S | monoseq
  1  MIMANQPLWL DSEVEMNHYQ QSHIKSKSPY FPEDKHICWI KIFKAFGTMI MANQPLWLDS
 61  EVEMNHYQQS HIKSKSPYFP EDKHICWIKI FKAFGTMIMA NQPLWLDSEV EMNHYQQSHI
121  KSKSPYFPED KHICWIKIFK AFGTMIMANQ PLWLDSEVEM NHYQQSHIKS KSPYFPEDKH
181  ICWIKIFKAF GT
```

1.3.1 Formatting options

The number of letters per block can be specified with the `-b` argument and the `-l` argument sets the number of blocks per line.

1.3.2 Annotations

Subsequences can be specified for annotation with the `-a` argument followed by the first and the last position of the subsequence, both one-based. For example, to annotate the first 10 bases and the 17th base, you would add `-a 1 10 -a 17 17`.

In addition, annotation is read from the BED track specified with the `-e` argument. If the input is a raw sequence, only the first chromosome is used from the BED track. If the input is a FASTA file, chromosomes are matched with record names.

1.3.3 More information

Use the `--help` argument for more information.

CHAPTER 2

API reference

Documentation on a specific function, class or method can be found in the API reference.

2.1 API reference

2.1.1 `monoseq`

`monoseq`, a Python library for pretty-printing DNA and protein sequences using a monospace font.

`monoseq.`**`PlaintextFormat`**
Plaintext output format.

`monoseq.`**`AnsiFormat`**
Plaintext output format with ANSI escape codes.

`monoseq.`**`HtmlFormat`**
HTML output format.

class `monoseq.`**`Format`**
Type of output formats for pretty-printed sequences.

Parameters

- **`annotations`** (*list*) – For each annotation level, a pair (*left*, *right*) of delimiters to use for enclosing a subsequence at that level.
- **`margin`** (*tuple*) – A pair (*left*, *right*) of delimiters to use for enclosing the margin (containing sequence positions).

The *annotations* field can have any number of items, any subsequent annotation levels will be ignored in pretty-printing sequences.

`monoseq.`**`partition_range`**`(`*stop*, *annotations=None*`)`
Partition the range from 0 to *stop* based on annotations.

```
>>> partition_range(50, annotations=[[(0, 21), (30, 35)],
...                                  [(15, 32), (40, 46)]])
[(0, 15, {0}),
 (15, 21, {0, 1}),
 (21, 30, {1}),
 (30, 32, {0, 1}),
 (32, 35, {0}),
 (35, 40, set()),
 (40, 46, {1}),
 (46, 50, set())]
```

Parameters

- **stop** (*int*) – End point (not included) of the range (similar to the *stop* argument of the built-in `range()` function).
- **annotations** (*list*) – For each annotation level, a list of (*start*, *stop*) pairs defining an annotated region.

Returns Partitioning of the range as (*start*, *stop*, *levels*) tuples defining a region with a set of annotation levels.

Return type list

All regions (*start*, *stop*) are defined as in slicing notation, so zero-based and *stop* is not included.

The *annotations* argument is a list of annotations. An annotation is a list of regions as (*start*, *stop*) tuples. The level of each annotation is its index in *annotations*.

Annotation regions can overlap (overlap within one level is ignored) and do not need to be sorted.

`monoseq.`**`pprint_sequence`**(*sequence*, *annotations=None*, *block_length=10*, *blocks_per_line=6*, *format=Format(annotations=[]*, *margin=('', ''))*)

Pretty-print sequence for use with a monospace font.

```
>>> sequence = 'MIMANQPLWLDSEVEMNHYQQSHIKSKSPYFPEDKHICWIKIFKAFGT' * 4
>>> print pprint_sequence(sequence, format=PlaintextFormat)
  1  MIMANQPLWL DSEVEMNHYQ QSHIKSKSPY FPEDKHICWI KIFKAFGTMI MANQPLWLDS
 61  EVEMNHYQQS HIKSKSPYFP EDKHICWIKI FKAFGTMIMA NQPLWLDSEV EMNHYQQSHI
121  KSKSPYFPED KHICWIKIFK AFGTMIMANQ PLWLDSEVEM NHYQQSHIKS KSPYFPEDKH
181  ICWIKIFKAF GT
```

Parameters

- **sequence** (*str or any sliceable yielding slices representable as strings.*) – Sequence to pretty-print.
- **annotations** (*list*) – For each annotation level, a list of (*start*, *stop*) pairs defining an annotated region.
- **block_length** (*int*) – Length of space-separated blocks.
- **blocks_per_line** (*int*) – Number of blocks per line.
- **format** (`Format`) – Output format to use for pretty-printing. Some formats are pre-defined as `HtmlFormat`, `AnsiFormat`, and `PlaintextFormat`.

Returns Pretty-printed version of *sequence*.

Return type str

All regions (*start*, *stop*) are defined as in slicing notation, so zero-based and *stop* is not included.

The *annotations* argument is a list of annotations. An annotation is a list of regions as (*start*, *stop*) tuples. The level of each annotation is its index in *annotations*.

Annotation regions can overlap (overlap within one level is ignored) and do not need to be sorted.

The number of annotation levels supported depends on *format*. `HtmlFormat` supports 10 levels, `AnsiFormat` supports 3 levels and annotations are ignored completely with `PlaintextFormat`.

2.1.2 `monoseq.ipynb`

CHAPTER 3

Additional notes

3.1 Development

Development of `monoseq` happens on GitHub: https://github.com/martijnvermaat/monoseq

3.1.1 Contributing

Contributions to `monoseq` are very welcome! They can be feature requests, bug reports, bug fixes, unit tests, documentation updates, or anything els you may come up with.

3.1.2 Coding style

In general, try to follow the PEP 8 guidelines for Python code and PEP 257 for docstrings.

3.1.3 Unit tests

To run the unit tests with nose, just run `nosetests -v`.

3.1.4 Versioning

A normal version number takes the form X.Y.Z where X is the major version, Y is the minor version, and Z is the patch version. Development versions take the form X.Y.Z.dev where X.Y.Z is the closest future release version.

Note that this scheme is not 100% compatible with SemVer which would require X.Y.Z-dev instead of X.Y.Z.dev but compatibility with setuptools is more important for us. Other than that, version semantics are as described by SemVer.

Releases are published at PyPI and available from the GitHub git repository as tags.

Release procedure

Releasing a new version is done as follows:

1. Make sure the section in the `CHANGES` file for this release is complete and there are no uncommitted changes.

   Note: Commits since release X.Y.Z can be listed with `git log vX.Y.Z..` for quick inspection.

2. Update the `CHANGES` file to state the current date for this release and edit `monoseq/__init__.py` by updating *__date__* and removing the `dev` value from *__version_info__*.

   Commit and tag the version update:

```
git commit -am 'Bump version to X.Y.Z'
git tag -a 'vX.Y.Z'
git push --tags
```

3. Upload the package to PyPI:

```
python setup.py sdist upload
```

4. Add a new entry at the top of the `CHANGES` file like this:

```
Version X.Y.Z+1
---------------

Release date to be decided.
```

   Increment the patch version and add a `dev` value to *__version_info__* in `monoseq/__init__.py` and commit these changes:

```
git commit -am 'Open development for X.Y.Z+1'
```

3.1.5 Todo

These are some general todo notes. More specific notes can be found by grepping the source code for `Todo`.

- Nothing to be done at the moment?

3.2 Changelog

Here you can see the full list of changes between each `monoseq` release.

3.2.1 Version 1.2.1

Released on May 10th 2015.

- Custom styling for *monoseq.ipynb.Seq*.

3.2.2 Version 1.2.0

Released on May 10th 2015.

- *monoseq.ipynb.Seq* for use in the IPython Notebook.

3.2.3 Version 1.1.1

Released on July 18th 2013.

- Actually support 10 annotation levels in *HtmlFormat* as documented.
- User guide and documentation for command line interface.

3.2.4 Version 1.1.0

Released on July 14th 2013.

- Read annotation from BED track.
- Better class names in *HtmlFormat* (`monoseq-annotation-*` and `monoseq-margin`).

3.2.5 Version 1.0.0

Released on July 13th 2013.

- User-definable formatting stylings. This breaks the 0.1.0 API for *monoseq.pprint_sequence* (the *mode* argument is superseded by *format*).
- Read sequences from a FASTA file.
- Command line interface.

3.2.6 Version 0.1.0

Released on July 12th 2013.

First public release.

3.3 Copyright

`monoseq` is licensed under the MIT License, meaning you can do whatever you want with it as long as all copies include these license terms. The full license text can be found below.

3.3.1 Authors

`monoseq` is written and maintained by Martijn Vermaat.

- Martijn Vermaat <martijn@vermaat.name>

3.3.2 License

Copyright (c) 2013 by Martijn Vermaat and contributors (see AUTHORS.rst for details).

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

Python Module Index