fSeq Documentation

Release 1.0.0a

Martin Zackrisson

June 24, 2014

fSeq is an extensible toolkit aimed at fast reading of sequence data with a common interface for all types of sequences. Upon that, the ability to encode the contents of the data into suitable numeric formats prepares for the second part.

After reading contents *fSeq* allows for post-processing and report-making as integral parts of its framework. *fSeq* comes with a small set of report builders and report makers, but these have equally been designed to allow fast development of further capabilities.

CHAPTER

ONE

CONTENTS

1.1 fseq

1.1.1 fseq package

Subpackages

fseq.reading package

Submodules

fseq.reading.seq_encoder module

class fseq.reading.seq_encoder.**FastQ**

Bases: fseq.reading.seq_encoder.SeqFormat

Detector of FASTQ format

Note: This class can be subclassed to detect the different quality-encoding schemes in the future.

See also:

SeqFormat Base class for detectors

Examples

Valid format:

```
@Contig_1
AACAATACGA
+Contig_1
@+>AACADGH
@Contig_2
CCATTTACGA
+
>CCAGJDFGH
```

Attributes

HEADER_LINE	
SEUENCE_LINE	
QUALITY_LINE	

HEADER_LINE = 0

QUALITY_LINE = 3

SEQUENCE_LINE = 1

expects(*line*)

Test for if line fits into required pattern for the format

Parameters **line: str**

A line as read from file

Returns bool

Raises **FormatImplementationError**

If `SeqFormat.expects` (the base class method) is called.

hasQuality

If format quality information: bool

hasSequence

If format has sequence information: bool

itemSize

The size (lines) of each entry: int

name

Human readable description of format: str

class `fseq.reading.seq_encoder.`**FastaMultiline**

Bases: `fseq.reading.seq_encoder.SeqFormat`

Detector of multi-line FASTA

Note: This format is not supported in the current implementation as it has no predictable item-size.

See also:

SeqFormat Base class for detectors

FastaSingleline Derived class, with more restricitons

Examples

Valid format:

```
>Contig_1
ACAATACA
GATTACA
>Contig_2
ACCCACA
>Contig_3
ACCAAACA
CCAACACA
ACAACCAC
```

Attributes

HEADER_LINE	
SEUENCE_LINE	
QUALITY_LINE	

HEADER_LINE = 0

QUALITY_LINE = None

SEQUENCE_LINE = -1

expects(*line*)
Test for if line fits into required pattern for the format

Parameters **line: str**

A line as read from file

Returns bool

Raises **FormatImplementationError**

If `SeqFormat.expects` (the base class method) is called.

hasQuality
If format quality information: bool

hasSequence
If format has sequence information: bool

name
Human readable description of format: str

class `fseq.reading.seq_encoder.`**FastaSingleline**
Bases: `fseq.reading.seq_encoder.FastaMultiline`

Detector of single-line FASTA format.

See also:

SeqFormat Base class for detectors

FastaMultiline Parent class

Examples

Valid format:

```
>Contig_1
AACAATACGA
>Contig_2
CCATTTACGA
```

Attributes

HEADER_LINE	
SEUENCE_LINE	
QUALITY_LINE	

HEADER_LINE = 0

QUALITY_LINE = None

SEQUENCE_LINE = 1

expects(*line*)
Test for if line fits into required pattern for the format

Parameters **line: str**

A line as read from file

Returns bool

Raises **FormatImplementationError**

If `SeqFormat.expects` (the base class method) is called.

itemSize
The size (lines) of each entry: int

name
Human readable description of format: str

exception `fseq.reading.seq_encoder.`**FormatError**
Bases: `exceptions.Exception`

Sequence Format Error for exceptions relating to missmatches between encoders and sequence formats as well as lacking formattings in encoders and unknown sequence formats.

exception `fseq.reading.seq_encoder.`**FormatImplementationError**
Bases: `fseq.reading.seq_encoder.FormatError`

Error for exposing parts of interface that needs to be overwritten in subclasses

exception `fseq.reading.seq_encoder.`**FormatUnknown**
Bases: `fseq.reading.seq_encoder.FormatError`

Error for having no available detectors left

class `fseq.reading.seq_encoder.`**SeqEncoder**(*expectedInputFormat=None*, *useSequence=True*, *useQuality=False*, *sequenceEncoding=None*, *qualityEncoding=None*, *requestReports=()*)
Bases: `object`

Base class for managing encoding of raw input data.

The encoder coordinates detection of raw data format and then extracts the information that the encoder is tasked to extract, convering it into suitable encoding.

Note: The base class is not intended to be used directly, but to be extended.

See also:

SeqEncoderGC GC encoder

Attributes

`format`	The input format currently expected.
`initiated`	If the sequence encoder is ready to start parsing: bool
`itemSize`	The size in number of lines for each item in the source.

Continued on next page

Table 1.1 – continued from previous page

`qualityEncoding`	Map for translating quality chars to numeric values.
`sequenceEncoding`	Map for translating sequence chars to numeric values.
`useQuality`	If quality-line is to be used by the encoder.
`useSequence`	If sequence-line is to be used by the encoder

detectFormat()
Detect format based on current input stream

Returns fseq.SeqEncoder

Returns `self`

Raises **FormatError**

If data is not compatible with information requested

See also:

SeqEncoder.detectFormat Guessing

feedDetection(*line*)
Give detection a line to work with.

Parameters **line: str**

A line from the source file

Returns fseq.SeqEncoder

Returns `self`

format
The input format currently expected.

Returns SeqFormatDetector

Raises **TypeError**

If attempting to set format with object not derived from `SeqFormatDetector` or `SeqFormat`

initiated
If the sequence encoder is ready to start parsing: bool

itemSize
The size in number of lines for each item in the source.

Returns int

Raises **fseq.FormatError**

If encoder is not fully initiated

See also:

SeqEncoder.initiated The initiation status

SeqEncoder.detectFormat Detecting the format

SeqEncoder.format Manually setting the format

parse(*lines*, *out*, *outindex*)
Placeholder parser overwritten when subclassing

Parameters **lines: iterable of str**

Iterable of length equal to `self.itemSize` containing the raw data for one item

out: numpy.ndarray

Array that will have values written to it

outIndex: object

Index for where the parse output should be written in the `out` array such that `out[outIndex]` gives a sufficiently large array that the result of parsing will fit in it.

Raises **NotImplemented**

If base class parse not overwritten or base class used directly

qualityEncoding

Map for translating quality chars to numeric values.

Returns Object with key-lookup (implementing `__getitem__`)

Raises **TypeError**

If attempting to set with object not having char key-lookup

requestReports

The reports that the encoder likes to be produced by the reader

reset()

Clears the sequence format

Returns fseq.SeqEncoder

Returns `self`

See also:

SeqEncoder.detectFormat Guessing

sequenceEncoding

Map for translating sequence chars to numeric values.

Returns Object with key-lookup (implementing `__getitem__`)

Raises **TypeError**

If attempting to set with object not having char key-lookup

useQuality

If quality-line is to be used by the encoder.

Returns bool

useSequence

If sequence-line is to be used by the encoder

Returns bool

class `fseq.reading.seq_encoder.`**SeqEncoderGC**(*expectedInputFormat=None*, *sequenceEncoding=None*)

Bases: `fseq.reading.seq_encoder.SeqEncoder`

GC Encoder, but useful for any sequence to numerical value encoding.

The encoder uses the sequence information of the raw data and encodes the data according to the following:

Input	Encoding
~~G / C~~	~~1.0~~
	A / T 0.0
N	0.5

To change this behaviour, simply submit a new mappable object such as e.g. a `dict` as the `sequenceEncoding`-parameter.

`parse`(*lines*, *out*, *outindex*)

Encoder of suitable aspects of `lines` into `out`.

The sequence line of `lines` will be encoded onto the index `outindex` of `out`. If the sequence line is shorter than the data structure of `out[outindex]`, the remainder of `out[outindex]` will be left untouched. If the line is longer, it will only encode up until the the length of `out[outindex]`.

Parameters **lines: iterable of str**

Iterable of length equal to `self.itemSize` containing the raw data for one item

out: numpy.ndarray

Array that will have values written to it

outIndex: object

Index for where the parse output should be written in the `out` array such that `out[outIndex]` gives a sufficiently large array that the result of parsing will fit in it.

class `fseq.reading.seq_encoder.`**`SeqFormat`**

Bases: `object`

Base Class for implementing data format detectors.

The attributes present in subclass should overwrite the parent properties. All subclasses must also overwrite the `SeqFormat.expects(line)`.

Attributes

`name`	Human readable description of format: str
`itemSize`	The size (lines) of each entry: int
`hasSequence`	If format has sequence information: bool
`hasQuality`	If format quality information: bool
`qualityEncoding`	If format comes with a known encoding of quality.

MATCH_AA	
MATCH_AA_S	
MATCH_NT	
MATCH_NT_S	
HEADER_LINE	
SEUENCE_LINE	
QUALITY_LINE	

`HEADER_LINE` **= 0**

`MATCH_AA` **= <_sre.SRE_Pattern object at 0x2b5ec97644e0>**

Matches any complete line of A-Z characters allowing for asterisc at end.

MATCH_AA_S = <_sre.SRE_Pattern object at 0x2b5ec7f32ed0>
Matches as `MATCH_AA_S` but extends to include space

MATCH_NT = <_sre.SRE_Pattern object at 0x2b5ec9764418>
Matches any complete line of only A T C G or N

MATCH_NT_S = <_sre.SRE_Pattern object at 0x2b5ec97645a8>
Matches as `MATCH_NT` but extends to include space

QUALITY_LINE = None

SEQUENCE_LINE = None

expects(*line*)
Test for if line fits into required pattern for the format

Parameters **line: str**

A line as read from file

Returns bool

Raises **FormatImplementationError**

If `SeqFormat.expects` (the base class method) is called.

givenUp
Reports if format has given up even though everything still was matching.

The purpose is to be able to promote more restricted formats that represents a subset of the more general

Returns bool

The status

hasQuality
If format quality information: bool

hasSequence
If format has sequence information: bool

itemSize
The size (lines) of each entry: int

name
Human readable description of format: str

qualityEncoding
If format comes with a known encoding of quality.

See also:

SeqEncoder.qualityEncoding Setter of encoder quality.

class `fseq.reading.seq_encoder.`**SeqFormatDetector**(*forceFormat=None*)
Bases: `object`

Detection of data-format manager

Given a set of initially specified formats the detector feeds them lines of data until only one remains `True`. It then further continues a little while to be more certain that it was not a mere fluke.

See also:

SeqFormat Base class for formats that can be detected.

Attributes

detecting	If attempting to detect: bool
format	The format name of the detected format: str
itemSize	The size (lines) of each entry: int
hasSequence	If format has sequence information: bool
hasQuality	If format quality information: bool
qualityEncoding	If format comes with a known encoding of quality.
headerLine	The line index in the item for the header
sequenceLine	The line index in the item for the sequence
qualityLine	The line index in the item for the quality
headerLine	The line index in the item for the header
sequenceLine	The line index in the item for the sequence
qualityLine	The line index in the item for the quality

FORMATS	

FORMATS = [<class 'fseq.reading.seq_encoder.FastaSingleline'>, <class 'fseq.reading.seq_encoder.FastaMultiline'>, <class

compatible(*encoder*)

Evaluates if encoder is compatible with the detected format

Returns bool

Compatibility

Raises **FormatError**

If attempting to test compatibility before format is detected

detecting

If attempting to detect: bool

feed(*line*)

Supply a new line to format detector.

Parameters **line: str**

The next line in the data

Returns fseq.SeqEncoder

Returns `self`

Raises **FormatUnknown**

If no known formatters are left

format

The format name of the detected format: str

hasQuality

If format quality information: bool

hasSequence

If format has sequence information: bool

headerLine

The line index in the item for the header

Returns int

Raises **FormatError**

If attempting to use before format is detected

itemSize
The size (lines) of each entry: int

qualityEncoding
If format comes with a known encoding of quality.

See also:

`SeqEncoder.qualityEncoding` Setter of encoder quality.

qualityLine
The line index in the item for the quality

Returns int

Raises **FormatError**

If attempting to use before format is detected

sequenceLine
The line index in the item for the sequence

Returns int

Raises **FormatError**

If attempting to use before format is detected

`fseq.reading.seq_encoder.`**inheritDocFromSeqFormat**(*f*)
This decorator will copy the docstring from `SeqFormat` for the matching function `f` if such exists and no docstring has been added manually.

fseq.reading.seq_reader module Module for reading sequence data

class `fseq.reading.seq_reader.`**SeqReader**(*seqEncoder=None*, *dataSourcePaths=None*, *dataTargetPaths=None*, *reportBuilders=None*, *popDataSources=True*, *resetSeqEncoder=True*, *popEncodingResults=None*, *dataArrayConstructor=<built-in function zeros>*, *dataWidth=101*, *dataType=<type 'numpy.float16'>*, *verbose=False*)
Bases: `object`

Reads sequence data and encodes it.

The length of the sequence reader reflects the number of inputs to be processed.

The entire stack of inputs can be processed in bulk by invoking `SeqReader.run()` but the results of each encoding, omitting any report building can also be produced iteratively as shown in the examples.

Examples

To invoke the reader it can either be run :

```
>>> seqReader.run()
<fseq.reading.seq_reader.SeqReader at 0x7faf6970fd10>
```

or if more control is required, it can be iterated over:

```
>>> for res in seqReader:
...     reportBuilder.distill(res, dirname=seqReader.reportDirectory)
```

Both methods above yielding the same result with the difference that the first may store results in `seqReader.results` depending on the `seqReader.popEncodingResults` settings while the latter never keeps the results in the state of the instance.

Attributes

`popDataSources`	If sequence reader should remove data sources from list of sources when they have been read.
`popEncodingResults`	If the outcome of an encoding should be remove from memory as soon as report as been produced or ite
`jobQueue`	The data sources to be read and their respective targets.
`reportBuilders`	The report builders associated with the reader.
`reportDirectory`	The directory where reports for the last made encoding should go.
`resetSeqEncoder`	If each data source will detect format anew or if all files are
`results`	A list of the results of the encodings.

dataArrayConstructor	
dataWidth	
dataType	
SeqEncoder	

DATA_INITIAL_SIZE = 100000

DEBUG = False

WORKERS = 32

addData (*sourcePaths*, *targetPaths=None*)

Add a data source path to be analysed.

Parameters **sourcePaths: string or iterable object**

Either a path string or a collection of paths to data files

targetPaths: string or iterable object, optional

Either a relative path string or collection of relative paths. The path is relative to the respective data source.

(Default: will create a folder in the same directory as the data source with the same name as the input suffixed by `.reports`.)

Note: If supplied, must reflect equal number of outputs as inputs in `sourcePaths`

Returns fseq.SeqReader

Returns `self`

Raises **ValueError**

If target paths are supplied but don't match in lenght with the number of source-paths

addReportBuilders (**reportBuilders*)

Add a report builder to the set of reports done upon analysis.

Parameters **reportBuilders: fseq.ReportBuilder, optional**

Any number of report builder to be added

Returns fseq.SeqReader

Returns `self`

Raises **TypeError**

If an item in `reportBuilders` is not a valid `fseq.ReportBuilder`

clearJobQueue()
Removes all jobs in the queue.

Returns fseq.SeqReader

Returns `self`

clearResults()
Removes all stored results of encodings

Returns fseq.SeqReader

Returns `self`

dataArrayConstructor

dataType

dataWidth

jobQueue
The data sources to be read and their respective targets.

Returns list of tuples

Each tuple representing a source - target pair.

next()
Part of iter interface, produces the encoding of the next data-source.

Returns numpy.ndarray

Encoding output.

Raises **ValueError**

If no encoder has been assigned.

StopIteration

If no more data-source exists.

popDataSources
If sequence reader should remove data sources from list of sources when they have been read.

The general use case is to set this to `True`, but omitting popping can be useful if data may be needed to be re-read with a different encoder.

Returns bool

popEncodingResults
If the outcome of an encoding should be remove from memory as soon as report as been produced or iteration completed.

Note: The process will require large amounts of memory if results are not popped and several large files analysed.

Returns bool

removeReportBuilders(**builders*)
Removes all builders supplied, or all builders if no specific builder is supplied.

Parameters ***args: fseq.ReportBuilder, optional**

Any number of report builder references for report builders to be removed. If none is supplied, all report builders will be removed.

Returns fseq.SeqReader

Returns `self`

Examples

If current instance has three builders:

```
>>> tuple(seqEncoder.reportBuilders)
(b1, b2, b3)
```

The `b2` and `b3` can be removed by:

```
>>> seqEncoder.removeReportBuilders(b2, b3)
<fseq.reading.seq_reader.SeqReader at 0x7faf696e5810>

>>> tuple(seqEncoder.reportBuilders)
(b1, )
```

Alternatively all builders can be removed by:

```
>>> seqEncoder.removeReportBuilders()
<fseq.reading.seq_reader.SeqReader at 0x7faf696e5810>

>>> tuple(seqEncoder.reportBuilders)
()
```

reportBuilders
The report builders associated with the reader.

If any, the reports will be destilled automatically at the end of `SeqReader.run()`.

Returns tuple

The currently assigned report builders

See also:

fseq.reporting.report_builder.ReportBuilder.distill Method for distilling encoded data.

reportDirectory
The directory where reports for the last made encoding should go.

Returns str

resetSeqEncoder
If each data source will detect format anew or if all files are assumed to be of the same format

Returns bool

results
A list of the results of the encodings.

Returns list

See also:

SeqEncoder.clearResults Clearing the list of results

run()
Runs through all sources and produces reports if such have been attached.

Returns fseq.SeqReader

Returns `self`

seqEncoder
The encoder attached to the sequence reader that will parse the input strings into values.

Returns fseq.SeqEncoder

Current encoder

Raises TypeError

If trying to assign object that is not a `fseq.SeqEncoder`

verbose

Module contents Reading-related modules of fseq.

The reading package contains of two modules: *seq_encoder* and *seq_reader*.

The reader contains the generic reader that coordinates actions and works as the mainframe of *fseq*. This module should need no extensions to increase the funcitonality of the *fseq*.

The encoder-module contains both the different types of encoders available as well as the format-detectors for variaous types of input formats. Here further formats can be added and new encoders written to extend the functionality of *fseq*.

fseq.reporting package

Submodules

fseq.reporting.report_builder module The report builders are classes that coordinate report productions

class `fseq.reporting.report_builder.`**ReportBuilderBase**(**reports*, ***kwargs*)
Bases: `object`

Base class for common report builder features.

Most prominently, the distill-method needs to be implemented in subclasses to make much sence in most use cases.

Parameters outputRoot: str, optional

Path to the directory where all reports should be put

(Default: `None`)

outputNamePrefix: str, optional

Partial name to be added to all reports done by the builder

(Default: `None`)

***reports: objects, optional**

Any number of reports to be added from start

Attributes

`outputRoot`	The base for saving out reports: str
`outputNamePrefix`	Partial file name to prepend the individual reports: str

DEFAULT_REPORTS

DEFAULT_REPORTS = ()

addReports(**reports*)

Adds any number of reports given that the reports exposes a distill method.

Parameters ***reports: objects, optional**

Returns fseq.ReportBuilderBase

Returns `self`

Raises **ValueError**

If a report lacks a method named `distill` or can't be hashed

distill(**args*, ***kwargs*)

The base distiller will not process any data passed to it, it will send all arguments and keyword arguments to the individual reports.

If either `outputRoot` or `outputNamePrefix` are passed as kwargs, the corresponding values preset in the system will be added to the kwargs sent to the subreports.

Returns fseq.ReportBuilderBase

Returns `self`

outputNamePrefix

Partial file name to prepend the individual reports: str

outputRoot

The base for saving out reports: str

class `fseq.reporting.report_builder.`**ReportBuilderFFT**(**reports*, ***kwargs*)

Bases: `fseq.reporting.report_builder.ReportBuilderBase`

Samples part of data set and performs FFT-based analysis on it.

Parameters **outputRoot: str, optional**

Path to the directory where all reports should be put

(Default: `None`)

outputNamePrefix: str, optional

Partial name to be added to all reports done by the builder

(Default: `None`)

sampleSize: int, optional

Size of sample to be randomly drawn

(Default: 1000)

distanceMetric: str, optional

Name of distance metric to be used. See `METRICS`-attribute for allowed metrics.

(Default: 'correlation')

***reports: objects, optional**

Any number of reports to be added from start

(Default: A fseq.HeatMap)

See also:

`ReportBuilderBase` Base class implementing more attributes.

Attributes

`distanceMetric`	The ReportBuilderFFT.METRIC used: str
`sampleSize`	Size of data subsample to analyze: int

DEFAULT_REPORTS = (<class 'fseq.reporting.reports.HeatMap'>,)

METRICS = set(['kulsinski', 'chebyshev', 'yule', 'sokalmichener', 'dice', 'canberra', 'jaccard', 'minkowski', 'seuclidean',

distanceMetric
The ReportBuilderFFT.METRIC used: str

distill(*data*, *distanceMetric=None*, *clusterOnAbsOnly=True*, **args*, ***kwargs*)
Make reports from data.

Produces two analyses:

Amplitude evaluation A clustered FFT-amplitude analysis

Angle evaluation A clustered FFT-angle analysis

Parameters data: numpy.ndarray

The 2D-array of data given

distanceMetric: str, optional

A distance metric to overwrite the default one of the instance.

(Default: Value of `self.distanceMetric`)

clusterOnAbsOnly: bool, optional

If clustering should be performed only on the amplitude (abs-values) or if amplitude and angle be clustered independently.

(Default: Cluster only on amplitude)

sampleSize
Size of data subsample to analyze: int

class fseq.reporting.report_builder.**ReportBuilderPositionAverage**(**reports*, ***kwargs*)

Bases: `fseq.reporting.report_builder.ReportBuilderBase`

Per position analysis builder.

Parameters **outputRoot: str, optional**

Path to the directory where all reports should be put

(Default: `None`)

outputNamePrefix: str, optional

Partial name to be added to all reports done by the builder

(Default: `None`)

undecidedValue: int, optional

The value for which undecided items were encoded (so it can be omited and calculated separately for 2 of 3 graphs).

(Default: 0.5)

***reports: objects, optional**

Any number of reports to be added from start

(Default: fseq.LinePlot)

See also:

`ReportBuilderBase` Base class which implements some more attributes.

Attributes

undecidedValue	

`DEFAULT_REPORTS` = **(<class 'fseq.reporting.reports.LinePlot'>,)**

`distill`(*data*, *undecidedValue=None*, **args*, ***kwargs*)

The distiller will create reports for several position-type informations.

Average Lacking Data Frequency The number of undecided values.

Average Non-Lacking Data The per position average for all non-lacking data values.

Average Combined Data The per position average as encoded.

Parameters **data: numpy.ndarray**

An array of numerically encoded sequence information

undecidedValue: float, optional

The value that undecided sequence positions are encoded as. If not supplied, the previously set value of the class instance will be used. (Default: 0.5)

***args:**

Any args will be passed to the `ReportBuilderBase.distill`

****kwargs:**

Any kwargs will be passed to the `ReportBuilderBase.distill`

Note: `outputNamePrefix` will be overwritten/added

Returns fseq.ReportBuilderPositionAverage

Returns `self`

undecidedValue

fseq.reporting.reports module Module for holding the various implemented reporting classes

class fseq.reporting.reports.**HeatMap**(*name='heatmap.pdf'*, *saveArgs=()*, *saveKwargs={}*)
Bases: fseq.reporting.reports.ReportBase

Makes heatmaps from data

Parameters **name: str, optional**

A specific name of the report

(Default: "heatmap.pdf")

saveArgs: tuple or list, optional

Any args to be passed to `matplotlib.savefig` after the figure

(Default: Empty tuple)

saveKwargs: dict, optional

Any keyword args to be passed to `matplotlib.savefig`

(Default: Empty dict)

distill(*data*, *name=None*, *outputRoot=None*, *outputNamePrefix=None*, *title=None*, *text=None*, *ylabel=None*, *xlabel=None*, *saveArgs=()*, *saveKwargs={}*, *vmin=None*, *vmax=None*, *aspect='auto'*, *axisOff=True*, *cmap=<matplotlib.colors.LinearSegmentedColormap object at 0x2b5eca180bd0>*, **args*, ***kwargs*)
Creates the actual heatmap.

Parameters **data: numpy.ndarray**

The data to be plotted

name: str, optional

If the default name of the HeatMap instance should be overwritten

(Default: Use the value of `self.name`)

outputRoot: str, optional

The directory in which to place the report

(Default: `None`)

outputNamePrefix: str, optional

A prefix to prepend the name when saving the output. Typically set by the builder to indicate what post-processing was done to the data shown in the report.

(Default: `None`)

title: str, optional

A title to be put over the heatmap (Default: `None`, value automatically scaled by matplotlib)

(Default: `None`)

text: str, optional

An explanatory text to put under the plot

(Default: `None`)

Note: This feature has not been implemented yet.

ylabel: str, optional

A label for the y-axis

(Default: `None`)

xlabel: str, optional

A label for the x-axis

(Default: `None`)

saveArgs: tuple or list, optional

A set of arguments to overwrite the instance's default save args

(Default: empty tuple)

saveKwargs: dict, optional

A set of keyword arguments to overwrite the instanc's default

(Default: empty dict)

vmin: number, optional

To set a minimum color-scale number for the heatmap

(Default: `None`, value automatically scaled by matplotlib)

vmax: number, optional

To set a maximum color-scale number for the heatmap

(Default: `None`, value automatically scaled by matplotlib)

aspect: str, optional

The aspect ratio of the blocks/pixels in the heatmap

(Default: 'auto', which allows for rectangular pixels)

axisOff: bool, optional

If the axis of the plot should not be rendered

(Default: True)

cmap: matplotlib.cmap, optional

A colormap to be used when plotting.

(Default: Red – Blue)

class `fseq.reporting.reports.`**`LinePlot`**(*name='line.pdf'*, *saveArgs=()*, *saveKwargs={}*)

Bases: `fseq.reporting.reports.ReportBase`

Makes lines from data

Parameters **name: str, optional**

A specific name of the report

(Default: "line.pdf")

saveArgs: tuple or list, optional

Any args to be passed to `matplotlib.savefig` after the figure

(Default: Empty tuple)

saveKwargs: dict, optional

Any keyword args to be passed to `matplotlib.savefig`

(Default: Empty dict)

distill(*data*, *name=None*, *outputRoot=None*, *outputNamePrefix=None*, *title=None*, *text=None*, *ylabel=None*, *xlabel=None*, *saveArgs=()*, *saveKwargs={}*, *logX=False*, *logY=False*, *basex=None*, *basey=None*, *labels=None*, **args*, ***kwargs*)

Creates the actual heatmap.

Parameters data: numpy.ndarray

The data to be plotted

name: str, optional

If the default name of the HeatMap instance should be overwritten

(Default: Use the value of `self.name`)

outputRoot: str, optional

The directory in which to place the report

(Default: `None`)

outputNamePrefix: str, optional

A prefix to prepend the name when saving the output. Typically set by the builder to indicate what post-processing was done to the data shown in the report.

(Default: `None`)

title: str, optional

A title to be put over the heatmap (Default: `None`, value automatically scaled by matplotlib)

(Default: `None`)

text: str, optional

An explanatory text to put under the plot

(Default: `None`)

Note: This feature has not been implemented yet.

ylabel: str, optional

A label for the y-axis

(Default: `None`)

xlabel: str, optional

A label for the x-axis

(Default: `None`)

saveArgs: tuple or list, optional

A set of arguments to overwrite the instance's default save args

(Default: empty `tuple`)

saveKwargs: dict, optional

A set of keyword arguments to overwrite the instanc's default

(Default: empty `dict`)

logX: bool, optional

If X-axis should be logged

(Default: `False`)

logY: bool, optional

If Y-axis should be logged

(Default: `False`)

basex: number, optional

To specify other then 10-base logging

(Default: `None`, uses 10-base)

basey: number, optional

To specify other than 10-base logging

(Default: `None`, uses 10-base)

labels: str, optional

To name the line plotted and thus add a legend to the plot.

(Default: `None`)

class `fseq.reporting.reports.`**`ReportBase`**(*name=None*, *saveArgs=()*, *saveKwargs={}*)

Bases: `object`

Base class for simple report creations.

Main purpose is to make a common interface for figure saving using matplotlib figures.

Parameters **name: str, optional**

A specific name of the report

(Default: `None`)

saveArgs: tuple or list, optional

Any args to be passed to `matplotlib.savefig` after the figure

(Default: Empty `tuple`)

saveKwargs: dict, optional

Any keyword args to be passed to `matplotlib.savefig`

(Default: Empty `dict`)

Attributes

name	Name of the plot, used to name the file: str
saveArgs	Save args passed to matplotlib.pyplot.figure.savefig: tuple
saveKwargs	Save keyword args passed to matplotlib.pyplot.figure.savefig:

distill (*data*, *outputRoot=None*, *outputNamePrefix=None*, **args*, ***kwargs*)
Placeholder distill interface not to be used.

Raises **NotImplementedError**

Always raises this exception

name
Name of the plot, used to name the file: str

saveArgs
Save args passed to matplotlib.pyplot.figure.savefig: tuple

saveFig (*fig*, *outputRoot*, *outputNamePrefix*, *name=None*, **args*, ***kwargs*)
Saves a figure and creates directories if needed.

Parameters **fig: matplotlib.figure**

The figure to be saved

outputRoot: str

The root directory for reports

outputNamePrefix: str

If the name of the file should be prepended by some string. Normally added by the *Report Builder*

name: str, optional

A specific name for this figure-file. If none supplied the default name for the report will be used,

(Default: Use the self.name of the instance)

Note: If none supplied and none set for instance, ValueError is raised.

***args:**

Any arguments to be sent to matplotlib.Figure.save. If none added the ReportBase.saveArgs will be used.

****kwargs:**

Any keyword arguments to be sent to matplotlib.Figure.save. If notn addd the ReportBase.saveKwargs will be used.

Returns ReportBase

Returns self

Raises **ValueError**

If no name has been given.

saveKwargs
Save keyword args passed to matplotlib.pyplot.figure.savefig: dict

Module contents Reporting-related modules of fseq.

The sub-package contains *report_builder* which post-processes data and sends it off to make various *reports*.

The builders contains no graphics information, but simply prepares data. A new builder should be written if a new type of analysis is needed based on an abstraction of the data that is not already present in any of the previous builders.

A report makes an image from data sent to it from the builder. Purpose of breaking this out is to allow for changing library that produces the reports and to allow for quick reuse with similar graphics for identical types of graphs for several report-builders.

Module contents

fSeq is a toolbox for sequence analysis in the frequency domain.

The module is organized around two phases of work: *reading* and *reporting*.

The reading uses a data format detector *SeqFormatDetector*, which detects the current *SeqFormat* that is being read by the *SeqReader* and passed to the *SeqEncoder* to translate into a numpy array

The *Report's and 'ReportBuilder* use the output of the import phase to produce the output. They are organized as such that a builder pre-processes the data without any graphics done and the reporters the takes the pre-processed data and make displays out of them.

All relevant parts of *reading* and *reporting* are directly imported to the package root.

Reading

There is one generic reader that is intended to handle all use cases.

fseq.SeqReader Root object that coordinates reading, encoding and reporting

The encoders translates and manages format detection

fseq.SeqEncoder Base class encoder

fseq.SeqEncoderGC Encoder that translates Gs and Cs to 1 while A and T become 0

There's a general format detector, and several data-formats.

fseq.SeqFormatDetector Detector of formats

fseq.SeqFormat The base class from which all formats must be derived

fseq.FastaMultiline Format detecting a fasta-file where sequence may span more than one line

fseq.FastaSingleline Format detecting a fasta-file where sequence is in a signle line

fseq.FastQ Format detecting fastq, but not which quality encoding

Reporting

There is a report builder base from which all report builders should be made, and two specific report builders.

fseq.ReportBuilderBase The base class for all builders

fseq.ReportBuilderPositionAverage A report builder that averages data per position

fseq.ReportBuilderFFT A report builder that subsamples and then does clustered FFT analysis

There are two reports included and an optional base class.

fseq.ReportBase Base class to make constructing new reports more efficient

fseq.LinePlot Plots a line from the data sent to it

fseq.HeatMap Plots a heat-map from the data sent to it.

Exceptions

fseq.FormatError Base exception for error with data-formats

fseq.FormatImplementationError If a format is not correctly implemented

fseq.FormatUnknown If data is of unknown format

1.2 Tutorial

1.2.1 Installing

The program is installed for current user by running:

```
$ python setup.py install --user
```

Or for all users by running:

```
$ sudo python setup.py install
```

The following dependencies needs to be installed separately:

```
numpy, scipy, matplotlib
```

On Debian systems copy:

```
$ sudo apt-get update && sudo apt-get install python-numpy python-scipy python-matplotlib
```

1.2.2 Command Line Use

The following example runs default analysis on two different files:

```
$ fseq ~/Data/Mysc_24_ATCACG_L008_R1_001.fastq ~/Data/Mysc_74_GTTTCG_L008_R1_001.fastq
14-06-23 17:55 SeqReader    INFO     Has 2 jobs
14-06-23 17:55 SeqReader    INFO     Reading: /home/martin/Data/Mysc_24_ATCACG_L008_R1_001.fastq
14-06-23 17:57 SeqReader    INFO     Reading Complete: /home/martin/Data/Mysc_24_ATCACG_L008_R1_001.f
14-06-23 17:57 SeqReader    INFO     Reporting <class 'fseq.reporting.report_builder.ReportBuilderFFT
14-06-23 17:57 SeqReader    INFO     Reporting <class 'fseq.reporting.report_builder.ReportBuilderPos
14-06-23 17:57 SeqReader    INFO     Reading: /home/martin/Data/Mysc_74_GTTTCG_L008_R1_001.fastq
Saving -> /home/martin/Data/Mysc_24_ATCACG_L008_R1_001.fastq.reports/fft-sample.abs.heatmap.pdf
Saving -> /home/martin/Data/Mysc_24_ATCACG_L008_R1_001.fastq.reports/average.total.line.pdf
Saving -> /home/martin/Data/Mysc_24_ATCACG_L008_R1_001.fastq.reports/fft-sample.angle.heatmap.pdf
14-06-23 18:01 SeqReader    INFO     Reading Complete: /home/martin/Data/Mysc_74_GTTTCG_L008_R1_001.f
14-06-23 18:01 SeqReader    INFO     Reporting <class 'fseq.reporting.report_builder.ReportBuilderFFT
14-06-23 18:01 SeqReader    INFO     Reporting <class 'fseq.reporting.report_builder.ReportBuilderPos
14-06-23 18:01 SeqReader    INFO     Waiting for 4 report builders to finish
Saving -> /home/martin/Data/Mysc_74_GTTTCG_L008_R1_001.fastq.reports/average.total.line.pdf
Saving -> /home/martin/Data/Mysc_74_GTTTCG_L008_R1_001.fastq.reports/fft-sample.abs.heatmap.pdf
Saving -> /home/martin/Data/Mysc_74_GTTTCG_L008_R1_001.fastq.reports/fft-sample.angle.heatmap.pdf
Saving -> /home/martin/Data/Mysc_24_ATCACG_L008_R1_001.fastq.reports/average.lacking.line.pdf
```

```
Saving -> /home/martin/Data/Mysc_24_ATCACG_L008_R1_001.fastq.reports/average.not-lacking.line.pdf
Saving -> /home/martin/Data/Mysc_74_GTTTCG_L008_R1_001.fastq.reports/average.lacking.line.pdf
Saving -> /home/martin/Data/Mysc_74_GTTTCG_L008_R1_001.fastq.reports/average.not-lacking.line.pdf
14-06-23 18:03 SeqReader    INFO     All jobs complete`
```

Note: Running above consumes quite a lot of memory and CPU and takes about 10 minutes.

Note: If `fseq` is not found on your system, it usually is due to the default target of scripts for user install is not in your PATH. To amend this, check where install copied the file *scripts/fseq* and append that to your current PATH.

1.2.3 Python Use

For all scenarios it should suffice to import the package:

```
>>> import fseq
```

To create a reader that will run the analysis:

```
>>> r = fseq.SeqReader(dataSourcePaths=("~/Data/Mysc_24_ATCACG_L008_R1_001.fastq", "~/Data/Mysc_74_G
```

We can see how many jobs the reader has left:

```
>>> len(r)
2
```

And we can see the encoding that will be performed (and change it):

```
>>> r.seqEncoder
<fseq.reading.seq_encoder.SeqEncoderGC at 0x7fa9f9539b10>
```

This encoder is the default and will translate Gs and Cs to 1 while As and Ts are made into 0s.

We can also see which reports were requested by the encoder and thus added to the reader since we didn't say what reports we wanted:

```
>>> tuple(r.reportBuilders)
(<fseq.reporting.report_builder.ReportBuilderFFT at 0x7fa9f95399d0>,
 <fseq.reporting.report_builder.ReportBuilderPositionAverage at 0x7fa9f9539c50>)
```

To run encoding and produce results, simply:

```
>>> r.run()
```

Note that this will take some time and consume quite a lot of resources. It took about 10 minutes on a standard desktop for the two files in the command line example, and the python use is no different.

1.3 Developers

`fseq` has been written to initially take care of a very limited set of analysis and sequence formats, while at the same time be written to be highly extensible.

Some examples of suitable features to be included in the future:

- **FastQ SeqFormat Subclasses**

  Sub-classing `fseq.reading.seq_encoder.FastQ` to automatically detect which quality encoding was used based on the range of values in the quality lines fed to it.

- **SeqEncoderQaulity SeqEncoder Subclass**

  Subclass `fseq.reading.seq_encoder.SeqEncoder` such that it encodes the quality line using the quality encoding supplied by the `fseq.reading.seq_encoder.SeqFormat` detected.

1.3.1 Git

The source code project is hosted at:

https://gitorious.org/fseq

For merge requests, the code is expected to:

- Be documented in accordance with `numpydoc`-format (see https://github.com/numpy/numpy/blob/master/doc/HOWTO_DOCUM
- Code to be PEP8 compliant (see http://legacy.python.org/dev/peps/pep-0008/)
- Coding style in general to follow the style established in *fseq*
- Git commits to in general be *informative* and have *one aspect changed per commit*
- Unit-tests to cover new functionality

1.3.2 Reading

To extend the functionality by adding further encoders, these encoders should be derived from `fseq.reading.seq_encoder.SeqEncoder` and as a minimal requirement need to overwrite the `fseq.reading.seq_encoder.SeqEncoder.parse()`-method.

To extend the functionality by adding support for more input formats, classes should be derived from `fseq.reading.seq_encoder.SeqFormat` or any of the already implemented formats if they partially solve detection for the new format. Minimal requirement is overwriting the `fseq.reading.seq_encoder.SeqFormat.expects()`-method, but typically many of the properties of the base class as well as the constructor needs replacing.

`SeqEncoder.parse(self, lines, out, outindex)` overwriting

Important 1: The overwritten `fseq.reading.seq_encoder.SeqEncoder.parse()` must have identical parameter set. If further information is needed, this should be dealt with during initiation or by separate methods.

Important 2: The overwritten method may not throw any errors and should silently handle scenarios where the length of the information to be encoded mismatches the length of the corresponding slot of the `out` object.

Example of how out the second important note can be achieved (adapted from `fseq.reading.seq_encoder.SeqEncoderGC.parse()`):

```
#Point to line of interes
l = lines[self._sequence_line]

#Put the contents of that line directly into out
out[outindex][:len(l)] = l[:out.shape(1)]
```

The above example is quite useless as an encoder as it doesn't translate the contents of the input in any way, but the `[:len(l)]` on `out` ensures the target slot of `out` is not too large, while the `[:out.shape(1)]` ensures that *l* is not too large for the slot in `out`.

Important 3: The `parse`-method may use the state of the class instance (as in the above example), but due to concurrency issues, *it should not alter the state*.

`SeqFormat` sub-classing

** Important 1:** If a class is parent to further sub-classing such that the class will conform to all data that the more specific subclass will do (e.g. FastQ will be expect all lines/return `True` for all scenarios that a FastQ_Q33-subclass that detects fastq-files with encoding starting at 33), then:

- The *parent* should implement the `fseq.reading.seq_encoder.SeqFormat._decay()` method similar to the base class and have a suitable `self._giveup` set in its init.
- The specific *child* should overwrite the `_decay`-method so that it never gives up *or alternatively* takes longer before it gives up by having ha higher number set to `self._giveup` in init.

The `self.expect(line)` should return a boolean if the line fits what was expected as the next line, this method doesn't need to continue reporting `False` after its first occurrence. As soon as an `expect`-method returns a `False`, that `SeqFormat` is removed from possible formats by the `fseq.reading.seq_encoder.SeqFormatDetector`.

1.3.3 Reporting

To extend the available abstractions/analysis done to the encoded data, new derived `fseq.reporting.report_builder.ReportBuilderBase` classes should be made. Typically the `__init__` and `distill` would be overwritten (but the super class methods called), and the `DEFAULT_REPORTS` attribute replaced. Potentially the interface extended by more relevant methods and properties needed for user customization of the post-processing.

For creating new reports any object having a `distill`-method will do, but using `fseq.reporting.reports.ReportBase` will save some implementation by having implemented the common aspects of saving figures in `matplotlib`.

`ReportBuilderBase` sub-classing

To maintain the constructor interface it is highly recommended that the init has the following structure:

```
def __init__(self, *reports, **kwargs):

    if len(reports) == 0:
        reports = tuple(r() for r in self.DEFAULT_REPORTS)

    super(MyReportBuilder, self).__init__(*reports, **kwargs)

    #Emulating default values for keywords is done by getting
    #the key with default values as follows
    self.someKey = kwargs.get('someKey', defaultValue)
```

To push some data to all attached reports make a `super` call to `fseq.reporting.report_builder.ReportBuilderBase.d`

`ReportBase` sub-classing or not

Using `ReportBase` to create new reports is entirely optional, but if the report is a `matplotlib`-report, then it is probably useful.

If sub-classing, then the `fseq.reporting.reports.ReportBase.distill()` must be overwritten and subclass should use a call to the inherited `saveFig`-method to do the actually saving once the figure has been setup within the `distill` method.

If using other modules than `matplotlib` and thereby not sub-classing `ReportBase`, the report should as a minimal requirement have a `distill` method that takes the main data as the first argument and that accepts any number of argument and keyword arguments by having something like `*args, **kwargs` at the end of the parameter list.

1.4 fSeq Project Report for C3SE Graduate Course: Python and High Performance Computing 2014

1.4.1 Abstract

fSeq was created for the C3SE Graduate Course [c3se], in part for want of suitable existing project or code base. Therefor only part of the course content could be covered – `numpy`, `scipy`, `matplotlib`, `unittest` and `sphinx` being the most prominent. The package contains classes to do simple per base analysis of the data as is existing elsewhere in e.g. [fastqc]. However, it also extends sequence quality analysis with heat maps of clustered Fourier data, which to the best of my knowledge, is novel to the field. These reflect previously identified issues of sequence data but also introduces new frequency features, which requires further investigation.

1.4.2 Solution Method

Design Analysis

The task was split up into several well defined components:

- **Reading data**

  This feature can be held generic, it will need to be able to talk to both encoders and format detectors on one side as well as post-processing classes on the other.

  The class can do several parts in parallel:

  - Reading data from file
  - Encoding chunks of data
  - Reporting on encoded data

- **Detecting input data format**

  To be able to support several formats and allow for future extension of formats supported, the task should be separated into *selecting suitable detector* among a collection of format collectors and *detecting particular formats*.

  The latter should have a common interface for the former to use, thus a base class that the detectors can subclass is suitable.

  The logic of the formats differ enough that making a factory design pattern would probably require more than is gained.

- **Encoding input**

  Potentially there is an unknown number of ways to encode the data.

  The data reader needs an interface where objects can be sent such that no concurrency issues can arise, that is, it may not alter the state. This interface needs to be common to all encoders.

  To simplify the use, the encoder can manage its format detector

The encoder can suggest default post-processors to be coupled with it in the reader. This will make the invocation less transparent, but greatly reduce the workload of the user. Thus the user must be able to override the default coupling, and this manner must be clear.

- **Post-processing encoded input**

  One type of encoding can potentially be used for several downstream analyses. However, it is not inherently clear if some aspects should be done by the encoder directly or the post-processing class.

  The post-processing class should coordinate all outputs made from its data.

  It should assist in naming and annotation to clarify the contents of the report.

  Reports from one post-processor should be grouped by file name.

  Post-processors need to have a common interface that doesn't alter the state for the reader to use.

- **Producing outputs**

  Dependence on graphics libraries such as `matplotlib` should be restricted to the output generators.

  These classes also need to have a common interface for the post-processors and may not alter the state (due to concurrencies).

  An output producer should be reusable for several post-processors.

General

The default invocation of analysis should require a minimum of user input. For each deviation for what is considered *default*, a new set of default behaviors should exist.

The interdependence of the classes should be kept simple and clean such that each class can be instantiated and as a maximum be needed as a parameter to one other class.

The complete setup and configuration of a class should be possible via its constructor and methods returning `self` such that all imaginable combinations of settings can be setup and run in a single line.

Parallelism

As described above, the main target for explicit coding of parallel computing is the reader object.

To be able to easily share the data storing `numpy.ndarray`, `treading` was selected. Other considered modules were `multiprocessing` and `pycuda`, however due to limitations in implementation time the former was used.

Still parallelism poses several issues, the size of the array cannot be changed without creating a new object, while the needed size cannot be known before the contents of the file has been read. If the whole file is scanned, reading is impossible due to the large amount of memory needed, a large benefit of threading would have been lost.

Therefore, a design was opted for where the main thread creates a large array and several threads are used to fill that array up from data read from the source file. When the array is full, the main thread must pause, reading data and wait for all live threads to finish, after which the array can be extended. Then, reading and threading can start again.

Unittesting

Test driven development [tdd] was considered, but as development and especially design time was extremely limited, testing was decided to:

- Verify all isolated behaviors

- Verify all interdependent behaviors that don't require running the entire analysis nor would produce files on the hard drive.

The tests were decided to be placed inside the package but not be part of the distribution.

Documentation

The documentation of the code should be compatible with `sphinx` [sphinx] and follow the `numpydoc` [npd] standard of restructured text [rst].

1.4.3 Implementation

Package structure

The relevant folder tree for the package was devised as follows:

- fseq (root of *git*-repository)
  - fseq (package/source root)
    - reading
    - reporting
    - tests (testings modules, not included in distribution)
  - scripts (run-scripts installed)
  - doc (sphinx-documentation)

The *setup.py* file was structured so that the scripts in the script folder were installed as executables so that the package can be run as a stand alone command line program.

A *MANIFEST.in* was created in accordance with `distutil`'s recommendations [distutil] to allow for distribution of packages via the *setup.py* file. The tests in the *testing* folder were purposely kept out of packaging as they were not considered part of the deployment code, but rather the development source code.

Design

The structure and interfaces of the classes kept as designed, making the following basic types:

- `SeqReader`
- `SeqEncoder` to encode data and manage format detection if not predefined.

  A specific subclass `SeqEncoderGC` was made to fulfill the goal of doing GC-analysis
- `SeqFormat` the object that detects specific formats for which three different formats are supported `FastaSingleline`, `FastaMultiline`, and `FastQ`
- `SeqFormatDetector` to select which format an input stream is.
- `ReportBuilderBase` the post-processing coordinator, for which two specific post-processors were created to allow `fseq` to produce usable Fourier reports: `ReportBuilderFFT` and `ReportBuilderPositionAverage`.
- `ReportBase` conforms with output producer, for which two specific graph producers (`LinePlot` and `HeatMap`) were created.

To comply with the general design criteria, all relevant classes are imported into the package root such that the user only needs to use `import fseq`.

Default behavior is simple as the following is sufficient:

```
>>> fseq.SeqReader(dataSourcePaths="some/path/to/file.fastq").run()
```

Further, full customization can be performed and expressed in a single line. The expression can also be split to several lines increase readability.

Unittests

In total 78 different tests were created in four different files. Each file corresponding to one of the four modules in the package. A test exclusively tested one aspect of the functionality, but many of the tests asserted more than one behavior for that aspect.

For example, `TestSeqFormatDetector.test_FormatUnknown` that ascertains that an exception is raised for when the detector runs out of available formats both when it was initiated with and without a forced format.

Documentation

All classes were fully documented as decided and several `sphinx` used to produce a complete documentation with several supporting extra documents.

1.4.4 Results

Technical results

A run took less than 10 minutes on a standard Intel i5 desktop with 4GB RAM and a 2TB HDD. Typically more than 100% CPU was used, though during resizing of the array, a dipping of CPU was clear due to main thread waiting for all threads to join. The memory usage peaked around 75% when using 16-bit float point precision, in *numpy*. With default settings, five report pdf:s were created for each file analyzed.

The unit tests typically ran for a fraction of a second and succeeded in reporting previously undetected errors as well as alerting to inconsistencies caused by minor changes of interfaces during development.

Analysis of two files

Two real data files were analyzed *Mysc_24_ATCACG_L008_R1_001.fastq* and *Mysc_74_GTTTCG_L008_R1_001.fastq*. The two files were multiplexed in the same Illumina MiSeq lane, but are two distinct species. Therefore, technical aspects of the sequencing can possibly be seen as recurring features in the two, while aspects pertaining to the DNA in each sample should be private.

As an example, the occurrence of undecided nucleotides is highly concurrent in both data files: `Mysc 24` `Mysc 74`

While the GC bias over the two files are distinctly different: `Mysc 24` `Mysc 74`

The *Myst 24* having a highly structured bias as averaged over the ~5M reads.

The random sample of 1000 reads, Fourier Transformed and clustered based on their amplitudes show little obvious structure in their angles:

`Mysc 24` `Mysc 74`

While the corresponding amplitudes for the same 1000 reads share two clear features. First, for the 0-frequency, an obvious large spread in overall GC bias is evident with a small subset of around 90% GC a majority around 40-50 and

another smaller cluster close to 0%. The second feature, which shows clearly in both is that the 1/34 frequency and its neighbors behave distinctively.

`Mysc 24` `Mysc 74`

1.4.5 Discussion

Package

The general design of the project was maintained during development and the extension of functionality during worked as intended. The package therefore shows promise of being well structured and designed.

The `threading` had some inherent issues with sleeping threads not appearing alive causing jumbled and random encodings initially until sufficiently slow implementation ensured threads are truly joined before reshaping of encoding array. There are some possibilities for further improving the performance of the `SeqReader` by decoupling the data reading from the managing of the encoding threads as well as taking an active part in managing the number of the latter. Moving away from single processing should also be feasible and could be the target of further performance development.

The use of unit tests worked well in assisting the development and as they were written in junction with the code they were not merely a *post-hoc* addition to prove the correctness of the implementation, but actively discovered issues previously unknown.

In general, the time plan was kept with the exception of documentation and report writing, for which much more time would have been needed to learn `sphinx` and `numpydoc` sufficiently well to produce both this report and the general package documentation.

Bioinformatics

The analyses included in the package reproduces know result where comparison is applicable. For example, the uneven bias of GC initially due to faulty timing of adapters – a known issue. More interestingly the implicated a recurring frequency on the amplitude analysis of clustered FFT data around 34/101. The implication of this needs to be further investigated. Potentially, protein coding regions in the sequence, for which triplicates of nucleotides form the information unit in translation of DNA to amino acids of the protein, could be related as it implies the factor 3. However, *why* and if this information can be useful remains to be investigated.

1.4.6 References

1.4.7 Appendix A: Project Plan

The `project plan` submitted for the project.

1.4.8 Appendix B: Code

The current code is accessible from *Gitorious* at:

https://gitorious.org/fseq

Alternatively, each class implementation can be accessed here:

- `fseq.reading`

        fseq.reading.seq_reader.SeqReader

        fseq.reading.seq_encoder.SeqEncoder

        fseq.reading.seq_encoder.SeqFormat

            fseq.reading.seq_encoder.FastQ

            fseq.reading.seq_encoder.FastaMultiline

            fseq.reading.seq_encoder.FastaSingleline

        fseq.reading.seq_encoder.SeqFormatDetector

- fseq.reporting

        fseq.reporting.reports.ReportBase

            fseq.reporting.reports.HeatMap

            fseq.reporting.reports.LinePlot

        fseq.reporting.report_builder.ReportBuilderBase

            fseq.reporting.report_builder.ReportBuilderFFT

            fseq.reporting.report_builder.ReportBuilderPositionAverage

1.5 License

CHAPTER

TWO

INDICES

- *genindex*
- *modindex*

CHAPTER

THREE

CONTACT

The package was written and is maintained by Martin Zackrisson.

Developers, look at *Developers* about Git and use *Gitorious* as mode of contact.

For end-users, questions can be addressed to the e-mail:

martin[dot]zackrisson[at]gu[dot]se

[c3se] http://www.c3se.chalmers.se/index.php/Python_and_High_Performance_Computing_2014

[distutil] https://docs.python.org/2/distutils/sourcedist.html#the-manifest-in-template

[tdd] http://en.wikipedia.org/wiki/Test-driven_development

[npd] https://github.com/numpy/numpy/blob/master/doc/HOWTO_DOCUMENT.rst.txt#common-rest-concepts

[sphinx] http://sphinx-doc.org/

[rst] http://docutils.sourceforge.net/docs/ref/rst/restructuredtext.html#bullet-lists

[fastqc] http://www.bioinformatics.babraham.ac.uk/projects/fastqc/

f