SSuMMo Documentation

Release 0.5a

Alex LB Leach

September 30, 2013

1 THE SSUMMO SOFTWARE **1**
- 1.1 The main algorithm - SSUMMO.py . . . 1
- 1.2 v0.0.1 . . . 1
- 1.3 v0.0.2 . . . 2
- 1.4 v0.0.3 . . . 2
- 1.5 v.0.0.4 . . . 2

2 Tutorial **3**
- 2.1 STORAGE OF RESULT DATA . . . 3
- 2.2 CONVERSION SCRIPTS . . . 3

3 `ssummo` Library **7**
- 3.1 Dependencies . . . 7
- 3.2 Compatibility . . . 7
- 3.3 `ArbIO` Module . . . 7
- 3.4 `CONFIG` Module . . . 8
- 3.5 `HMMcompare` Module . . . 9
- 3.6 `Phylogeny` Module . . . 9
- 3.7 `acc_finder` Module . . . 10
- 3.8 `alignDB` Module . . . 10
- 3.9 `cmd_options` Module . . . 11
- 3.10 `colours` Module . . . 11
- 3.11 `compare_trees` Module . . . 12
- 3.12 `count_hmms` Module . . . 14
- 3.13 `count_seqs` Module . . . 15
- 3.14 `dict_to_html` Module . . . 15
- 3.15 `dictify` Module . . . 16
- 3.16 `find_taxa` Module . . . 18
- 3.17 `graphs` Module . . . 19
- 3.18 `link_EMBL_taxonomy` Module . . . 20
- 3.19 `phyloxml` Module . . . 21
- 3.20 `pressGenus` Module . . . 22
- 3.21 `resample` Module . . . 22
- 3.22 `seqDB` Module . . . 23
- 3.23 `seqLen` Module . . . 24
- 3.24 `simulate_lengths` Module . . . 25
- 3.25 `ssummolib` Module . . . 25
- 3.26 `taxonomy` Module . . . 26
- 3.27 `traverse` Module . . . 27

4 `ssummo` Executables **29**
- 4.1 ACGTCounts.py . . . 29
- 4.2 SSUMMO.py . . . 29
- 4.3 comparative_results.py . . . 32
- 4.4 dict_to_phyloxml.py . . . 32

4.5 plot_data.py 32
4.6 rankAbundance.py 32
4.7 rarefactionCurve.py 33

Python Module Index **35**

Index **37**

CHAPTER

ONE

THE SSUMMO SOFTWARE

Firstly, thanks for your interest in SSUMMO! This manual has been written for the end user and maintainer, to explain SSuMMo and how it evolved at an algorithmic level. I also describe how data is stored and converted between various file formats and how to update the SSuMMo database to use the latest ARB Silva sequence database. Also included is a tutorial, listing some practical examples of using SSuMMo to annotate sequences, reconstruct taxonomies from sequence data, and compare multiple sequence datasets.

SSuMMo is designed to read in a file of DNA or RNA sequences, and assign those sequences to taxonomic ranks. The sequence file can be read in a wide variety of sequence file formats, (fasta, fastq, sff, Stockholm, etc.), and each of those sequences will be allocated to taxa in the tree of life, so long as they are found to contain the gene, or genes of interest. The current version of SSuMMo only recognizes small subunit ribosomal RNA sequences, but other marker genes can be added to the SSuMMo database, albeit with a bit of work. Let's not get ahead of ourselves! First of all, I feel I should explain how SSuMMo evolved from a simple collection of python scripts, to an object-oriented, multiprocessing program that it is today.

1.1 The main algorithm - SSUMMO.py

The first task of the SSuMMo program is to load a pre-compiled index of ARB taxonomies into memory, for fast traversion of the taxonomy database. When first loaded, this object takes up a lot of memory (~50MB), but as the tree of life is traversed, taxa which aren't identified amongst the query sequences are completely dropped (from memory), but identified taxa are saved into a taxonomy object which grows as taxa at higher resolutions are found. As subsequent nodes are encountered, accessions allocated to each taxon are "popped" from parent to the winning child node, unless the sequence is just as likely to have derived from two or more taxa, in which case the accession is left with the parent, as well as with all equally probable child taxa. This design, of recursively allocating sequences to increasingly specific nodes in the tree of life, was incorporated into all versions of SSuMMo, but could still be implemented in a number of ways. In order to minimize processing times, we built and tested SSuMMo using hmmer in 3 different methods, and took the fastest of these forward for development as a parallel-processing program.

1.2 v0.0.1

Sequences were parsed individually and compared to each node's child HMMs using Hmmer v3.0's Viterbi algorithm, built into hmmsearch. Bit-scores returned from hmmsearch for each query sequence vs. HMM comparison are collated at each node, and the highest bit-score is chosen as the taxon whence the sequence is most likely to have derived. The algorithm proceeds to score the query sequence against each child of the previous round's winning taxon. This process continues until there are either no further HMMs with which to compare, or there are two or more HMMs with the same parent which result in the same bit-score. In the case where there is no clear winner (multiple HMMs with equal alignment bit- scores), the sequence is compared to the child nodes of each winning node, continuing until there is a clear winner. If no unique winner is found and there are no further children, then this sequence is assigned to the last single winning node, and classed as an "ambiguous sequence".

1.3 v0.0.2

Every sequence is simultaneously piped to an hmmsearch program to be aligned and scored against a single HMM at a time. After each round of running hmmsearch against a node's children, the highest bit-scores for each sequence are collated, and the query sequence accessions are assigned to the relevant best- scoring HMM. Similar to v.0.0.1, if multiple children result in equal scores, hmmsearch continues to run on all subsequent top- scoring children until a single, highest bit-score is found. The accession is retained with the last uniquely winning parent node if no clear winner is found.

1.4 v0.0.3

The database of HMMs was first modified to include "pressed" HMMs at each node in the decision tree. These are created by using HMMer v3.0's hmmpress program, which reads in multiple HMMs and converts them into several optimized, binary files. SSuMMo v.0.0.3 use hmmscan in place of hmmsearch, which can score single or multiple sequences against each set of pressed HMMs. Otherwise, SSuMMo v.0.0.3 used similar application logic to the previous versions.

1.5 v.0.0.4

After profiling all the above versions for speed, we noted that SSuMMo v.0.0.2 performed the fastest on the same test dataset, and using the same test environment (same machine, HMM database and hmmer version). We thus chose this version for further development, and optimized it to take advantage of Python's multiprocessing module. This was used to parallelize the hmmsearch program calls, so that a variable number of hmmsearch processes can be started simultaneously and thus the sequences can be compared against multiple HMMs at the same time, across multiple processors.

CHAPTER

TWO

TUTORIAL

2.1 STORAGE OF RESULT DATA

All results files are stored in standard files denoted by their file name suffix (html, csv, jpg etc.) or as optimized "pickled" objects, created using Python's cPickle module. Phyloxml trees are saved with a .xml extension. These pickled objects can be used by SSuMMo post-analysis programs for conversion to a wide number of formats or used for further population analyses.

2.2 CONVERSION SCRIPTS

2.2.1 dict_to_html.py

dict_to_html.py can create a dynamic html file which can be uploaded to a webserver or viewed in a local web browser. Template html, Javascript and css files are provided as templates with which to create dynamic html representations of an identified taxonomy.

2.2.2 dict_to_phyloxml.py

No prior template files are necessary to create a phyloxml representation of SSuMMo results; only the .pkl file or files which are produced by SSuMMo. However, this script can also output files which automatically colour and annotate the trees when viewed in IToL, according to taxonomy.

2.2.3 Turning results files into figures.

The results dictionary is saved in "pickled" format, and 3 scripts have been written to convert this results dictionary to either dhtml for browsing in a web browser, phyloxml for visualizing in a tree viewer, or a tab- or comma-delimited text file for tabular representation of each sequence's final results data, in Microsoft Excel for example. Perhaps the best functionality is realized with multiple results files, when multiple trees can be combined and compared alongside one another. **comparative_results.py** is designed for this purpose, and is the swiss-army knife of the SSuMMo toolkit for manipulating results data files. See below for examples of it's useage.

2.2.4 Preparing test sequence data

The ssummo database was prepared with ARB's SSURef 102 dataset, which contains 460,783 non-redundant* small subunit rRNA sequences and was released on 15th February 2010 [http://beta.arb-silva.de/]. An updated release, SSUParc 103, which contains 1,353,677 sequences, was made available on the 1st June 2010. This provides an opportunity to test up to 892,894 full-length, pre-annotated sequences not present in the training data. Sequences exclusively present in SSUParc 103 were determined and placed into a blast database. The accession numbers for all of these sequences were stored as a binary index file (a pickled python dictionary with accession number as key, and EMBL taxonomy as the value).

* "non-redundant" refers to the fact that highly similar sequences have been removed, so that the dataset is a minimised but still providing a comprehensive reflection of the diversity in small subunit RNA sequences.

2.2.5 Sequence length vs. accuracy of genus assignment

Full length test sequences were randomly chosen and extracted in aligned fasta format. Two commonly used universal primers and their target regions were identified for both prokaryotic 16S rRNA and eukaryotic 18S rRNA. Each universal primer is designed to target a specific region of the reference alignments, and these regions, as reported in the literature are presented in Table **??**.

Subsequences were extracted to mimic data from a real pyrosequencing experiment, by choosing start residues as would be targeted by each primer, and then cutting the sequences to lengths varying from a maximum of 700 residues (depending on length of sequence from primer annealing site) to 40 residues, in steps of 10 bases. Each subsequence was passed through SSuMMo and the genus assignments recorded and tallied.

Table 2.1: Conserved regions identified in 16S rRNA

Primer	Primer sequence 5′-3′	Alignment Position
E343F	TACGGRAGGCAGCAG	343–357
U519F	CAGCMGCCGCGGTAATWC	519-537
U789F	TAGATACCCSSGTAGTCC	789-807
A906R	CCCGCCAATTCCTTTAAGTTTC	906-927
U1053F	GCATGGCYGYCGTCAG	1053-1068

In pyrosequencing experiments, universal primers are designed to target conserved regions in SSU rRNA, so that a single universal primer can be used to determine sequence information from one or more groups of organisms.

BUILDING THE SSUMMO DATABASE FROM A NEW ARB RELEASE

The SSuMMo software first needs to be downloaded (from http://ssummo.googlecode.com/p/ssummo).

Edit `ssummo.CONFIG` pointing `ssummo.CONFIG.top` and `ssummo.CONFIG.arbDBdir` to directories where you have write-permission. `ssummo.CONFIG.top` should be a directory where the index file (`ssummo.CONFIG.taxIndex`) will be stored. `ssummo.CONFIG.arbDBdir` will be the root directory for the ARB Silva database of SSU rRNA HMMs.

e.g.

```
top = '/var/lib/ssummo'
arbDBdir = top + '/arbDB106'
taxIndex = 'arbDB106.pkl'
```

You need a copy of the ARB Silva sequence databases to continue. Go to ftp://ftp.arb-silva.de/release_106/Exports/ and get the SSURef_*tax* and SSURef_*full_align* file/s.

Then, to build the database

```
# Change to directory with SSURef sequences.
$ cd /path/to/top/arbDB106

# Create the ARB taxonomy index.
$ python /path/to/dictify.py --indexTaxa SSURef_[version]_tax_silva.fasta

# Remove gapped sequences and convert leading and trailing '.'s to '-'s
$ python /path/to/dictify.py --rewrite SSURef_[ver]_tax_silva.fasta
```

After downloading the SSURef...full_align...tgz file from the ARB Silva FTP server

```
# Extract it.
$ tar xzf <file_name.tgz>
```

```
# Index the byte locations of each sequence.
python ../bin/dictify.py –-indexSeqs SSU*full_align*.fasta
```

This outputs a file with the same name, but with ".fas" suffix. Now, split sequences into files named by their domain (e.g. 'Bacteria.fas', etc.)

```
python ../bin/dictify.py –-splitTaxa SSU*full_align*.fas
```

SSUMMO COMMANDS

- Assign fasta formatted sequences to taxa

  ```
  python /path/to/SSUMMO.py <sequencefile>
  ```

- Annotate sequences from sff format and rewrite to fasta format

  ```
  python /path/to/SSUMMO.py –format sff –in sequencefile.sff \
      -out annotatedseqs.fas –outformat fasta
  ```

2.2.6 Loading the SSuMMo MySQL database

The SSuMMo database can be loaded like so, on any POSIX compliant operating system

```
$ mysql –h <hostname> –u <username> –p <database> < taxdump.sql
```

CHAPTER

THREE

SSUMMO LIBRARY

SSuMMo import hook. Prints GPL copyright string on import and hacks the import path, in order to make everything in the ssummo package directory directly importable..

With a slightly more modularised design, this should become unnecessary.

3.1 Dependencies

Make sure that SSuMMo can import the following modules:-

- MySQLdb (from the python-mysql package)
- Bio (the python-biopython package)
- hmmer (at least version 3.0 - available http://hmmer.janelia.org/)

3.2 Compatibility

SSUMMO is a package written purely in CPython and has been tested on Ubuntu & Mac OS X, both running python 2.6.x. & python 2.7.x

IronPython in Windows won't work because it doesn't allow the use of CPython's built-in 'signal' module. CPython in windows probably won't work either unless you've got hmmer compiled and installed on Cygwin, as a POSIX subsystem is required.

3.3 `ArbIO` Module

Description:

Class to work with alignments from the ARB database. Their alignments are big and some of the sequences contain gaps. This can output those sequences and give the option of whether to keep those gaps or not.

Example:- python ArbIO.py SSU_Ref_104_full_align.fasta

class `ssummo.ArbIO.`**`ArbIO`**(*inHandle=<open file '<stdin>', mode 'r' at 0x1007240c0>*, *out=<open file '<stdout>', mode 'w' at 0x100724150>*, *index=None*)

Bases: `object`

Works with large files... Keep the file object open, and move around using the ArbIO.index This contains byte locations of each record, but if indexed, then the sequence will be returned.

`FastaIterator`(*skipGapped=True*)

`arbSeqToStr`(*sequence*, *skipGapped=True*)

close()
Closes 3 open handles:- self.inFile, self.outFile, self.indexFile (if it's open, dumping contents of self.indexes there)

dumpAndIndex(*outHandle*, *skipGapped=True*, *format='fasta'*)
Writes converted arb sequence file to a file handle. Updates the self.index simultaneously.

fetch(*accessions*)
Give a list of accessions and this will yield them as SeqRecord objects by reading them from the rewritten file.

index()
Indexes self.inFile for sequence locations. Dumps the index to self.indexFile, which is named with the same prefix as self.inFile, but with '.pklindex' as the suffix.

indexAndInfo()
Indexes self.inFile for sequence locations, whilst creating dictionaries containing the sequence header information. If a species is presented in [...], this is stored in a dictionary called self.speciesDict, with accessions as the keys. Everything else in a sequence header is stored in another dictionary self.info

parse(*handle*, *skipGapped=True*)
An iterator function that yield's SeqRecord objects from an ARB sequence alignment. Parsing includes converting dots '.' at the start and end of each alignment to dashes '-'. Also, will check for dots in the middle of the sequence, which represent gaps in the alignment. If these dots are found and skipGapped=True (default), then these SeqRecords will not be yielded.

parseHeader(*seqHeader*)
Given the header of an ARB sequence or fasta sequence, will return the tuple:- (id, name, description). id is the everything up until the first bit of whitespace. description is the rest up until '[' if found. name is anything that's in the square brackets.

pipeSequences(*accessions*, *outPipe*)
SeqIOIndex - An Index created by IndexDB / SeqIO.index accessions - a list of accessions to be returned in fasta format

regreplace(*matchObject*)

rereplace(*regObj*)

taxonomify(*taxonomy*, *accession*)
Given the taxonomy in the sequence header (stored as ArbSeqRecord.description), update the taxonomy dictionary (self.taxonomy)

class ssummo.ArbIO.**ArbSeqRecord**(*seq*, ***kwargs*)
Bases: Bio.SeqRecord.SeqRecord

storeindex(*index*)

tell()

exception ssummo.ArbIO.**DotsException**
Bases: exceptions.Exception

ssummo.ArbIO.**main**(*options*)

3.4 CONFIG Module

This is the configuration file. You are expected to make sure all the path names and account details in here are right for you!

It is also worth creating an IToL account, requesting a batch-upload account and entering the username and password they provide into this file.

3.5 HMMcompare Module

Reads an HMM file and converts each natural log probability into a position specific scoring matrix.

class ssummo.HMMcompare.**AlignSeq**(*inPipe*, *outPipe*, *options*)
Bases: multiprocessing.process.Process

run()

class ssummo.HMMcompare.**CalcProb**(*inQ*, *sumQ*, *lock*)
Bases: multiprocessing.process.Process

run()

exception ssummo.HMMcompare.**ColumnError**
Bases: exceptions.Exception

class ssummo.HMMcompare.**LocalOptions**(*args=None*)
Bases: ssummo.cmd_options.Options

help_text = {'-ncpus': 'Number of processes to intitiate which will do the math calculations', '-desc': 'Filter input sequ

options = {'-ncpus': 3, '-desc': [], '-accs': [], '-format': 'fasta', '-out': <open file '<stdout>', mode 'w' at 0x100724150>, '-c

parse_args(*args*)

print_help()

class ssummo.HMMcompare.**SumProbs**(*inQ*, *outQ*)
Bases: multiprocessing.process.Process

run()

ssummo.HMMcompare.**get_consensus**(*HMM*)

ssummo.HMMcompare.**parseHMM**(*options*, *prob_calc_procs*)

ssummo.HMMcompare.**parse_seqs**(*options*)

ssummo.HMMcompare.**print_HMMs_vs_seq**(*accession_names*, *matrices*, *resultseq*, *out=<open file '<stdout>', mode 'w' at 0x100724150>*)
This will write (to out) the cumulative sequence scores a single sequence against multiple HMMs. The columns are subdivided into groups, each group corresponding to an HMM. Within each group of columns, every sequence gets a column of it's own. Note that consensus sequence scores show the cumulative scores for consensus sequences that are unique for each HMM.

ssummo.HMMcompare.**print_seqs_vs_HMM**(*accession_names*, *matrices*, *out=<open file '<stdout>', mode 'w' at 0x100724150>*)
This prints multiple seqeunces against each sequential HMM. So the output will contain columns containing the cumulative sequence scores of each sequence against just one model. If there's multiple HMMs, then the next HMM will be printed underneath.

ssummo.HMMcompare.**score_alignment**(*alignment_residues*, *matrices*, *processes*)

3.6 Phylogeny Module

ssummo.Phylogeny.**combine_dicts**(*results_dicts*, *merge=False*)
Give a list of SSUMMO results dictionaries. This shall return a dictionary containing each & every node from all of those dictionaries.

Where accessions are found assigned to a node, this will combine the accessions from all results_dicts into a list of lists; one list of accessions per results dictionary, in the same order as are passed to this function.

ssummo.Phylogeny.**merge_dicts**(*results_dicts*)
Give a list of SSUMMO results dictionaries. This shall return a dictionary containing each & every node from all of those dictionaries.

Where accessions are found assigned to a node, this will combine the accessions from all results_dicts into a list of lists; one list of accessions per results dictionary, in the same order as are passed to this function.

3.7 acc_finder Module

Script to retrieve accessions assigned to specific taxa. Searches through SSUMMO results files (.pkl files) and retrieves all accessions asigned to the taxa of interest.

class ssummo.acc_finder.**Finder**(*cwd='.'*)

find_seq_file(*dataset_name*)

get_format(*file_name*)

get_sequences(*dataset_name*, *accessions*)
Dataset_name is the name of the dataset file. sequence format will be auto-detected. accessions should be an iterable item that yields accession numbers to search for in the file.

search(*results_dict*, *name*)

3.8 alignDB Module

Simple command line tool to extract sequences of interest from a sequence file.

Description:

Program to reformat or filter sequences in a variety of formats. See http://www.biopython.org/wiki/SeqIO for a list of formats, although there are currently more supported by biopython.

e.g. *sff* format (Roche 454's binary sequence format) is supported by biopython, but not mentioned on the webpage.

e.g.

```
alignDB.py  -db infile.sto
            [-format stockholm]
            [-accs "acc_1" "acc_2" "acc_N"]
            [-desc "Methanobacter maripuladis" "Methanococcus"]
```

class ssummo.alignDB.**LocalOptions**(*args=None*)
Bases: ssummo.cmd_options.Options

help_text = {'-desc': 'Descriptions to look for within db. Again, is relaxed, so returns all matches. Envelope each descr

options = {'-desc': [], '-accs': [], '-db': None, '-format': 'fasta', '-out': <open file '<stdout>', mode 'w' at 0x100724150>, '-o

parse_args(*args*)

print_help()

class ssummo.alignDB.**Reader**(*options*)

reverser(*seq*)

writer(*seq*)

ssummo.alignDB.**parse_input**(*options*)

3.9 cmd_options Module

Provides class:*Options*, to assist with command line option parsing and printing of help messages. Developed to be independent of core Python library modules: `argparse` and `optparse`, as the former was only introduced to Python recently, while the latter was deprecated.

class `ssummo.cmd_options.`**`Options`**(*args=None*)

Bases: `object`

Class to parse command line options. Overwrite the class attributes to customise.

`check_groups`()

Called automatically if any regular expression arguments are defined and found when parsing command line options.

This checks to make sure that there are the same number of grouped files as there are input files. If they differ, then this will overwrite files provided to -in with files provided in the groups.

`example` = **None**

Show how the program is used (in programmer lingo).

`expand_stars`()

`help_text` = {'-out': 'Suffix of all the output files. By default, use -in, appending something to the suffix.', '-in': 'Input se

A dictionary containing keys and default arguments for all options.

`multiargs` = []

List containing arguments that can take only one value.

`options` = {'-out': None, '-in': []}

`parse_args`(*args*)

Give the command line options and this will update self.options and return those updated options as a dictionary.

`post_checks` = []

List of regular expressions for creating a dynamic number of groups. By default, groups are defined by -groupN where N is any positive integer. Subsequent arguments are file names that are coverged with -in when parse_args() is called.

`print_help`()

Prints help info for the options defined in self.options

`regargs` = [<_sre.SRE_Pattern object at 0x103030490>]

List containing the arguments that can take multiple values (e.g. can often provide multiple input files).

`regopts` = {}

for internal use.

`singleargs` = []

Give an example of how to use it.

`switches` = {}

A dictionary with same keys as `options`, containing help text for all options.

`useage` = **None**

A dictionary containing help text for all options not requiring commands. All switches start with two dashes '–', and when provided will be changed to have a value of True.

3.10 colours Module

Library for working with colours. If run from the command line, provide a number N, and this shall return N colours (in HEX format) as distant as possible, according to the HSL model.

Example usage:-

python colours.py 5

Description: Library for working with colours. If run from the command line, provide a number N, and this shall return N colours (in HEX format) as distant as possible, according to the HSL model.

class ssummo.colours.**HSLColour**(*angle*, *degrees=360*)

ssummo.colours.**HSL_to_HEX**(*HSL*)

ssummo.colours.**HSL_to_RGB**(*HSL*)

class ssummo.colours.**Heat**(*N=360*, *groups=None*)

BLUE_HUE = (180, 240)

BLUE_LUM = (99, 70.71067811865476)

LAST_HUE = (0, 360)

LAST_LUM = (10, 86.60254037844386)

RED_HUE = (25, -31)

RED_LUM = (50, 10)

YELLOW_HUE = (80, 30)

YELLOW_LUM = (86.60254037844386, 29.289321881345245)

classmethod last_test(*col*, *colour_number*)

classmethod make_HSL_heatmap(*N*, *groups=None*)
Given a number N, will yield that many colours, splitting N into 3 main colour groups.

If optional groups is given, this should be a list indicating the number of shades per colour group.

If N is not divisible by 3, more shades are put into the latter groups.

sin30 = 50

sin45 = 70.71067811865476

sin60 = 86.60254037844386

ssummo.colours.**RGB_to_HEX**(*HSL*)

ssummo.colours.**degrees_within_360**(*degrees*)

ssummo.colours.**draw_heatmap**(*N*, *groups=None*)

ssummo.colours.**draw_vert_heatmap**(*N*, *groups=None*)

ssummo.colours.**generate_HEX_colours**(*N*, *shift=81*, *degrees=360.0*)

ssummo.colours.**generate_HSL_colours**(*N*, *shift=81*, *degrees=360.0*)
Given an integer N, return N unique colours as distinct as possible according to the HSL model

ssummo.colours.**hue_to_RGB**(*m1*, *m2*, *hue*)

3.11 compare_trees Module

Backend module for **comparative_results.py**

exception ssummo.compare_trees.**ArgumentError**
Bases: exceptions.BaseException

class ssummo.compare_trees.**GraphWriter**(*results*, *options*, *methods='default'*)

add_method(*method*, *fn*, **args*, ***kwargs*)
When comparing results, which is a recursive process, there are a number of places with which to attach methods. *method* indicates where in the process to attach the function *fn*. :param method: Allowed values:

- `'pre'` - Methods called before we start recursing the tree. By default, this initialises output files and combines the results dictionaries.
- `'iter'` - A generator function for iterating through results dictionaries. Each iter method needs it's own consumer method too. The generator will thus yield an item that is passed to the given method. This must be defined with the extra kwarg: `meth`
- `'in'` - The default 'meth' is itereator. This calls every method referenced by 'in', in the order specified by self.methods.in
- `'post'` - Closing methods.

fn should be a reference to a function.

`*args` and `**kwargs` are dependent on the given value of `fn`.

close_files()

close_xml(**args*, ***kwargs*)

colour_node(*unique_name*, *node*)

combine_dicts()

default()
This method adds the default method sets to the object model prior to processing the SSuMMo results files.

diff_groups(*unique_name*, *node*)
Counts how many groups contain this particularly node. If there's only one group containing this node, then we store the unique name of the taxa in self.group_uniques['-groupX']

generator()

init_colorstrip(**args*, ***kwargs*)

init_colours(**args*, ***kwargs*)

init_graph(**args*, ***kwargs*)

init_heatmap(**args*, ***kwargs*)

init_xml(**args*, ***kwargs*)

iter_groups(**args*, ***kwargs*)

itereater(*taxpath_node*)

merge_dicts(*results_dicts*)
Like ssummolib.combine_dicts, except that accessions aren't grouped by dataset, but by group.

Parameters **results_dicts** – A sequence of SSuMMo results dictionaries.

Returns The total observed population structure in all the `results_dicts`. At leaves and ambiguous nodes, the dictionary key `'accessions'` contains a list of lists as long as the number of groups, with inner each list containing file names in each group.

print_groups(**args*, ***kwargs*)

run()

unique_taxon(*tax_nodes*)

uniqueify()

unknowns()
Either merges or deletes unknown nodes, assigning the accessions to a single unknown clade, or the parent clade, respectively.

write_colorstrip(*unique_name*, *node*)

write_heatmap(*unique_name*, *node*)

write_xml(*unique_name*, *node*)

ssummo.compare_trees.**ITOLdownload**(*graph_name, itol_uploader, ITOLDownloadOptions={}, formats=['pdf']*)

ssummo.compare_trees.**ITOLupload**(*graph_name*, *heatmap_name*, *ITOLOptionDict*)

class ssummo.compare_trees.**LocalOptions**(*args=None*)
Bases: ssummo.cmd_options.Options

Class to parse command line options.

get_file_name(*tried=None*)

local_checks()

local_parse_args(*args*)
Give the command line options and this will update self.options and return those updated options as a dictionary.

class ssummo.compare_trees.**Population**(***kwargs*)
Bases: object

name

taxa

class ssummo.compare_trees.**Taxon**(***kwargs*)
Bases: object

name

rank

taxid

ssummo.compare_trees.**load_dicts**(*file_names*)

ssummo.compare_trees.**main**(*options*)

3.12 count_hmms Module

If called from a Unix shell, please provide the function name you want to call, and any required arguments, exactly as presented here.

e.g.:

```
python count_hmms.py countseqs /path/to/dir
```

or:

```
python count_hmms.py
```

This will traverse all directories from Archaea, Bacteria and Eukaryota, and will return the total number of hmms that exist and the number that are in the right place.

Description:

- Some functions to look through the SSUMMO directory structure, checking either the HMMs or the sequences found in that directory.

- If a function argument is presented here with a default value, then that argument is optional. Default values are the ones with an equal sign.

e.g. The function 'walkall' has an argument 'delete', which has a default value of False.

Defined functions:- walkall(dir_list = ['Bacteria','Eukaryota','Archaea'] , delete=False) countseqs(pathname, file_name='accessions.txt'): avseqlens(pathname): os_walk(dir_list, delete=False) counthmms(cwd):

Other help??:- pydoc count_hmms.py

ssummo.count_hmms.**avseqlens**(*pathname*)
Given a pathname, find the sequences file, and average the length of all the sequences

ssummo.count_hmms.**counthmms**(*cwd*)
Given a pathname, this function will return the number of HMMs in that directory, and the number of directories in that directory

ssummo.count_hmms.**countseqs**(*pathname*, *file_name='accessions.txt'*)
Given a pathname to a taxonomic id, Return the number of representative 16S sequences in a taxonomies folder directory.

ssummo.count_hmms.**os_walk**(*dir_list*, *delete=False*)
Uses os.walk() and traverses the directories presented in the list 'dir_list' and checks to see if hmms are present in the correct place. If delete is True, then these hmms will be deleted. If delete is False or not specified, then a count of misplaced hmms will be printed & returned.

ssummo.count_hmms.**walkall**(*dir_list=None*, *delete=False*)

3.13 count_seqs Module

class ssummo.count_seqs.**Counter**(*options*)

count()

class ssummo.count_seqs.**MyOptions**(*args*)
Bases: ssummo.cmd_options.Options

3.14 dict_to_html Module

This script creates an html file that represents the directory hierarchy which it searches. Provide a path as first system argument, and this will represent the subdirectory hierarchies as a dynamic web page.

Useage:-

python dict_to_html.py [SSUMMO_output].pkl

Description:

Creates browse-able html page from SSuMMo results dictionary.

note This script requires the file `folder_initiate.html`, which is looked for in the directory

```
CONFIG.top + '/html/'
```

This file is distributed with the SSuMMo package.

For the generated html page to work correctly, the following files need to be found by the web server, as are linked to by the generated html:

- simpletree.css
- simpletreemenu.js

When placed in a web server directory, need to ensure that the following files are linked to properly:-

html/simpletree.css html/simpletreemenu.js

By default, the generated html embeds links to the following web server directories (relative to site root):-

- /styles/simpletree.css
- /js/simpletreemenu.js

If you want or need to change the location of these files, then you'll need to change the links in html/folder_initiate.html (lines 2 & 12)

`ssummo.dict_to_html.`**`close_html`**(*write_handle*)
This function will append the appropriate closing tags to the end of the html file

`ssummo.dict_to_html.`**`cursor`**()

`ssummo.dict_to_html.`**`find_start`**(*resultsDict*, *dbDir=' EDIT ME '*, *tdict='arbDB104.pkl'*)

`ssummo.dict_to_html.`**`get_all_ranks`**(*tdict*, *startDir*, *MySQLcur*, *parent='root'*)
Given the results dict and a MySQLdb cursor instance, returns a list of all unique taxonomic ranks present in the results dictionary

`ssummo.dict_to_html.`**`get_rank`**(*MySQLcur*, *OTU*, *parent*, *table*)

`ssummo.dict_to_html.`**`initiate_html`**(*top*, *write_handle*, *ranks*)
This function initiates an html directory file by appending the appropriate head tags and css information to create a dynamic tree representation from the directory hierarchy

`ssummo.dict_to_html.`**`write_css`**(*handle*, *ranks*, *colours*)

`ssummo.dict_to_html.`**`write_html`**(*startDir*, *html_out*, *taxDict*)

`ssummo.dict_to_html.`**`write_key`**(*write_handle*, *ranks*)

3.15 `dictify` Module

This script looks through a fasta file of SSU... file downloaded from ARB, creating a directory hierarchy that reflects the taxonomic ordering as presented by the ARB data file.

In each of these directories a file ('accessions.txt') is created that contains all the accessions that are descendants of that taxonomic rank.

class `ssummo.dictify.`**`HMMBuilder`**
More of a checker than a builder. Provides methods to check whether or not to rebuild an HMM, and manages Locks, Pipes and Queues controlling their building.

`buildhmm`(*cwd*, *nSeqs*)
Checks an HMM given in cwd, returns True if it should be rebuilt, along with the NCBI taxonomic identifier and it's rank.

`checkhmm`(*cwd*, *hmmName*, *NSeqs*)

`receiveTaxID`(*tax_name_ID*, *OTU_name*)
Receive information from the `TaxDB` process.

Parameters

- **tax_name_ID** – A dictionary sent by pipe from `TaxDB._tax_ID_thread()`.
- **OTU_name** – is the name of the OTU according to the ARB and the SSUMMO database.

Returns The tuple `(tax_ID, rank)`, if `tax_name_ID‘ is is unique and ‘‘OTU_name` matches. If the received tax_id is *o*, then the returned *tax_ID* is the matched OTU name, instead of the taxonomic ID.

`ssummo.dictify.`**`IBuildHmms`**(*options*)

`ssummo.dictify.`**`ICheck`**(*options*)

ssummo.dictify.**IDeleteNonUniques**(*options*)

ssummo.dictify.**IGapBGone**(*options*)

ssummo.dictify.**IIndexSeqs**(*options*)

ssummo.dictify.**IIndexTaxa**(*options*)

ssummo.dictify.**IPressHmms**(*options*)

ssummo.dictify.**IProcessOptions**(*options*)

ssummo.dictify.**IReduceToGenus**(*options*)

ssummo.dictify.**IRemoveAccessions**(*options*)

ssummo.dictify.**IRewrite**(*options*)

This rewrites all ARB sequences to a file in a specified format. This'll also dump a byte-index of the output sequences to the same name as the output file. This causes problems if writing to standard out, as won't have write permission to save to /dev/stdout.pklindex, so will then save byte index to the name of the input file(.pklindex).

ssummo.dictify.**ISplitTaxa**(*options*)

class ssummo.dictify.**MyOptions**(**args*)

Bases: ssummo.cmd_options.Options

print_help()

class ssummo.dictify.**SeqDB**

Bases: multiprocessing.process.Process

Asynchronous process for managing large sequence files released by the ARB Silva database. General workflow should look something like this:

```
arb_db_process = SeqDB()
arb_db_process.start()
```

The order of the following arb_db_process method calls is important, and can be repeated as many times as necessary, before finally calling `finalise()`.

```
arb_db_process.init_seq_file('SSU_Ref_...fas')
accessions = ['NC_123456', ...]
arb_db_process.put_ids(accessions)
seqs = []
for acc in accessions:
    seq = arb_db_process.get_seq()
    seqs.append(seq)
arb_db_process.stop()
```

Once no more sequence files need to be read, call

```
arb_db_process.finalise()
```

finalise(*ids*)

get_seq()

Used by client threads, to retrieve a sequence from the SeqDB process.

init_seq_file(*path*)

Initialise a sequence file for reading. Should be called by client thread immediately after thread is started and before `put_id()` is called.

Internally, (i.e. in the SeqDB subprocess) create a new `ArbIO.ArbIO` instance, on the sequence file, and reads sequences using that.

put_ids(*ids*)

Send accession numbers to the SeqDB server, for asynchronous retrieval from `get_seq()`.

Parameters **ids** – should be a sequence of accession numbers identifiable in the sequence file.

run()

stop(*ids*)
Stop reading from seq_file, waiting for either a new seq file, or `finalise()` to be called.

ssummo.dictify.**buildhmms**(*path*, *in_file*, *tdict*, *TaxDBObj*, *seq_db*, *node='all'*, *threads=None*)
Traverse the (indexed) directory hierarchy and runs hmmalign, hmmbuild in each of the directories from top onwards.

ssummo.dictify.**deleteNonUniques**(*tdict*, *tDB*)

ssummo.dictify.**fasta_to_index_file**(*in_file*)
Reads fasta file into dictionary, outs it to file of directory indexes.

ssummo.dictify.**gapbgone**(*files*, *threshold='100'*)

ssummo.dictify.**getSequences**(*ArbIOobj*, *accessions*, *out_pipe*)
SeqIOIndex - An Index created by IndexDB / SeqIO.index accessions - a list of accessions to be returned in fasta format

ssummo.dictify.**hmm_checker**(*in_file*, *options={'-start': ' EDIT ME '}*)
Walks all directories and prints out the number of sequences.

Parameters

- **in_file** – file handle for taxonomic index.
- **options** – dictionary of configuration options, passed to `find_start()`.

Usage

```
from dictify import hmm_checker
index_handle = file('file_list.txt', 'r')
remaining_seqs, remaining_hmms = hmm_checker(index_handle)
```

This same command can be run directly from the command line with

```
python dictify.py check Bacteria Archaea
```

Note This function relies on the path configured in CONFIG.py `ssummo.CONFIG.arbDBdir`.

ssummo.dictify.**presshmms**(*tdict*)
Traverses tdict, pressing sub-node HMMs into the parent node

ssummo.dictify.**splitTaxa**(*SeqFile*)

ssummo.dictify.**taxonomic_dictionary**(*file_handle=''*, *filetype='dir_index'*)
As input, give a file handle to a fasta file from the ARB. This will return a dictionary of dictionaries representing all the taxonomic identities of all the contained sequences.

If using the pickled dictionary file (taxIndex in CONFIG.py), be sure to open the file in BINARY mode, using the 'b' flag.

e.g.

```
with handle as file(taxIndex, 'rb'):
    tdict = taxonomic_dictionary(handle)
```

3.16 find_taxa Module

Program to find all taxa containing a name of interest. For example, if you want to find all nodes with Mycobacteria in the name, this program will return the path to those nodes on one line, and the ranks of each of those taxa on the next.

Example:- python findTaxa.py -in Mycobacteria Escherichia

ssummo.find_taxa.**main**(*args*)

ssummo.find_taxa.**parse_args**(*args*)

ssummo.find_taxa.**print_matches**(*nameRanks*, *callback=<built-in method write of file object at 0x100724150>*)

Traverse through the tree nameRanks, which should be a nested dictionary of dictionaries, printing the nodes at each level with consistent indentation. Results are written using the given `callback()` function.

3.17 graphs Module

Generic graph plotting functionality.

class ssummo.graphs.**Plotter**

Bases: `object`

Class providing generic graph plotting methods.

artists = []

index_artists = []

index_heights = []

index_labels = []

key = '8'

marker_ind = 0

marker_sizes = {'d': 49, '+': 49, 'o': 9, 's': 49, '8': 49, 'v': 49, 'x': 49, '<': 49}

Available scatter plot markers for matplotlib

markers = ['x', '+', 'o', 'v', 'd', '<', 's', '8']

plot_box(*xvals*, *yvals*, *label=None*, *widths=None*, **args*, ***kwargs*)

Given one or more x-value, will plot all given y-values as a box-and-whisker plot.

This is designed to be called once per dataset, so that each dataset has the same colour.

See http://matplotlib.sourceforge.net/api/pyplot_api.html#matplotlib.pyplot.boxplot for more information

plot_scatter(*xvals*, *yvals*, *marker=None*, *label=None*, **args*, ***kwargs*)

Parameters

- **xvals** – Passed directly to plt.scatter(), so must be compatible with matplotlib.
- **yvals** – Can optionally be a dictionary of vectors, in which case all the keys must be present in `xvals`. If not a dictionary, assume that it is a sequence with same length as xvals.
- **marker** – Can be given a number indexing which of self.markers to use, or a string to pass directly to matplotlib. By default, rotate through markers, using whatever marker is currently selected with: *self.markers[self.marker_ind]*

label can optionally be used to indicate the label in the legend.

Optional `args` and `kwargs` are passed directly to `matplotlib.pyplot.scatter()`

plot_scatter_errors(*xvals*, *yvals*, *mid=None*, *xerr=None*, *yerr=None*, *label=None*, **args*, ***kwargs*)

Plot xvals against yvals with error bars. Uses `self.plot_scatter` to draw the points, but adds error bars.

param mid This is the function which calculates the centre of crosses. By default, use `numpy.mean()`.

param xerr By default, don't draw x error bars. Otherwise, pass a function reference which will return the error values when passed xvals.

param yerr By default, draw 95% Confidence Intervals. i.e. :math:'pm 1.96 * rac{(STD. dev.}{sqrt{replicates}}'

x_max = 0

y_max = 0

`ssummo.graphs.`**make_patch_spines_invisible**(*ax*)

`ssummo.graphs.`**make_spine_invisible**(*ax*, *direction*)

3.18 link_EMBL_taxonomy Module

Populates a MySQL database with a combination of ARB's and NCBI's taxonomy databases.

class `ssummo.link_EMBL_taxonomy.`**ISqlDatabase**

Bases: `object`

Abstract class providing database connectivity hooks for sublcasses to implement.

connect(*user*, *host*, *passwd*, *db*)

Connect to MySQL database, creating `self.cnx` and `self.cur`

db = **None**

MySQL database name

delete_tables()

Drop all tables from database. Use with care!!!

host = **None**

MySQL server host name

passwd = **None**

MySQL password

show_tables()

Populate `tables` from the SQL command:

```
(mysql) SHOW TABLES
```

user = **None**

MySQL user name

class `ssummo.link_EMBL_taxonomy.`**NcbiTaxDb**

Bases: `ssummo.link_EMBL_taxonomy.ISqlDatabase`

Class used to retrieve the latest EMBL taxonomy listing from the NCBI ftp repository and save it to a local mysql database.

_write_to_mysql may need to be edited to match the database security info that is in use on your system.

General useage: x = NCBI_tax_to_SQL() x.automate()

By default this will use the database metagenomics (will crash if the database doesn't exist) and then create the table 'nodes' within that database.

automate(*db_name='taxonomies'*)

iter_lines(*fname*)

class ssummo.link_EMBL_taxonomy.**Translate**
Bases: ssummo.link_EMBL_taxonomy.ISqlDatabase

automate()

check_exist(*OTU*, *parent*, *table*)

create_output_table()
Creates a MySQL table called EMBL_Vs_ARB and a text file called EMBL_Vs_ARB.txt. These are empty but can be written to by calling:- self.cur.execute(...) or self.out_handle.write(...) Columns are in the order: OTU Name*, EMBL Count, ARB Count, Rank, Depth*, Max Depth The *'s indicate compound primary keys.

find_parent(*childIDs*, *parent*, *child*)
Given a list of ID numbers and the name of the parent this will find the ID with the same parent (from IDs) and will return the correct ID and the rank.

get_Ids(*OTU*, *parent*)

get_rank(*ID*)

rows_to_list(*rows*)
Given a single column of results from mysql.cur.fetchall() convert to a single list

rows_to_set(*rows*)

walk(*dict_to_walk*, *parent='root'*)
Provide a dictionary, and this will traverse it top down, yielding each OTU, the depth it is found and the maximum depth from that point.

3.19 phyloxml Module

This is the backend library for the SSuMMo script dict_to_phyloxml.py, which can create phyloxml formatted trees from SSuMMo results files.

It's contains the functions necessary to make phyloxml, so is also used by comparative_results and SSUMMO.

Description: Library module for creating phyloxml formatted trees from SSuMMo results dictionaries

ssummo.phyloxml.**colour_node**(*path_list*, *colour_indexes*, *out_handle*, *colours*)
Colours each node by the 2nd level from root, by creating an ITOL-compatible colour index file for uploading to colour the leaves.

ssummo.phyloxml.**exec_dict_to_phyloxml**(*argv=None*)
Main entry point for dict_to_phyloxml

ssummo.phyloxml.**make_colours**(*results_dict*, *taxDict*, *colour_rank='phylum'*)
Results_dict is the results we're gonna be turning into phyloxml. taxDict is the complete ARB taxonomy dictionary. colour_rank is the rank at which you want to colour.

ssummo.phyloxml.**number_of_leaves**(*node*)
Given a taxonomy node, count the number of assigned sequences to it, from that point in the tree.

ssummo.phyloxml.**write_xml**(*top*, *tree_out*, *results_dict*, *tax_dict*, *colour_rank='phylum'*)
Generate PhyloXml output from a SSuMMo results dictionary

Parameters

- **top** – Path string to top of tree.
- **tree_out** – File handle to which results are written.
- **results_dict** – SSuMMo results dictionary.
- **tax_dict** – Global taxIndex.
- **colour_rank** – Rank at which to highlight branches (default='phylum').

3.20 pressGenus Module

Script to find all the genuses present in the ARB database, get all of their HMMs, and save all of those HMMs to a single file. Once all those HMMs are in a single file, they're going to be pressed by hmmbuild.

class ssummo.pressGenus.**FindHmmPath**(*tdict*, *inQ*, *outQ*)
Bases: threading.Thread

This thread continually loops through the dictionary looking for path names that match the OTU names piped over. When the OTU name & parent name match, the path to that HMM is passed back to the main thread.

run()

ssummo.pressGenus.**getGenuses**(*domain*)

ssummo.pressGenus.**main**()

ssummo.pressGenus.**pressHMM**(*inName*)

3.21 resample Module

Library module for code shared between rankAbundance, rarefactionCurve and others. Contains code for selecting random subsamples from full datasets.

class ssummo.resample.**Indices**(*out*, *collapse_rank=None*)
Number crunching class. Methods calculate biodiversity statistics for SSuMMo results files.

SPlusJackknife(*counts*, *total*)
Implementation of JackKnife formula from: http://www.jstatsoft.org/v15/i03/paper

Smith, CD. Pontius, JS. Jackknife Estimator of Species Richness with S-PLUS, 2006, 15 (3)

$$J_n(S) = S_o + \frac{(n-1)}{n} \cdot \sum_{i=1}^{n} r_i$$

calc_all()
Automates all calculations so that we can print the output.

jack = **None**
Jackknife value

jackknife(*counts*, *total*)

load(*results_dictionary*, *combined=False*, *name=None*)
Load the dictionary. If the option '-collapse-at-rank' is given, then we shall also collapse the dictionary at the specificied rank. Results dictionary is not returned, but is stored as:- self.dictionary

name = **None**
Dataset name

out = **None**
Output file

print_header()

shannon = **None**
Shannon value

shannon_index(*probs*)
Give the probabilities as returned from tally() and total species count. :returns: The tuple $(H', H_{max},)$

simpson = **None**
Simpson index

simpsons_index(*counts*, *total*)
Simpson's index of Diversity. Returns a value from 0 - 1, where 0 shows no diversity, and 1 shows infinite diversity.

tally(*combined=False*)
Writes rank abundance information to `self.out`. This is represented as just a single column, showing the number of sequences allocated to each taxon.

To use this, first define self.dictionary, preferably with `Indices.load()`

The function returns 3 items:-

1. counts-a sorted list, containing the number of sequences allocated to each taxa, probabilities and total number of taxa.
2. probs - a list of floating point numbers, showing the percentage of sequences assigned to each taxon.
3. total - the total number of sequences found in the dataset.

These 3 items are needed to calculate the other indices...

class ssummo.resample.**RandomNumberThread**(*Q*, *N*, *replicates*, *increment*)
Bases: `threading.Thread`

run()

class ssummo.resample.**Replicate**
Store statistics on a single dataset. Supports addition and division.

nseqs = 0

rarefaction = 0.0
number of sequences sampled

shannon = 0.0

shannon_max = 0.0
Shannon H' value

simpson = 0.0
Shannon H_{max} value

ssummo.resample.**Results**
Object for storing multiple `Replicate` instance statistics.

3.22 seqDB Module

Command line tool to extract, filter or convert biological sequences.

Usage:

```
$ python seqDB.py -db infile.sto -format stockholm -accs acc1 acc2 \
                  -desc foo bar
```

This will read the Stockholm formatted sequence file *infile.sto*, and print out any sequences whose accession or description matches those given.

Description:

Program to reformat or filter sequences in a variety of formats. See http://www.biopython.org/wiki/SeqIO for a list of formats, although there are currently more supported by biopython. e.g. sff format (Roche 454's binary sequence format) is supported by biopython, but not mentioned on the webpage.

class ssummo.seqDB.**Options**

help_text = (('-db', 'Sequences input file (Default: stdin)'), ('-accs', 'List of accessions to look for within <-db>. This is r

options = {'-outformat': 'fasta', '-desc': [], '-accs': [], '-format': 'fasta', '-db': None, '–reverse': False, '-out': <open file '<st

parse_args(*args*)

print_help()

class ssummo.seqDB.**Reader**(*inhandle*, *descriptions=None*, *accessions=None*, *maxlen=None*, *minlen=None*)

Bases: object

Class for reading sequences of any format and filtering them, based on various criteria.

filter(*test*)
Filter sequence by length. Test should be a callable object which returns True or False, indicating whether or not to return the sequence.

search_acc(*seq*)
Returns True if a give sequence's accession matches one of those given to initialiser.

search_desc(*seq*)
Returns True if a give sequence's descriptions matches one of those given to initialiser.

sequences(*format='fasta'*)
Generator method, yielding sequences which match this instance's filtering criteria.

class ssummo.seqDB.**Writer**(*reverse=False*)

Bases: object

static reverser(*seq*, *form*)

static writer(*seq*, *form*)

ssummo.seqDB.**main**(*options*)
SeqDB main loop. Just give an Options instance and this function will do the rest.

3.23 seqLen Module

class ssummo.seqLen.**Lengths**

getLens(*inhandle*, *format=None*)

getMean()

getSTD()

nBases()

ssummo.seqLen.**bases**(*nBases*)

ssummo.seqLen.**get_headers**()

ssummo.seqLen.**get_stats**(*file_name*, *lengths=None*)

ssummo.seqLen.**guess_format**(*file_name*)
Given a file name, will extract the file suffix, and try and guess the file type from that. Returns a format that can be accepted as an argument to SeqIO.parse(....)

ssummo.seqLen.**iterfiles**(*file_names*, *format=None*)

ssummo.seqLen.**main**(*file_names*, *format=None*)

ssummo.seqLen.**update_formats**(*formats=None*)

3.24 simulate_lengths Module

class ssummo.simulate_lengths.**Options**(*args=None*)
Class to parse command line options.

calc_defaults()

parse_args(*args*)
Give the command line options and this will update self.options and return those updated options as a dictionary.

print_help()
Prints help info for the options defined in self.options

class ssummo.simulate_lengths.**Writer**

replSub = <_sre.SRE_Pattern object at 0x105455818>

classmethod reverser(*seqRecord*, *length*, *start=0*)
N.B. This doesn't take the reverse complement. It returns a sequence slice from the opposite end.

classmethod writer(*seqRecord*, *length*, *start=0*)

ssummo.simulate_lengths.**callSSUMMO**(*inName*, *SSUMMO_Options*)

ssummo.simulate_lengths.**callTallyDict**(*inName*)

ssummo.simulate_lengths.**getMinMeanMaxSeqLengths**(*in_file_name*, *seq_file_format*)
This searches through the given file name and returns a tuple with the minimum and maximum sequnce lengths in that file

ssummo.simulate_lengths.**getRegion**(*inFileName*, *length*, *Vregion*, *kingdom='archaea'*)

ssummo.simulate_lengths.**hmmsearch_to_domtbl**(*seq_file_name*, *path*)
Given the a sequence file name and a path to an HMM, this will run hmmsearch on them and return a pipe to the domain table.

ssummo.simulate_lengths.**parse_hmmsearch_domtbl**(*pipe*)
Given a pipe to hmmsearch domtbl results, will return the results lines in a list of lists.

ssummo.simulate_lengths.**rewriteSeqsAtLength**(*options*, *length*, *seq_file_format='fasta'*, *start=0*, *min_length=10*)
Given an Options instance, a maximum sequence length, this will rewrite those sequences to *options['-out']*. Optionally, provide a start location. Otherwise it'll cut the sequence from the start up until the given length (if the sequence is that long)

3.25 ssummolib Module

General library functions commonly needed for SSUMMO modules. Not to be called from command line.

class ssummo.ssummolib.**ITOLCGI**(*out_file=None*, *colour_file=None*)

ITOLdownload(*formats=['pdf']*)

ITOLupload()

add_dataset(*dataset_file*, *options='dataset_options'*)

downloadOptions = {'rangesCover': 'leaves', 'showInternalLabels': 1, 'showInternalIDs': 0, 'fontSize': 28, 'lineWidth': 1

class ssummo.ssummolib.**SsummoOptions**(*args=None*)
Bases: ssummo.cmd_options.Options

local_parse_args(*args*)

ssummo.ssummolib.**collapse_at_rank**(*results_dict*, *collapse_at_rank*, *TaxDbObj=None*, *tax_dict=None*, *combined=False*, *in_files=None*)

Collapses a results dictionary at desired rank.

Parameters

- **results_dict** – load (using `pickle.load()`) a SSUMMO results file (.pkl file).
- **collapse_at_rank** – The desired rank to collapse as the value.
- **TaxDbObj** – Instance of `TaxDB`. Created if None.
- **tax_dict** – The full taxonomic dictionary index (usually referred to by taxIndex in CONFIG.py).
- **combined** – Have results files already been combined?
- **in_files** – If combined, should be a list. Used to count number of datasets.

ssummo.ssummolib.**combine_dicts**(*results_dicts*)

Give a list of SSUMMO results dictionaries. This shall return a dictionary containing each & every node from all of those dictionaries.

Where accessions are found assigned to a node, this will combine the accessions from all results_dicts into a list of lists; one list of accessions per results dictionary, in the same order as are passed to this function.

ssummo.ssummolib.**find_start_node**(*resultsDict*, *taxDict*)

Locates the directory where to enter the SSUMMO loop. Default is to start in arbDBdir, which is configured in CONFIG.py. To change, give the command option '-start /some/path/to/dir'

ssummo.ssummolib.**get_combined_accessions**(*startNode*, *accessions*)

ssummo.ssummolib.**load_index**(*silent=False*)

ssummo.ssummolib.**reduceToGenus**(*tdict*, *TaxDB*)

class ssummo.ssummolib.**seqDB**(*seqfile*, *prefetchQ*, *outQ*, *pipes*, *format='fasta'*)

Bases: `multiprocessing.process.Process`

run()

3.26 taxonomy Module

Helper module for traversing and querying the ARB and NCBI taxonomy databases.

class ssummo.taxonomy.**TaxDB**

create_thread(*in_queue*, *out_pipe*)

Create and return a `threading.Thread` instance, meant for running `self._tax_ID_thread()` in a separate thread.

Sa `TaxDB._tax_ID_thread()`

fetch_tax_id(*table*, *OTUName*, *parentName*, *Queue=None*)

get_ranks(*table='Prokaryotes'*)

ssummo.taxonomy.**fetch_rank**(*TaxDBObj*, *table*, *OTUName*, *parent_name=None*)

Checks MySQL database for taxon with name OTUName, and parent parent_name. If a unique rank is found, return that, otherwise return False.

ssummo.taxonomy.**find_taxa**(*tdict*, *options*)

ssummo.taxonomy.**get_ranks**(*taxPathList*, *TaxDBObj*)

Give a list of all taxonomies you've got, and a TaxDB object. This will return a list of all the ranks, mirroring the taxPathList given, and also the table name.

ssummo.taxonomy.**match_ranks**(*taxa*)

ssummo.taxonomy.**walk_ranks**(*tdict*, *path=''*)

3.27 traverse Module

Library module for functions and classes related to traversing SSUMMO results dictionaries, stored as a nested dictionary of dictionaries. These data types are loadable directly from from the .pkl files, saved by SSuMMo.

ssummo.traverse.**dict_walk**(*top*, *taxdict*, *topdown=True*, *random=True*)
Generator function to recursively traverse a nested, hierarchical dictionary of taxonomic nodes. Yields tuples of (path name, node) for each node encounted in the tree.

Parameters

- **top** – Path to prepend to each to each node.
- **taxdict** – Tree to traverse.
- **topdown** – If False, start by yielding leaf nodes and work backwards up the tree.
- **random** – If False, sort the node names before yielding results.

ssummo.traverse.**find_max_depth**(*node*, *depth=0*, *max_depth=0*, *deepest=[]*)
Given a node, finds the deepest node and returns it's depth and all nodes of thatdepth

ssummo.traverse.**find_node_name**(*target_name*, *dict_node*)
Searches the given tree (`dict_node`) for a node with the name `target_name`.

Returns the full path to the given node.

ssummo.traverse.**find_start**(*tdict*, *options*)
Locates the directory where to enter the SSUMMO loop. Default is to start in arbDBdir, which is configured in CONFIG.py. To change, give the command option '-start /some/path/to/dir'

ssummo.traverse.**get_accessions**(*start_node*, *accessions=None*)
Find and return all accessions from a given start_node.

ssummo.traverse.**my_walk**(*top*, *taxdict*, *topdown=True*, *followlinks=False*, *random=False*)
Traverses the dictionary taxdict and yields the full path appended to top, and a list of child nodes within the path yielded.

CHAPTER

FOUR

SSUMMO EXECUTABLES

4.1 ACGTCounts.py

Command line utility to calculate a position specific scoring matrix for any set of sequences.

Writes tab-delimited table with residue letters in the header line and in each column the probability of observing each of those residues calculated from the sequence file.

Example:- $ python bin/ACGTCounts.py [insequencefile].fas > PSM.txt or $ python ACGTCounts.py [insequence-file].fas -out PSM.txt

Description: Calculates position specific scoring matrix from as sequence file.

ACGTCounts.**parse_args**(*args*)

ACGTCounts.**pos_counts**(*inFile*, *fileFormat='fasta'*)

ACGTCounts.**tabularise**(*PSMDict*, *nseqs*, *out_file*, *residues_seen=[]*)

4.2 SSUMMO.py

class SSUMMO.**Application**(*options=None*)

Print(*string*)

Write(*string*)

annotate()

call()

call_iter()

enter_loop()
Given the name of a sequence file, makes a blast database and provides the entry point for the SSUMMO algorithm. Returns a dictionary of results containing taxonomic nodes with matches and at each node the accessions that have been assigned there.

find_start()
If the -start option was given, then this shuffles the results along accordingly and also calls SSuMMo on a single node, whereas at the root, need to iterate through all 3 domains.

This therefore sets self.start_node and self.start_dir, indicating which node we should start SSuMMo.

reverser()

separate()

shuffle_along(*results_dict*, *start_dir*, *start_node*)

shuffle_back(*results*, *start_dir*)

shutdown()

SSUMMO.**SSUMMO**(*node*, *path*, *results_node*, *seq_db*, *result_pipe*, *diverge_dict={}*)
Main SSuMMo function loop. Recurses through the nested dictionary *node*, searching each sequence loaded in seq_db against each HMM built for taxa specified by the tree *node*.

Parameters

- **node** – Nested dictionary representing current node in tree of life.
- **path** – Absolute or relative file path to current node's HMM directory.
- **results_node** – Where results are stored [in/out].
- **seq_db** – SeqDB instance containing all query sequences.
- **result_pipe** – Used for communication with Scorer instance.
- **diverge_dict** – For internal use.

class SSUMMO.**Scorer**(*hmmsearch_outQ*, *result_pipe*, *distrib_end_queue*, *seq_DbQ*)
Bases: threading.Thread

Let's put (through inQ) results here as they're produced by parseHMMThread. Scorer will continue to collate and order results as they're produced. As soon as all hmmsearch results for a sequence are ready, then they're piped to score.

n_done = 0

n_to_do = 0

run()

class SSUMMO.**SeqDB**(*seqfile*, *prefetchQ*, *distribute_Q*, *distributor_end_queue*, *threads*, *lock*, *reverse_pipe=None*, *format='fasta'*)
Bases: threading.Thread

annotate(*outfile*)
SeqDB knows nothing of the taxonomic assignments, just accessions and sequences. We therefore read tuple pairs of:

(<accession>, <assignment>)

through self.prefetchQ, until the tuple (False, False) is read.

When calling annotate (by sending the string 'annotate' through self.prefetchQ), also give the name of the output sequence file. Sequences are saved in the same format as they were read.

Once all assigned sequences have been saved, the other sequences which were not assigned to a taxa are appended to the input file. If this is undesired, use separate instead.

die(*e*)

forget(*accessions*)
Deletes all provided accessions from memory.

forget_inverse(*accessions*)

get(*accessions*)
Given a list of accessions, puts each one into self.outQ.

get_all(**args*)
Puts all sequences, in fasta format, into self.outQ.

load_sequences()

proceed(**args*)
Put 'Proceed' into outQ, for SequenceDistributor.

reverse()
Called once before traversing the directory. The input pipe (self.rpipe) is closed before returning None.

run()

separate(*outfile*)
Works similarly to SeqDB.annotate. Reads (accession, annotation) tuple pairs through self.prefetchQ, and saves them to outfile. Typically, this saves all sequences that could be assigned to a taxa to <outfile>.

shutdown()
Closes down all threads.

skip(**args*)
Put 'Skip' into outQ, for SequenceDistributor.

slice(*accession*, *start*, *end*)
This will slice sequence with seq.id accession, from start to end, INCLUSIVE. i.e. using python indexing [start -1 : end + 1]. This was chosen to comply with hmmer alignment co-ordinates.

slice_all()
Keeps reading self.prefetchQ, expecting the tuple (accession, start, end). For each tuple received, this will slice the relevant sequence accordingly. This will continue to expect tuples until it gets "END" from self.prefetchQ.

N.B. Sequences are sliced inclusive of given position.

class SSUMMO.**SequenceDistributor**(*seq_DbQ*, *hmmsearch_threads*, *jobEndQ*, *SeqDBInQ*)
Bases: threading.Thread

distribute()

get_free_thread()

run()

SSUMMO.**clean_converged**(*accession*, *diverge_dict_item*)

SSUMMO.**final_merge**(*resultsDict*, *DivDict*)

SSUMMO.**find_rel_path**(*node*, *OTUname*, *rel_path=[]*)
Given a dictionary node, and an OTU name, will search for OTUname from the top of node and return the relative path, delimited by '/'.

SSUMMO.**find_start**(*tdict*)
Locates the directory where to enter the SSUMMO loop. Default is to start in arbDBdir, which is configured in CONFIG.py. To change, give the command option '-start /some/path/to/dir'

SSUMMO.**get_tax_index**(*silent=False*)
Returns the full ARB taxonomy as a nested dictionary, but only for the main three kingdoms: Archaea, Bacteria and Eukaryota.

SSUMMO.**main**(*options*)

SSUMMO.**merge_diverge**(*resultsDict*, *DivDict*, *tax_node*)
Given the results_node dictionary (resultsDict), add accessions present in DivDict to the appropriate nodes in resultsDict.

SSUMMO.**print_ambiguous_results**(*amb_nodes*)

SSUMMO.**print_tree_line**(*choice*, *results_node*, *depth*)

SSUMMO.**save**(*save_name*, *object*)

SSUMMO.**score**(*scorer_pipe*, *tax_node*, *path*, *results_dict=None*, *diverge_dict=None*)
Called once per iteration. Therefore needs to receive the node names once at the start. Then it should start receiving lots of hmmmsearchThread results.

SSUMMO.**sep_diverge**(*resultsDict*, *DivDict*, *node_keys*, *seq_db*)
Given the results dictionary and the diverge dictionary, check the scores present in DivDict, looking for a single highest score. If there is a single highest score, place that in the appropriate location and delete it from DivDict. Otherwise, don't change anything.

4.3 comparative_results.py

Provide a set of SSUMMO results files (.pkl format) and this will combine the results into a single phyloxml formatted tree and an ITOL compatible text file that is used to display multi-valued bar charts showing the abundance of each member at each observed rank.

Useage:- comparative_results.py [options] results1.pkl results2.pkl [results3.pkl] [...]

4.4 dict_to_phyloxml.py

This script creates a phyloxml formatted tree from a ssummo results file. It's most useful because it contains the functions in order to make that phyloxml, so is called by comparative_results.py and SSUMMO.py.

Description: Creates phyloxml formatted tree from a SSuMMo results dictionary. Simple as that.

Useage:- dict_to_phyloxml.py /path/to/SSuMMoresults.pkl

4.5 plot_data.py

class plot_data.**LocalOptions**(*args=None*)
Bases: ssummo.cmd_options.Options

integers()

reqs()

plot_data.**join_results**(*results*, *taxpath*, *species*)

plot_data.**main**(*options*)

plot_data.**separate**(*options*)

4.6 rankAbundance.py

rankAbundance.py

Description:-

Give one or more pickle files as input and this shall output a white space delimited table showing calculated biodiversity information. This includes Shannon, Simpson & JackKnife Indices, which are indices which can be interpreted as representations of species evenness, richness, and sampling bias, repsectively.

Some useful options included with this program:-

-collapse-at-rank <rank> :- Collapse the result tree at rank <rank>.

-groupX [group1files] [...] [-group2 ... [...]] :-

- used to combine and average diversity indices over

multiple results files.

class rankAbundance.**LocalOptions**(*args*)
Bases: ssummo.cmd_options.Options

rankAbundance.**main**(*options*)
Wrapper function to print everything! If you want to use it, just initiate a copy of LocalOptions, edit it's namespace variables as necessary (i.e. the options), and pass to this function.

e.g.

```
>>> input_files = ['one.pkl', 'two.pkl']
>>> myoptions = LocalOptions(input_files)
>>> myoptions['-collapse-at-rank'] = 'genus'
>>> main( myoptions )
```

4.7 rarefactionCurve.py

class rarefactionCurve.**Loader**(*results_dict*, *options*, *queue=None*, *semaphore=None*)
Bases: multiprocessing.process.Process

We have one Loader instance for each SSuMMo results file. When run() is called, (usually with Loader().start() to run as a separate process), a Results instance will be created and returned via queue, if it exists, or just returned otherwise.

rarefy()
Given a results dictionary, this will sample random accessions from that dictionary and will _yield_ the number of accessions sampled as well as the number of genera / species / OTUs that a random sample of that number of accessions finds.

Therefore, this is an interator generator function, yielding the tuple (nsampled, found_nodes) for each sample replicate.

run()
Loads the results in results_dict and populates a dictionary, with x-values as the keys, and y-values as the values. This allows multiple y-values to be stored per x-value. Optional queue will be used to send the results instead of returning them

class rarefactionCurve.**LocalOptions**(*args=None*)
Bases: ssummo.cmd_options.Options

groupify()
If there are any groups defined on the command line, this'll just create a list contianing the lengths of each group.

indices()
If you specify –indices, append all.

class rarefactionCurve.**PlotRarefaction**(*options*)
Bases: ssummo.graphs.Plotter

draw()
Just provide the x-axis and y-axis limits. The x-axis is rounded up to the nearest 100, and the y-axis is rounded up to the nearest 500.

draw_indices(*index_artists*, *index_labels*)

init_axes()
Here define methods to initialise additional axes. This is called during initialisation if options['–indices'] is True.

load_rarefaction(*data*)

reinit()
To be called if need to reload the colours. This needs doing with –draw, when PlotRarefaction is initiated before the previous results have been loaded (i.e. before the original options have been loaded).

resume(*results*, *loc=4*)

Parameters

- **results** – Should be a dictionary of Results instances. If you want them presented in a specific order, then give the keys of the results dictionary in the order which you want them displayed.

- **loc** – The location of the legend according to matplotlib co-ordinates.

rarefactionCurve.**calc_increment**(*n_sequences*, *steps=None*, *max_fn=<built-in function max>*)

Calculates a decent (x-axis) increment. Will by default figure out the increment widths for 20 steps, for a number of sequences determined by `max_fn()`.

Parameters

- **n_sequences** – Should be a sequence containing the number of sequences for each set of results being plotted.
- **steps** – Number of increments desired for the graph (default=20)
- **max_fn** – A callback function to be called with `n_sequences`. The returned value will be divided by *steps* to calculate the increment.

e.g.

```
>>> import numpy as np
>>> # lambda function to calculate 95% confidence limits.
>>> my_maximum = lambda x : np.average(x) +  (1.96 * np.std(x) )
>>> # generate some random data
>>> n_seqs = [np.random.random(10) for i in xrange(10)]
>>> increment = calc_increment(n_seqs, steps=40, max_fn=my_maximum)
```

rarefactionCurve.**get_accession_places**(*node*, *accessions={}*, *path='/'*)

rarefactionCurve.**get_save_name**()

Asks a user for a file name. If it exists, ask again, if user types nothing, ask again... Return None if user types 'n' or 'no'.

rarefactionCurve.**main**(*myoptions*)

rarefactionCurve.**truncate_name**(*file_name*)

a

ACGTCounts, 29

c

comparative_results, 32

d

dict_to_phyloxml, 32

p

plot_data, 32

r

rankAbundance, 32
rarefactionCurve, 33

s

SSUMMO, 29
ssummo.__init__, 7
ssummo.acc_finder, 10
ssummo.alignDB, 10
ssummo.ArbIO, 7
ssummo.cmd_options, 11
ssummo.colours, 11
ssummo.compare_trees, 12
ssummo.CONFIG, 8
ssummo.count_hmms, 14
ssummo.count_seqs, 15
ssummo.dict_to_html, 15
ssummo.dictify, 16
ssummo.find_taxa, 18
ssummo.graphs, 19
ssummo.HMMcompare, 9
ssummo.link_EMBL_taxonomy, 20
ssummo.Phylogeny, 9
ssummo.phyloxml, 21
ssummo.pressGenus, 22
ssummo.resample, 22
ssummo.seqDB, 23
ssummo.seqLen, 24
ssummo.simulate_lengths, 25
ssummo.ssummolib, 25
ssummo.taxonomy, 26
ssummo.traverse, 27

A

ACGTCounts (module), 29
add_dataset() (ssummo.ssummolib.ITOLCGI method), 25
add_method() (ssummo.compare_trees.GraphWriter method), 12
AlignSeq (class in ssummo.HMMcompare), 9
annotate() (SSUMMO.Application method), 29
annotate() (SSUMMO.SeqDB method), 30
Application (class in SSUMMO), 29
ArbIO (class in ssummo.ArbIO), 7
ArbSeqRecord (class in ssummo.ArbIO), 8
arbSeqToStr() (ssummo.ArbIO.ArbIO method), 7
ArgumentError, 12
artists (ssummo.graphs.Plotter attribute), 19
automate() (ssummo.link_EMBL_taxonomy.NcbiTaxDb method), 20
automate() (ssummo.link_EMBL_taxonomy.Translate method), 21
avseqlens() (in module ssummo.count_hmms), 15

B

bases() (in module ssummo.seqLen), 24
BLUE_HUE (ssummo.colours.Heat attribute), 12
BLUE_LUM (ssummo.colours.Heat attribute), 12
buildhmm() (ssummo.dictify.HMMBuilder method), 16
buildhmms() (in module ssummo.dictify), 18

C

calc_all() (ssummo.resample.Indices method), 22
calc_defaults() (ssummo.simulate_lengths.Options method), 25
calc_increment() (in module rarefactionCurve), 34
CalcProb (class in ssummo.HMMcompare), 9
call() (SSUMMO.Application method), 29
call_iter() (SSUMMO.Application method), 29
callSSUMMO() (in module ssummo.simulate_lengths), 25
callTallyDict() (in module ssummo.simulate_lengths), 25
check_exist() (ssummo.link_EMBL_taxonomy.Translate method), 21
check_groups() (ssummo.cmd_options.Options method), 11
checkhmm() (ssummo.dictify.HMMBuilder method), 16
clean_converged() (in module SSUMMO), 31
close() (ssummo.ArbIO.ArbIO method), 7
close_files() (ssummo.compare_trees.GraphWriter method), 13
close_html() (in module ssummo.dict_to_html), 16
close_xml() (ssummo.compare_trees.GraphWriter method), 13
collapse_at_rank() (in module ssummo.ssummolib), 25
colour_node() (in module ssummo.phyloxml), 21
colour_node() (ssummo.compare_trees.GraphWriter method), 13
ColumnError, 9
combine_dicts() (in module ssummo.Phylogeny), 9
combine_dicts() (in module ssummo.ssummolib), 26
combine_dicts() (ssummo.compare_trees.GraphWriter method), 13
comparative_results (module), 32
connect() (ssummo.link_EMBL_taxonomy.ISqlDatabase method), 20
count() (ssummo.count_seqs.Counter method), 15
Counter (class in ssummo.count_seqs), 15
counthmms() (in module ssummo.count_hmms), 15
countseqs() (in module ssummo.count_hmms), 15
create_output_table() (ssummo.link_EMBL_taxonomy.Translate method), 21
create_thread() (ssummo.taxonomy.TaxDB method), 26
cursor() (in module ssummo.dict_to_html), 16

D

db (ssummo.link_EMBL_taxonomy.ISqlDatabase attribute), 20
default() (ssummo.compare_trees.GraphWriter method), 13
degrees_within_360() (in module ssummo.colours), 12
delete_tables() (ssummo.link_EMBL_taxonomy.ISqlDatabase method), 20
deleteNonUniques() (in module ssummo.dictify), 18
dict_to_phyloxml (module), 32
dict_walk() (in module ssummo.traverse), 27
die() (SSUMMO.SeqDB method), 30
diff_groups() (ssummo.compare_trees.GraphWriter method), 13
distribute() (SSUMMO.SequenceDistributor method), 31
DotsException, 8
downloadOptions (ssummo.ssummolib.ITOLCGI attribute), 25
draw() (rarefactionCurve.PlotRarefaction method), 33
draw_heatmap() (in module ssummo.colours), 12

draw_indices() (rarefactionCurve.PlotRarefaction method), 33
draw_vert_heatmap() (in module ssummo.colours), 12
dumpAndIndex() (ssummo.ArbIO.ArbIO method), 8

E

enter_loop() (SSUMMO.Application method), 29
example (ssummo.cmd_options.Options attribute), 11
exec_dict_to_phyloxml() (in module ssummo.phyloxml), 21
expand_stars() (ssummo.cmd_options.Options method), 11

F

fasta_to_index_file() (in module ssummo.dictify), 18
FastaIterator() (ssummo.ArbIO.ArbIO method), 7
fetch() (ssummo.ArbIO.ArbIO method), 8
fetch_rank() (in module ssummo.taxonomy), 26
fetch_tax_id() (ssummo.taxonomy.TaxDB method), 26
filter() (ssummo.seqDB.Reader method), 24
final_merge() (in module SSUMMO), 31
finalise() (ssummo.dictify.SeqDB method), 17
find_max_depth() (in module ssummo.traverse), 27
find_node_name() (in module ssummo.traverse), 27
find_parent() (ssummo.link_EMBL_taxonomy.Translate method), 21
find_rel_path() (in module SSUMMO), 31
find_seq_file() (ssummo.acc_finder.Finder method), 10
find_start() (in module SSUMMO), 31
find_start() (in module ssummo.dict_to_html), 16
find_start() (in module ssummo.traverse), 27
find_start() (SSUMMO.Application method), 29
find_start_node() (in module ssummo.ssummolib), 26
find_taxa() (in module ssummo.taxonomy), 26
Finder (class in ssummo.acc_finder), 10
FindHmmPath (class in ssummo.pressGenus), 22
forget() (SSUMMO.SeqDB method), 30
forget_inverse() (SSUMMO.SeqDB method), 30

G

gapbgone() (in module ssummo.dictify), 18
generate_HEX_colours() (in module ssummo.colours), 12
generate_HSL_colours() (in module ssummo.colours), 12
generator() (ssummo.compare_trees.GraphWriter method), 13
get() (SSUMMO.SeqDB method), 30
get_accession_places() (in module rarefactionCurve), 34
get_accessions() (in module ssummo.traverse), 27
get_all() (SSUMMO.SeqDB method), 30
get_all_ranks() (in module ssummo.dict_to_html), 16
get_combined_accessions() (in module ssummo.ssummolib), 26
get_consensus() (in module ssummo.HMMcompare), 9
get_file_name() (ssummo.compare_trees.LocalOptions method), 14
get_format() (ssummo.acc_finder.Finder method), 10
get_free_thread() (SSUMMO.SequenceDistributor method), 31
get_headers() (in module ssummo.seqLen), 24
get_Ids() (ssummo.link_EMBL_taxonomy.Translate method), 21
get_rank() (in module ssummo.dict_to_html), 16
get_rank() (ssummo.link_EMBL_taxonomy.Translate method), 21
get_ranks() (in module ssummo.taxonomy), 26
get_ranks() (ssummo.taxonomy.TaxDB method), 26
get_save_name() (in module rarefactionCurve), 34
get_seq() (ssummo.dictify.SeqDB method), 17
get_sequences() (ssummo.acc_finder.Finder method), 10
get_stats() (in module ssummo.seqLen), 24
get_tax_index() (in module SSUMMO), 31
getGenuses() (in module ssummo.pressGenus), 22
getLens() (ssummo.seqLen.Lengths method), 24
getMean() (ssummo.seqLen.Lengths method), 24
getMinMeanMaxSeqLengths() (in module ssummo.simulate_lengths), 25
getRegion() (in module ssummo.simulate_lengths), 25
getSequences() (in module ssummo.dictify), 18
getSTD() (ssummo.seqLen.Lengths method), 24
GraphWriter (class in ssummo.compare_trees), 12
groupify() (rarefactionCurve.LocalOptions method), 33
guess_format() (in module ssummo.seqLen), 24

H

Heat (class in ssummo.colours), 12
help_text (ssummo.alignDB.LocalOptions attribute), 10
help_text (ssummo.cmd_options.Options attribute), 11
help_text (ssummo.HMMcompare.LocalOptions attribute), 9
help_text (ssummo.seqDB.Options attribute), 23
hmm_checker() (in module ssummo.dictify), 18
HMMBuilder (class in ssummo.dictify), 16
hmmsearch_to_domtbl() (in module ssummo.simulate_lengths), 25
host (ssummo.link_EMBL_taxonomy.ISqlDatabase attribute), 20
HSL_to_HEX() (in module ssummo.colours), 12
HSL_to_RGB() (in module ssummo.colours), 12
HSLColour (class in ssummo.colours), 12
hue_to_RGB() (in module ssummo.colours), 12

I

IBuildHmms() (in module ssummo.dictify), 16
ICheck() (in module ssummo.dictify), 16
IDeleteNonUniques() (in module ssummo.dictify), 16
IGapBGone() (in module ssummo.dictify), 17
IIndexSeqs() (in module ssummo.dictify), 17
IIndexTaxa() (in module ssummo.dictify), 17
index() (ssummo.ArbIO.ArbIO method), 8
index_artists (ssummo.graphs.Plotter attribute), 19
index_heights (ssummo.graphs.Plotter attribute), 19
index_labels (ssummo.graphs.Plotter attribute), 19
indexAndInfo() (ssummo.ArbIO.ArbIO method), 8
Indices (class in ssummo.resample), 22

indices() (rarefactionCurve.LocalOptions method), 33
init_axes() (rarefactionCurve.PlotRarefaction method), 33
init_colorstrip() (ssummo.compare_trees.GraphWriter method), 13
init_colours() (ssummo.compare_trees.GraphWriter method), 13
init_graph() (ssummo.compare_trees.GraphWriter method), 13
init_heatmap() (ssummo.compare_trees.GraphWriter method), 13
init_seq_file() (ssummo.dictify.SeqDB method), 17
init_xml() (ssummo.compare_trees.GraphWriter method), 13
initiate_html() (in module ssummo.dict_to_html), 16
integers() (plot_data.LocalOptions method), 32
IPressHmms() (in module ssummo.dictify), 17
IProcessOptions() (in module ssummo.dictify), 17
IReduceToGenus() (in module ssummo.dictify), 17
IRemoveAccessions() (in module ssummo.dictify), 17
IRewrite() (in module ssummo.dictify), 17
ISplitTaxa() (in module ssummo.dictify), 17
ISqlDatabase (class in ssummo.link_EMBL_taxonomy), 20
iter_groups() (ssummo.compare_trees.GraphWriter method), 13
iter_lines() (ssummo.link_EMBL_taxonomy.NcbiTaxDb method), 20
itereater() (ssummo.compare_trees.GraphWriter method), 13
iterfiles() (in module ssummo.seqLen), 24
ITOLCGI (class in ssummo.ssummolib), 25
ITOLdownload() (in module ssummo.compare_trees), 14
ITOLdownload() (ssummo.ssummolib.ITOLCGI method), 25
ITOLupload() (in module ssummo.compare_trees), 14
ITOLupload() (ssummo.ssummolib.ITOLCGI method), 25

J

jack (ssummo.resample.Indices attribute), 22
jackknife() (ssummo.resample.Indices method), 22
join_results() (in module plot_data), 32

K

key (ssummo.graphs.Plotter attribute), 19

L

LAST_HUE (ssummo.colours.Heat attribute), 12
LAST_LUM (ssummo.colours.Heat attribute), 12
last_test() (ssummo.colours.Heat class method), 12
Lengths (class in ssummo.seqLen), 24
load() (ssummo.resample.Indices method), 22
load_dicts() (in module ssummo.compare_trees), 14
load_index() (in module ssummo.ssummolib), 26
load_rarefaction() (rarefactionCurve.PlotRarefaction method), 33
load_sequences() (SSUMMO.SeqDB method), 30
Loader (class in rarefactionCurve), 33
local_checks() (ssummo.compare_trees.LocalOptions method), 14
local_parse_args() (ssummo.compare_trees.LocalOptions method), 14
local_parse_args() (ssummo.ssummolib.SsummoOptions method), 25
LocalOptions (class in plot_data), 32
LocalOptions (class in rankAbundance), 32
LocalOptions (class in rarefactionCurve), 33
LocalOptions (class in ssummo.alignDB), 10
LocalOptions (class in ssummo.compare_trees), 14
LocalOptions (class in ssummo.HMMcompare), 9

M

main() (in module plot_data), 32
main() (in module rankAbundance), 32
main() (in module rarefactionCurve), 34
main() (in module SSUMMO), 31
main() (in module ssummo.ArbIO), 8
main() (in module ssummo.compare_trees), 14
main() (in module ssummo.find_taxa), 19
main() (in module ssummo.pressGenus), 22
main() (in module ssummo.seqDB), 24
main() (in module ssummo.seqLen), 24
make_colours() (in module ssummo.phyloxml), 21
make_HSL_heatmap() (ssummo.colours.Heat class method), 12
make_patch_spines_invisible() (in module ssummo.graphs), 20
make_spine_invisible() (in module ssummo.graphs), 20
marker_ind (ssummo.graphs.Plotter attribute), 19
marker_sizes (ssummo.graphs.Plotter attribute), 19
markers (ssummo.graphs.Plotter attribute), 19
match_ranks() (in module ssummo.taxonomy), 26
merge_dicts() (in module ssummo.Phylogeny), 9
merge_dicts() (ssummo.compare_trees.GraphWriter method), 13
merge_diverge() (in module SSUMMO), 31
multiargs (ssummo.cmd_options.Options attribute), 11
my_walk() (in module ssummo.traverse), 27
MyOptions (class in ssummo.count_seqs), 15
MyOptions (class in ssummo.dictify), 17

N

n_done (SSUMMO.Scorer attribute), 30
n_to_do (SSUMMO.Scorer attribute), 30
name (ssummo.compare_trees.Population attribute), 14
name (ssummo.compare_trees.Taxon attribute), 14
name (ssummo.resample.Indices attribute), 22
nBases() (ssummo.seqLen.Lengths method), 24
NcbiTaxDb (class in ssummo.link_EMBL_taxonomy), 20
nseqs (ssummo.resample.Replicate attribute), 23
number_of_leaves() (in module ssummo.phyloxml), 21

O

Options (class in ssummo.cmd_options), 11

Options (class in ssummo.seqDB), 23
Options (class in ssummo.simulate_lengths), 25
options (ssummo.alignDB.LocalOptions attribute), 10
options (ssummo.cmd_options.Options attribute), 11
options (ssummo.HMMcompare.LocalOptions attribute), 9
options (ssummo.seqDB.Options attribute), 23
os_walk() (in module ssummo.count_hmms), 15
out (ssummo.resample.Indices attribute), 22

P

parse() (ssummo.ArbIO.ArbIO method), 8
parse_args() (in module ACGTCounts), 29
parse_args() (in module ssummo.find_taxa), 19
parse_args() (ssummo.alignDB.LocalOptions method), 10
parse_args() (ssummo.cmd_options.Options method), 11
parse_args() (ssummo.HMMcompare.LocalOptions method), 9
parse_args() (ssummo.seqDB.Options method), 24
parse_args() (ssummo.simulate_lengths.Options method), 25
parse_hmmsearch_domtbl() (in module ssummo.simulate_lengths), 25
parse_input() (in module ssummo.alignDB), 10
parse_seqs() (in module ssummo.HMMcompare), 9
parseHeader() (ssummo.ArbIO.ArbIO method), 8
parseHMM() (in module ssummo.HMMcompare), 9
passwd (ssummo.link_EMBL_taxonomy.ISqlDatabase attribute), 20
pipeSequences() (ssummo.ArbIO.ArbIO method), 8
plot_box() (ssummo.graphs.Plotter method), 19
plot_data (module), 32
plot_scatter() (ssummo.graphs.Plotter method), 19
plot_scatter_errors() (ssummo.graphs.Plotter method), 19
PlotRarefaction (class in rarefactionCurve), 33
Plotter (class in ssummo.graphs), 19
Population (class in ssummo.compare_trees), 14
pos_counts() (in module ACGTCounts), 29
post_checks (ssummo.cmd_options.Options attribute), 11
pressHMM() (in module ssummo.pressGenus), 22
presshmms() (in module ssummo.dictify), 18
Print() (SSUMMO.Application method), 29
print_ambiguous_results() (in module SSUMMO), 31
print_groups() (ssummo.compare_trees.GraphWriter method), 13
print_header() (ssummo.resample.Indices method), 22
print_help() (ssummo.alignDB.LocalOptions method), 10
print_help() (ssummo.cmd_options.Options method), 11
print_help() (ssummo.dictify.MyOptions method), 17
print_help() (ssummo.HMMcompare.LocalOptions method), 9
print_help() (ssummo.seqDB.Options method), 24
print_help() (ssummo.simulate_lengths.Options method), 25
print_HMMs_vs_seq() (in module ssummo.HMMcompare), 9
print_matches() (in module ssummo.find_taxa), 19
print_seqs_vs_HMM() (in module ssummo.HMMcompare), 9
print_tree_line() (in module SSUMMO), 31
proceed() (SSUMMO.SeqDB method), 30
put_ids() (ssummo.dictify.SeqDB method), 17

R

RandomNumberThread (class in ssummo.resample), 23
rank (ssummo.compare_trees.Taxon attribute), 14
rankAbundance (module), 32
rarefaction (ssummo.resample.Replicate attribute), 23
rarefactionCurve (module), 33
rarefy() (rarefactionCurve.Loader method), 33
Reader (class in ssummo.alignDB), 10
Reader (class in ssummo.seqDB), 24
receiveTaxID() (ssummo.dictify.HMMBuilder method), 16
RED_HUE (ssummo.colours.Heat attribute), 12
RED_LUM (ssummo.colours.Heat attribute), 12
reduceToGenus() (in module ssummo.ssummolib), 26
regargs (ssummo.cmd_options.Options attribute), 11
regopts (ssummo.cmd_options.Options attribute), 11
regreplace() (ssummo.ArbIO.ArbIO method), 8
reinit() (rarefactionCurve.PlotRarefaction method), 33
Replicate (class in ssummo.resample), 23
replSub (ssummo.simulate_lengths.Writer attribute), 25
reqs() (plot_data.LocalOptions method), 32
rereplace() (ssummo.ArbIO.ArbIO method), 8
Results (in module ssummo.resample), 23
resume() (rarefactionCurve.PlotRarefaction method), 33
reverse() (SSUMMO.SeqDB method), 30
reverser() (ssummo.alignDB.Reader method), 10
reverser() (SSUMMO.Application method), 29
reverser() (ssummo.seqDB.Writer static method), 24
reverser() (ssummo.simulate_lengths.Writer class method), 25
rewriteSeqsAtLength() (in module ssummo.simulate_lengths), 25
RGB_to_HEX() (in module ssummo.colours), 12
rows_to_list() (ssummo.link_EMBL_taxonomy.Translate method), 21
rows_to_set() (ssummo.link_EMBL_taxonomy.Translate method), 21
run() (rarefactionCurve.Loader method), 33
run() (ssummo.compare_trees.GraphWriter method), 13
run() (ssummo.dictify.SeqDB method), 18
run() (ssummo.HMMcompare.AlignSeq method), 9
run() (ssummo.HMMcompare.CalcProb method), 9
run() (ssummo.HMMcompare.SumProbs method), 9
run() (ssummo.pressGenus.FindHmmPath method), 22
run() (ssummo.resample.RandomNumberThread method), 23
run() (SSUMMO.Scorer method), 30
run() (SSUMMO.SeqDB method), 30
run() (SSUMMO.SequenceDistributor method), 31

run() (ssummo.ssummolib.seqDB method), 26

S

save() (in module SSUMMO), 31
score() (in module SSUMMO), 31
score_alignment() (in module ssummo.HMMcompare), 9
Scorer (class in SSUMMO), 30
search() (ssummo.acc_finder.Finder method), 10
search_acc() (ssummo.seqDB.Reader method), 24
search_desc() (ssummo.seqDB.Reader method), 24
sep_diverge() (in module SSUMMO), 31
separate() (in module plot_data), 32
separate() (SSUMMO.Application method), 29
separate() (SSUMMO.SeqDB method), 31
SeqDB (class in SSUMMO), 30
SeqDB (class in ssummo.dictify), 17
seqDB (class in ssummo.ssummolib), 26
SequenceDistributor (class in SSUMMO), 31
sequences() (ssummo.seqDB.Reader method), 24
shannon (ssummo.resample.Indices attribute), 22
shannon (ssummo.resample.Replicate attribute), 23
shannon_index() (ssummo.resample.Indices method), 22
shannon_max (ssummo.resample.Replicate attribute), 23
show_tables() (ssummo.link_EMBL_taxonomy.ISqlDatabase method), 20
shuffle_along() (SSUMMO.Application method), 29
shuffle_back() (SSUMMO.Application method), 29
shutdown() (SSUMMO.Application method), 29
shutdown() (SSUMMO.SeqDB method), 31
simpson (ssummo.resample.Indices attribute), 22
simpson (ssummo.resample.Replicate attribute), 23
simpsons_index() (ssummo.resample.Indices method), 22
sin30 (ssummo.colours.Heat attribute), 12
sin45 (ssummo.colours.Heat attribute), 12
sin60 (ssummo.colours.Heat attribute), 12
singleargs (ssummo.cmd_options.Options attribute), 11
skip() (SSUMMO.SeqDB method), 31
slice() (SSUMMO.SeqDB method), 31
slice_all() (SSUMMO.SeqDB method), 31
splitTaxa() (in module ssummo.dictify), 18
SPlusJackknife() (ssummo.resample.Indices method), 22
SSUMMO (module), 29
SSUMMO() (in module SSUMMO), 30
ssummo.__init__ (module), 7
ssummo.acc_finder (module), 10
ssummo.alignDB (module), 10
ssummo.ArbIO (module), 7
ssummo.cmd_options (module), 11
ssummo.colours (module), 11
ssummo.compare_trees (module), 12
ssummo.CONFIG (module), 8
ssummo.count_hmms (module), 14
ssummo.count_seqs (module), 15
ssummo.dict_to_html (module), 15
ssummo.dictify (module), 16
ssummo.find_taxa (module), 18
ssummo.graphs (module), 19
ssummo.HMMcompare (module), 9
ssummo.link_EMBL_taxonomy (module), 20
ssummo.Phylogeny (module), 9
ssummo.phyloxml (module), 21
ssummo.pressGenus (module), 22
ssummo.resample (module), 22
ssummo.seqDB (module), 23
ssummo.seqLen (module), 24
ssummo.simulate_lengths (module), 25
ssummo.ssummolib (module), 25
ssummo.taxonomy (module), 26
ssummo.traverse (module), 27
SsummoOptions (class in ssummo.ssummolib), 25
stop() (ssummo.dictify.SeqDB method), 18
storeindex() (ssummo.ArbIO.ArbSeqRecord method), 8
SumProbs (class in ssummo.HMMcompare), 9
switches (ssummo.cmd_options.Options attribute), 11

T

tabularise() (in module ACGTCounts), 29
tally() (ssummo.resample.Indices method), 23
taxa (ssummo.compare_trees.Population attribute), 14
TaxDB (class in ssummo.taxonomy), 26
taxid (ssummo.compare_trees.Taxon attribute), 14
Taxon (class in ssummo.compare_trees), 14
taxonomic_dictionary() (in module ssummo.dictify), 18
taxonomify() (ssummo.ArbIO.ArbIO method), 8
tell() (ssummo.ArbIO.ArbSeqRecord method), 8
Translate (class in ssummo.link_EMBL_taxonomy), 20
truncate_name() (in module rarefactionCurve), 34

U

unique_taxon() (ssummo.compare_trees.GraphWriter method), 13
uniqueify() (ssummo.compare_trees.GraphWriter method), 13
unknowns() (ssummo.compare_trees.GraphWriter method), 13
update_formats() (in module ssummo.seqLen), 24
useage (ssummo.cmd_options.Options attribute), 11
user (ssummo.link_EMBL_taxonomy.ISqlDatabase attribute), 20

W

walk() (ssummo.link_EMBL_taxonomy.Translate method), 21
walk_ranks() (in module ssummo.taxonomy), 27
walkall() (in module ssummo.count_hmms), 15
Write() (SSUMMO.Application method), 29
write_colorstrip() (ssummo.compare_trees.GraphWriter method), 14
write_css() (in module ssummo.dict_to_html), 16
write_heatmap() (ssummo.compare_trees.GraphWriter method), 14
write_html() (in module ssummo.dict_to_html), 16
write_key() (in module ssummo.dict_to_html), 16
write_xml() (in module ssummo.phyloxml), 21

write_xml() (ssummo.compare_trees.GraphWriter method), 14
Writer (class in ssummo.seqDB), 24
Writer (class in ssummo.simulate_lengths), 25
writer() (ssummo.alignDB.Reader method), 10
writer() (ssummo.seqDB.Writer static method), 24
writer() (ssummo.simulate_lengths.Writer class method), 25

X

x_max (ssummo.graphs.Plotter attribute), 20

Y

y_max (ssummo.graphs.Plotter attribute), 20
YELLOW_HUE (ssummo.colours.Heat attribute), 12
YELLOW_LUM (ssummo.colours.Heat attribute), 12