Metagenomic framework Documentation

Release 0.4.1

Francesco Rubino

Nov 08, 2019

Contents

Contents:

CHAPTER 1

Introduction

The aim of this library[8] is to provide a series of useful modules and packages to make it easier to build custom pipelines for metagenomics or any kind of bioinformatics analysis. It integrates other well known python libraries in bioinformatics, like HTSeq[1], pysam[2], numpy[3] and scipy[4].

A tutorial pipeline is provided in the Documentation[5].

A discussion mailing list is available at mgkit-users[6].

1.1 Citing

Rubino, F. and Creevey, C.J. 2014. MGkit: Metagenomic Framework For The Study Of Microbial Communities. . Available at: figshare[7] [doi:10.6084/m9.figshare.1269288].‘

a citation is also available using the `mgkit.cite()` function or using the *–cite* option on all scripts included. For example:

```
blast2gff --cite
```

1.2 Links

[8] https://github.com/frubino/mgkit
[1] http://www-huber.embl.de/users/anders/HTSeq/
[2] https://code.google.com/p/pysam/
[3] http://www.numpy.org
[4] http://www.scipy.org
[5] https://mgkit.readthedocs.io/en/latest/
[6] https://groups.google.com/forum/#!forum/mgkit-users
[7] http://figshare.com/articles/MGkit_Metagenomic_Framework_For_The_Study_Of_Microbial_Communities/1269288

CHAPTER 2

Installation

The library is both in PyPI and Bioconda[9]. Any new version is first released on PyPI and then ported to the Bioconda repository. Installation is easier from Bioconda, since some dependencies (specifically pysam[10]) can have requirements that are difficult to resolve.

Warning: The preferred version of Python to use is >=3.5 as this is the one I'm using MGKit with and Python 2.7 will not be mantained anymore starting with 1st January 2020. From MGKit 0.3.4 support for Python 3 was added and from 0.4.x you can expect Python 2.7 supoort to be gradually removed.

2.1 Using Conda

Note: this was tested in conda version 4.5.4 and Python 3.6.5

Installing in a conda environment is relatively straight forward, and I recommend installing mgkit in its own environment:

```
$ conda create --name mgkit mgkit
```

This will install *mgkit* in a virtual environment called *mgkit*. Conda will write on screen the way to activate the virtual environment, which may different according to your installation.

2.2 Docker Instance (with Jupyter Notebook)

A preconfigured Docker instance (user: mgkit, no password) has been configured at Docker Hub[11], including more packages for testing, available at Docker Hub (frubino/mgkit), with more instruction on its use available there. The version of MGKit targeted is the last development branch, but can be customised using the files available at github[12], specifically in the *bootstrap.sh* file.

[9] https://bioconda.github.io
[10] https://github.com/pysam-developers/pysam
[11] https://hub.docker.com/r/frubino/mgkit/
[12] https://github.com/frubino/mgkit-docker-repo

2.3 Pip and pipenv

pipenv[13] is the environment I use for developing and test, *conda* should be preferred for end users.

As of version 0.4.0, only python >=3.5 is used. The library has been ported to Python 3 (tested under version 3.5 and laters), but a layer of compatibility with Python 2.7 is used (the **future** package). Forward, version 0.3.4 is the last one that targets primarly Python 2.7 (but is partially compatible with Python 3). To test the version of Python installed use:

```
$ python --version
```

Warning: when installing pysam from source, the compilation of its libraries need system packages installed. Read the INSTALL[14] file to help you install the library

Assuming you have installed the required packages for *pysam* and other dependencies, *pipenv* can be used:

```
$ pipenv install mgkit
```

and the environment activated with:

```
$ pipenv shell
```

Note: *enum34* is installed with Python version < 3.4

To recreate the *.c* files for the *mgkit.io.utils.sequences*, set the environment variable USE_CYTHON:

```
$ export USE_CYTHON=TRUE
```

and then:

```
$ python setup.py build
```

2.3.1 Using the repository

The source code can also be obtained from the Bitbucket repository[15].

2.4 Running Tests

Tests are performed with tox[16], which you can use to run tests on specific versions of the interpreter. You can look at *tox.ini* in the distribution for the version tested.

The tests requires the *pytest* package and some plugins (*pytest-datadir* and *pytest-console-scripts*):

```
$ pip install pytest pytest-datadir pytest-console-scripts
```

You can run the tests with:

```
$ python setup.py test
```

Some test won't be run if the required library/data is not found. Consult the output for more information.

[13] https://pipenv.readthedocs.io/
[14] https://github.com/pysam-developers/pysam/blob/master/htslib/INSTALL
[15] https://bitbucket.org/setsuna80/mgkit
[16] https://tox.readthedocs.io/en/latest/

2.5 Building Documentation

The requirements are detailed in *docs_req.txt* of the repository. Other libraries:

- latex (for pdf output - *make latexpdf*)
- pandoc

2.6 Troubleshooting (for older versions)

Some of the dependencies require available compilers to finish the installation. At the mimimum a system that provides the full GNU compiler suite, including a fortran compiler is required to install those dependencies by source.

If a compilation error is raised during installation, it's adviced to install each dependency manually. I'll try to keep this section updated, but there's not that many OS that I can keep working on (mostly MacOSX and GNU/Linux).

2.6.1 Older versions of MacOSX

Version 10.19 of MacOSX comes with Python 2.7 installed. To install every dependency from source, however it's needed to install the *Xcode* app from the **App Store** which install the compilers, with the exception of *gfortran*. Another solution is using Homebrew[17] or Macports[18], to install the compilers needed.

If you want to use Xcode, you need to install the gfortran compiler, with the package provided here[19]. This should be enough to install most packages from source.

> **Warning:** There seems to be a problem with *pandas* version 0.13.1 on MacOSX, with a segmentation fault happening when using DataFrames. The 0.14.1 version is the one tested.

Note: if there's a problem building a python package because of a compile error, dealing with an unknown command line option, use:

```
export ARCHFLAGS=-Wno-error=unused-command-line-argument-hard-error-in-future
```

It's related to the clang toolchain included with Xcode

Matplotlib

The tricky package to install in MacOSX is actually matplotlib[20], with one of many solutions being posted on a disccusion on stackoverflow[21]. In our case, installing *freetype2* and *libpng* through Homebrew it's the less painful:

```
$ brew install libpng freetype2
```

Note: If you get a compilation error which refers to freetype2 in the */opt/X11/* I found it easy to delete XQuartz installing matplotlib and then reinstall XQuartz.

Or use:

[17] http://brew.sh
[18] http://www.macports.org
[19] http://gcc.gnu.org/wiki/GFortranBinariesMacOS
[20] http://matplotlib.org
[21] http://stackoverflow.com/questions/4092994/unable-to-install-matplotlib-on-mac-os-x

```
export PKG_CONFIG_PATH=/usr/local/Cellar/freetype/2.6_1/lib/pkgconfig/:/usr/local/
↪Cellar/libpng/1.6.19/lib/pkgconfig/
```

Note that the versions may be different.

2.6.2 Installing Scipy from source on Linux

A full description on how to install the scipy on Linux from source can be found at this address[22], be aware that the compilation of the *math-atlas* and *lapack* libraries takes a long time.

Installation in a virtual environment:

```
# create virtual environment, if needed, otherwise activate the one desired
virtualenv venv
source venv/bin/activate
# create temporary directory to compile math-atlas and lapack
mkdir dep-build; cd dep-build
wget http://www.netlib.org/lapack/lapack.tgz
wget http://sourceforge.net/projects/math-atlas/files/Stable/3.10.2/atlas3.10.2.
↪tar.bz2/download
tar xfvj download
cd ATLAS
mkdir build; cd build
../configure -Fa alg -fPIC --with-netlib-lapack-tarfile=../../lapack.tgz --prefix=
↪$VIRTUAL_ENV
make
cd lib; make shared; make ptshared; cd ..
make install
```

This will compile math-atlas with full lapack support in the virtual environment; change the *–prefix=$VIRTUAL_ENV* to *–prefix=$HOME* if you want to install the dependencies in you home directory.

2.7 Notes

Not all packages are required to use the part of the library, but it's recommended to install all of them. Requirements are bound to change, but pandas, scipy, numpy, pysam and matplotlib are the bases of the library.

To avoid problems with the system installation, I suggest using the excellent virtualenv[23]. This will avoid problems with installing packages system-wide and breaking a working installation.

[22] http://www.scipy.org/scipylib/building/linux.html
[23] http://www.virtualenv.org/

CHAPTER 3

Metagenomic Pipeline

This section detailed information about example pipelines made using the framework Changed in version 0.3.4: updates

3.1 Tutorial

The aim of this tutorial is to show how to build a pipeline to analyse metagenomic samples. Moreover, the SNPs calling part was made to show how diversity estimates can be calculated from metagenomic data, hence it should be changed to be more strict.

We're going to use Peru Margin Subseafloor Biosphere[25] as an example, which can be download from the ENA website.

This tutorial is expected to run on a UNIX (Linux/MacOSX/Solaris), with the *bash* shell running, because of some of the loops (not tested with other shells).

Note: We assume that all scripts/commands are run in the same directory.

Warning: It is advised to run the tutorial on a cluster/server: the memory requirements for the programs used are quite high (external to the library).

3.1.1 Initial setup

We will assume that the pipeline and it's relative packages are already installed on the system where the tutorial is run, either through a system-wide install or a virtual environment (advised). The details are in the *Installation* section of the documentation.

Also for the rest of the tutorial we assume that the following software are installed and accessible system-wide:

- Velvet[26]

[25] https://www.ebi.ac.uk/metagenomics/project/SRP000183
[26] https://www.ebi.ac.uk/~zerbino/velvet/

- Bowtie 2[27]
- samtools and bcftools 1.8[28]
- Picard Tools[29][34]
- GATK[30][35]
- BLAST[31] or RAPSearch2[32]

3.1.2 Getting Sequence Data

The data is stored on the EBI ftp as well, and can be downloaded with the following command (on Linux):

```
$ wget ftp://ftp.sra.ebi.ac.uk/vol1/fastq/SRR001/SRR001326/SRR001326.fastq.gz
$ wget ftp://ftp.sra.ebi.ac.uk/vol1/fastq/SRR001/SRR001325/SRR001325.fastq.gz
$ wget ftp://ftp.sra.ebi.ac.uk/vol1/fastq/SRR001/SRR001323/SRR001323.fastq.gz
$ wget ftp://ftp.sra.ebi.ac.uk/vol1/fastq/SRR001/SRR001322/SRR001322.fastq.gz
```

on MacOSX you can replace *wget* with *curl -O*.

And then uncompress with:

```
$ gunzip *.fastq.gz
```

Taxonomy Data

We only need the taxonomy for an optional part of the gene prediction for the analysis. It can be downloaded using the command:

```
$ download-taxonomy.sh
```

The data will be saved in the file *taxonomy.pickle* to which we'll refer from now on. More information can be found in *Download Taxonomy*

3.1.3 Metagenome Assembly

We're going to use velvet to assemble the metagenomics sample, using the following commands in *bash*:

```
$ velveth velvet_work 31 -fmtAuto  *.fastq
$ velvetg velvet_work -min_contig_lgth 50
```

The contigs are in the *velvet_work/contigs.fa* file. We want to take out some of the information in each sequence header, to make it easier to identify them. We decided to keep only *NODE_#*, where # is a unique number in the file (e.g. from *>NODE_27_length_157_cov_703.121033* we keep only *>NODE_27*). We used this command in bash:

```
$ cat velvet_work/contigs.fa | sed -E 's/(>NODE_[0-9]+)_.+/\1/g' > assembly.fa
```

Alternatively, *fasta-utils uid* can be used to avoid problems with spaces in the headers:

[27] http://bowtie-bio.sourceforge.net/bowtie2/
[28] http://samtools.sourceforge.net
[29] http://picard.sourceforge.net
[34] Picard Tools needs to be found in the directory picard-tools in the same directory as this tutorial.
[30] http://www.broadinstitute.org/gatk/
[35] GATK directory is expected to be called *GATK* and inside the tutorial directory. It also needs java v1.7.x in newer versions.
[31] http://www.ncbi.nlm.nih.gov/books/NBK279690/
[32] http://omics.informatics.indiana.edu/mg/RAPSearch2/

```
$ fasta-utils uid cat velvet_work/contigs.fa assembly.fa
```

3.1.4 Gene Prediction

Gene prediction can be done with any software that supports the tab format that BLAST outputs. Besides BLAST, RAPSearch can be used as well.

Before that a suitable DB must be downloaded. In this tutorial we'll use the SwissProt portion of *Uniprot <http://www.uniprot.org>* that can be downloaded using the following commands:

```
$ wget ftp://ftp.uniprot.org/pub/databases/uniprot/current_release/knowledgebase/
↪complete/uniprot_sprot.fasta.gz
$ gunzip uniprot_sprot.fasta.gz
```

Using BLAST

BLAST needs the DB to be indexed using the following command:

```
$ makeblastdb -dbtype prot -in uniprot_sprot.fasta
```

After which BLAST can be run:

```
$ blastx -query assembly.fasta -db uniprot_sprot.fasta -out \
    assembly.uniprot.tab -outfmt 6
```

Using RAPSearch

RAPSearch is faster than BLAST, while giving similar results. As with BLAST, there is a command to be executed before it can predict genes:

```
$ prerapsearch -d uniprot_sprot.fasta -n uniprot_sprot
```

After this command is complete its execution, RAPSearch can be started:

```
$ rapsearch -q assembly.fa -d uniprot_sprot -o assembly.uniprot.tab
```

RAPSearch will produce two files, *assembly.uniprot.tab.m8* and *assembly.uniprot.tab.aln*. *assembly.uniprot.tab.m8* is the file in the correct format, so we can rename it and remove the other one:

```
$ rm assembly.uniprot.tab.aln
$ mv assembly.uniprot.tab.m8 assembly.uniprot.tab
```

3.1.5 Create the GFF

After BLAST or RAPSearch are finished, we can convert all predictions to a GFF file:

```
$ blast2gff uniprot -b 40 -db UNIPROT-SP -dbq 10 assembly.uniprot.tab \
    assembly.uniprot.gff
```

And then, because the number of annotations is high, we filter them to reduce the number of overlapping annotations:

```
$ filter-gff overlap assembly.uniprot.gff assembly.uniprot-filt.gff
```

This will result in a smaller file. Both script supports piping, so they can be used together, for example to save a compressed file:

```
$ blast2gff uniprot -b 40 -db UNIPROT-SP -dbq 10 assembly.uniprot.tab | \
    filter-gff overlap - assembly.uniprot-filt.gff
```

Finally, rename the filtered GFF file:

```
$ mv assembly.uniprot-filt.gff assembly.uniprot.gff
```

Warning: filter-gff may require a lot of memory, so it's recommended to read its documentation for strategies on lowering the memory requirements for big datasets (a small script to sort a GFF is included *sort-gff.sh*)

Taxonomic Refinement

This section is optional, as taxonomic identifiers are assigned using Uniprot, but it can result in better identification. It requires the the *nt* database from NCBI to be found on the system, in the *ncbi-db* directory.

if you don't have the *nt* database installed, it can be downloaded (> 80GB uncompressed, about 30 compressed) with this command (you'll need to install *ncftpget*):

```
$ mkdir ncbi-db
$ cd ncbi-db
$ ncftpget ftp://ftp.ncbi.nlm.nih.gov/blast/db/nt*.gz
$ tar xfvz *.tar.gz
$ cd ..
```

To do it, first the nucleotide sequences must be extracted and then use blastn against the *nt* database:

```
$ get-gff-info sequence -f assembly.fa assembly.uniprot.gff \
    assembly.uniprot.frag.fasta
$ blastn -query assembly.uniprot.frag.fasta -db ncbi-db/nt -out \
    assembly.uniprot.frag.tab -outfmt 6
```

After BLAST completes, we need to download supporting file to associate the results with the taxonomic information:

```
$ download-ncbi-taxa.sh
$ gunzip -c ncbi-nucl-taxa.gz | taxon-utils to_hdf -n nt
```

We now need to run the *taxon-utils* (*taxon-utils - Taxonomy Utilities*) script to find the LCA for each annotation. BLAST will output too many matches, so we want to also filter this file first, with *filter-gff*. First we convert into GFF the BLAST tab file, then use *filter-gff* to pick only the 95% quantile of hit length out of all hits and finally filter to get the 95% of identities. Finally run *taxon-utils* to get the LCA table:

```
$ blast2gff blastdb -i 3 -r assembly.uniprot.frag.tab | \
    filter-gff sequence -t -a length -f quantile -l 0.95 -c gt | \
    filter-gff sequence -t -a identity -f quantile -l 0.95 -c gt | \
    add-gff-info addtaxa -f taxa-table.hf5:nt | \
    taxon-utils lca -b 40 -t taxonomy.pickle -s -p - lca.tab
```

What we do is convert the BLAST results into a GFF file, removing the version information from the accession. Then filter the GFF keeping only the annotation which are in the top 5% of indentity scores, but also use only annotations that have a bitscore of 40 and write the result as a 2 columns table.

We can now run the script to add the taxonomic information to the GFF file, with:

```
$ add-gff-info addtaxa -v -t lca.tab -a seq_id -db NCBI-NT \
    assembly.uniprot.gff assembly.uniprot-taxa.gff
```

after it completes, it is safe to rename the output GFF:

```
$ mv assembly.uniprot-taxa.gff assembly.uniprot.gff
```

Complete GFF

To add the remaining information, mapping to KO[33] and others, including the taxonomic information, a script is provided that downloads this information into a GFF file:

```
$ add-gff-info uniprot --buffer 500 -t -e -ec -ko \
    assembly.uniprot.gff assembly.uniprot-final.gff
```

After which we can rename the GFF file:

```
$ mv assembly.uniprot-final.gff assembly.uniprot.gff
```

3.1.6 Alignment

The alignment of all reads to the assembly we'll be made with *bowtie2*. The first step is to build the index for the reference (out assembly) with the following command:

```
$ bowtie2-build assembly.fa assembly.fa
```

and subsequently start the alignment, using bowtie2 and piping the output SAM file to *samtools* to convert it into BAM files with this command:

```
for file in *.fastq; do
    BASENAME=`basename $file .fastq`
    bowtie2 -N 1 -x assembly.fa -U $file \
    --very-sensitive-local \
    --rg-id $BASENAME --rg PL:454 --rg PU:454 \
    --rg SM:$BASENAME | samtools view -Sb - > $BASENAME.bam;
done
```

We'll have BAM files which we need to sort and index:

```
for file in *.bam; do
    samtools sort -o `basename $file .bam`-sort.bam $file;
    mv `basename $file .bam`-sort.bam $file
    samtools index $file;
done
```

3.1.7 Coverage and SNP Info

The coverage information is added to the GFF and needs to be added for later SNP analysis, including information about the expected number of synonymous and non-synonymous changes. The following lines can do it, using one of the scripts included with the library:

```
$ export SAMPLES=$(for file in *.bam; do echo -a `basename $file .bam`,$file ;done)
$ add-gff-info coverage $SAMPLES assembly.uniprot.gff | add-gff-info \
    exp_syn -r assembly.fa > assembly.uniprot-update.gff

$ mv assembly.uniprot-update.gff assembly.uniprot.gff
$ unset SAMPLES
```

[33] http://www.kegg.jp

The first line prepares part of the command line for the script and stores it into an environment variable, while the last command unsets the variable, as it's not needed anymore. The second command adds the expected number of synonymous and non-synonymous changes for each annotation.

A faster way to add the coverage to a GFF file is to use the *cov_samtools* command instead:

```
$ for x in *.bam; do samtools depth -aa $x > `basename $x .bam`.depth; done
$ add-gff-info cov_samtools $(for file in *.depth; do echo -s `basename $file .
↪depth` -d $file ;done) assembly.uniprot.gff assembly.uniprot-update.gff
$ mv assembly.uniprot-update.gff assembly.uniprot.gff
```

This requires the creation of *depth* files from samtools, which can be fairly big. The script will accept files compressed with gzip, bzip2 (and xz if the module is available), but will be slower. For this tutorial, each uncompressed depth file is aboud 110MB.

The *coverage* command memory footprint is tied to the GFF file (kept in memory). The *cov_samtools* reads the depth information one line at a time and keeps a numpy array for each sequence in memory (and each sample), while the GFF is streamed.

3.1.8 SNP Calling

bcftools

For calling SNPs, we can use *bcftools* (v 1.8 was tested)

```
bcftools mpileup -f assembly.fa -Ou *.bam | bcftools call -m -v -O v --ploidy 1 -A␣
↪-o assembly.vcf
```

3.1.9 Data Preparation

Diversity Analysis

To use diversity estimates (pN/pS) for the data, we need to first first is aggregate all SNP information from the vcf file into data structures that can be read and analysed by the library. This can be done using the included script *snp_parser*, with this lines of bash:

```
$ export SAMPLES=$(for file in *.bam; do echo -m `basename $file .bam`;done)
$ snp_parser -s -v -g assembly.uniprot.gff -p assembly.vcf -a assembly.fa $SAMPLES
$ unset SAMPLES
```

Note: The *-s* options must be added if the VCF file was created with *bcftools*

Count Data

To evaluate the abundance of taxa and functional categories in the data we need to produce one file for each sample using htseq-count, from the HTSeq library.

```
for file in *.bam; do
    htseq-count -f bam -r pos -s no -t CDS -i uid -a 8 \
    -m intersection-nonempty $file assembly.uniprot.gff \
    > `basename $file .bam`-counts.txt
done
```

And to add the counts to the GFF file:

```
$ add-gff-info counts `for x in *.bam; do echo -s $(basename $x .bam); done` \
    `for x in *-counts.txt; do echo -c $x; done` assembly.uniprot.gff tmp.gff
$ mv tmp.gff assembly.uniprot.gff
```

Alternatively *featureCounts* from the *subread* package can be used:

```
$ featureCounts -a assembly.uniprot.gff -g uid -O  -t CDS -o counts-featureCounts.
↪txt *.bam
```

And adding it to the GFF is similar:

```
$ add-gff-info counts `for x in *.bam; do echo -s $(basename $x .bam); done` -c␣
↪counts-featureCounts.txt -e assembly.uniprot.gff tmp.gff
$ mv tmp.gff assembly.uniprot.gff
```

Note however that there will be one file only made by featureCounts and that is allowed when using *add-gff-info counts* when the *-e* option is passed.

Additional Downloads

The following files needs to be downloaded to analyse the functional categories in the following script:

```
$ wget http://eggnog.embl.de/version_3.0/data/downloads/COG.members.txt.gz
$ wget http://eggnog.embl.de/version_3.0/data/downloads/NOG.members.txt.gz
$ wget http://eggnog.embl.de/version_3.0/data/downloads/COG.funccat.txt.gz
$ wget http://eggnog.embl.de/version_3.0/data/downloads/NOG.funccat.txt.gz
```

and this for Enzyme Classification:

```
$ wget ftp://ftp.expasy.org/databases/enzyme/enzclass.txt
```

3.1.10 Full Bash Script

```
#!/bin/bash

#download data
#50 meters
wget ftp://ftp.sra.ebi.ac.uk/vol1/fastq/SRR001/SRR001326/SRR001326.fastq.gz
#1 meter
wget ftp://ftp.sra.ebi.ac.uk/vol1/fastq/SRR001/SRR001325/SRR001325.fastq.gz
#32 meters
wget ftp://ftp.sra.ebi.ac.uk/vol1/fastq/SRR001/SRR001323/SRR001323.fastq.gz
#16 meters
wget ftp://ftp.sra.ebi.ac.uk/vol1/fastq/SRR001/SRR001322/SRR001322.fastq.gz
#uncompress data
gunzip -v *.fastq.gz

#assembly - preparatory phase
velveth velvet_work 31 -fmtAuto *.fastq
#assembly
velvetg velvet_work -min_contig_lgth 50
#change sequence names
cat velvet_work/contigs.fa | sed -E 's/(>NODE_[0-9]+)_.+/\1/g' > assembly.fa
#alternative
#fasta-utils uid cat velvet_work/contigs.fa assembly.fa
#remove velvet working directory
rm -R velvet_work

```

(continues on next page)

(continued from previous page)

```
#To use the LCA option and other analysis we need a taxonomy file
download-taxonomy.sh

#Uniprot SwissProt DB
wget ftp://ftp.uniprot.org/pub/databases/uniprot/current_release/knowledgebase/
↪complete/uniprot_sprot.fasta.gz
#Uncompress it
gunzip uniprot_sprot.fasta.gz

########
#Gene prediction

###BLAST
#index Uniprot
makeblastdb -dbtype prot -in uniprot_sprot.fasta
#Run blastx
blastx -query assembly.fa -db uniprot_sprot.fasta -out \
        assembly.uniprot.tab -outfmt 6

###RAPSearch
#Index
# prerapsearch -d uniprot_sprot.fasta -n uniprot_sprot
#Run
# rapsearch -q assembly.fa -d uniprot_sprot -o assembly.uniprot.tab
#rename .m8 file to assembly.uniprot.tab and delete .aln
#rm assembly.uniprot.tab.aln
#mv assembly.uniprot.tab.m8 assembly.uniprot.tab

########
#Converts gene prediction into GFF annotations
blast2gff uniprot -b 40 -db UNIPROT-SP -dbq 10 assembly.uniprot.tab \
        assembly.uniprot.gff
filter-gff overlap assembly.uniprot.gff assembly.uniprot-filt.gff
#rename the new filtered file
mv assembly.uniprot-filt.gff assembly.uniprot.gff

########
#Taxonomic refinement - requires NCBI nt DB installed and indexed
export NCBINT_DIR=ncbi-db
if [ -d "$NCBINT_DIR" ]; then
        echo "Taxonomic refinement";
        #Extract annotations sequences
        get-gff-info sequence -f assembly.fa assembly.uniprot.gff \
        assembly.uniprot.frag.fasta
        #Use blastn to match against NCBI NT
        blastn -query assembly.uniprot.frag.fasta -db ncbi-db/nt -out \
                assembly.uniprot.frag.tab -outfmt 6

        #Download necessary data
        download-ncbi-taxa.sh
        gunzip -c ncbi-nucl-taxa.gz | taxon-utils to_hdf -n nt

        # Get the LCA for the sequences
        blast2gff blastdb -i 3 -r assembly.uniprot.frag.tab | \
        filter-gff sequence -t -a length -f quantile -l 0.95 -c gt | \
        filter-gff sequence -t -a identity -f quantile -l 0.95 -c gt | \
        add-gff-info addtaxa -f taxa-table.hf5:nt | \
        taxon-utils lca -b 40 -t taxonomy.pickle -s -p - lca.tab

        # Add the LCA info to the GFF
        add-gff-info addtaxa -v -t lca.tab -a seq_id -db NCBI-NT \
```

(continues on next page)

(continued from previous page)

```
        assembly.uniprot.gff assembly.uniprot-taxa.gff

        #rename the file to continue the script
        mv assembly.uniprot-taxa.gff assembly.uniprot.gff
fi
unset NCBINT_DIR

########
#Finalise information from Gene Prediction
#Adds remaining taxonomy, EC numbers, KO and eggNOG mappings
add-gff-info uniprot --buffer 500 -t -e -ec -ko \
        assembly.uniprot.gff assembly.uniprot-final.gff
#Rename the GFF
mv assembly.uniprot-final.gff assembly.uniprot.gff

########
#Alignments
bowtie2-build assembly.fa assembly.fa
for file in *.fastq; do
        BASENAME=`basename $file .fastq`;
        bowtie2 -N 1 -x assembly.fasta -U $file \
        --very-sensitive-local \
        --rg-id $BASENAME --rg PL:454 --rg PU:454 \
        --rg SM:$BASENAME | samtools view -Sb - > $BASENAME.bam;
done
#sort and index BAM files with samtools
for file in *.bam; do
        samtools sort $file `basename $file .bam`-sort;
        #removes the unsorted file, it's not needed
        mv `basename $file .bam`-sort.bam $file
        samtools index $file;
done

########
#Add coverage and expected changes to GFF file
#export SAMPLES=$(for file in *.bam; do echo -a `basename $file .bam`,$file ;done)
#Coverage info
#add-gff-info coverage $SAMPLES assembly.uniprot.gff | add-gff-info \
#        exp_syn -r assembly.fa > assembly.uniprot-update.gff
#rename to continue the script
#mv assembly.uniprot-update.gff assembly.uniprot.gff
#unset SAMPLES
#Faster
for x in *.bam; do samtools depth -aa $x > `basename $x .bam`.depth; done
add-gff-info cov_samtools $(for file in *.depth; do echo -s `basename file .depth`
↪-d $file ;done) assembly.uniprot.gff assembly.uniprot-update.gff
mv assembly.uniprot-update.gff assembly.uniprot.gff

########
#SNP calling using samtools
bcftools mpileup -f assembly.fa -Ou *.bam | bcftools call -v -O v --ploidy 1 -A -o
↪assembly.vcf


#Index fasta with Picard tools - GATK requires it
#java -jar picard-tools/picard.jar CreateSequenceDictionary \
#        R=assembly.fa O=assembly.dict

#merge vcf
#export SAMPLES=$(for file in *.bam.vcf; do echo -V:`basename $file .bam.vcf`
↪$file ;done)
```

(continues on next page)

(continued from previous page)

```
# java -Xmx10g -jar GATK/GenomeAnalysisTK.jar \
#           -R assembly.fa -T CombineVariants -o assembly.vcf \
#           -genotypeMergeOptions UNIQUIFY \
#           $SAMPLES
# unset SAMPLES

########
#snp_parser
export SAMPLES=$(for file in *.bam; do echo -m `basename $file .bam`;done)
snp_parser -s -v -g assembly.uniprot.gff -p assembly.vcf -a assembly.fa $SAMPLES
unset SAMPLES

########
#Count reads
# Using HTSeq
for file in *.bam; do
        htseq-count -f bam -r pos -s no -t CDS -i uid -a 8 \
        -m intersection-nonempty $file assembly.uniprot.gff \
        > `basename $file .bam`-counts.txt
done

# Adding counts to GFF
add-gff-info counts `for x in *.bam; do echo -s $(basename $x .bam); done` \
        `for x in *-counts.txt; do echo -c $x; done` assembly.uniprot.gff tmp.gff

mv tmp.gff assembly.uniprot.gff
# Using featureCounts
#featureCounts -a assembly.uniprot.gff -g uid -O  -t CDS -o counts-featureCounts.
↪txt *.bam
#add-gff-info counts `for x in *.bam; do echo -s $(basename $x .bam); done` -c␣
↪counts-featureCounts.txt -e assembly.uniprot.gff tmp.gff
#mv tmp.gff assembly.uniprot.gff

########
#eggNOG mappings
wget http://eggnog.embl.de/version_3.0/data/downloads/COG.members.txt.gz
wget http://eggnog.embl.de/version_3.0/data/downloads/NOG.members.txt.gz
wget http://eggnog.embl.de/version_3.0/data/downloads/COG.funccat.txt.gz
wget http://eggnog.embl.de/version_3.0/data/downloads/NOG.funccat.txt.gz

########
#EC names
wget ftp://ftp.expasy.org/databases/enzyme/enzclass.txt
```

3.2 Tutorial - Exploring the Data

The following section requires that:

- the tutorial has been completed
- the data from it is in the same directory

In alternative the data required to run this example can be download from figshare[36] and uncrompressed.

[36] http://files.figshare.com/2598711/tutorial_data.zip

3.2.1 Imports

[1]:
```
from __future__ import print_function

#Python Standard Library
import glob
import pickle
import sys
import functools
#External Dependencies (install via pip or anaconda)

# Check if running interactively or not
import matplotlib as mpl # http://matplotlib.org
# from:
# http://stackoverflow.com/questions/15411967/how-can-i-check-if-code-is-executed-
↪in-the-ipython-notebook
# and
# http://stackoverflow.com/questions/15455029/python-matplotlib-agg-vs-interactive-
↪plotting-and-tight-layout
import __main__ as main
if hasattr(main, '__file__'):
    # Non interactive, force the use of Agg backend instead
    # of the default one
    mpl.use('Agg')

import numpy # http://www.numpy.org
import pandas # http://pandas.pydata.org
import seaborn # http://stanford.edu/~mwaskom/software/seaborn/
import scipy # http://www.scipy.org
import matplotlib.pyplot as plt


#MGKit Import
from mgkit.io import gff, fasta
from mgkit.mappings import eggnog
import mgkit.counts, mgkit.taxon, mgkit.snps, mgkit.plots
import mgkit.snps
import mgkit.mappings.enzyme
```

```
/home/frubino/mgkit/dev-env/local/lib/python2.7/site-packages/statsmodels/compat/
↪pandas.py:56: FutureWarning: The pandas.core.datetools module is deprecated and␣
↪will be removed in a future version. Please use the pandas.tseries module␣
↪instead.
  from pandas.core import datetools
```

[2]:
```
mgkit.logger.config_log()
```

[3]:
```
mgkit.cite(sys.stdout)
```

```
 _|      _|    _|_|_|  _|    _|  _|    _|
 _|_|  _|_|  _|        _|  _|        _|_|_|_|
 _|  _|  _|  _|  _|_|  _|_|      _|    _|
 _|      _|  _|    _|  _|  _|    _|    _|
 _|      _|    _|_|_|  _|    _|  _|      _|_|


MGKit Version: 0.3.4

Rubino, F. and Creevey, C.J. (2014).
MGkit: Metagenomic Framework For The Study Of Microbial Communities.
```

(continues on next page)

(continued from previous page)

```
Available at: http://figshare.com/articles/
↪MGkit_Metagenomic_Framework_For_The_Study_Of_Microbial_Communities/1269288

[doi:10.6084/m9.figshare.1269288]
```

3.2.2 Download Complete Data

If the tutorial can't be completed, the download data can be downloaded from: %%

[4]:
```
# the following variable is used to indicate where the tutorial data is stored
data_dir = 'tutorial-data/'
```

3.2.3 Read Necessary Data

[5]:
```
# Keeps a list of the count data file outputted by
# htseq-count
counts = glob.glob('{}*-counts.txt'.format(data_dir))
```

[6]:
```
# This file contains the SNPs information and it is the output
# of the snp_parser script
snp_data = pickle.load(open('{}snp_data.pickle'.format(data_dir), 'r'))
```

[7]:
```
# Taxonomy needs to be download beforehand. It is loaded into an an
# instance of mgkit.taxon.UniprotTaxonomy. It is used in filtering
# data and to map taxon IDs to different levels in the taxonomy
taxonomy = mgkit.taxon.UniprotTaxonomy('{}/taxonomy.pickle'.format(data_dir))
```

```
2018-05-22 14:06:05,570 - INFO - mgkit.taxon->load_data: Loading taxonomy from␣
↪file tutorial-data//taxonomy.pickle
```

[8]:
```
# Loads all annotations in a dictionary, with the unique ID (uid) as key
# and the mgkit.io.gff.Annotation instance that represent the line in the
# GFF file as value
annotations = {x.uid: x for x in gff.parse_gff('{}assembly.uniprot.gff'.
↪format(data_dir))}
```

```
2018-05-22 14:06:14,637 - INFO - mgkit.io.gff->parse_gff: Loading GFF from file␣
↪(tutorial-data/assembly.uniprot.gff)
2018-05-22 14:06:15,578 - INFO - mgkit.io.gff->parse_gff: Read 9136 line from file␣
↪(tutorial-data/assembly.uniprot.gff)
```

[9]:
```
# Used to extract the sample ID from the count file names
file_name_to_sample = lambda x: x.rsplit('/')[-1].split('-')[0]
```

[10]:
```
# Used to rename the DataFrame columns
sample_names = {
    'SRR001326': '50m',
    'SRR001325': '01m',
    'SRR001323': '32m',
    'SRR001322': '16m'
}
```

3.2.4 Explore Count Data

Load Taxa Table

Build a pandas.DataFrame instance. It is NOT required, but it is easier to manipulate. load_sample_counts_to_taxon returns a pandas.Series instance.

The DataFrame will have the sample names as columns names and the different taxon IDs as rows names. There are 3 different function to map counts and annotations to a pandas.Series instance:

- mgkit.counts.load_sample_counts
- mgkit.counts.load_sample_counts_to_taxon
- mgkit.counts.load_sample_counts_to_genes

The three differs primarly by the index for the pandas.Series they return, which is (gene_id, taxon_id), taxon_id and gene_id, respectively. Another change is the possibility to map a gene_id to another and a taxon_id to a different rank. In this contexts, as it is interesting to assess the abundance of each organism, mgkit.counts.load_sample_counts_to_taxon can be used. It provides a **rank** parameter that can be changed to map all counts to the *order* level in this case, but can be changed to any rank in mgkit.taxon.TAXON_RANKS, for example *genus*, *phylum*.

[11]:
```
taxa_counts = pandas.DataFrame({
    # Get the sample names
    file_name_to_sample(file_name): mgkit.counts.load_sample_counts_to_taxon(
        # A function accept a uid as only parameter and returns only the
        # gene_id and taxon_id, so we set it to a lambda that does
        # exactly that
        lambda x: (annotations[x].gene_id, annotations[x].taxon_id),
        # An iterator that yields (uid, count) is needed and MGKit
        # has a function that does that for htseq-count files.
        # This can be adapted to any count data file format
        mgkit.counts.load_htseq_counts(file_name),
        # A mgkit.taxon.UniprotTaxonomy instance is necessary to filter
        # the data and map it to a different rank
        taxonomy,
        # A taxonomic rank to map each taxon_id to. Must be lowercase
        rank='order',
        # If False, any taxon_id that can not be resolved at the taxonomic
        # rank requested is excluded from the results
        include_higher=False
    )
    # iterate over all count files
    for file_name in counts
})
```

```
2018-05-22 14:06:15,644 - INFO - mgkit.counts.func->load_htseq_counts: Loading␣
↪HTSeq-count file tutorial-data/SRR001322-counts.txt
2018-05-22 14:06:15,733 - INFO - mgkit.counts.func->load_htseq_counts: Loading␣
↪HTSeq-count file tutorial-data/SRR001323-counts.txt
2018-05-22 14:06:15,837 - INFO - mgkit.counts.func->load_htseq_counts: Loading␣
↪HTSeq-count file tutorial-data/SRR001325-counts.txt
2018-05-22 14:06:15,921 - INFO - mgkit.counts.func->load_htseq_counts: Loading␣
↪HTSeq-count file tutorial-data/SRR001326-counts.txt
```

[12]:
```
# This is an alternative if the counts are stored in the annotations
# ann_func = functools.partial(
#     taxonomy.get_ranked_id,
#     rank='order',
#     it=True,
#     include_higher=False
# )
# taxa_counts = mgkit.counts.func.from_gff(annotations.values(), sample_names, ann_
↪func=lambda x: ann_func(x.taxon_id))
# taxa_counts = taxa_counts[taxa_counts.index.notnull()]
```

Scaling (DESeq method) and Rename Rows/Columns

Because each sample has different yields in total DNA from the sequencing, the table should be scaled. The are a few approaches, RPKM, scaling by the minimum. MGKit offers mgkit.counts.scaling.scale_factor_deseq and mgkit.counts.scaling.scale_rpkm that scale using the DESeq method and RPKM respectively.

[12]:
```
# the DESeq method doesn't require information about the gene length
taxa_counts = mgkit.counts.scale_deseq(taxa_counts)
```

```
/home/frubino/mgkit/dev-env/local/lib/python2.7/site-packages/scipy/stats/stats.
↪py:305: RuntimeWarning: divide by zero encountered in log
  log_a = np.log(np.array(a, dtype=dtype))
```

One of the powers of pandas data structures is the metadata associated and the possibility to modify them with ease. In this case, the columns are named after the sample IDs from ENA and the row names are the taxon IDs. To make it easier to analyse, columns and rows can be renamed and sorted by name and the rows sorted in descending order by the first colum (1 meter).

To rename the columns the dictionary **sample_name** can be supplied and for the rows the name of each taxon ID can be accessed through the taxonomy instance, because it works as a dictionary and the returned object has a **s_name** attribute with the scientific name (lowercase).

[13]:
```
taxa_counts = taxa_counts.rename(
    index=lambda x: taxonomy[x].s_name,
    columns=sample_names
).sort_index(axis='columns')
```

[14]:
```
taxa_counts = taxa_counts.loc[taxa_counts.mean(axis='columns').sort_
↪values(ascending=False).index]
```

[15]:
```
# the *describe* method of a pandas.Series or pandas.DataFrame
# gives some insights into the data
taxa_counts.describe()
```

[15]:

	01m	16m	32m	50m
count	194.000000	194.000000	194.000000	194.000000
mean	27.781155	33.622739	30.885116	34.487249
std	69.125920	93.515885	86.123968	92.818806
min	0.000000	0.000000	0.000000	0.000000
25%	0.000000	0.000000	0.713044	0.000000
50%	5.345444	3.198502	4.278267	3.465247
75%	19.711325	19.990636	17.826111	21.441218
max	529.198964	722.861412	713.044453	738.097692

[16]:
```
#Save a CSV to disk, but Excel and other file formats are available
taxa_counts.to_csv('{}taxa_counts.csv'.format(data_dir))
```

[17]:
```
# This will give an idea of the counts for each order
taxa_counts.head(n=10)
```

[17]:

	01m	16m	32m	50m
Methanococcales	529.198964	722.861412	713.044453	738.097692
Thermococcales	354.135670	637.568030	577.566007	595.156237
Bacillales	525.189881	554.406983	440.661472	528.450225
Methanobacteriales	163.036044	358.232204	290.922137	330.931125
Archaeoglobales	208.472319	313.453179	278.087337	274.620855
Dehalococcoidales	203.126875	229.225964	286.643870	253.829370
Clostridiales	225.845012	217.498124	205.356802	206.182219
Enterobacterales	199.117792	165.255928	146.887157	252.963059
Methanosarcinales	146.999712	238.821470	213.913336	129.080465
Desulfurococcales	66.818051	197.240946	249.565558	182.791799

Plots for Top40 Taxa

Distribution of Each Taxon Over Depth

How to visualise the data depends on the question we want to ask and the experimental design. As a starting point, it may be interesting to visualise the variation of a taxonomic order abundance over the samples. This can be done using boxplots, among other methods.

MGKit offers a few functions to make complex plots, with a starting point in mgkit.plots.boxplot.boxplot_dataframe. However, as the data produced is in fact a pandas DataFrame, which is widely supported, a host of different specialised libraries tht offer similar functions can be used.

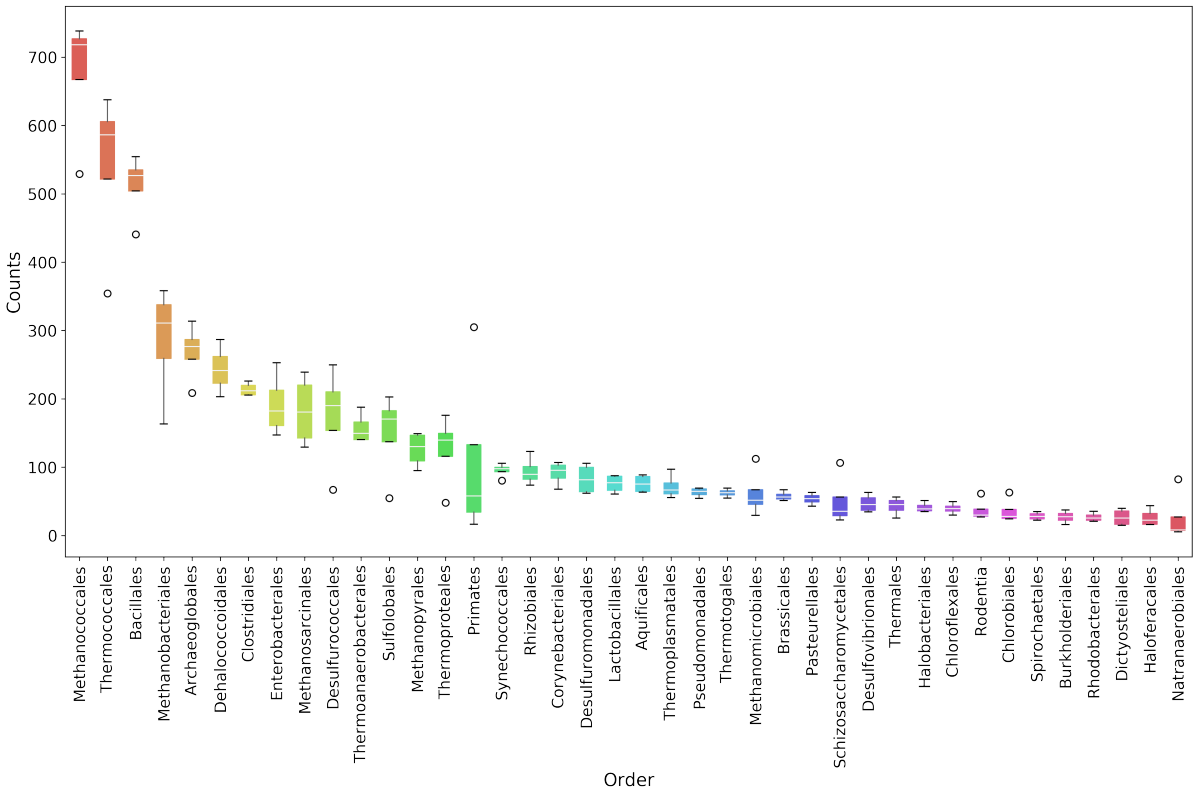
[18]:
```
# A matplotlib Figure instance and a single axis can be returned
# by this MGKit function. It is an helper function, the axis is
# needed to plot and the figure object to save the file to disk
fig, ax = mgkit.plots.get_single_figure(figsize=(15, 10))
# The return value of mgkit.plots.boxplot.boxplot_dataframe is
# passed to the **_** special variable, as it is not needed and
# it would be printed, otherwise
_ = mgkit.plots.boxplot.boxplot_dataframe(
    # The full dataframe can be passed
    taxa_counts,
    # this variable is used to tell the function
    # which rows and in which order they need to
    # be plot. In this case only the first 40 are
    # plot
    taxa_counts.index[:40],
    # A matplotlib axis instance
    ax,
    # a dictionary with options related to the labels
    # on both the X and Y axes. In this case it changes
    # the size of the labels
    fonts=dict(fontsize=14),
    # The default is to use the same colors for all
    # boxes. A dictionary can be passed to change this
    # in this case, the 'hls' palette from seaborn is
    # used.
    data_colours={
        x: color
        for x, color in zip(taxa_counts.index[:40], seaborn.color_palette('hls',␣
↪40))
    }
)
# Adds labels to the axes
ax.set_xlabel('Order', fontsize=16)
ax.set_ylabel('Counts', fontsize=16)
# Ensure the correct layout before writing to disk
fig.set_tight_layout(True)
# Saves a PDF file, or any other supported format by matplotlib
fig.savefig('{}taxa_counts-boxplot_top40_taxa.pdf'.format(data_dir))
```

```
2018-05-22 14:06:28,310 - DEBUG - matplotlib.font_manager->findfont: findfont:␣
↪Matching␣
↪:family=sans-serif:style=normal:variant=normal:weight=normal:stretch=normal:size=14.
↪0 to DejaVu Sans (u'/home/frubino/mgkit/dev-env/local/lib/python2.7/
↪site-packages/matplotlib/mpl-data/fonts/ttf/DejaVuSans.ttf') with score of 0.
↪050000
2018-05-22 14:06:28,357 - DEBUG - matplotlib.font_manager->findfont: findfont:␣
↪Matching␣
↪:family=sans-serif:style=normal:variant=normal:weight=normal:stretch=normal:size=16.
↪0 to DejaVu Sans (u'/home/frubino/mgkit/dev-env/local/lib/python2.7/
↪site-packages/matplotlib/mpl-data/fonts/ttf/DejaVuSans.ttf') with score of 0.
↪050000
```

(continues on next page)

(continued from previous page)

```
2018-05-22 14:06:28,386 - DEBUG - matplotlib.backends.backend_pdf->fontName:␣
↪Assigning font /F1 = u'/home/frubino/mgkit/dev-env/local/lib/python2.7/
↪site-packages/matplotlib/mpl-data/fonts/ttf/DejaVuSans.ttf'
2018-05-22 14:06:28,537 - DEBUG - matplotlib.backends.backend_pdf->writeFonts:␣
↪Embedding font /home/frubino/mgkit/dev-env/local/lib/python2.7/site-packages/
↪matplotlib/mpl-data/fonts/ttf/DejaVuSans.ttf.
2018-05-22 14:06:28,538 - DEBUG - matplotlib.backends.backend_pdf->writeFonts:␣
↪Writing TrueType font.
/home/frubino/mgkit/dev-env/local/lib/python2.7/site-packages/matplotlib/figure.
↪py:2267: UserWarning: This figure includes Axes that are not compatible with␣
↪tight_layout, so results might be incorrect.
  warnings.warn("This figure includes Axes that are not compatible "
```



Distribution of Taxa at Each Depth

Seaborn offers a KDE plot, which is useful to display the distribution of taxa counts for each sampling depth.

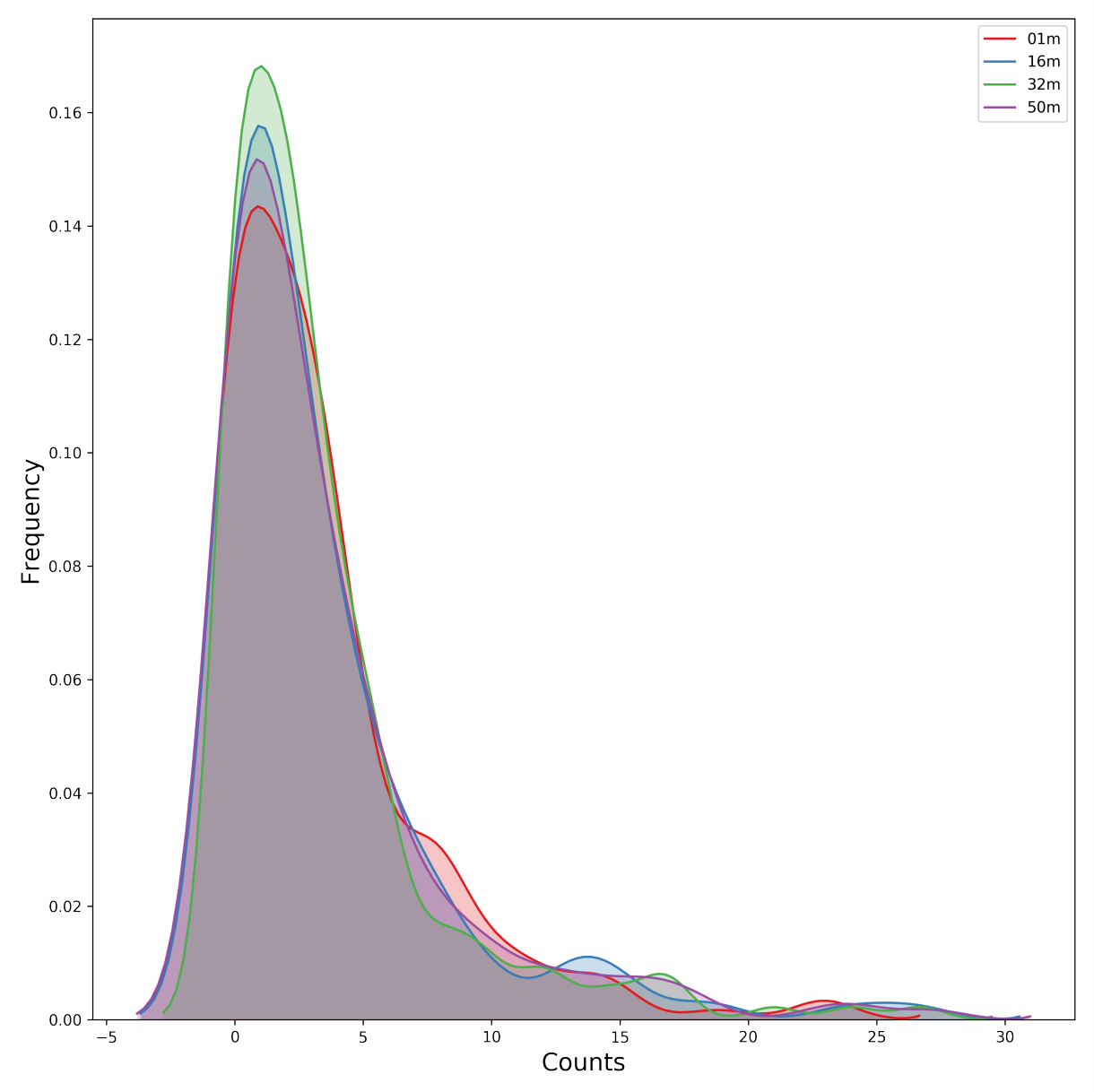
[19]:
```
fig, ax = mgkit.plots.get_single_figure(figsize=(10, 10))
# iterate over the columns, which are the samples and assign a color to each one
for column, color in zip(taxa_counts.columns, seaborn.color_palette('Set1',␣
↪len(taxa_counts.columns))):
    seaborn.kdeplot(
        # The data can transformed with the sqrt function of numpy
        numpy.sqrt(taxa_counts[column]),
        # Assign the color
        color=color,
        # Assign the label to the sample name to appear
        # in the legend
        label=column,
        # Add a shade under the KDE function
        shade=True
```

(continues on next page)

(continued from previous page)

```
    )
# Adds a legend
ax.legend()
ax.set_xlabel('Counts', fontsize=16)
ax.set_ylabel('Frequency', fontsize=16)
fig.set_tight_layout(True)
fig.savefig('{}taxa_counts-distribution_top40_taxa.pdf'.format(data_dir))
```

```
2018-05-22 14:06:32,367 - DEBUG - matplotlib.font_manager->findfont: findfont:␣
↪Matching␣
↪:family=sans-serif:style=normal:variant=normal:weight=normal:stretch=normal:size=10.
↪0 to DejaVu Sans (u'/home/frubino/mgkit/dev-env/local/lib/python2.7/
↪site-packages/matplotlib/mpl-data/fonts/ttf/DejaVuSans.ttf') with score of 0.
↪050000
2018-05-22 14:06:32,404 - DEBUG - matplotlib.backends.backend_pdf->fontName:␣
↪Assigning font /F1 = u'/home/frubino/mgkit/dev-env/local/lib/python2.7/
↪site-packages/matplotlib/mpl-data/fonts/ttf/DejaVuSans.ttf'
2018-05-22 14:06:32,461 - DEBUG - matplotlib.backends.backend_pdf->writeFonts:␣
↪Embedding font /home/frubino/mgkit/dev-env/local/lib/python2.7/site-packages/
↪matplotlib/mpl-data/fonts/ttf/DejaVuSans.ttf.
2018-05-22 14:06:32,463 - DEBUG - matplotlib.backends.backend_pdf->writeFonts:␣
↪Writing TrueType font.
```



Heatmap of the Table

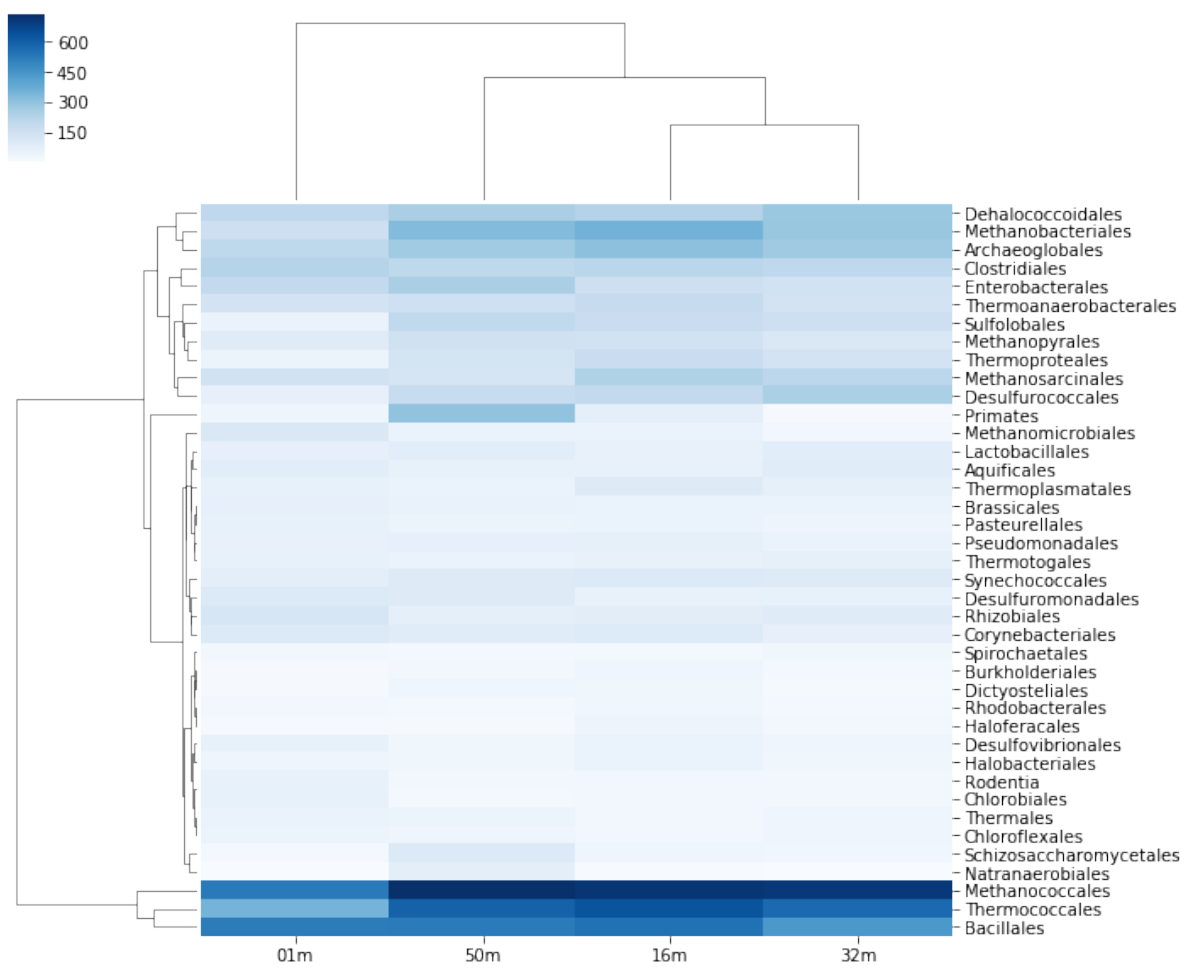
[20]:
```
# An heatmap can be created to provide information on the table
clfig = seaborn.clustermap(taxa_counts.iloc[:40], cbar=True, cmap='Blues')
clfig.fig.set_tight_layout(True)
for text in clfig.ax_heatmap.get_yticklabels():
    text.set_rotation('horizontal')
clfig.savefig('{}taxa_counts-heatmap-top40.pdf'.format(data_dir))
```

```
2018-05-22 14:06:34,833 - DEBUG - matplotlib.backends.backend_pdf->fontName:␣
↪Assigning font /F1 = u'/home/frubino/mgkit/dev-env/local/lib/python2.7/
↪site-packages/matplotlib/mpl-data/fonts/ttf/DejaVuSans.ttf'
2018-05-22 14:06:34,936 - DEBUG - matplotlib.backends.backend_pdf->writeFonts:␣
↪Embedding font /home/frubino/mgkit/dev-env/local/lib/python2.7/site-packages/
↪matplotlib/mpl-data/fonts/ttf/DejaVuSans.ttf.
2018-05-22 14:06:34,938 - DEBUG - matplotlib.backends.backend_pdf->writeFonts:␣
↪Writing TrueType font.
2018-05-22 14:06:35,020 - DEBUG - matplotlib.backends.backend_pdf->fontName:␣
↪Assigning font /F1 = u'/home/frubino/mgkit/dev-env/local/lib/python2.7/
↪site-packages/matplotlib/mpl-data/fonts/ttf/DejaVuSans.ttf'
```

(continues on next page)

(continued from previous page)

```
2018-05-22 14:06:35,129 - DEBUG - matplotlib.backends.backend_pdf->writeFonts:␣
↪Embedding font /home/frubino/mgkit/dev-env/local/lib/python2.7/site-packages/
↪matplotlib/mpl-data/fonts/ttf/DejaVuSans.ttf.
2018-05-22 14:06:35,131 - DEBUG - matplotlib.backends.backend_pdf->writeFonts:␣
↪Writing TrueType font.
```



Functional Categories

Besides looking at specific taxa, it is possible to map each gene_id to functional categories. eggNOG[37] provides this. **v3 must be used**, as the mappings in Uniprot points to that version.

Load Necessary Data

[21]:
```
eg = eggnog.NOGInfo()
```

[39]:
```
#Build mapping Uniprot IDs -> eggNOG functional categories
fc_map = {
    # An Annotation instance provide a method to access the list of IDs for the
    # specific mapping. For example eggnog mappings are store into the
    # map_EGGNOG attribute
    annotation.gene_id: eg.get_nogs_funccat(annotation.get_mapping('eggnog'))
```

(continues on next page)

[37] http://eggnog.embl.de/

(continued from previous page)

```
    for annotation in annotations.itervalues()
}
```

[41]:
```
# Just a few to speed up the analysis, but other can be used
# Should have been downloaded by the full tutorial script
eg.load_members('{}COG.members.txt.gz'.format(data_dir))
eg.load_members('{}NOG.members.txt.gz'.format(data_dir))
eg.load_funccat('{}COG.funccat.txt.gz'.format(data_dir))
eg.load_funccat('{}NOG.funccat.txt.gz'.format(data_dir))
```

```
2018-05-22 14:09:38,473 - INFO - mgkit.mappings.eggnog->load_members: Reading␣
↪Members from tutorial-data/COG.members.txt.gz
2018-05-22 14:09:56,921 - INFO - mgkit.mappings.eggnog->load_members: Reading␣
↪Members from tutorial-data/NOG.members.txt.gz
2018-05-22 14:10:03,884 - INFO - mgkit.mappings.eggnog->load_funccat: Reading␣
↪Functional Categories from tutorial-data/COG.funccat.txt.gz
2018-05-22 14:10:03,896 - INFO - mgkit.mappings.eggnog->load_funccat: Reading␣
↪Functional Categories from tutorial-data/NOG.funccat.txt.gz
```

Build FC Table

As mentioned above, mgkit.counts.load_sample_counts_to_genes works in the same way as mgkit.counts.load_sample_counts_to_taxon, with the difference of giving **gene_id** as the only index.

In this case, however, as a mapping to functional categories is wanted, to the **gene_map** parameter a dictionary where for each *gene_id* an iterable of *mappings* is assigned. These are the values used in the index of the returned pandas.Series, which ends up as rows in the **fc_counts** DataFrame.

[45]:
```
fc_counts = pandas.DataFrame({
    file_name_to_sample(file_name): mgkit.counts.load_sample_counts_to_genes(
        lambda x: (annotations[x].gene_id, annotations[x].taxon_id),
        mgkit.counts.load_htseq_counts(file_name),
        taxonomy,
        gene_map=fc_map
    )
    for file_name in counts
})
```

```
2018-05-22 14:10:11,210 - INFO - mgkit.counts.func->load_htseq_counts: Loading␣
↪HTSeq-count file tutorial-data/SRR001322-counts.txt
2018-05-22 14:10:11,293 - INFO - mgkit.counts.func->load_htseq_counts: Loading␣
↪HTSeq-count file tutorial-data/SRR001323-counts.txt
2018-05-22 14:10:11,370 - INFO - mgkit.counts.func->load_htseq_counts: Loading␣
↪HTSeq-count file tutorial-data/SRR001325-counts.txt
2018-05-22 14:10:11,445 - INFO - mgkit.counts.func->load_htseq_counts: Loading␣
↪HTSeq-count file tutorial-data/SRR001326-counts.txt
```

Scale the Table and Rename Rows/Columns

[46]:
```
fc_counts = mgkit.counts.scale_deseq(fc_counts).rename(
    columns=sample_names,
    index=eggnog.EGGNOG_CAT
)
```

[47]:
```
fc_counts.describe()
```

[47]:
```
                16m           32m          01m           50m
count     23.000000     23.000000    23.000000     23.000000
```

(continues on next page)

(continued from previous page)

```
mean    288.923364   289.089215  255.206073   287.413551
std     288.130525   286.446124  205.412833   283.111728
min       0.000000     0.000000    0.000000     3.471907
25%      66.085234    78.138939  111.201303    69.004160
50%     209.794392   232.510989  220.994995   223.070050
75%     356.650467   388.026535  344.864801   376.701953
max    1103.518503  1147.308320  734.773169  1177.844584
```

[49]:
```
fc_counts
```

[49]:

	16m	32m	\
RNA processing and modification	4.195888	3.049324	
Chromatin structure and dynamics	27.273271	21.345271	
Energy production and conversion	832.883737	735.649521	
Cell cycle control, cell division, chromosome p...	73.428037	97.578382	
Amino acid transport and metabolism	676.586915	658.654079	
Nucleotide transport and metabolism	336.719999	374.304575	
Carbohydrate transport and metabolism	333.573084	354.483966	
Coenzyme transport and metabolism	323.083364	304.170113	
Lipid transport and metabolism	232.871775	238.609637	
Translation, ribosomal structure and biogenesis	1103.518503	1147.308320	
Transcription	290.565233	276.726193	
Replication, recombination and repair	528.681868	554.214717	
Cell wall/membrane/envelope biogenesis	206.647476	174.573824	
Cell motility	27.273271	19.058278	
Posttranslational modification, protein turnove...	376.580934	401.748495	
Inorganic ion transport and metabolism	186.717009	200.493082	
Secondary metabolites biosynthesis, transport a...	58.742430	86.143415	
General function prediction only	577.983550	563.362690	
Function unknown	209.794392	232.510989	
Signal transduction mechanisms	35.665047	38.116555	
Intracellular trafficking, secretion, and vesic...	87.064673	96.816051	
Defense mechanisms	115.386916	70.134462	
Cytoskeleton	0.000000	0.000000	

	01m	50m
RNA processing and modification	25.337006	45.134796
Chromatin structure and dynamics	12.668503	68.570171
Energy production and conversion	609.495751	703.061248
Cell cycle control, cell division, chromosome p...	121.054583	52.078611
Amino acid transport and metabolism	609.495751	574.600674
Nucleotide transport and metabolism	271.669007	291.640221
Carbohydrate transport and metabolism	285.745121	275.148661
Coenzyme transport and metabolism	368.794196	374.098022
Lipid transport and metabolism	243.516778	243.901495
Translation, ribosomal structure and biogenesis	734.773169	1177.844584
Transcription	237.886332	365.418254
Replication, recombination and repair	439.174767	451.347962
Cell wall/membrane/envelope biogenesis	216.772161	199.634675
Cell motility	33.782674	33.851097
Posttranslational modification, protein turnove...	320.935407	379.305883
Inorganic ion transport and metabolism	190.027544	173.595370
Secondary metabolites biosynthesis, transport a...	106.978469	96.345430
General function prediction only	494.071613	675.285989
Function unknown	220.994995	223.070050
Signal transduction mechanisms	115.424138	46.870750
Intracellular trafficking, secretion, and vesic...	121.054583	86.797685
Defense mechanisms	90.087132	69.438148
Cytoskeleton	0.000000	3.471907

[48]:
```
#Save table to disk
fc_counts.to_csv('{}fc_counts.csv'.format(data_dir))
```

Heatmap to Explore Functional Categories

[50]:
```
clfig = seaborn.clustermap(fc_counts, cbar=True, cmap='Greens')
clfig.fig.set_tight_layout(True)
for text in clfig.ax_heatmap.get_yticklabels():
    text.set_rotation('horizontal')
clfig.savefig('{}fc_counts-heatmap.pdf'.format(data_dir))
```

```
2018-05-22 14:10:17,521 - DEBUG - matplotlib.backends.backend_pdf->fontName:␣
↪Assigning font /F1 = u'/home/frubino/mgkit/dev-env/local/lib/python2.7/
↪site-packages/matplotlib/mpl-data/fonts/ttf/DejaVuSans.ttf'
2018-05-22 14:10:17,626 - DEBUG - matplotlib.backends.backend_pdf->writeFonts:␣
↪Embedding font /home/frubino/mgkit/dev-env/local/lib/python2.7/site-packages/
↪matplotlib/mpl-data/fonts/ttf/DejaVuSans.ttf.
2018-05-22 14:10:17,627 - DEBUG - matplotlib.backends.backend_pdf->writeFonts:␣
↪Writing TrueType font.
2018-05-22 14:10:17,693 - DEBUG - matplotlib.backends.backend_pdf->fontName:␣
↪Assigning font /F1 = u'/home/frubino/mgkit/dev-env/local/lib/python2.7/
↪site-packages/matplotlib/mpl-data/fonts/ttf/DejaVuSans.ttf'
2018-05-22 14:10:17,798 - DEBUG - matplotlib.backends.backend_pdf->writeFonts:␣
↪Embedding font /home/frubino/mgkit/dev-env/local/lib/python2.7/site-packages/
↪matplotlib/mpl-data/fonts/ttf/DejaVuSans.ttf.
2018-05-22 14:10:17,799 - DEBUG - matplotlib.backends.backend_pdf->writeFonts:␣
↪Writing TrueType font.
```

Enzyme Classification

Enzyme classification number were added the *add-gff-info* script, so they can be used in a similar way to functional categories. The specificity level requested is **2**.

[51]:
```
ec_map = {
    # EC numbers are store into the EC attribute in a GFF file and
    # an Annotation instance provide a get_ec method that returns
    # a list. A level of specificity can be used to the mapping
    # less specific, as it ranges from 1 to 4 included. Right
    # now a list is returned, so it is a good idea to convert
    # the list into a set so if any duplicate appears (as effect
    # of the change in level) it won't inflate the number later.
    # In later versions (0.2) a set will be returned instead of
    # a list.
    # We also want to remove any hanging ".-" to use the labels
    # from expasy
    annotation.gene_id: set(x.replace('.-', '') for x in annotation.get_
↪ec(level=2))
    for annotation in annotations.itervalues()
}
```

[52]:
```
# The only difference with the functional categories is the mapping
# used.
ec_counts = pandas.DataFrame({
    file_name_to_sample(file_name): mgkit.counts.load_sample_counts_to_genes(
        lambda x: (annotations[x].gene_id, annotations[x].taxon_id),
        mgkit.counts.load_htseq_counts(file_name),
        taxonomy,
        gene_map=ec_map
    )
    for file_name in counts
})
```

```
2018-05-22 14:10:20,682 - INFO - mgkit.counts.func->load_htseq_counts: Loading␣
↪HTSeq-count file tutorial-data/SRR001322-counts.txt
2018-05-22 14:10:20,768 - INFO - mgkit.counts.func->load_htseq_counts: Loading␣
↪HTSeq-count file tutorial-data/SRR001323-counts.txt
2018-05-22 14:10:20,844 - INFO - mgkit.counts.func->load_htseq_counts: Loading␣
↪HTSeq-count file tutorial-data/SRR001325-counts.txt
2018-05-22 14:10:20,922 - INFO - mgkit.counts.func->load_htseq_counts: Loading␣
↪HTSeq-count file tutorial-data/SRR001326-counts.txt
```

[53]:
```
# This file contains the names of each enzyme class and can be downloaded
# from ftp://ftp.expasy.org/databases/enzyme/enzclass.txt
# It should be downloaded at the end of the tutorial script
ec_names = mgkit.mappings.enzyme.parse_expasy_file('{}enzclass.txt'.format(data_
↪dir))
```

[54]:
```
# Rename columns and row. Rows will include the full label the enzyme class
ec_counts = mgkit.counts.scale_deseq(ec_counts).rename(
    index=lambda x: "{} {} [EC {}.-]".format(
        # A name of the second level doesn't include the first level
        # definition, so if it is level 2, we add the level 1 label
        '' if len(x) == 1 else ec_names[x[0]] + " - ",
        # The EC label for the specific class (e.g. 3.2)
        ec_names[x],
        # The EC number
        x
    ),
    columns=sample_names
)
```

[55]:
```
plot_order = ec_counts.median(axis=1).sort_values(ascending=True, inplace=False).
↪index
```

[56]:
```
ec_counts.describe()
```

[56]:
```
              16m         32m         01m         50m
count   59.000000   59.000000   59.000000   59.000000
mean    74.276636   78.371254   79.636608   80.406521
std     98.400500  108.265549  108.003831  115.948168
min      0.000000    0.000000    0.000000    0.000000
25%      4.786284    6.764489    5.672632    3.995495
50%     36.375758   42.090153   34.035791   37.291289
75%     96.204307  105.225384  127.229027  121.640634
max    465.226796  537.401066  546.193404  674.794758
```

[57]:
```
ec_counts
```

[57]:
```
                                                      16m         32m  \
 Oxidoreductases [EC 1.-]                           30.632217   18.038637
Oxidoreductases -  Acting on the CH-OH group of...  159.861883  134.538169
Oxidoreductases -  Acting on a peroxide as acce...    6.700797   14.280588
Oxidoreductases -  Acting on hydrogen as donors...   65.093461   81.173867
Oxidoreductases -  Acting on single donors with...    0.000000    7.516099
Oxidoreductases -  Acting on paired donors, wit...    0.000000    0.000000
Oxidoreductases -  Acting on superoxide as acce...    4.786284    3.006440
Oxidoreductases -  Oxidizing metal ions [EC 1.1...   14.358852   10.522538
Oxidoreductases -  Acting on CH or CH(2) groups...   54.563637   61.632010
Oxidoreductases -  Acting on iron-sulfur protei...   36.375758   15.783808
Oxidoreductases -  Acting on the aldehyde or ox...  150.289315  145.812317
Oxidoreductases -  Catalyzing the reaction X-H ...    0.957257    6.012879
Oxidoreductases -  Acting on the CH-CH group of...   28.717703   42.090153
Oxidoreductases -  Acting on the CH-NH(2) group...   36.375758   71.402939
Oxidoreductases -  Acting on the CH-NH group of...    0.957257    9.019319
Oxidoreductases -  Acting on NADH or NADPH [EC ...   71.794259   83.428697
Oxidoreductases -  Acting on other nitrogenous ...    8.615311    0.000000
Oxidoreductases -  Acting on a sulfur group of ...  125.400639  102.970554
Oxidoreductases -  Acting on a heme group of do...    1.914514    0.000000
Oxidoreductases -  Other oxidoreductases [EC 1....    1.914514    6.012879
 Transferases [EC 2.-]                              0.000000    0.000000
Transferases -  Transferring one-carbon groups ...  192.408613  223.228135
Transferases -  Transferring molybdenum- or tun...    0.000000    0.000000
Transferases -  Transferring aldehyde or ketoni...   27.760447   23.299906
Transferases -  Acyltransferases [EC 2.3.-]         107.212760  126.270460
Transferases -  Glycosyltransferases [EC 2.4.-]      82.324083  106.728603
Transferases -  Transferring alkyl or aryl grou...   79.452313   96.957675
Transferases -  Transferring nitrogenous groups...   85.195854  103.722164
Transferases -  Transferring phosphorus-contain...  465.226796  537.401066
Transferases -  Transferring sulfur-containing ...   53.606380   60.880401
 Hydrolases [EC 3.-]                                2.871770   12.777368
Hydrolases -  Acting on ester bonds [EC 3.1.-]      189.536843  239.763553
Hydrolases -  Acting on carbon-sulfur bonds [EC...    0.000000    1.503220
Hydrolases -  Glycosylases [EC 3.2.-]               39.247528   24.051516
Hydrolases -  Acting on ether bonds [EC 3.3.-]      11.487081    9.019319
Hydrolases -  Acting on peptide bonds (peptidas...  156.990112  190.157300
Hydrolases -  Acting on carbon-nitrogen bonds, ...  203.895695  153.328416
Hydrolases -  Acting on acid anhydrides [EC 3.6.-]  331.210847  338.224447
Hydrolases -  Acting on carbon-carbon bonds [EC...    8.615311    1.503220
Hydrolases -  Acting on phosphorus-nitrogen bon...    0.000000    0.000000
 Lyases [EC 4.-]                                    2.871770    4.509659
Lyases -  Carbon-carbon lyases [EC 4.1.-]           123.486125  131.531730
Lyases -  Carbon-oxygen lyases [EC 4.2.-]           201.981181  171.367053
Lyases -  Carbon-nitrogen lyases [EC 4.3.-]          50.734609   46.599813
Lyases -  Carbon-sulfur lyases [EC 4.4.-]            13.401595   13.528978
Lyases -  Phosphorus-oxygen lyases [EC 4.6.-]        15.316109    9.019319
Lyases -  Other lyases [EC 4.99.-]                    0.000000    0.000000
```

(continues on next page)

(continued from previous page)

```
Isomerases -  Racemases and epimerases [EC 5.1.-]    71.794259   51.861082
Isomerases -  Cis-trans-isomerases [EC 5.2.-]         4.786284    9.770928
Isomerases -  Intramolecular oxidoreductases [E...   76.580543   50.357862
Isomerases -  Intramolecular transferases [EC 5...   71.794259   92.448016
Isomerases -  Intramolecular lyases [EC 5.5.-]       13.401595    5.261269
Isomerases -  Other isomerases [EC 5.99.-]           84.238597   75.160988
 Ligases [EC 6.-]                                     0.000000    0.000000
Ligases -  Forming carbon-oxygen bonds [EC 6.1.-]   386.731740  450.214320
Ligases -  Forming carbon-sulfur bonds [EC 6.2.-]   146.460288  129.276900
Ligases -  Forming carbon-nitrogen bonds [EC 6....  218.254546  222.476525
Ligases -  Forming carbon-carbon bonds [EC 6.4.-]    19.145136   19.541857
Ligases -  Forming phosphoric ester bonds [EC 6...   44.991069   78.919038

                                                          01m         50m
 Oxidoreductases [EC 1.-]                            47.001806   64.815812
Oxidoreductases -  Acting on the CH-OH group of...  215.560008  161.595587
Oxidoreductases -  Acting on a peroxide as acce...    0.000000    3.551551
Oxidoreductases -  Acting on hydrogen as donors...   51.864062   46.170168
Oxidoreductases -  Acting on single donors with...    6.483008    3.551551
Oxidoreductases -  Acting on paired donors, wit...    0.000000    1.775776
Oxidoreductases -  Acting on superoxide as acce...    0.000000    2.663664
Oxidoreductases -  Oxidizing metal ions [EC 1.1...    3.241504   11.542542
Oxidoreductases -  Acting on CH or CH(2) groups...  102.107372   43.506504
Oxidoreductases -  Acting on iron-sulfur protei...   11.345264   37.291289
Oxidoreductases -  Acting on the aldehyde or ox...  153.971434  139.398391
Oxidoreductases -  Catalyzing the reaction X-H ...    0.000000    0.000000
Oxidoreductases -  Acting on the CH-CH group of...   98.865868   63.927924
Oxidoreductases -  Acting on the CH-NH(2) group...   30.794287   36.403401
Oxidoreductases -  Acting on the CH-NH group of...   16.207519    2.663664
Oxidoreductases -  Acting on NADH or NADPH [EC ...   53.484814   59.488485
Oxidoreductases -  Acting on other nitrogenous ...    9.724512    1.775776
Oxidoreductases -  Acting on a sulfur group of ...  111.831884  128.743737
Oxidoreductases -  Acting on a heme group of do...    0.000000    0.000000
Oxidoreductases -  Other oxidoreductases [EC 1....    4.862256    6.215215
 Transferases [EC 2.-]                                9.724512    0.000000
Transferases -  Transferring one-carbon groups ...  222.043016  198.886876
Transferases -  Transferring molybdenum- or tun...    3.241504    0.000000
Transferases -  Transferring aldehyde or ketoni...   21.069775   25.748747
Transferases -  Acyltransferases [EC 2.3.-]          87.520605  146.501494
Transferases -  Glycosyltransferases [EC 2.4.-]     129.660155  150.053045
Transferases -  Transferring alkyl or aryl grou...  147.488427  114.537531
Transferases -  Transferring nitrogenous groups...  144.246923   66.591588
Transferases -  Transferring phosphorus-contain...  546.193404  674.794758
Transferases -  Transferring sulfur-containing ...   61.588574   71.031027
 Hydrolases [EC 3.-]                                  3.241504    1.775776
Hydrolases -  Acting on ester bonds [EC 3.1.-]      152.350682  227.299287
Hydrolases -  Acting on carbon-sulfur bonds [EC...    6.483008    0.000000
Hydrolases -  Glycosylases [EC 3.2.-]                47.001806   31.963962
Hydrolases -  Acting on ether bonds [EC 3.3.-]       11.345264    9.766766
Hydrolases -  Acting on peptide bonds (peptidas...  163.695946  233.514502
Hydrolases -  Acting on carbon-nitrogen bonds, ...  149.109179  191.783773
Hydrolases -  Acting on acid anhydrides [EC 3.6.-]  398.704977  350.715697
Hydrolases -  Acting on carbon-carbon bonds [EC...   27.552783    4.439439
Hydrolases -  Acting on phosphorus-nitrogen bon...    0.000000    4.439439
 Lyases [EC 4.-]                                     12.966016   15.094093
Lyases -  Carbon-carbon lyases [EC 4.1.-]           134.522411  107.434429
Lyases -  Carbon-oxygen lyases [EC 4.2.-]           225.284520  133.183176
Lyases -  Carbon-nitrogen lyases [EC 4.3.-]          27.552783   39.067065
Lyases -  Carbon-sulfur lyases [EC 4.4.-]             0.000000    1.775776
Lyases -  Phosphorus-oxygen lyases [EC 4.6.-]         4.862256    6.215215
Lyases -  Other lyases [EC 4.99.-]                    3.241504    0.000000
```

(continues on next page)

(continued from previous page)

```
Isomerases -  Racemases and epimerases [EC 5.1.-]     43.760302   31.076074
Isomerases -  Cis-trans-isomerases [EC 5.2.-]          1.620752   14.206205
Isomerases -  Intramolecular oxidoreductases [E...    34.035791   78.134130
Isomerases -  Intramolecular transferases [EC 5...    48.622558   85.237233
Isomerases -  Intramolecular lyases [EC 5.5.-]        19.449023   15.981981
Isomerases -  Other isomerases [EC 5.99.-]            59.967822   55.936934
 Ligases [EC 6.-]                                      3.241504    0.000000
Ligases -  Forming carbon-oxygen bonds [EC 6.1.-]    333.874900  338.285267
Ligases -  Forming carbon-sulfur bonds [EC 6.2.-]    124.797899  190.895886
Ligases -  Forming carbon-nitrogen bonds [EC 6....   270.665574  198.886876
Ligases -  Forming carbon-carbon bonds [EC 6.4.-]     32.415039    6.215215
Ligases -  Forming phosphoric ester bonds [EC 6...    68.071581  107.434429
```

[58]:
```
ec_counts.to_csv('{}ec_counts.csv'.format(data_dir))
```

[59]:
```
fig, ax = mgkit.plots.get_single_figure(figsize=(15, 12))
_ = mgkit.plots.boxplot.boxplot_dataframe(
    ec_counts,
    plot_order,
    ax,
    # a dictionary with options related to the labels
    # on both the X and Y axes. In this case it changes
    # the size of the labels and the rotation - the default
    # is 'vertical', as the box_vert=True by default
    fonts=dict(fontsize=12, rotation='horizontal'),
    data_colours={
        x: color
        for x, color in zip(plot_order, seaborn.color_palette('hls', len(plot_
↪order)))
    },
    # Changes the direction of the boxplot. The rotation of
    # the labels must be set to 'horizontal' in the *fonts*
    # dictionary
    box_vert=False
)
# Adds labels to the axes
ax.set_xlabel('Counts', fontsize=16)
ax.set_ylabel('Enzyme Class', fontsize=16)
# Ensure the correct layout before writing to disk
fig.set_tight_layout(True)
# Saves a PDF file, or any other supported format by matplotlib
fig.savefig('{}ec_counts-boxplot.pdf'.format(data_dir))
```

```
2018-05-22 14:10:25,953 - DEBUG - matplotlib.font_manager->findfont: findfont:␣
↪Matching␣
↪:family=sans-serif:style=normal:variant=normal:weight=normal:stretch=normal:size=12.
↪0 to DejaVu Sans (u'/home/frubino/mgkit/dev-env/local/lib/python2.7/
↪site-packages/matplotlib/mpl-data/fonts/ttf/DejaVuSans.ttf') with score of 0.
↪050000
2018-05-22 14:10:26,264 - DEBUG - matplotlib.backends.backend_pdf->fontName:␣
↪Assigning font /F1 = u'/home/frubino/mgkit/dev-env/local/lib/python2.7/
↪site-packages/matplotlib/mpl-data/fonts/ttf/DejaVuSans.ttf'
2018-05-22 14:10:26,958 - DEBUG - matplotlib.backends.backend_pdf->writeFonts:␣
↪Embedding font /home/frubino/mgkit/dev-env/local/lib/python2.7/site-packages/
↪matplotlib/mpl-data/fonts/ttf/DejaVuSans.ttf.
2018-05-22 14:10:26,960 - DEBUG - matplotlib.backends.backend_pdf->writeFonts:␣
↪Writing TrueType font.
```

3.2.5 Explore Diversity

Diversity in metagenomic samples can be analysed using pN/pS values. The data required to do this was produced in the tutorial by the *snp_parser* script. Here are some examples of how to calculate diversity estimates from this data.

The complete toolset to map diversity estimates can be found in the **mgkit.snps** package, with the *mgkit.snps.funcs.combine_sample_snps* function building the final pandas DataFrame. As the use of the function requires the initialisation of different functions, a few easier to use ones are available in the **mgkit.snps.conv_func** module:

- get_rank_dataframe
- get_gene_map_dataframe
- get_full_dataframe
- get_gene_taxon_dataframe

The first is used to get diversity estimates for taxa, the second for genes/functions. The other two provides functionality to return estimates tied to both taxon and function.

Taxa

[60]:
```
# Sets the minimum coverage for an annotation to be
# included into the table (defaults to 4)
mgkit.consts.DEFAULT_SNP_FILTER['min_cov'] = 2
```

[61]:
```
# To get diversity estimates for taxa *mgkit.snps.conv_func.get_rank_dataframe* can be used
# It is also imported and accesible from the *mgkit.snps* package
pnps = mgkit.snps.get_rank_dataframe(snp_data, taxonomy, min_num=3, rank='order', index_type='taxon')
```

```
2018-05-22 14:10:34,437 - INFO - mgkit.snps.funcs->combine_sample_snps: Analysing SNP from sample SRR001322
2018-05-22 14:10:34,491 - INFO - mgkit.snps.funcs->combine_sample_snps: Analysing SNP from sample SRR001323
2018-05-22 14:10:34,552 - INFO - mgkit.snps.funcs->combine_sample_snps: Analysing SNP from sample SRR001325
2018-05-22 14:10:34,610 - INFO - mgkit.snps.funcs->combine_sample_snps: Analysing SNP from sample SRR001326
```

[62]:
```
pnps = pnps.rename(
    columns=sample_names,
    index=lambda x: taxonomy[x].s_name
)
```

[63]:
```
pnps.describe()
```

[63]:

	16m	32m	01m	50m
count	83.000000	83.000000	86.0	75.000000
mean	0.027889	0.035167	0.0	0.020063
std	0.086763	0.136137	0.0	0.100393
min	0.000000	0.000000	0.0	0.000000
25%	0.000000	0.000000	0.0	0.000000
50%	0.000000	0.000000	0.0	0.000000
75%	0.000000	0.000000	0.0	0.000000
max	0.360621	0.882326	0.0	0.604127

[64]:
```
pnps
```

[64]:

	16m	32m	01m	50m
Rubrobacterales	0.000000	0.000000	0.0	0.000000
Sphaerobacterales	0.000000	0.000000	0.0	NaN
Bifidobacteriales	0.000000	0.000000	0.0	0.000000
Micrococcales	0.000000	0.000000	0.0	0.000000
Neisseriales	0.000000	0.000000	0.0	0.000000
Micromonosporales	0.000000	NaN	0.0	0.000000
Pseudonocardiales	0.000000	0.000000	0.0	NaN
Streptomycetales	0.000000	0.000000	0.0	0.000000
Streptosporangiales	0.000000	0.000000	0.0	0.000000
Haloferacales	0.000000	0.000000	0.0	0.000000
Bacteroidales	0.000000	0.000000	0.0	0.000000
Mycoplasmatales	0.000000	0.000000	0.0	0.000000
Corynebacteriales	0.000000	0.000000	0.0	0.000000
Schizosaccharomycetales	0.306546	NaN	0.0	0.000000
Clostridiales	NaN	0.000000	0.0	0.000000
Rhodocyclales	0.000000	0.000000	0.0	NaN
Thiotrichales	NaN	0.000000	0.0	0.000000
Pseudomonadales	0.000000	0.000000	0.0	0.000000
Legionellales	0.314474	0.000000	0.0	0.000000
Chlamydiales	0.000000	0.000000	0.0	0.000000
Chroococcales	0.000000	0.000000	0.0	NaN
Methanobacteriales	0.301844	0.304830	0.0	0.000000
Planctomycetales	0.000000	0.000000	0.0	0.000000
Synechococcales	0.000000	0.000000	0.0	0.397696
Desulfovibrionales	0.000000	0.000000	0.0	0.000000
Cenarchaeales	0.000000	0.000000	NaN	0.000000
Oscillatoriales	0.000000	0.000000	0.0	0.000000

(continues on next page)

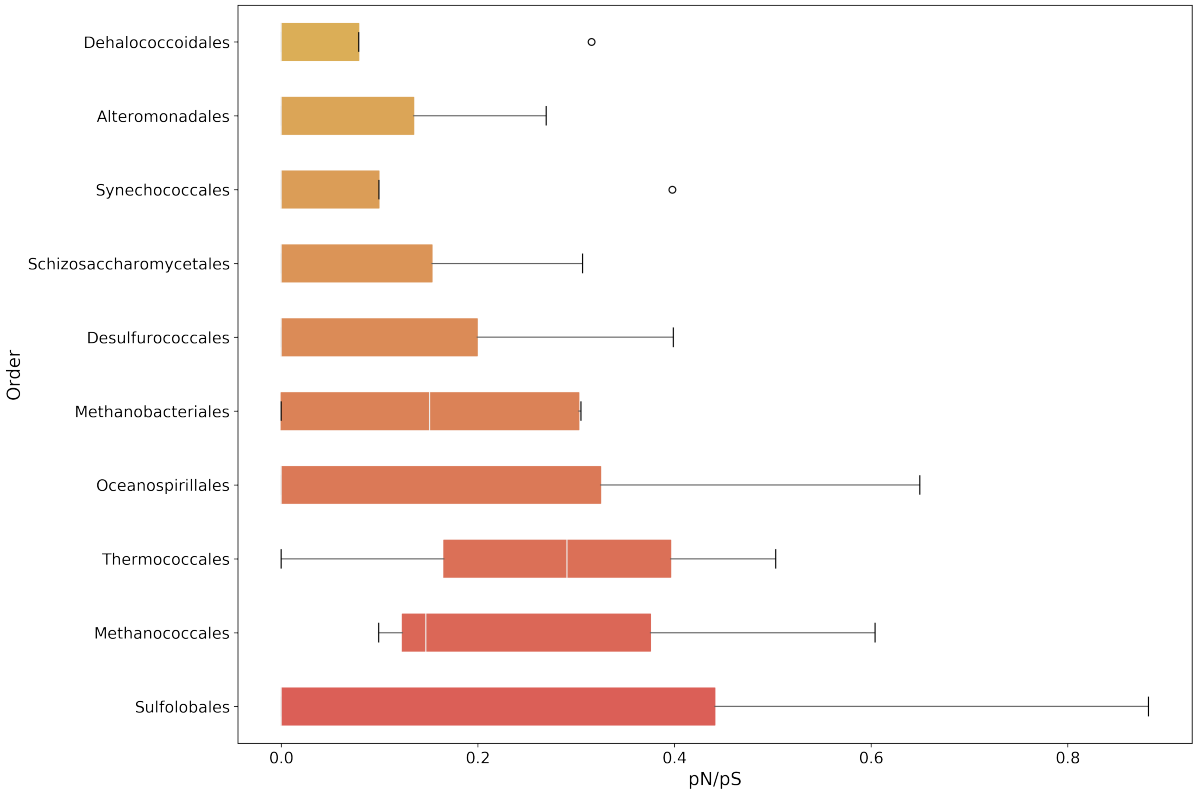
(continued from previous page)

```
Methanococcales          0.099162  0.147193  NaN  0.604127
Spirochaetales           0.000000  0.000000  0.0  0.000000
Nostocales               0.000000  0.000000  0.0       NaN
...                           ...       ...  ...       ...
Herpetosiphonales        0.000000  0.000000  0.0  0.000000
Thermomicrobiales        0.000000       NaN  0.0  0.000000
Nitrospirales            0.000000  0.000000  0.0  0.000000
Campylobacterales        0.000000  0.000000  0.0  0.000000
Acidithiobacillales      0.000000  0.000000  0.0  0.000000
Parachlamydiales         0.000000  0.000000  0.0       NaN
Thermotogales            0.000000  0.000000  0.0  0.000000
Methanopyrales           0.000000  0.000000  0.0  0.000000
Solibacterales           0.000000  0.000000  0.0       NaN
Cellvibrionales          0.000000  0.000000  0.0       NaN
Natranaerobiales         0.000000  0.000000  0.0  0.000000
Gloeobacterales          0.000000  0.000000  0.0  0.000000
Methanocellales          0.000000  0.000000  0.0  0.000000
Aquificales              0.000000  0.000000  0.0  0.000000
Petrotogales             0.000000  0.000000  0.0       NaN
Eurotiales               0.000000  0.000000  0.0  0.000000
Chlorobiales             0.000000  0.000000  0.0  0.000000
Onygenales                    NaN  0.000000  0.0  0.000000
Chromatiales             0.000000  0.000000  0.0  0.000000
Xanthomonadales          0.000000  0.000000  0.0  0.000000
Oceanospirillales             NaN  0.649600  0.0  0.000000
Flavobacteriales         0.000000  0.000000  0.0  0.000000
Alteromonadales          0.269426       NaN  0.0  0.000000
Vibrionales              0.000000  0.000000  0.0  0.000000
Aeromonadales            0.000000  0.000000  0.0       NaN
Pasteurellales           0.299672  0.000000  0.0  0.000000
Syntrophobacterales      0.000000  0.000000  0.0  0.000000
Rickettsiales            0.000000  0.000000  0.0  0.000000
Methanosarcinales        0.059651       NaN  0.0  0.000000
Cytophagales             0.000000  0.000000  0.0       NaN

[90 rows x 4 columns]
```

[65]:
```
pnps.to_csv('{}pnps-taxa.csv'.format(data_dir))
```

[66]:
```
#sort the DataFrame to plot them by mean value
plot_order = pnps.mean(axis=1).sort_values(inplace=False, ascending=False).
↪index[:10]

fig, ax = mgkit.plots.get_single_figure(figsize=(15, 10))
_ = mgkit.plots.boxplot.boxplot_dataframe(
    pnps,
    plot_order,
    ax,
    fonts=dict(fontsize=14, rotation='horizontal'),
    data_colours={
        x: color
        for x, color in zip(plot_order, seaborn.color_palette('hls', len(pnps.
↪index)))
    },
    box_vert=False
)
ax.set_xlabel('pN/pS', fontsize=16)
ax.set_ylabel('Order', fontsize=16)
fig.set_tight_layout(True)
fig.savefig('{}pnps-taxa-boxplot.pdf'.format(data_dir))
```

```
/home/frubino/mgkit/dev-env/local/lib/python2.7/site-packages/numpy/core/
↪fromnumeric.py:52: FutureWarning: reshape is deprecated and will raise in a␣
↪subsequent release. Please use .values.reshape(...) instead
  return getattr(obj, method)(*args, **kwds)
2018-05-22 14:10:34,970 - DEBUG - matplotlib.backends.backend_pdf->fontName:␣
↪Assigning font /F1 = u'/home/frubino/mgkit/dev-env/local/lib/python2.7/
↪site-packages/matplotlib/mpl-data/fonts/ttf/DejaVuSans.ttf'
2018-05-22 14:10:35,015 - DEBUG - matplotlib.backends.backend_pdf->writeFonts:␣
↪Embedding font /home/frubino/mgkit/dev-env/local/lib/python2.7/site-packages/
↪matplotlib/mpl-data/fonts/ttf/DejaVuSans.ttf.
2018-05-22 14:10:35,016 - DEBUG - matplotlib.backends.backend_pdf->writeFonts:␣
↪Writing TrueType font.
```



Functional Categories

[67]:
```
# To get diversity estimates of functions, *mgkit.snps.conv_func.get_gene_map_
↪dataframe* can be used
# This is available in the *mgkit.snps* package as well
fc_pnps = mgkit.snps.get_gene_map_dataframe(snp_data, taxonomy, min_num=3, gene_
↪map=fc_map, index_type='gene')
```

```
2018-05-22 14:10:38,138 - INFO - mgkit.snps.funcs->combine_sample_snps: Analysing␣
↪SNP from sample SRR001322
2018-05-22 14:10:38,174 - INFO - mgkit.snps.funcs->combine_sample_snps: Analysing␣
↪SNP from sample SRR001323
2018-05-22 14:10:38,215 - INFO - mgkit.snps.funcs->combine_sample_snps: Analysing␣
↪SNP from sample SRR001325
2018-05-22 14:10:38,256 - INFO - mgkit.snps.funcs->combine_sample_snps: Analysing␣
↪SNP from sample SRR001326
```

[68]:
```
fc_pnps = fc_pnps.rename(
    columns=sample_names,
```

(continues on next page)

(continued from previous page)

```
    index=eggnog.EGGNOG_CAT
)
```

[69]:
```
fc_pnps.describe()
```

[69]:
```
              16m        32m   01m        50m
count  18.000000  15.000000  18.0  19.000000
mean    0.106591   0.195602   0.0   0.085110
std     0.149325   0.257863   0.0   0.173999
min     0.000000   0.000000   0.0   0.000000
25%     0.000000   0.000000   0.0   0.000000
50%     0.000000   0.076031   0.0   0.000000
75%     0.275874   0.225485   0.0   0.127011
max     0.410372   0.761215   0.0   0.701625
```

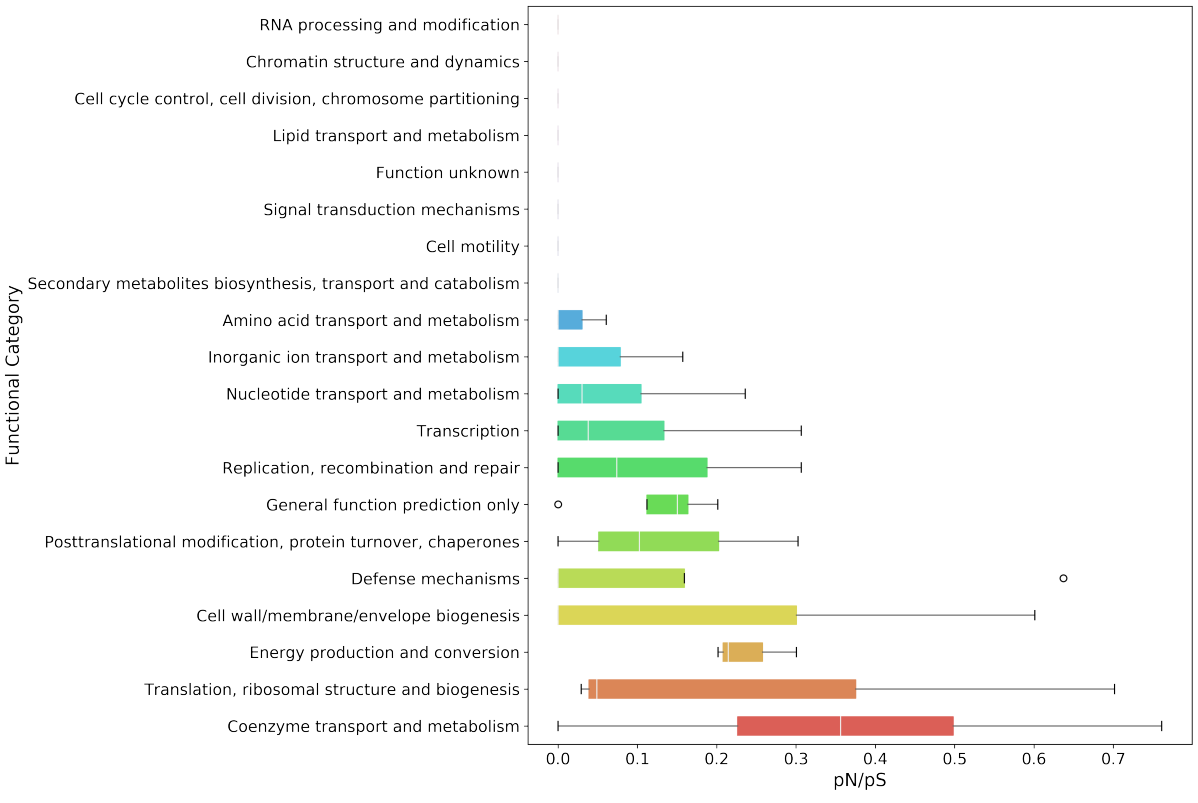
[70]:
```
fc_pnps
```

[70]:
```
                                                   16m       32m  01m  \
RNA processing and modification                    NaN  0.000000  0.0
Energy production and conversion              0.300669  0.214854  NaN
Chromatin structure and dynamics              0.000000  0.000000  0.0
Amino acid transport and metabolism                NaN  0.060911  0.0
Cell cycle control, cell division, chromosome p...  0.000000       NaN  0.0
Nucleotide transport and metabolism           0.060956  0.236117  0.0
Lipid transport and metabolism                0.000000       NaN  0.0
Coenzyme transport and metabolism             0.410372  0.761215  0.0
Transcription                                 0.306611  0.076031  0.0
Translation, ribosomal structure and biogenesis  0.029312  0.048914  NaN
Cell wall/membrane/envelope biogenesis        0.000000  0.601078  0.0
Replication, recombination and repair         0.306580  0.148139  0.0
Posttranslational modification, protein turnove...  0.302645       NaN  0.0
Cell motility                                 0.000000  0.000000  0.0
Secondary metabolites biosynthesis, transport a...  0.000000  0.000000  0.0
Inorganic ion transport and metabolism        0.000000       NaN  0.0
Function unknown                              0.000000       NaN  0.0
General function prediction only              0.201490  0.149640  0.0
Signal transduction mechanisms                0.000000  0.000000  0.0
Defense mechanisms                            0.000000  0.637127  0.0

                                                   50m
RNA processing and modification               0.000000
Energy production and conversion              0.201721
Chromatin structure and dynamics              0.000000
Amino acid transport and metabolism           0.000000
Cell cycle control, cell division, chromosome p...  0.000000
Nucleotide transport and metabolism           0.000000
Lipid transport and metabolism                0.000000
Coenzyme transport and metabolism             0.302295
Transcription                                 0.000000
Translation, ribosomal structure and biogenesis  0.701625
Cell wall/membrane/envelope biogenesis             NaN
Replication, recombination and repair         0.000000
Posttranslational modification, protein turnove...  0.102591
Cell motility                                 0.000000
Secondary metabolites biosynthesis, transport a...  0.000000
Inorganic ion transport and metabolism        0.157425
Function unknown                              0.000000
General function prediction only              0.151431
Signal transduction mechanisms                0.000000
Defense mechanisms                            0.000000
```

[71]:
```
fc_pnps.to_csv('{}pnps-fc.csv'.format(data_dir))
```

[72]:
```
#sort the DataFrame to plot them by median value
plot_order = fc_pnps.mean(axis=1).sort_values(inplace=False, ascending=False).index

fig, ax = mgkit.plots.get_single_figure(figsize=(15, 10))
_ = mgkit.plots.boxplot.boxplot_dataframe(
    fc_pnps,
    plot_order,
    ax,
    fonts=dict(fontsize=14, rotation='horizontal'),
    data_colours={
        x: color
        for x, color in zip(plot_order, seaborn.color_palette('hls', len(fc_pnps.
↪index)))
    },
    box_vert=False
)
ax.set_xlabel('pN/pS', fontsize=16)
ax.set_ylabel('Functional Category', fontsize=16)
fig.set_tight_layout(True)
fig.savefig('{}pnps-fc-boxplot.pdf'.format(data_dir))
```

```
2018-05-22 14:10:38,683 - DEBUG - matplotlib.backends.backend_pdf->fontName:␣
↪Assigning font /F1 = u'/home/frubino/mgkit/dev-env/local/lib/python2.7/
↪site-packages/matplotlib/mpl-data/fonts/ttf/DejaVuSans.ttf'
2018-05-22 14:10:38,777 - DEBUG - matplotlib.backends.backend_pdf->writeFonts:␣
↪Embedding font /home/frubino/mgkit/dev-env/local/lib/python2.7/site-packages/
↪matplotlib/mpl-data/fonts/ttf/DejaVuSans.ttf.
2018-05-22 14:10:38,778 - DEBUG - matplotlib.backends.backend_pdf->writeFonts:␣
↪Writing TrueType font.
```



Enzyme Classification

[73]:
```
ec_map = {
    # Using only the first level
    annotation.gene_id: set(x.replace('.-', '') for x in annotation.get_
↪ec(level=1))
    for annotation in annotations.itervalues()
}
```

[74]:
```
ec_pnps = mgkit.snps.get_gene_map_dataframe(snp_data, taxonomy, min_num=3, gene_
↪map=ec_map, index_type='gene')
```

```
2018-05-22 14:10:42,201 - INFO - mgkit.snps.funcs->combine_sample_snps: Analysing␣
↪SNP from sample SRR001322
2018-05-22 14:10:42,238 - INFO - mgkit.snps.funcs->combine_sample_snps: Analysing␣
↪SNP from sample SRR001323
2018-05-22 14:10:42,277 - INFO - mgkit.snps.funcs->combine_sample_snps: Analysing␣
↪SNP from sample SRR001325
2018-05-22 14:10:42,319 - INFO - mgkit.snps.funcs->combine_sample_snps: Analysing␣
↪SNP from sample SRR001326
```

[75]:
```
# Rename columns and row. Rows will include the full label the enzyme class
ec_pnps = ec_pnps.rename(
    index=lambda x: "{} {} [EC {}.-]".format(
        # A name of the second level doesn't include the first level
        # definition, so if it is level 2, we add the level 1 label
        '' if len(x) == 1 else ec_names[x[0]] + " - ",
        # The EC label for the specific class (e.g. 3.2)
        ec_names[x],
        # The EC number
        x
    ),
    columns=sample_names
)
```

[76]:
```
ec_pnps.describe()
```

[76]:

	16m	32m	01m	50m
count	3.000000	3.000000	2.0	3.000000
mean	0.351120	0.187620	0.0	0.173303
std	0.248257	0.139590	0.0	0.185190
min	0.100278	0.100437	0.0	0.000000
25%	0.228326	0.107119	0.0	0.075732
50%	0.356374	0.113802	0.0	0.151464
75%	0.476541	0.231211	0.0	0.259954
max	0.596708	0.348620	0.0	0.368444

[77]:
```
ec_pnps
```

[77]:

	16m	32m	01m	50m
Oxidoreductases [EC 1.-]	0.100278	0.113802	NaN	0.151464
Transferases [EC 2.-]	0.356374	0.348620	0.0	0.368444
Isomerases [EC 5.-]	0.596708	0.100437	0.0	0.000000

[78]:
```
ec_pnps.to_csv('{}pnps-ec.csv'.format(data_dir))
```

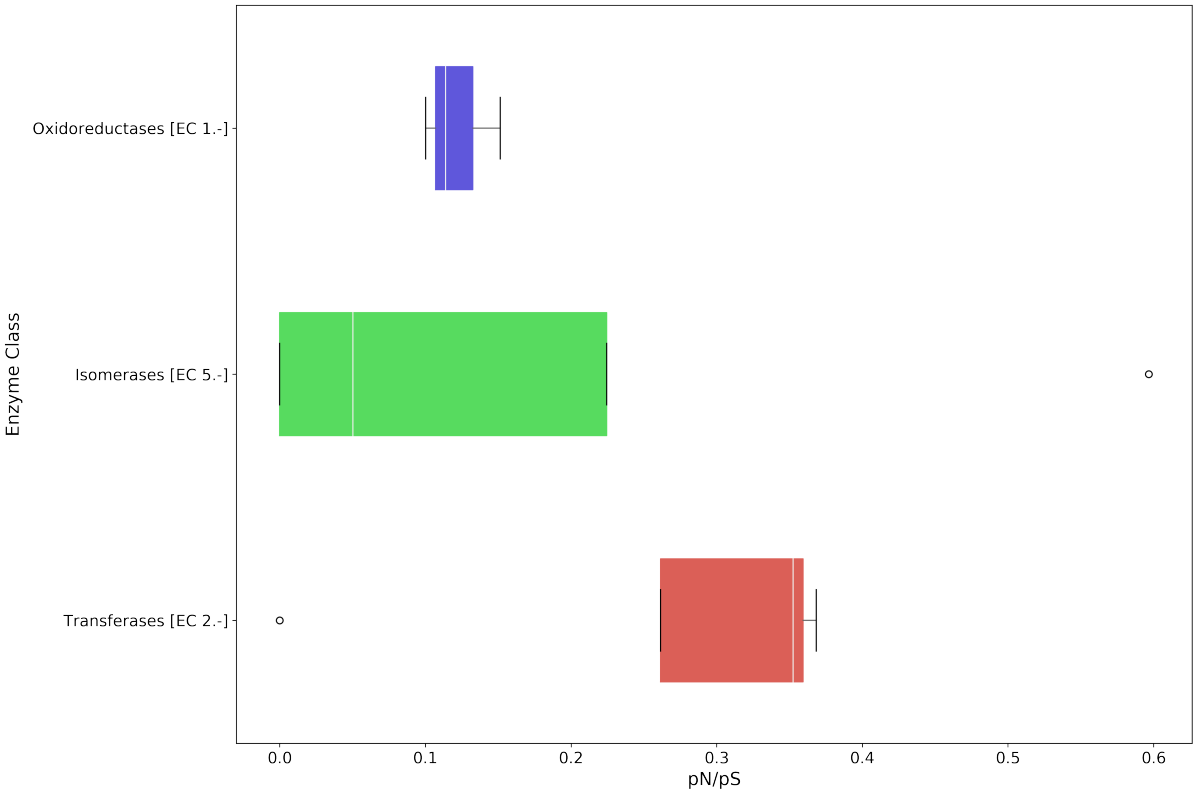
[79]:
```
#sort the DataFrame to plot them by median value
plot_order = ec_pnps.mean(axis=1).sort_values(inplace=False, ascending=False).index

fig, ax = mgkit.plots.get_single_figure(figsize=(15, 10))
_ = mgkit.plots.boxplot.boxplot_dataframe(
```

(continues on next page)

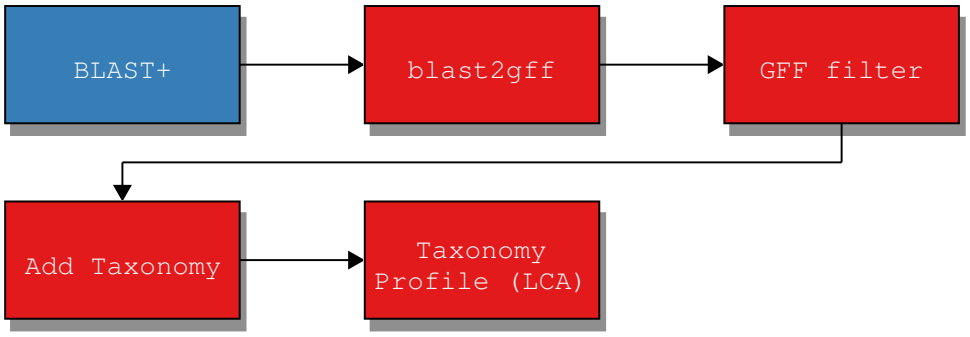
(continued from previous page)

```
    ec_pnps,
    plot_order,
    ax,
    fonts=dict(fontsize=14, rotation='horizontal'),
    data_colours={
        x: color
        for x, color in zip(plot_order, seaborn.color_palette('hls', len(plot_
↪order)))
    },
    box_vert=False
)
ax.set_xlabel('pN/pS', fontsize=16)
ax.set_ylabel('Enzyme Class', fontsize=16)
fig.set_tight_layout(True)
fig.savefig('{}pnps-ec-boxplot.pdf'.format(data_dir))
```

```
2018-05-22 14:10:43,431 - DEBUG - matplotlib.backends.backend_pdf->fontName:␣
↪Assigning font /F1 = u'/home/frubino/mgkit/dev-env/local/lib/python2.7/
↪site-packages/matplotlib/mpl-data/fonts/ttf/DejaVuSans.ttf'
2018-05-22 14:10:43,455 - DEBUG - matplotlib.backends.backend_pdf->writeFonts:␣
↪Embedding font /home/frubino/mgkit/dev-env/local/lib/python2.7/site-packages/
↪matplotlib/mpl-data/fonts/ttf/DejaVuSans.ttf.
2018-05-22 14:10:43,456 - DEBUG - matplotlib.backends.backend_pdf->writeFonts:␣
↪Writing TrueType font.
```



3.3 Profile a Community with BLAST



The above diagram shows the process of getting a community profile from a BLAST run against a DB of choice. The choice of DB is up to the user, but any DB that provides a NCBI *taxon_id* can be used. Such DBs include the ones provided by NCBI (e.g. nt, nr, viral) as well as Uniprot (SwissProt, TrEMBL).

The community profile will use an assembly and we want to assign each of its contigs to taxon. This can be done a BLAST output and a series of scripts that ends with the *lca* command of the *taxon_utils* script (*taxon-utils - Taxonomy Utilities*). *lca* stands for *last common ancestor*, which indicates that given a number of taxa, we try to resolve the taxon they all have in common. This can be of any level, from a specific strain to a kingdom, such as Bacteria.

There a cases when there's no *lca* that can be resolved and this is due to the way NCBI taxonomy is structured, with multiple top levels, such as *cellular organisms*, *viruses* and so on.

Note: Other DB may provide the *taxon_id* from NCBI, but this should be checked by the user

---

3.3.1 Considerations

Since the assembly of a metagenome is a time consuming process, the assembly from the hmmer-tutorial tutorial will be used. We'll try to use all results from the *nt* DB from NCBI, as well as separate *viruses* and *cellular organisms* and resolve the *lca* for those annotations separately.

Another thing to consider is how to filter the annotations. That's up to the user to decide which suits the specific task, but these options will be examined:

1. filter based on a static threshold, such as > 50 bitscore (using *filter-gff values*)
2. filter based on a dynamically chosen value, keeping only the X top options (using *filter-gff sequence*)
3. filter based on overlap (using *filter-gff overlap*)

The results may differ, and they are listed from the fastest to the slowest.

3.3.2 Requirements

MGKit should be installed and its scripts can be run from the command line. Refer to the installation for this, but it's assumed that it was installed with:

```
$ pip install mgkit[full]
```

Moreover, the tutorial makes use of UNIX command line utilities, so a version of GNU/Linux, *BSD or MacOS X should be used to run this tutorial. *BASH* is expected to be the shell running.

The following should be installed as well:

- BLAST+ (blastn will be used)
- ncftp (used to download the *NCBI nt* DB)

MacOS X

The software requirements can be installed with *homebrew*, using the following command:

```
$ brew install ncftp blast
```

3.3.3 Download Data

Assembly

TODO

NCBI nt

We'll be using the NCBI nt which will be stored in the *ncbi-nt* directory. If a copy is already somewhere, just create a symbolic link to that directory, for example:

```
$ ln -s path-to-the-db ncbi-nt
```

Otherwise, a copy can be downloaded and prepared with the following commands:

```
$ mkdir ncbi-nt
cd ncbi-nt
ncftpget ftp://ftp.ncbi.nlm.nih.gov/blast/db/nt*.gz
for x in *.tar.gz; do tar xfvz $x ;done
rm *.tar.gz
cd ..
```

ID to Taxonomy

The following file contains the *taxon_id* for all the IDs in the *NCBI nt* DB. It will be used to add taxonomic information before running the *lca* step:

```
$ wget ftp://ftp.ncbi.nih.gov/pub/taxonomy/accession2taxid/nucl_gb.accession2taxid.
↪gz
```

NCBI Taxonomy

This can be installed using the following script included in MGKit:

```
$ download-taxonomy.sh
```

Which will create a file called *taxonomy.pickle*

3.3.4 Community Profiling

To make the tutorial faster, we'll filter the assembly file to include only contigs of at least 500bp:

```
$ python - <<END
from mgkit.io import fasta
with open('final-contigs-filt.fa', 'w') as f:
    for name, seq in fasta.load_fasta('final-contigs.fa'):
        if len(seq) >= 500:
            fasta.write_fasta_sequence(f, name, seq)
END
```

BLAST

The *blastn* command will be used to search for similar sequences in the *NCBI nt* DB. The following command will create a tab separated file with the results:

```
$ blastn -query final-contigs-filt.fa -db ncbi-nt/nt -outfmt 6 -out assembly-nt.
↪tab -evalue 0.001
```

Convert into a GFF

The following command will create a GFF file from the BLAST output:

```
$ blast2gff blastdb -i 3 -r assembly-nt.tab assembly-nt.gff
```

We're using the *blastdb* command of the *blast2gff* command, since it gives more control over the way the header file is formatted:

```
gi|118501159|gb|CP000482.1|
```

At the moment, the header format of the *NCBI nt* DB is a | (pipe) list that contains two type of identifiers. The first element is *gi*, to indicate that the following element (second) is the GI identifier that it's being retired in September 2016. The third is indicates the DB from where the other ID originates from (GenBank in this case) and the fourth is the identifier that we'll use.

By default *blast2gff blastdb* used the second element (*118501159*) of the header as *gene_id*, so we use:

1. *-i 3* to instead use the fourth element (*CP000482.1*) as *gene_id*
2. *-r* will remove the versioning information from the *gene_id*, so *CP000482.1* will become *CP000482*

The reason for this is that the file containing the *taxon_id* for each identifier is better used with a fourth element of the header without the versioning information.

Adding the Taxonomic Information

The *add-gff-info addtaxa* command allows to insert taxonomic information (in the GFF *taxon_id* attribute) into the GFF file. This step integrates the content of the *nucl_gb.accession2taxid.gz* file with the GFF file. The structure of this file is:

ACCESSION ACCESSION.VERSION TAXONID GI

Warning: this command has to load all the GFF in memory, so a high memory machine should be used (~30GB). The GFF can be split into smaller files to save memory and the subsection here will describe the process.

Since we used the *ACCESSION* as *gene_id*, we need to edit the file to pass it to the *add-gff-info addtaxa* command *-t* option. This can be don on the fly and the following command adds information to the GFF file created:

```
$ add-gff-info addtaxa -t <(gunzip -c nucl_gb.accession2taxid.gz | cut -f 1,3) -e␣
↪assembly-nt.gff assembly-nt-taxa.gff; mv assembly-nt-taxa.gff assembly-nt.gff
```

The *-t* option is the file that can contains the *taxon_id* for each *gene_id*, the script accept a tab separated file. After the this we rename the output file to keep less files around. The *-e* option was used to remove from the output file any annotation for which a *taxon_id* was not found. Since we need them for the LCA later, it makes sense to remove them before filtering.

Reduce Memory Usage

First we need to split the *assembly-nt.gff* file, with a good option being using the *split* command in Unix. The following command will create the files:

```
$ split -l 1000000 -d assembly-nt.gff split-gff
```

This command will create 12 GFF files (of at most 1 milion lines each), whose names start with *split-gff*. Since we split the files we can use a loop to add the taxonomic information to all of them:

```
$ for x in split-gff*; do
add-gff-info addtaxa -t <(gunzip -c nucl_gb.accession2taxid.gz | cut -f 1,3) -e $x
↪$x-taxa;
done
```

This reduces the memory usage to ~2.5GB, but it takes longer to re-read the *nucl_gb.accession2taxid.gz* 12 times. There are way to parallelise it, but they are beyond the scope of this tutorial.

After the command has finished running, the content of the files can be concatenated into a single file again and delete the split files:

```
$ cat split-gff*-taxa > assembly-nt.gff; rm split-gff*
```

Filter the GFF

As mentioned we'll provide three different way to filter a GFF, before passing it to the script that will output the *lca* information. This way we can compare the different filtering strategies.

Filter by Value

Let's assume a scenario where we're working on reads or very short contigs. We may decide to use a threshold, so the filtering is fast, but doesn't compromise the quality of the assignment. This can achieved using the *filter-gff values* command:

```
$ filter-gff values -b 50 assembly-nt.gff assembly-nt_filt-value.gff
```

The command will read the GFF file and keep only the hits that are greater than or equal to 50, which we're assuming is a good compromise for the assignment. This filtering strategy has the advantage of operating on a per-annotation basis, so the memory usage is low and no grouping or calculation is required.

Filter Dynamically

While the above can be give good results, we can think of cases where the number of hits that pass that threshold may be high (e.g. a conserved sequence in multiple organisms). In this case a more sensible choice would be to keep only the hits that are in the top 5-10% of the hits on that contig, all those over the median, mean or any other

threshold based on the distribution of a sequence's hits. The *filter-gff sequence* command can be used to filter the GFF:

```
$ filter-gff sequence -t -q .95 -c ge assembly-nt.gff assembly-nt_filt-sequence.gff
```

The options used will keep only the hits that have a bitscore (evalue and identity can also be used) greater than or equal to the top 5% of the bitscore distribution for that contig.

This threshold will include also contigs that have only one hit (that's the reason to use *-c ge* instead of *-c gt*). We also assume that the input GFF is sorted (*-t* option) by contig name, to use less memory.

Filter Ovelaps

Let's assume that in some cases the we think there may be cases where the contig contains regions that different rates of conservation. The first filter may keep too many taxa with similar sequences in a portion of the contig, while the second one may not provide enough coverage of the contig, keeping only the very best hits.

In this case, we can use the *filter-gff overlap* command to keep of all overlapping hits only the best one. And since we want to make sure that we still have good homology, we could still filter by value the hits, before that filter.

The following command will make that type of filtering:

```
$ filter-gff values -b 50 assembly-nt.gff | sort -s -k 1,1 -k 7,7 | filter-gff␣
↪overlap -t -s 1 - assembly-nt_filt-overlap.gff
```

We just chained the filtering from the *values* command, keeping only annotations with at least 50 bitscore and passing it to the sort command. This passage is not necessary it the the *-t* option is not used with *filter-gff overlap*, but it uses less memory by pre-sorting the GFF by contig/strand first, since the *filter-gff overlap* works on each strand separately. We also used the *-s* options to trigger the filter for annotations that overlap for as much as 1 bp.

More information about this type of filter can be found in simple-tutorial and *filter-gff - Filter GFF annotations*.

Getting the Profile

We'll have 3 GFF files ending in *final.gff*, one per each type of filtering, that contain the *taxon_id* for each annotation they contain.

Note: these files are available at this page[38] if you want to skip

---

Since the filtered files are available now, we can create a file that contains the LCA assignments. We can ouput 2 type of files (see *taxon-utils - Taxonomy Utilities*), but for the purpose of this tutorial, we'll get a GFF file that we can also use in a assembly viewer. The command to create them is:

```
$ for x in *filt-*.gff; do
taxon_utils lca -v -t taxonomy.pickle -r final-contigs-filt.fa -s -n `basename $x .
↪gff`-nolca.tab -ft LCA-`echo $x | egrep -o 'value|overlap|sequence' | tr␣
↪[:lower:] [:upper:]` $x `basename $x .gff`-lca.gff;
done
```

The options used are:

- *-t* to direct the script to the taxonomy that we already downloaded
- *-r* to output a GFF with one annotation per contig that covers the whole sequence
- *-s* indicates that the input is sorted by reference sequence
- *-n* outputs a tab separated file with the contigs that could not be assigned

[38] http://bitbucket.org

- *-ft* is used to change the *feature type* column in the GFF, from the dafault *LCA* to one which includes the type of filtering used

The file ending in *-nolca.tab* contain the contigs that could not be assigned, while the files ending in *-lca.gff* contain the taxonomic assignments, with the *taxon_id* pointing to the assigned taxon identifier, *taxon_name* for the taxon scientific name (or common name if none is found) and *lineage* contains the whole lineage of the taxon.

Using Krona

Besides having a file with the assignments and a GFF that can be used in Tablet, a quick profile can be produced using Krona[39] and its associated *Krona Tools*. To produce a file that can be used with Krona Tools the **-k** can be used with the *taxon_utils lca* command. An additional option is to give the tool the total number of sequences in the assembly with the **-kt** option, to have a complete profile of the assembly:

```
$ for x in *filt-{overlap,sequence,value}.gff; do
taxon_utils lca -v -t taxonomy.pickle -k -kt `grep -c '>' final-contigs-filt.fa` -
↪s $x `basename $x .gff`-lca.krona;
done
```

To the *-kt* option was passed the total number of sequences (just used grep to count how many headers are in the fasta file).

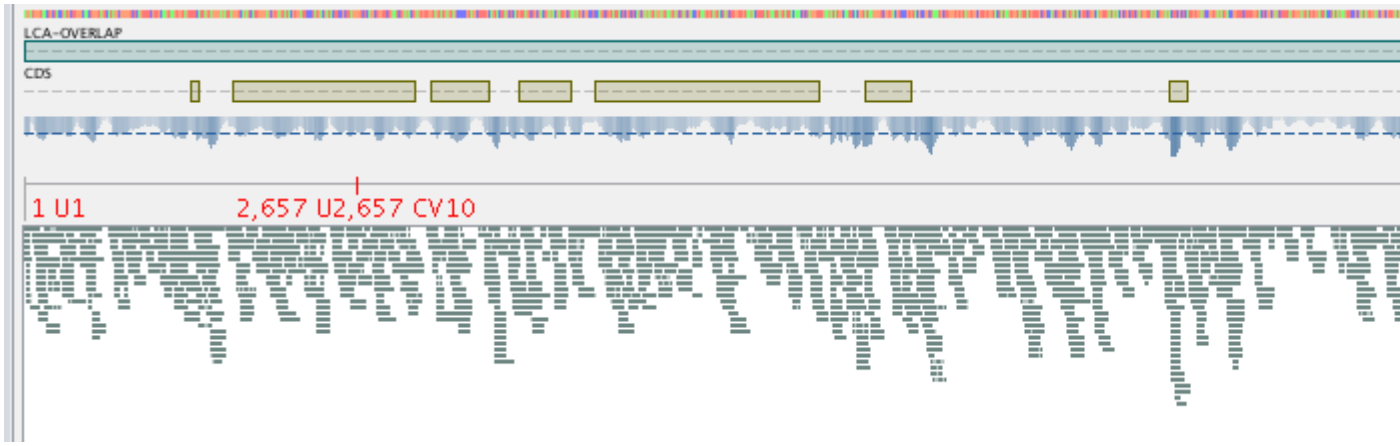
The produced files with **krona** extension can be the be used with the **ktImportText** (or **ImportText** if Krona Tools were not installed). The **-q** option of the script must be used:

```
$ for x in *.krona; do
ktImportText -q -o `basename $x .krona`.html $x;
done
```

This will create an HTML file for each one that can be read in a web browser.

Using Tablet

The GFF created can be used in software such as Tablet[40]. The image below shows a contig with the features loaded from the filtered (overlap) GFF and the GFF LCA file produced by *taxon_utils lca*.



3.4 Gene Prediction

Gene prediction is an essential portion of a metagenomic pipeline, because there is no a priori knowledge of what genes are in the samples. Moreover, a gene must be taxonomically annotated to correlate its function to the

[39] https://github.com/marbl/Krona/wiki
[40] https://ics.hutton.ac.uk/tablet/

taxonomic group it belongs to.

There different ways to predict genes, with some relying on general function domains like the ones from PFam[41] or others. This type of collections is very useful in identifying proteins in an unknown sequence. The main drawback for the examined datasets is that it's not possible to identify the organism but only the general function of a sequence.

A second approach is to use orthologs, genes derived from the same ancestral sequence with their separation originated from a speciation process. As functionality is preserved among them, they are a good choice when approaching samples where multiple organisms are present. Two collections, eggNOG[42] and Kegg Orthologs[43], are highly curated. A single ortholog gene identifier maps to several genes from different organisms, so the characterisation of an ortholog gene propagate to all its associated genes. This includes links to pathways in the case of Kegg and to functional categories in the case of eggNOG.

These genes are shared between organisms, so a single ortholog gene corresponds to several genes in different organisms. In some cases this is a preferred approach, as it allows a good resolution in the function, especially because this collections are linked to pathways in the case of Kegg and to functional categories in the case of eggNOG. The downside is that the collection of gene is not extensive and it is not connected to a taxonomic identification.

Another approach is to use genes from general public databases, like Uniprot[44]. While more general a collection, compared to Kegg Orthologs or eggNOG, it offers mappings to these two collections, as well as others. It does contain when available taxonomic information of its genes and it is divided into a manually curated portion (SwissProt) and an automated one (TrEMBL). This separation allows to have mixing annotations from both portions while preferentially use the ones from SwissProt.

In general the framework does not enforce one collection over another and in fact ortholog genes were used in one study, while Uniprot genes were used in others.

3.4.1 Prediction Software

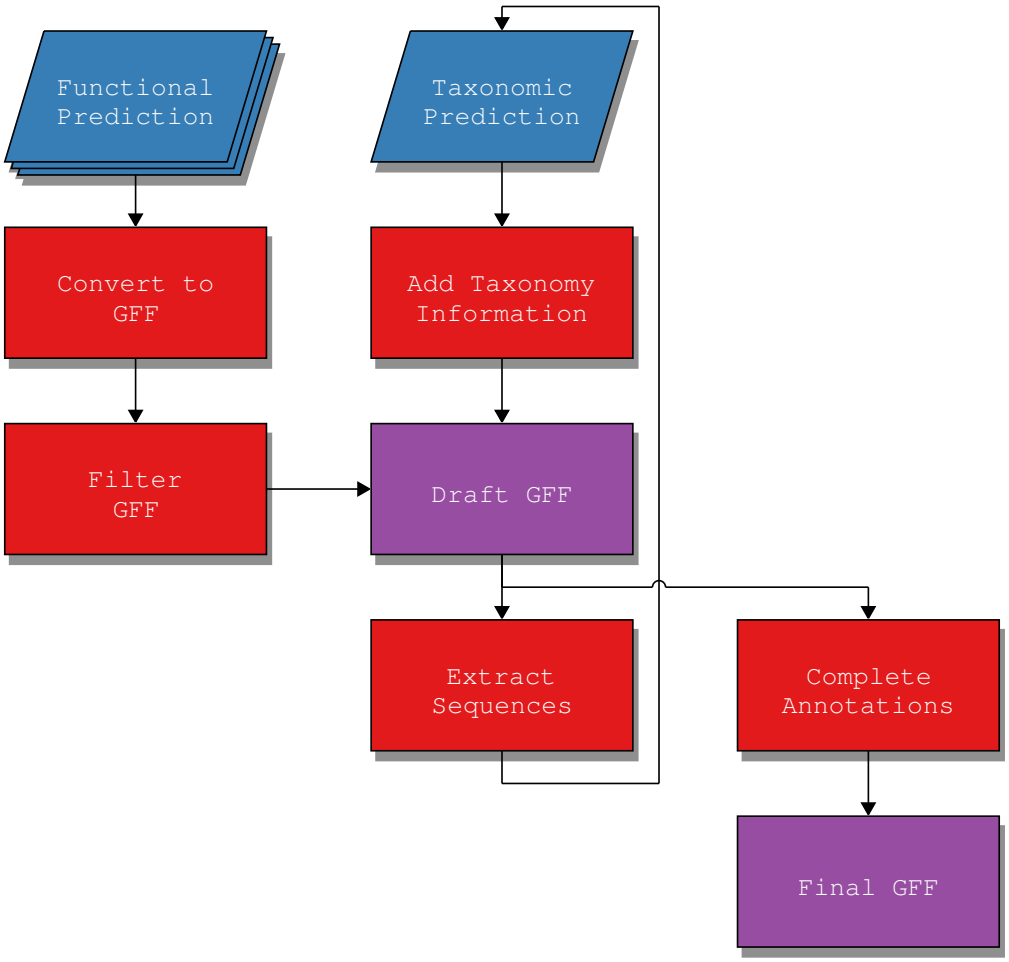
The prediction of genes requires both a collection and specific softwares to find homologous sequneces. There are two classes of software that can be used for gene prediction: one is profile based and the second uses similarity search.

An example of software using profile search is HMMER[45]. This approach uses an alignment of similar sequences to create an hidden Markov model (HMM) profile that is used to identify sequences that are similar to said profile. Curated profiles can be created from the eggNOG collection or other collection, but is also possible to automate the process of creating custom profiles.

The similarity search approach, is used in BLAST+[46], where a collection of sequences is first indexed and then all words in the index are searched against the query sequence and the most similar ones are investigated further to report a region of similarity.

[41] http://pfam.xfam.org/
[42] http://eggnog.embl.de
[43] http://www.kegg.jp/kegg/ko.html
[44] http://www.uniprot.org
[45] http://hmmer.janelia.org/
[46] http://blast.ncbi.nlm.nih.gov/Blast.cgi?CMD=Web&PAGE_TYPE=BlastDocs&DOC_TYPE=Download

3.4.2 General Procedure



The end result of the full process is a GFF file[47] including *gene_id taxon_id* and *uid* attributes. This attributes are need to identify univocally the annotation (*uid*), the gene that was functionally predicted (*gene_id*) and the organism it belongs to (*taxon_id*). A more detailed explanation of these attributes and others is in *MGKit GFF Specifications*.

The choice of the format is based on the fact that it is to manipulate without ad-hoc tools, as it is a text file, and it is accepted as input file by several bioinformatics tools.

3.4.3 Functional Prediction

The first step of the pipeline is to generate functional prediction information of the metagenomic sequences. This can be achieved using any tool of choice, with HMMER and BLAST+ preferred among others. While BLAST+ is being extensively tested, any program that outputs prediction data in the same tab separated format as BLAST+ can be used, including USEARCH[48] and RAPSearch2[49].

The framework provides two script, one for HMMER output *hmmer2gff - Convert HMMER output to GFF* and one for BLAST+ tab separated format *blast2gff - Convert BLAST output to GFF*, to convert predictions to GFF annotations.

[47] http://www.sequenceontology.org/gff3.shtml
[48] http://www.drive5.com/usearch/
[49] http://omics.informatics.indiana.edu/mg/RAPSearch2/

Usually a filter on the quality of the prediction is used, with 40 bit as a minimum indication of homology and 60 being a better one. If more than one gene collection is used, for example both SwissProt and TrEMBL, it is advised to keep track of which collection the annotation comes from and giving the chosen collection a quality index (*dbq* attribute in the GFF), with higher values assigned to preferred collections.

Generate Profiles

Note: More detailed information is in script-download-profiles

The framework provides, via a script and the following guidelines, a way to generate profiles for Kegg Orthologs genes, using Uniprot as repository of sequnece data.

The process of building the profiles to be used with HMMER is a step that involves several tasks:

1. download of data
2. alignment of sequences
3. conversion in HMMER profiles.

The first step involves, for all ortholog genes, to download all sequences available for each taxon level of interest: this will produce a series of file which contain the amino-acid sequences for each tuple gene-taxon. The sequences downloaded are aligned using Clustal Omega[50] and for each alignment a profile is built.

Building profiles in this way, by going through all ortholog genes and choosing the taxon level desired, opens the possibility of incrementally refining the profiling of a metagenome without having to rerun all profiles again, as only the new ones need to be run. Filtering the all the results is a much faster operation.

3.4.4 Filter Annotations

The number of predictions generated by the chosen prediction software can be very high, with a lot of them having just a few base pairs difference. When this involves the same functional prediction, it is safe to use the one with the best score.

However, when multiple genes are predicted on roughly the same region of a sequence, the choice of the annotation to keep is more difficult. Overlapping annotations can be either a weaker prediction, or the result of a chimeric sequence, as it can happen in metagenomic assemblies.

To solve this problem a script (filter-gff) was written that filters annotations when an overlap occurs. The algorithm scans the list of all annotations in a single sequence, sorted by their bit score, trying to find annotations that overlap. The filter is triggered when two annotations overlap, for at least 100bp by default, and the annotation to keep is chosen using a function that maximise three parameters: db quality (dbq), bit score (bitscore) and annotation length, in order of priority. This greatly reduces the number of annotations remaining and keeps the best possible annotations.

The choice of the 100 bp, as default value for an overlap to trigger filtering between two annotations, is based on the comparison of 36 prokaryotic genomes retrieved from UCSC[51] gene overlaps.

Table 1: Archaeal Genomes

Crenarchaea	Euryarchaea	Thaumarchaea
Acidianus hospitalis	Archaeoglobus fulgidus	Cenarchaeum symbiosum
Desulfurococcus kamchatkensis	Haloarcula marismortui	Nitrosopumilus maritimus
Hyperthermus butylicus	Methanobrevibacter ruminantium M1	
Pyrobaculum islandicum	Methanobrevibacter smithii	
Thermoproteus tenax Kra1	Thermococcus barophilus MP	
	Thermococcus onnurineus	

[50] http://www.clustal.org/omega/
[51] https://genome.ucsc.edu

Table 2: Bacterial Genomes

Actinobacteria	Aquificae	Bacteroidetes	Proteobacteria	Spirochaetes
Acidothermus cellulolyticus 11B	Aquifex aeolicus	Bacteroides thetaiotaomicron	Blochmannia floridanus	Borrelia burgdorferi
Bifidobacterium longum	Hydrogenivirga sp. 128	Cytophaga hutchinsonii	Candidatus Carsonella ruddii	Leptospira interogans
Mycobacterium tuberculosis	Hydrogenobaculum 3684	Gramella forsetii	Photobacterium profundum	Treponema pallidum
Nocardioides JS614	Persephonella marina	Salinibacter ruber	Salmonella typhi	
Rhodococcus RHA1	Sulfurihydrogenbium YO3AOP1		Shewanella oneidensis	
Tropheryma whipplei TW08 27	Sulfurihydrogenibium yellowstonense		Vibrio parahaemolyticus	

3.4.5 Taxonomic Prediction

When using Uniprot to functionally predict genes in a sequence, the metadata available for the gene may contain taxonomic information. However, while a gene from one species may have been predicted in the data, this prediction may be incorrect. There are various reasons, closely related organisms, lack of specific genes for a class of organisms or annotations, among others.

In this cases the approach taken in the framework is to extract the predicted nucleotide sequences using the tool of choice, provieded that it names the sequences using the *uid* attribute of an annotation, or the provided script (*get-gff-info - Extract informations to GFF annotations*). The sequences included in the file can be used with a similarity search program as BLAST to find the closest related sequences.

The collection used for this is the *nt* database from NCBI and a search against it can provide a better taxonomic assignment. The default behaviour is to take the taxonomic prediction with the highest score. It is recommended to use only predictions with a bit score of 60 or higher.

An included script *add-gff-info - Add informations to GFF annotations* provides the functionality necessary to add the taxonomic assignments to the GFF file. It also includes a last common ancestor (LCA) algorithm to resolve ambiguous assignments.

Last Common Ancestor

While the default behaviour is to take a prediction with the highest score, this may not be correct if more predictions have similar score. For this reason a last common ancestor (LCA) algorithm can be enabled on the predictions that are a set number of bits from the highest one, with the default value used 10.

The algorithm works by collecting all taxonomic predictions for a sequence, that falls within the chosen threshold, and traversing the taxonomy to find the last common ancestor. If no common ancestor can be found, the taxonomic predictions are discarded.

3.4.6 Complete Annotations

When a GFF file is produced by the framework, it can be integrated with the taxonomic information from Uniprot, if that was the collection used to predict genes.

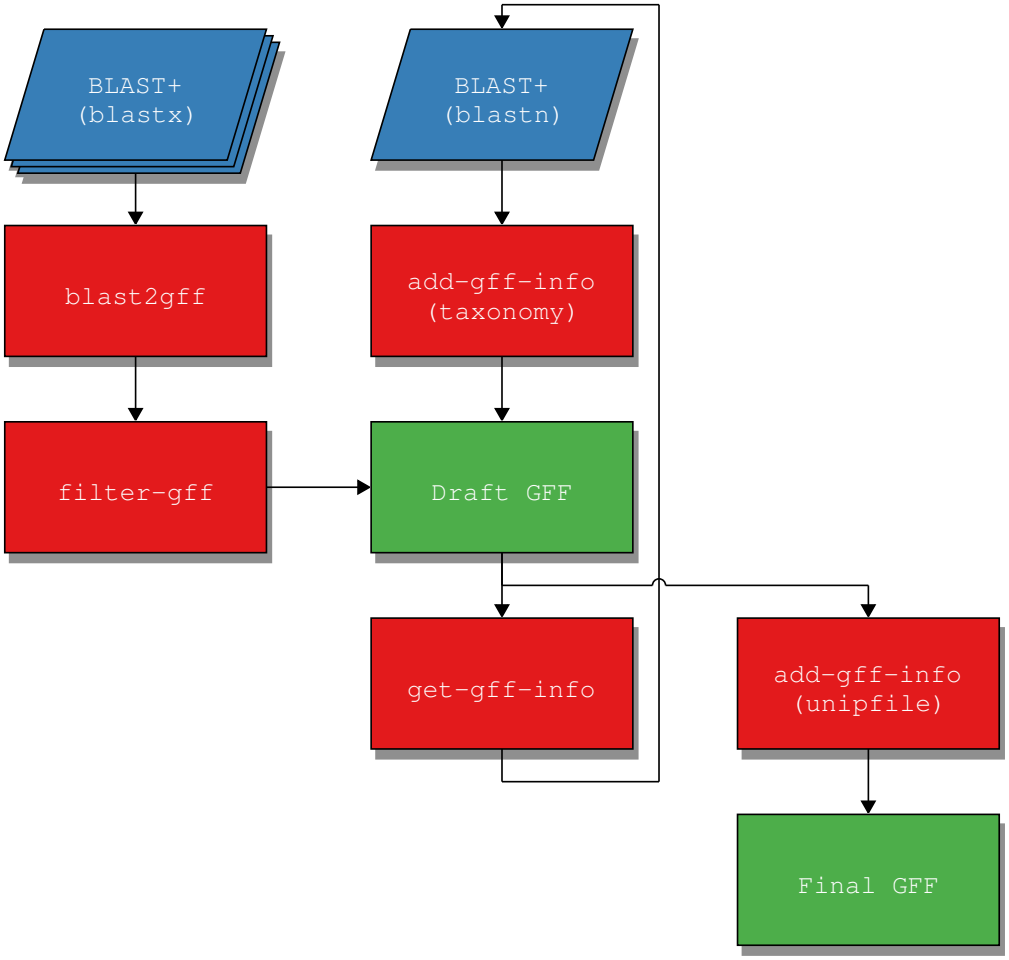
The process add mapping attributes to the GFF file, with eggNOG and Kegg Orthologs for example, while also completing the taxonomic assignment of annotations that were not assigned taxonomically. This can be done with an included script *add-gff-info - Add informations to GFF annotations* and the completed GFF can be used for further analysis

3.4.7 Examples

Gene Prediction with BLAST+

BLAST is another option to predict genes in a sequence and it is less difficult to set up as it only needs a FASTA file with the collection of genes to use.

The examples here use Uniprot DBs to predict genes, as it enables the mapping to several DBs, including eggNOG and Kegg Orthologs. It also assumes that an assembly has been produced for the gene prediction and that a DB to use blast with is already set up. Also, the BLAST+ package is expected to be installed on the system.



Functional Prediction

Assuming that BLAST is correctly installed and that the Uniprot DB is indexed, the only required parameter required by the scripts is *–outfmt 6*, which produces the BLAST tab format required by scripts that convert a BLAST output to a GFF *blast2gff - Convert BLAST output to GFF*. An example of the command line is this:

```
$ blastx -query assembly.fasta -db uniprot_sprot.fasta -out assembly.uniprot.tab -
↪outfmt 6
```

This will output a file that can be passed to the GFF creation script, *blast2gff*, with the following command:

```
$ blast2gff uniprot -b 40 -db UNIPROT-SP -dbq 10 assembly.uniprot.tab assembly.
↪uniprot.gff
```

The script documentation *blast2gff - Convert BLAST output to GFF* offer more information on the parameters. Suffice to say that *-b 40* excludes any BLAST hit with a bit score of less than 40 and *-dbq 10* point to the DB quality, as per *MGKit GFF Specifications*, which is important to filter annotations coming from multiple DBs with varying quality.

Filter GFF

The amount of prediction can be huge and most of them are overlapping annotations, so filtering the GFF annotations is important. A script is included to filter annotations (*filter-gff - Filter GFF annotations*), whose *overlap* command filters overlapping annotations. An example of the script execution is:

```
$ filter-gff overlap assembly.uniprot.gff assembly.uniprot-filt.gff
```

This will considerably reduce the size of the GFF file.

Taxonomic Prediction

Once the functional annotations are filtered, the next step is to assign taxonomic information to them, with the process being a two step process, to further refine the assignments.

The base process is to use the taxonomic assignment associated with the Uniprot ID predicted by BLAST, with a possible refinement of this by using the nucleotidic sequence associated with an annotation, whose similarity is then predicted using BLAST against a large collection of sequences, like the *nt* DB in NCBI.

Taxonomic Refinement

This part is entirely optional, but should be executed before the next one, to speed the scripts that follow.

First the sequences from the GFF file needs to be extracted with the *get-gff-info sequence* (*get-gff-info - Extract informations to GFF annotations*) command; an example execution is:

```
$ get-gff-info sequence -f assembly.fasta assembly.uniprot.gff assembly.uniprot.
↪frag.fasta
```

This will output a FASTA file called *assembly.uniprot.fasta* with the sequences used as query for the *blastn* command of the BLAST+ package against the *nt* DB:

```
$ blastn -query assembly.uniprot.frag.fasta -db nt -out assembly.uniprot.frag.tab -
↪outfmt 6
```

The ouput file *assembly.uniprot.frag.tab* is then passed to the *taxonomy* command of the *add-gff-info* script to incorporate the assignments information into the GFF file, an example of the execution of this command is the following:

```
$ add-gff-info taxonomy -t gi_taxid_nucl.dmp.gz -b assembly.uniprot.frag.tab -s 40␣
↪-d NCBI-NT assembly.uniprot.gff assembly.uniprot-taxa.gff
```

More information about the options used can be found at the script documentation (*get-gff-info - Extract informations to GFF annotations*), with an LCA option being available for assignments.

Complete Annotations

The rest of the taxonomic assignments, if not all, as well as additional informations can be added with *uniprot* or *unipfile* commands of the *add-gff-info* *add-gff-info - Add informations to GFF annotations* script. The main difference is that the *uniprot* command may be slower, as it connects to the internet and on a large number of annotations it takes a long time. The *unipfile* uses a file provided by Uniprot with additional information (in particular the taxonomy).

An example execution of the command is:

```
$ add-gff-info unipfile -i idmapping.dat.gz -m NCBI_TaxID assembly.uniprot.gff␣
↪assembly.uniprot-final.gff
```

Note: if you used the taxonomic refinement, use *assembly.uniprot-taxa.gff* instead of *assembly.uniprot.gff*

CHAPTER 4

Scripts Details

This section detailed information about the scripts included

4.1 blast2gff - Convert BLAST output to GFF

4.1.1 Overview

Blast output conversion in GFF requires a BLAST+ tabular format which can be obtained by using the *–outfmt 6* option with the default columns, as specified in `mgkit.io.blast.parse_blast_tab()`. The script can get data from the standard in and ouputs GFF lines on the standard output by default.

Uniprot

The Function `mgkit.io.blast.parse_uniprot_blast()` is used, which filters BLAST hits based on bitscore and adds by default a *db* attribute to the annotation with the value *UNIPROT-SP*, indicating that the SwissProt db is used and a *dbq* attribute with the value 10. The feature type used in the GFF is CDS.

BlastDB

If a BlastDB, such as *nt* or *nr* was used, the **blastdb** command offers some quick defaults to parse BLAST results.

It now includes options to control the way the sequence header are formatted. Options to change the separator used, as well as the column used as *gene_id*. This was added because at the moment the GI identifier (the second column in the header) is used, but it's being phased out in favour of the embl/gb/dbj (right now the fourth column in the header). This should easy the transition to the new format and makes it easier to adapt an older pipeline/blastdb to newer files (like the ID to TAXA files).

The header from the a *ncbi-nt* header looks like this:

```
gi|160361034|gb|CP000884.1
```

This is the default output accepted by the *blastdb* command. The fields are separated by | (pipe) and the GI is used (–gene-index 1, since internally the string is split by the separator and the second element is take - lists indices are 0-based in Python). This output uses the following options:

```
--header-sep '|' --gene-index 1
```

Notice the single quotes to pass the pipe symbol, since *bash* would interpret it as pipeing to the next coommand otherwise. This is the default.

In case, for the same header, we want to use the *gb* identifier, the only option to be specified is:

```
--gene-index 3
```

This will get the fourth element of the header (since we're splitting by pipe).

As in the *uniprot* command, the *gene_id* can be set to use the whole header, using the *-n* option. Useful in case the *BLAST* db that was used was custom made. While pipe is used in major databases, it was made the default, by if the db used has different conventions the separator can be changed. There's also the options of later changing the *gene_id* in the output GFF if necessary.

Changes

Changed in version 0.3.4: using *click* instead of *argparse*

Changed in version 0.2.6: added *-r* option to *blastdb*

Changed in version 0.2.5: added more options to give user control to the *blastdb* command

New in version 0.2.3: added *–fasta-file* option, added more data from a blsat hit

New in version 0.2.2: added *blastdb* command

Changed in version 0.2.1: added *-ft* option

Changed in version 0.1.13: added *-n* and *-k* parameters to *uniprot* command

New in version 0.1.12.

4.1.2 Options

blast2gff

Main function

```
blast2gff [OPTIONS] COMMAND [ARGS]...
```

Options

--version
Show the version and exit.

--cite

blastdb

Reads a BLAST output file [blast-file] in tabular format (using -outfmt 6) and outputs a GFF file [gff-file]

```
blast2gff blastdb [OPTIONS] [BLAST_FILE] [GFF_FILE]
```

Options

-v, --verbose

-db, --db-used <db_used>
blastdb used [default: NCBI-NT]

-n, --no-split
if used, the script assumes that the sequence header will be used as gene_id

-s, --header-sep <header_sep>
The separator for the header, defaults to 'l' (pipe) [default: l]

-i, --gene-index <gene_index>
Which of the header columns (0-based) to use as gene_id (defaults to 1 - the second column) [default: 1]

-r, --remove-version
if used, the script removes the *version* information from the gene_id

-a, --fasta-file <fasta_file>
Optional FASTA file with the query sequences

-dbq, --db-quality <db_quality>
Quality of the DB used [default: 10]

-b, --bitscore <bitscore>
Minimum bitscore to keep the annotation [default: 0.0]

-k, --attr-value <attr_value>
Additional attribute and value to add to each annotation, in the form attr:value

-ft, --feat-type <feat_type>
Feature type to use in the GFF [default: CDS]

--progress
Shows Progress Bar

Arguments

BLAST_FILE
Optional argument

GFF_FILE
Optional argument

uniprot

Reads a BLAST output file [blast-file] in tabular format (using -outfmt 6) from a Uniprot DB and outputs a GFF file [gff-file]

```
blast2gff uniprot [OPTIONS] [BLAST_FILE] [GFF_FILE]
```

Options

-v, --verbose

-db, --db-used <db_used>
Uniprot database used with BLAST [default: UNIPROT-SP]

-n, --no-split
if used, the script assumes that the sequence header will be used as gene_id

-a, --fasta-file <fasta_file>
Optional FASTA file with the query sequences

-dbq, --db-quality <db_quality>
Quality of the DB used [default: 10]

-b, --bitscore <bitscore>
Minimum bitscore to keep the annotation [default: 0.0]

-k, --attr-value <attr_value>
Additional attribute and value to add to each annotation, in the form attr:value

-ft, --feat-type <feat_type>
Feature type to use in the GFF [default: CDS]

--progress
Shows Progress Bar

Arguments

BLAST_FILE
Optional argument

GFF_FILE
Optional argument

4.2 filter-gff - Filter GFF annotations

4.2.1 Overview

Filters GFF annotations in different ways.

Value Filtering

Enables filtering of GFF annotations based on the the values of attributes of a GFF annotation. The filters are based on equality of numbers (internally converted into float) and strings, a string contained in the value of an attribute less or greater than are included as well. The length of annotation has the attribute *length* and can be tested.

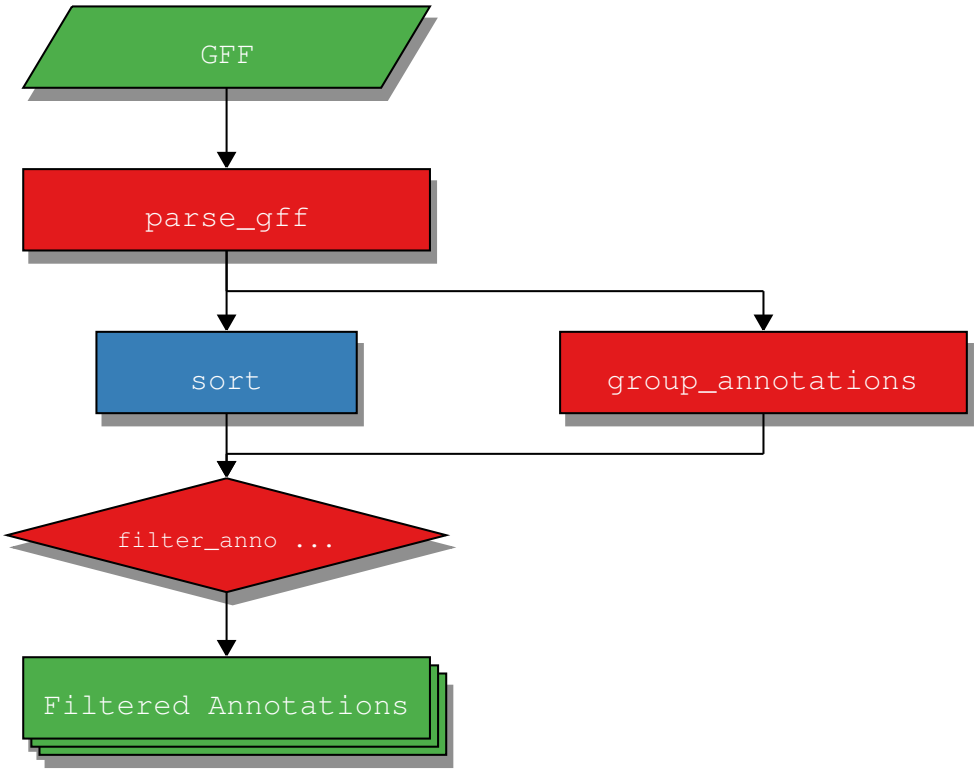
Overlap Filtering

Filters overlapping annotations using the functions *mgkit.filter.gff.choose_annotation()* and *mgkit.filter.gff.filter_annotations()*, after the annotations are grouped by both sequence and strand. If the GFF is sorted by sequence name and strand, the *-t* can be used to make the filtering use less memory. It can be sorted in Unix using *sort -s -k 1,1 -k 7,7 gff_file*, which applies a stable sort using the sequence name as the first key and the strand as the second key.

Note: It is also recommended to use:

```
export LC_ALL=C
```

To speed up the sorting



The above digram describes the internals of the script.

The annotations needs first to be grouped by seq_id and strand, forming a group that can be then be passed to `mgkit.filter.gff.filter_annotations()`. This function:

1. sort annotations by bit score, from the highest to the lowest
2. loop over all combination of N=2 annotations:
    1. choose which of the two annotations to discard if they overlap for a the required amount of bp (defaults to 100bp)
    2. in which case, the preference is given to the db quality first, than the bit score and finally the lenght of annotation, the one with the highest values is kept

While the default behaviour is the same, now it is posible to decided the function used to discard one the two annotations. It is possible to use the *-c* argument to pass a string that defines the function. The string passed must start with or without a **+**. Using **+** translates into the builtin function *max* while no **+** translates into *min* from the second character on, any number of attributes can be used, separated by commas. The attributes, however, must be one of the properties defined in `mgkit.io.gff.Annotation`, *bitscore* that returns the value converted in a *float*. Internally the attributes are stored as strings, so for attributes that have no properties in the class, such as *evalue*, the *float* builtin is applied.

The tuples built for both annotations are then passed to the comparison function to be selected and the value returned by it is **discarded**. The order of the elements in the string is important to define the priority given to each element in the comparison and the leftmost one has the highesst priority.

Examples of function strings:

- *-dbq,bitscore,length* becomes max((ann1.dbq, ann1.bitscore, ann1.length), (ann2.dbq, ann2.bitscore, ann2.length) - This is default and previously only choice
- *-bitscore,length,dbq* uses the same elements but gives lowest priority to *dbq*
- *+evalue*: will discard the annotation with the highest *evalue*

Per Sequence Values

The *sequence* command allows to filter on a per sequence basis, using functions such as the median, quantile and mean on attributes like evalue, bitscore and identity. The file can be passed as sorted already, saving memory (like in the *overlap* command), but it's not needed to sort the file by strand, only by the first column.

Coverage Filtering

The *cov* command calculates the coverage of annotations as a measure of the percentage of each reference sequence length. A minimum coverage percentage can be used to keep the annotations of sequences that have a greater or equal coverage than the specified one.

Changes

New in version 0.1.12.

Changed in version 0.1.13: added *–sorted* option

Changed in version 0.2.0: changed option *-c* to accept a string to filter overlap

Changed in version 0.2.5: added *sequence* command

Changed in version 0.2.6: added *length* as attribute and *min*/*max*, and *ge* is the default comparison for command *sequence*, *–sort-attr* to *overlap*

Changed in version 0.3.1: added *–num-gt* and *–num-lt* to *values* command, added *cov* command

Changed in version 0.3.4: moved to use *click* for argument parsing reworked the *values*, *sequence* commands

4.2.2 Options

filter-gff

Main function

```
filter-gff [OPTIONS] COMMAND [ARGS]...
```

Options

--version
Show the version and exit.

--cite

cov

Filter on a per coverage basis

```
filter-gff cov [OPTIONS] [INPUT_FILE] [OUTPUT_FILE]
```

Options

-v, --verbose

-f, --reference <reference>
Reference FASTA file for the GFF [required]

-s, --strand-specific
If the coverage must be calculated on each strand

-t, --sorted
Assumes the GFF to be correctly sorted

-c, --min-coverage <min_coverage>
Minimum coverage for the contig/strand

-r, --rename
Emulates BLAST in reading the FASTA file (keeps only the header before the first space)

--progress
Shows Progress Bar

Arguments

INPUT_FILE
Optional argument

OUTPUT_FILE
Optional argument

overlap

Use overlapping filter

```
filter-gff overlap [OPTIONS] [INPUT_FILE] [OUTPUT_FILE]
```

Options

-v, --verbose

-s, --size <size>
Size of the overlap that triggers the filter [default: 100]

-t, --sorted
If the GFF file is sorted (all of a sequence annotations are contiguos and sorted by strand) can use less memory, *sort -s -k 1,1 -k 7,7* can be used

-c, --choose-func <choose_func>
Function to choose between two overlapping annotations

-a, --sort-attr <sort_attr>
Attribute to sort annotations before filtering (default bitscore) [default: bitscore]

Options bitscoreidentityllength

--progress
Shows Progress Bar

Arguments

INPUT_FILE
Optional argument

OUTPUT_FILE
Optional argument

sequence

Filter on a per sequence basis

```
filter-gff sequence [OPTIONS] [INPUT_FILE] [OUTPUT_FILE]
```

Options

-v, --verbose

-t, --sorted
If the GFF file is sorted (all of a sequence annotations are contiguos) can use less memory, *sort -s -k 1,1* can be used

-a, --attribute <attribute>
Attribute on which to apply the filter [default: bitscore]

Options evalue|bitscore|identity|length

-f, --function <function>
Function for filtering [default: mean]

Options mean|median|quantile|std|max|min

-l, --value <value>
Value for the function (used for *std* and *quantile*)

-c, --comparison <comparison>
Type of comparison (e.g. ge -> greater than or equal to) [default: ge]

Options gt|ge|lt|le

--progress
Shows Progress Bar

Arguments

INPUT_FILE
Optional argument

OUTPUT_FILE
Optional argument

values

Filter based on values

```
filter-gff values [OPTIONS] [INPUT_FILE] [OUTPUT_FILE]
```

Options

-v, --verbose

--str-eq <str_eq>
filter by custom key:value, if the argument is 'key:value' the annotation is kept if it contains an attribute 'key' whose value is exactly 'value' as a string

--str-in <str_in>
Same as '–str-eq' but 'value' is contained in the attribute

--num-eq <num_eq>
Same as '–str-eq' but 'value' is a number which is equal or greater than

--num-ge <num_ge>
Same as '–str-eq' but 'value' is a number which is equal or greater than

--num-le <num_le>
Same as '–num-ge' but 'value' is a number which is equal or less than

--num-gt <num_gt>
Same as '–str-eq' but 'value' is a number which is greater than

--num-lt <num_lt>
Same as '–num-ge' but 'value' is a number which is less than

--progress
Shows Progress Bar

Arguments

INPUT_FILE
Optional argument

OUTPUT_FILE
Optional argument

4.3 add-gff-info - Add informations to GFF annotations

4.3.1 Overview

Add more information to GFF annotations: gene mappings, coverage, taxonomy, etc..

Uniprot Command

If the *gene_id* of an annotation is a Uniprot ID, the script queries Uniprot for the requested information. At the moment the information that can be added is the taxon_id, taxon_name, lineage and mapping to EC, KO, eggNOG IDs.

It's also possible to add mappings to other databases using the *-m* option with the correct identifier for the mapping, which can be found at this page[52]; for example if it's we want to add the mappings of uniprot IDs to *BioCyc*, in the *abbreviation* column of the mappings we find that it's identifier is *REACTOME_ID*, so we pass *-m REACTOME* to the script (leaving *_ID* out). Mapped IDs are separated by commas.

The taxonomy IDs are not overwritten if they are found in the annotations, the *-f* is provided to force the overwriting of those values.

See also *MGKit GFF Specifications* for more informations about the GFF specifications used.

Note: As the script needs to query Uniprot a lot, it is recommended to split the GFF in several files, so an error in the connection doesn't waste time.

However, a cache is kept to reduce the number of connections

---

[52] http://www.uniprot.org/faq/28

Coverage Command

Adds coverage information from BAM alignment files to a GFF file, using the function *mgkit.align.add_coverage_info()*, the user needs to supply for each sample a BAM file, using the *-a* option, whose parameter is in the form *sample,samplealg.bam*. More samples can be supplied adding more *-a* arguments.

Hint: As an example, to add coverage for *sample1*, *sample2* the command line is:

```
add-gff-info coverage -a sample1,sample1.bam -a sample2,sample2.bam \
inputgff outputgff
```

A total coverage for the annotation is also calculated and stored in the *cov* attribute, while each sample coverage is stored into *sample_cov* as per *MGKit GFF Specifications*.

Adding Coverage from samtools depth

The *cov_samtools* allows the use of the output of *samtools* **depth** command. The *-aa* options must be used to pass information about all base pairs and sequences coverage in the BAM/SAM file. The command work similarly to *coverage*, accepting compressed *depth* files as well. If only one *depth* file is passed and no sample is passed, the attribute in the GFF will be *cov*, otherwise the attribute will be *sample1_cov*, *sample2_cov*, etc.

To create samtools *depth* files, this command must be used:

```
$ samtools depth -aa bam_file
```

Uniprot Offline Mappings

Similar to the *uniprot* command, it uses the idmapping[53] file provided by Uniprot, which speeds up the process of adding mappings and taxonomy IDs from Uniprot gene IDs. It's not possible tough to add *EC* mappings with this command, as those are not included in the file.

Kegg Information

The *kegg* command allows to add information to each annotation. Right now the information that can be added is restricted to the pathway(s) (reference KO) a KO is part of and both the KO and pathway(s) descriptions. This information is stored in keys starting with **ko_**.

Expected Aminoacidic Changes

Some scripts, like *snp_parser - SNPs analysis*, require information about the expected number of synonymous and non-synonymous changes of an annotation. This can be done using *mgkit.io.gff.Annotation.add_exp_syn_count()* by the user of the command *exp_syn* of this script. The attributes added to each annotation are explained in the *MGKit GFF Specifications*

Adding Count Data

Count data on a per-sample basis can be added with the *counts* command. The accepted inputs are from HTSeq-count and featureCounts. The ouput produced by featureCounts, is the one from using its **-f** option must be used.

This script accept by default a tab separated file, with a uid in the first column and the other columns are the counts for each sample, in the same order as they are passed to the **-s** option. To use the featureCounts file format, this script **-e** option must be used.

[53] ftp://ftp.uniprot.org/pub/databases/uniprot/current_release/knowledgebase/idmapping/idmapping.dat.gz

The sample names must be provided in the same order as the columns in the input files. If the counts are FPKMS the *-f* option can be used.

Adding Taxonomy from a Table

There are cases where it may needed or preferred to add the taxonomy from a *gene_id* already provided in the GFF file. For such cases the *addtaxa* command can be used. It works in a similar way to the *taxonomy* command, only it expect three different type of inputs:

- *GI-Taxa* table from NCBI (e.g. gi_taxid_nucl.dmp,)
- tab separated table
- dictionary
- HDF5

The first two are tab separated files, where on each line, the first column is the *gene_id* that is found in the first column, while the second if the *taxon_id*.

The third option is a serialised Python *dict*/hash table, whose keys are the *gene_id* and the value is that gene corresponding *taxon_id*. The serialised formats accepted are msgpack, json and pickle. The *msgpack* module must be importable. The option to use json and msgpack allow to integrate this script with other languages without resorting to a text file.

The last option is a HDF5 created using the *to_hdf* command in *taxon-utils - Taxonomy Utilities*. This requires *pandas* installed and *pytables* and it provides faster lookup of IDs in the table.

While the default is to look for the *gene_id* attribute in the GFF annotation, another attribute can be specified, using the **-gene-attr** option.

Note: the dictionary content is loaded after the table files and its keys and corresponding values takes precedence over the text files.

Warning: from September 2016 NCBI will retire the GI. In that case the same kind of table can be built from the *nucl_gb.accession2taxid.gz* file The format is different, but some information can be found in `mgkit.io.blast.parse_accession_taxa_table()`

Adding information from Pfam

Adds the Pfam description for the annotation, by downloading the list from Pfam.

The options allow to specify in which attribute the ID/ACCESSION is stored (defaults to *gene_id*) and which one between ID/ACCESSION is the value of that attribute (defaults to *ID*). if no description is found for the family, a warning message is logged.

Changes

Changed in version 0.3.4: removed the *taxonomy* command, since a similar result can be obtained with *taxon-utils lca* and *add-gff-info addtaxa*. Removed *eggnog* command and added option to verbose the logging in *cov_samtools* (now is quiet), also changed the interface

Changed in version 0.3.3: changed how *addtaxa -a* works, to allow the use of *seq_id* as key to add the taxon_id

Changed in version 0.3.0: added *cov_samtools* command, *–split* option to *exp_syn*, *-c* option to *addtaxa*. *kegg* now does not skip annotations when the attribute is not found.

Changed in version 0.2.6: added *skip-no-taxon* option to *addtaxa*

Changed in version 0.2.5: if a dictionary is supplied to *addtaxa*, the GFF is not preloaded

Changed in version 0.2.3: added *pfam* command, renamed *gitaxa* to *addtaxa* and made it general

Changed in version 0.2.2: added *eggnog*, *gitaxa* and *counts* command

Changed in version 0.2.1.

- added *-d* to *uniprot* command
- added cache to *uniprot* command
- added *kegg* command (cached)

Changed in version 0.1.16: added *exp_syn* command

Changed in version 0.1.15: *taxonomy* command *-b* option changed

Changed in version 0.1.13.

- added *–force-taxon-id* option to the *uniprot* command
- added *coverage* command
- added *taxonomy* command
- added *unipfile* command

New in version 0.1.12.

4.3.2 Options

add-gff-info

Main function

```
add-gff-info [OPTIONS] COMMAND [ARGS]...
```

Options

--version
Show the version and exit.

--cite

addtaxa

Adds taxonomy information from a GI-Taxa, gene_id/taxon_id table or a dictionary serialised as a pickle/msgpack/json file, or a table in a HDF5 file

```
add-gff-info addtaxa [OPTIONS] [INPUT_FILE] [OUTPUT_FILE]
```

Options

-v, --verbose

-t, --gene-taxon-table <gene_taxon_table>
GIDs taxonomy table (e.g. gi_taxid_nucl.dmp.gz) or a similar file where GENE/TAXON are tab separated and one per line

-a, --gene-attr <gene_attr>
In which attribute the GENEID in the table is stored (defaults to *gene_id*)

-f, --hdf-table <hdf_table>
HDF5 file and table name to use for taxon_id lookups. The format to pass is the file name, colon and the table file hf5:taxa-table. The index in the table is the accession_id, while the column *taxon_id* stores the taxon_id as int

-x, --taxonomy <taxonomy>
Taxonomy file - If given, both *taxon_name* and *lineage* attributes will be set

-d, --dictionary <dictionary>
A serialised dictionary, where the key is the GENEID and the value is TAXONID. It can be in json or msgpack format (can be a compressed file) *Note*: the dictionary values takes precedence over the table files

-e, --skip-no-taxon
If used, annotations with no taxon_id won't be outputted

-db, --taxon-db <taxon_db>
DB used to add the taxonomic information

-c, --cache-table
If used, annotations are not preloaded, but the taxa table is cached, instead

--progress
Shows Progress Bar

Arguments

INPUT_FILE
Optional argument

OUTPUT_FILE
Optional argument

counts

Adds counts data to the GFF file

```
add-gff-info counts [OPTIONS] [INPUT_FILE] [OUTPUT_FILE]
```

Options

-v, --verbose

-s, --samples <samples>
Sample names, in the same order as the count files [required]

-c, --count-files <count_files>
Count file(s) [required]

-f, --fpkms
If the counts are FPKMS

-e, --featureCounts
If the counts files are from featureCounts

--progress
Shows Progress Bar

Arguments

INPUT_FILE
Optional argument

OUTPUT_FILE
Optional argument

cov_samtools

Adds information from samtools_depth

```
add-gff-info cov_samtools [OPTIONS] [INPUT_FILE] [OUTPUT_FILE]
```

Options

-v, --verbose

-s, --samples <samples>
Sample name, will add a *sample_cov* in the GFF file. If not passed, the attribute will be *cov*

-d, --depths <depths>
samtools depth -aa file [required]

-n, --num-seqs <num_seqs>
Number of sequences to update the log. If 0, no message is logged [default: 0]

--progress
Shows Progress Bar

Arguments

INPUT_FILE
Optional argument

OUTPUT_FILE
Optional argument

coverage

Adds coverage information from BAM Alignment files

```
add-gff-info coverage [OPTIONS] [INPUT_FILE] [OUTPUT_FILE]
```

Options

-v, --verbose

-a, --sample-alignment <sample_alignment>
sample name and correspondent alignment file separated by comma [required]

Arguments

INPUT_FILE
Optional argument

OUTPUT_FILE
Optional argument

exp_syn

Adds expected synonymous and non-synonymous changes information

```
add-gff-info exp_syn [OPTIONS] [INPUT_FILE] [OUTPUT_FILE]
```

Options

-v, --verbose

-r, --reference <reference>
reference sequence in fasta format [required]

-s, --split
Split the sequence header of the reference at the first space, to emulate BLAST behaviour

--progress
Shows Progress Bar

Arguments

INPUT_FILE
Optional argument

OUTPUT_FILE
Optional argument

kegg

Adds information and mapping from Kegg

```
add-gff-info kegg [OPTIONS] [INPUT_FILE] [OUTPUT_FILE]
```

Options

-v, --verbose

-c, --email <email>
Contact email [required]

-d, --description
Add Kegg description

-p, --pathways
Add pathways ID involved

-m, --kegg-id <kegg_id>
In which attribute the Kegg ID is stored (defaults to *gene_id*)

Arguments

INPUT_FILE
Optional argument

OUTPUT_FILE
Optional argument

pfam

Adds information from Pfam

```
add-gff-info pfam [OPTIONS] [INPUT_FILE] [OUTPUT_FILE]
```

Options

-v, --verbose

-i, --id-attr <id_attr>
In which attribute the Pfam ID/ACCESSION is stored (defaults to *gene_id*)

-a, --use-accession
If used, the attribute value is the Pfam ACCESSION (e.g. PF06894), not ID (e.g. Phage_TAC_2)

Arguments

INPUT_FILE
Optional argument

OUTPUT_FILE
Optional argument

unipfile

Adds expected synonymous and non-synonymous changes information

```
add-gff-info unipfile [OPTIONS] [INPUT_FILE] [OUTPUT_FILE]
```

Options

-v, --verbose

-i, --mapping-file <mapping_file>
Uniprot mapping file [required]

-f, --force-taxon-id
Overwrite taxon_id if already present

-m, --mapping <mapping>
Mappings to add [required]

Options NCBI_TaxID|eggNOG|KO|KEGG|BioCyc|UniPathway|EMBL|EMBL-CDS|GI|STRING

--progress
Shows Progress Bar

Arguments

INPUT_FILE
Optional argument

OUTPUT_FILE
Optional argument

uniprot

Adds information from GFF whose gene_id is from Uniprot

```
add-gff-info uniprot [OPTIONS] [INPUT_FILE] [OUTPUT_FILE]
```

Options

-v, --verbose

-c, --email <email>
Contact email [required]

--buffer <buffer>
Number of annotations to keep in memory [default: 50]

-f, --force-taxon-id
Overwrite taxon_id if already present

-t, --taxon-id
Add taxonomic ids to annotations, if taxon_id is found, it won't be Overwritten.

-l, --lineage
Add taxonomic lineage to annotations

-e, --eggnog
Add eggNOG mappings to annotations

-ec, --enzymes
Add EC mappings to annotations

-ko, --kegg_orthologs
Add KO mappings to annotations

-d, --protein-names
Add Uniprot description

-m, --mapping <mapping>
Add any DB mappings to annotations

Arguments

INPUT_FILE
Optional argument

OUTPUT_FILE
Optional argument

4.4 get-gff-info - Extract informations to GFF annotations

4.4.1 Overview

Extract information from GFF files

sequence command

Used to extract the nucleotidic sequences from GFF annotations. It requires the *fasta* file containing the sequences referenced in the GFF *seq_id* attribute (first column of the raw GFF).

The sequnces extract have as identifier the *uid* stored in the GFF file and by default the sequnece is not reverse complemented if the annotation is on the **-** strand, but this can be changed by using the *-r* option.

The sequences are wrapped at 60 characters, as per FASTA specs, but this behavior can be disabled by specifing the *-w* option.

> **Warning:** The reference file is loaded in memory

dbm command

Creates a dbm DB using the *semidbm* package. The database can then be loaded using `mgkit.db.dbm.GFFDB`

mongodb command

Outputs annotations in a format supported by MongoDB. More information about it can be found in `mgkit.db.mongo`

gtf command

Outputs annotations in the GTF format

split command

Splits a GFF file into smaller chunks, ensuring that all of a sequence annotations are in the same file.

cov command

Calculate annotation coverage for each contig in a GFF file. The command can be run as strand specific (not by default) and requires the reference file to which the annotation refer to. The output file is a tab separated one, with the first column being the sequence name, the second is the strand (+, -, or NA if not strand specific) and the third is the percentage of the sequence covered by annotations.

> **Warning:** The GFF file is assumed to be sorted, by sequence or sequence-strand if wanted. The GFF file can be sorted using *sort -s -k 1,1 -k 7,7* for strand specific, or *sort -s -k 1,1* if not strand specific.

Changes

Changed in version 0.3.4: using *click* instead of *argparse*, renamed *split* command *–json* to *–json-out*

Changed in version 0.3.1: added *cov* command

Changed in version 0.3.0: added *–split* option to *sequence* command

Changed in version 0.2.6: added *split* command, *–indent* option to *mongodb*

Changed in version 0.2.3: added *–gene-id* option to *gtf* command

New in version 0.2.2: added *gtf* command

New in version 0.2.1: *dbm* and *mongodb* commands

New in version 0.1.15.

4.4.2 Options

get-gff-info

Main function

```
get-gff-info [OPTIONS] COMMAND [ARGS]...
```

Options

--version
Show the version and exit.

--cite

cov

Report on how much a sequence length is covered by annotations in [gff-file]

```
get-gff-info cov [OPTIONS] [GFF_FILE] [OUTPUT_FILE]
```

Options

-v, --verbose

-f, --reference <reference>
Reference FASTA file for the GFF [required]

-j, --json-out
The output will be a JSON dictionary

-s, --strand-specific
If the coverage must be calculated on each strand

-r, --rename
Emulate BLAST output (use only the header part before the first space)

--progress
Shows Progress Bar

Arguments

GFF_FILE
Optional argument

OUTPUT_FILE
Optional argument

dbm

Creates a dbm database with annotations from file [gff-file] into db [output-dir]

```
get-gff-info dbm [OPTIONS] [GFF_FILE]
```

Options

-v, --verbose

-d, --output-dir <output_dir>
Directory for the database [default: gff-dbm]

Arguments

GFF_FILE
Optional argument

gtf

Extract annotations from a GFF file [gff-file] to a GTF file [gtf-file]

```
get-gff-info gtf [OPTIONS] [GFF_FILE] [GTF_FILE]
```

Options

-v, --verbose

-g, --gene-id <gene_id>
GFF attribute to use for the GTF *gene_id* attribute [default: gene_id]

Arguments

GFF_FILE
Optional argument

GTF_FILE
Optional argument

mongodb

Extract annotations from a GFF [gff-file] file and makes output for MongoDB [output-file]

```
get-gff-info mongodb [OPTIONS] [GFF_FILE] [OUTPUT_FILE]
```

Options

-v, --verbose

-t, --taxonomy <taxonomy>
Taxonomy used to populate the lineage

-c, --no-cache
No cache for the lineage function

-i, --indent <indent>
If used, the json will be written in a human readble form

--progress
Shows Progress Bar

Arguments

GFF_FILE
Optional argument

OUTPUT_FILE
Optional argument

sequence

Extract the nucleotidic sequences of annotations from [gff-file] to [fasta-file]

```
get-gff-info sequence [OPTIONS] [GFF_FILE] [FASTA_FILE]
```

Options

-v, --verbose

-r, --reverse
Reverse complement sequences on the - strand

-w, --no-wrap
Write the sequences on one line

-s, --split
Split the sequence header of the reference at the first space, to emulate BLAST behaviour

-f, --reference <reference>
Fasta file containing the reference sequences of the GFF file

--progress
Shows Progress Bar

Arguments

GFF_FILE
Optional argument

FASTA_FILE
Optional argument

split

Split annotations from a GFF file [gff-file] to several files starting with [prefix]

```
get-gff-info split [OPTIONS] [GFF_FILE]
```

Options

-v, --verbose

-p, --prefix <prefix>
Prefix for the file name in output [default: split]

-n, --number <number>
Number of chunks into which split the GFF file [default: 10]

-z, --gzip
gzip output files

Arguments

GFF_FILE
Optional argument

4.5 hmmer2gff - Convert HMMER output to GFF

4.5.1 Overview

Script to convert HMMER results files (domain table) to a GFF file, the name of the profiles are expected to be now in the form *GENEID_TAXONID_TAXON-NAME(-nr)* by default, but any other profile name is accepted.

The profiles tested are those made from Kegg Orthologs, from the *download_profiles* script. If the *–no-custom-profiles* options is used, the script can be used with any profile name. The profile name will be used for *gene_id*, *taxon_id* and *taxon_name* in the GFF file.

It is possible to use seuqnces not translated using mgkit, no information on the frame is assumed, so this script can be used against a protein DB. For example Uniprot can be searched for profiles, in which case the **–no-frame** options must be used.

Note: for GENEID, old documentation points to KOID, it is the same

Warning: The compatibility with old data has been **removed**, meaning that old experiments must use the scripts from those versions. It is possible to use multiple environments, with *virtualenv* for this purpose. An examples is given in *Installation*.

Changes

Changed in version 0.1.15: adapted to new GFF module and specs

Changed in version 0.2.1: added options to customise output and filters and old restrictions

Changed in version 0.3.1: added *–no-frame* option for non mgkit-translated proteins, sequence headers are handled the same way as HMMER (truncated at the first space)

4.5.2 Options

Convert HMMER data to GFF file

```
usage: hmmer2gff [-h] [-o [OUTPUT_FILE]] [-t DISCARD] [-d] [-c] [-db DATABASE]
                 [-f FEATURE_TYPE] [-n] [-v | --quiet] [--cite] [--manual]
                 [--version]
                 aa_file [hmmer_file]
```

Named Arguments

-v, --verbose	more verbose - includes debug messages
	Default: 20
--quiet	less verbose - only error and critical messages
--cite	Show citation for the framework
--manual	Show the script manual
--version	show program's version number and exit

File options

aa_file	Fasta file containing contigs translated to aa (used by HMMER)
hmmer_file	Default: -
-o, --output-file	Default: <_io.TextIOWrapper name='<stdout>' mode='w' encoding='UTF-8'>

Filters

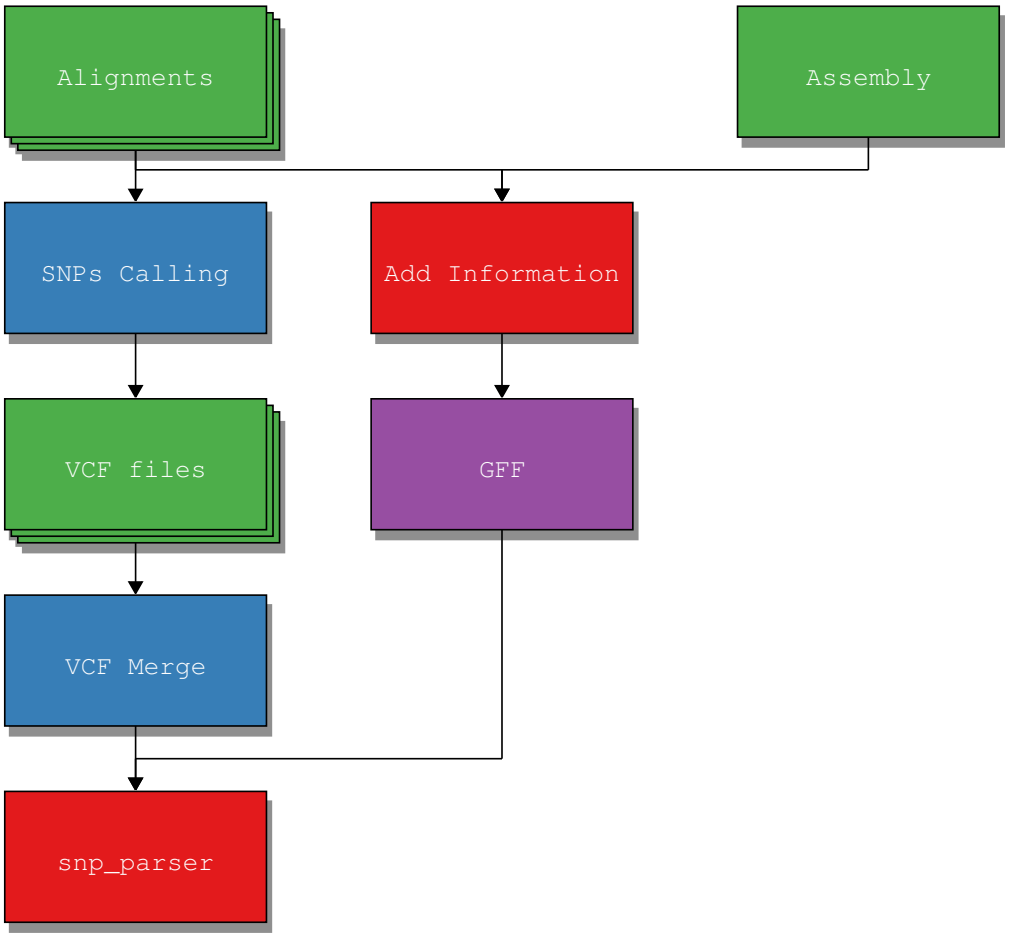
-t, --discard	Evalue over which an hit will be discarded
	Default: 0.05
-d, --disable-evalue	Disable Evalue filter
	Default: False

GFF

-c, --no-custom-profiles	Profiles names are not in the custom format
	Default: True
-db, --database	Database from which the profiles are generated " +" (e.g. PFAM)
	Default: "CUSTOM"
-f, --feature-type	Type of feature (e.g. gene)
	Default: "gene"
-n, --no-frame	Set if the sequences were not translated with translate_seq
	Default: False

4.6 snp_parser - SNPs analysis

4.6.1 Overview



The workflow starts with a number of alignments passed to the SNP calling software, which produces one VCF file per alignment/sample. These VCF files are used by SNPDat[54] along a GTF file and the reference genome to integrate the information in VCF files with synonymous/non-synonymous information.

All VCF files are merged into a VCF that includes information about all the SNPs called among all samples. This merged VCF is passed, along with the results from SNPDat and the GFF file to snp_parser.py which integrates information from all data sources and output files in a format that can be later used by the rest of the pipeline.[55]

Note: The GFF file passed to the parser must have per sample coverage information.

4.6.2 Script Reference

This script parses results of SNPs analysis from any tool for SNP calling[56] and integrates them into a format that can be later used for other scripts in the pipeline.

[54] http://code.google.com/p/snpdat/
[55] This step is done separately because it's both time consuming and can helps to paralellise later steps
[56] GATK pipeline was tested, but it is possible to use samtools and bcftools

It integrates coverage and expected number of syn/nonsyn change and taxonomy from a GFF file, SNP data from a VCF file.

Note: The script accept gzipped VCF files

Changes

Changed in version 0.2.1: added *-s* option for VCF files generated using bcftools

Changed in version 0.1.16: reworkked internals and removed SNPDat, syn/nonsyn evaluation is internal

Changed in version 0.1.13: reworked the internals and the classes used, including options -m and -s

4.6.3 Options

SNPs analysis, requires a vcf file and SNPDat results

```
usage: snp_parser [-h] [-o OUTPUT_FILE] [-q MIN_QUAL] [-f MIN_FREQ]
                  [-r MIN_READS] -g GFF_FILE -p VCF_FILE -a REFERENCE -m
                  SAMPLES_ID [-c COV_SUFF] [-s] [-v | --quiet] [--cite]
                  [--manual] [--version]
```

Named Arguments

-o, --output-file	Ouput file
	Default: snp_data.pickle
-q, --min-qual	Minimum SNP quality (Phred score)
	Default: 30
-f, --min-freq	Minimum allele frequency
	Default: 0.01
-r, --min-reads	Minimum number of reads to accept the SNP
	Default: 4
-g, --gff-file	GFF file with annotations
-p, --vcf-file	Merged VCF file
-a, --reference	Fasta file with the GFF Reference
-m, --samples-id	the ids of the samples used in the analysis
-c, --cov-suff	Per sample coverage suffix in the GFF
	Default: "_cov"
-s, --bcftools-vcf	bcftools call was used to produce the VCF file
	Default: False
-v, --verbose	more verbose - includes debug messages
	Default: 20
--quiet	less verbose - only error and critical messages
--cite	Show citation for the framework
--manual	Show the script manual

--version show program's version number and exit

4.7 Download Taxonomy

A bash script called **download-taxonomy.sh** is installed along with MGKit. This script download the relevant files from NCBI using *wget*, and save the taxonomy file that can be used with MGKit to a file called **taxonomy.pickle**.

Since the script uses *wget* to download the file taxdump.tar.gz[57], if *wget* can't be found, the scripts fails. To avoid this situation, the file can be downloaded in another way, and the script detects if the file exists, avoiding the call of *wget*.

The script can also save the file with another file name, if this is passed when the script is invoked. if the file extension contains *.msgpack*, the **msgpack** module is used to write the taxonomy, otherwise *pickle* is used.

The advantage of *msgpack* is faster read/write and better compression ratio; it needs an additional module (msgpack[58]) that is not installed by default.

4.8 Download Accession/TaxonID

There are 2 separate scripts to download these tables:

- *download-uniprot-taxa.sh* will download a table for Uniprot databases
- *download-ncbi-taxa.sh* for BLAST DBs from NCBI, by default for *nt*, but *nr* can be downloaded with *download-ncbi-taxa.sh prot*

In particular, **nr** refers to the protein database in NCBI, while **nt** refers to the nucleotidic one. Both Uniprot Swissprot and TrEMBL are downloaded by the first scripts.

4.9 taxon-utils - Taxonomy Utilities

4.9.1 Overview

The script contains commands used to access functionality related to taxonomy, without the need to write ad-hoc code for functionality that can be part of a workflow. One example is access to the the last common ancestor function contained in the `mgkit.taxon`.

Last Common Ancestor (lca and lca_line)

These commands expose the functionality of `last_common_ancestor_multiple()`, making it accessible via the command line. They differ in the input file format and the choice of output files.

the *lca* command can be used to define the last common ancestor of contigs from the annotation in a GFF file. The command uses the *taxon_ids* from all annotations belonging to a contig/sequence, if they have a **bitscore** higher or equal to the one passed (50 by default). The default output of the command is a tab separated file where the first column is the contig/sequence name, the taxon_id of the last common ancestor, its scientific/common name and its lineage.

For example:

```
contig_21   172788  uncultured phototrophic eukaryote   cellular organisms,
↪environmental samples
```

[57] ftp://ftp.ncbi.nlm.nih.gov/pub/taxonomy/taxdump.tar.gz
[58] https://github.com/msgpack/msgpack-python

If the *-r* is used, by passing the fasta file containing the nucleotide sequences the output file is a GFF where for each an annotation for the full contig length contains the same information of the tab separated file format.

The **lca_line** command accept as input a file where each line consist of a list of taxon_ids. The separator for the list can be changed and it defaults to TAB. The last common ancestor for all taxa on a line is searched. The ouput of this command is the same as the tab separated file of the **lca** command, with the difference that instead of the first column, which in this command becames a list of all *taxon_ids* that were used to find the last common ancestor for that line. The list of *taxon_ids* is separated by semicolon ";".

Note: Both also accept the *-n* option, to report the config/line and the taxon_ids that had no common ancestors. These are treated as errors and do not appear in the output file.

Krona Output

New in version 0.3.0.

The *lca* command supports the writing of a file compatible with Krona. The output file can be used with the *ktImportText/ImportText.pl* script included with KronaTools[59]. Specifically, the output from *taxon_utils* will be a file with all the lineages found (tab separated), that can be used with:

```
$ ktImportText -q taxon_utils_ouput
```

Note the use of *-q* to make the script count the lineages. Sequences with no LCA found will be marked as *No LCA* in the graph, the *-n* is not required.

Note: Please note that the output won't include any sequence that didn't have a hit with the software used. If that's important, the **-kt** option can be used to add a number of *Unknown* lines at the end, to read the total supplied.

Filter by Taxon

The **filter** command of this script allows to filter a GFF file using the *taxon_id* attribute to include only some annotations, or exclude some. The filter is based on the *mgkit.taxon.is_ancestor* function, and the *mgkit.filter.taxon.filter_taxon_by_id_list*. It can also filter a table (tab separated values) when the first element is an ID and the second is a taxon_id. An example of a table of this sort is the output of the *download-ncbi-taxa.sh* and *download-uniprot-taxa.sh*, where each accession of a database is associated to a taxon_id.

Multiple taxon_id can be passed, either for inclusion or exclusion. If both exclusion and inclusion is used, the first check is on the inclusion and then on the exclusion. In alternative to passing taxon_id, taxon_names can be passed, with values such as 'cellular organisms' that needs to be quoted. Example:

```
$ taxon-utils filter -i 2 -in archaea -en prevotella -t taxonomy.pickle in.gff out.
↪gff
```

Which will keep only line that are from Bacteria (taxon_id=2) and exclude those from the genus *Prevotella*. It will be also include Archaea.

Multiple inclusion and exclusion flags can be put:

```
$ taxon-utils filter -i 2 -i 2172 -t taxonomy in.gff out.gff
```

In particular, the inclusion flag is tested first and then the exclusion is tested. So a line like this one:

```
printf "TEST\t838\nTEST\t1485" | taxon-utils filter -p -t taxonomy.pickle -i 2 -i␣
↪1485 -e 838
```

[59] https://github.com/marbl/Krona/wiki

Will produce **TEST 1485**, because both Prevotella (838) and Clostridium (1485) are Bacteria (2) OR Prevotella, but Prevotella must be excluded according to the exclusion option. This line also illustrate that a tab-separated file, where the second column contains taxon IDs, can be filtered. In particular it can be applied to files produced by *download-ncbi-taxa.sh* or *download-uniprot-taxa.sh* (see *Download Taxonomy*).

Warning: Annotations with no taxon_id are not included in the output of both filters

Convert Taxa Tables to HDF5

This command is used to convert the taxa tables download from Uniprot and NCBI, using the scripts mentioned in download-data, *download-uniprot-taxa.sh* and *download-ncbi-taxa* into a HDF5 file that can be used with the *addtaxa* command in *add-gff-info - Add informations to GFF annotations*.

The advantage is a faster lookup of the IDs. The other is a smaller memory footprint when a great number of annotations are kept in memory.

Changes

Changed in version 0.3.4: changed interface and behaviour for *filter*, also now can filter tables; *lca* has changed the interface and allows the output of a 2 column table

Changed in version 0.3.1: added *to_hdf* command

Changed in version 0.3.1: added *-j* option to *lca*, which outputs a JSON file with the LCA results

Changed in version 0.3.0: added *-k* and *-kt* options for Krona output, lineage now includes the LCA also added *-a* option to select between lineages with only ranked taxa. Now it defaults to all components.

Changed in version 0.2.6: added *feat-type* option to *lca* command, added phylum output to nolca

New in version 0.2.5.

4.9.2 Options

taxon-utils

Main function

```
taxon-utils [OPTIONS] COMMAND [ARGS]...
```

Options

--version
Show the version and exit.

--cite

filter

Filter a GFF file or a table based on taxonomy

```
taxon-utils filter [OPTIONS] [INPUT_FILE] [OUTPUT_FILE]
```

Options

-v, --verbose

-p, --table

-t, --taxonomy <taxonomy>
Taxonomy file [required]

-i, --include-taxon-id <include_taxon_id>
Include only taxon_ids

-in, --include-taxon-name <include_taxon_name>
Include only taxon_names

-e, --exclude-taxon-id <exclude_taxon_id>
Exclude taxon_ids

-en, --exclude-taxon-name <exclude_taxon_name>
Exclude taxon_names

--progress
Shows Progress Bar

Arguments

INPUT_FILE
Optional argument

OUTPUT_FILE
Optional argument

lca

Finds the last common ancestor for each sequence in a GFF file

```
taxon-utils lca [OPTIONS] [GFF_FILE] [OUTPUT_FILE]
```

Options

-v, --verbose

-t, --taxonomy <taxonomy>
Taxonomy file [required]

-n, --no-lca <no_lca>
File to which write records with no LCA

-a, --only-ranked
If set, only taxa that have a rank will be used in the lineageself. This is not advised for lineages such as Viruses, where the top levels have no rank

-b, --bitscore <bitscore>
Minimum bitscore accepted [default: 0]

-m, --rename
Emulates BLAST behaviour for headers (keep left of first space)

-s, --sorted
If the GFF file is sorted (all of a sequence annotations are contiguos) can use less memory, *sort -s -k 1,1* can be used

-ft, --feat-type <feat_type>
Feature type used if the output is a GFF (default is *LCA*) [default: LCA]

-r, --reference <reference>
Reference file for the GFF, if supplied a GFF file is the output

-p, --simple-table
Uses a 2 column table format (seq_id taxon_id) TAB separated

-kt, --krona-total <krona_total>
Total number of raw sequences (used to output correct percentages in Krona

-f, --out-format <out_format>
Format of output file [default: tab]

Options krona|json|tab|gff

--progress
Shows Progress Bar

Arguments

GFF_FILE
Optional argument

OUTPUT_FILE
Optional argument

lca_line

Finds the last common ancestor for all IDs in a text file line

```
taxon-utils lca_line [OPTIONS] [INPUT_FILE] [OUTPUT_FILE]
```

Options

-v, --verbose

-t, --taxonomy <taxonomy>
Taxonomy file [required]

-n, --no-lca <no_lca>
File to which write records with no LCA

-a, --only-ranked
If set, only taxa that have a rank will be used in the lineageself. This is not advised for lineages such as Viruses, where the top levels have no rank

-s, --separator <separator>
separator for taxon_ids (defaults to TAB)

Arguments

INPUT_FILE
Optional argument

OUTPUT_FILE
Optional argument

to_hdf

Convert a taxa table to HDF5, with the input as tabular format, defaults to stdin. Output file, defaults to (taxa-table.hf5)

```
taxon-utils to_hdf [OPTIONS] [INPUT_FILE] [OUTPUT_FILE]
```

Options

-v, --verbose

-n, --table-name <table_name>
Name of the table/storage to use [default: taxa]

-w, --overwrite
Overwrite the file, instead of appending to it

-s, --index-size <index_size>
Maximum number of characters for the gene_id [default: 12]

-c, --chunk-size <chunk_size>
Chunk size to use when reading the input file [default: 5000000]

--progress
Shows Progress Bar

Arguments

INPUT_FILE
Optional argument

OUTPUT_FILE
Optional argument

4.10 fasta-utils - Fasta Utilities

4.10.1 Overview

New in version 0.3.0.

Scripts that includes some functionality to help use FASTA files with the framework

split command

Used to split a fasta file into smaller fragments

translate command

Used to translate nucleotide sequences into amino acids.

uid command

Used to change a FASTA file headers to a unique ID. A table (tab separated) with the changes made can be kept, using the *–table* option.

Changes

New in version 0.3.0.

Changed in version 0.3.1: added *translate* and *uid* command

Changed in version 0.3.4: ported to *click*

4.10.2 Options

fasta-utils

Main function

```
fasta-utils [OPTIONS] COMMAND [ARGS]...
```

Options

--version
Show the version and exit.

--cite

split

Splits a FASTA file [fasta-file] in a number of fragments

```
fasta-utils split [OPTIONS] [FASTA_FILE]
```

Options

-v, --verbose

-p, --prefix <prefix>
Prefix for the file name in output [default: split]

-n, --number <number>
Number of chunks into which split the FASTA file [default: 10]

-z, --gzip
gzip output files

Arguments

FASTA_FILE
Optional argument

translate

Translate FASTA file [fasta-file] in all 6 frames to [output-file]

```
fasta-utils translate [OPTIONS] [FASTA_FILE] [OUTPUT_FILE]
```

Options

-v, --verbose

-t, --trans-table <trans_table>
translation table [default: universal]

Options bac_plt|drs_mit|inv_mit|prt_mit|universal|vt_mit|yst_alt|yst_mit

--progress
Shows Progress Bar

Arguments

FASTA_FILE
Optional argument

OUTPUT_FILE
Optional argument

uid

Changes each header of a FASTA file [file-file] to a uid (unique ID)

```
fasta-utils uid [OPTIONS] [FASTA_FILE] [OUTPUT_FILE]
```

Options

-v, --verbose

-t, --table <table>
Filename of a table to record the changes (by default discards it)

Arguments

FASTA_FILE
Optional argument

OUTPUT_FILE
Optional argument

4.11 fastq-utils - Fastq Utilities

4.11.1 Overview

Commands

- Interleave/deinterleave paired-end fastq files.
- Converts to FASTA
- sort 2 files to sync the headers

Changes

Changed in version 0.3.4: moved to use click, internal fastq parsing, removed *rand* command

Changed in version 0.3.1: added stdin/stdout defaults for some commands

Changed in version 0.3.0: added *convert* command to FASTA

4.11.2 Options

fastq-utils

Main function

```
fastq-utils [OPTIONS] COMMAND [ARGS]...
```

Options

--version
Show the version and exit.

--cite

convert

Convert FastQ file [fastq-file] to FASTA file [fasta-file]

```
fastq-utils convert [OPTIONS] [FASTQ_FILE] [FASTA_FILE]
```

Options

-v, --verbose

Arguments

FASTQ_FILE
Optional argument

FASTA_FILE
Optional argument

di

Deinterleave sequences from [fastq-file], into [mate1-file] and [mate2-file]

```
fastq-utils di [OPTIONS] [FASTQ_FILE] MATE1_FILE MATE2_FILE
```

Options

-v, --verbose

-s, --strip
Strip additional info

Arguments

FASTQ_FILE
Optional argument

MATE1_FILE
Required argument

MATE2_FILE
Required argument

il

Interleave sequences from [mate1-file] and [mate2-file] into [fastq-file]

```
fastq-utils il [OPTIONS] MATE1_FILE MATE2_FILE [FASTQ_FILE]
```

Options

-v, --verbose

Arguments

MATE1_FILE
Required argument

MATE2_FILE
Required argument

FASTQ_FILE
Optional argument

sort

Sort paired-end sequences from [mate1-input] and [mate2-input] into files [mate1-output] and [mate2-output]

```
fastq-utils sort [OPTIONS] MATE1_INPUT MATE2_INPUT MATE1_OUTPUT MATE2_OUTPUT
```

Options

-v, --verbose

Arguments

MATE1_INPUT
Required argument

MATE2_INPUT
Required argument

MATE1_OUTPUT
Required argument

MATE2_OUTPUT
Required argument

4.12 json2gff - Convert JSON to GFF

4.12.1 Overview

Changed in version 0.3.4: using *click* instead of *argparse*

New in version 0.2.6.

This script converts annotations in JSON format that were created using MGKit back into GFF annotations.

mongodb command

Annotations converted into MongoDB records with *get-gff-info mongodb* can be converted back into a GFF file using this command. It can be useful to get a GFF file as output from a query to a MongoDB instance on the command line.

For example:

mongoexport -d db -c test | json2gff mongodb

will convert all the annotations in the database *db*, collection *test* to the standard out.

4.12.2 Options

json2gff

Main function

```
json2gff [OPTIONS] COMMAND [ARGS]...
```

Options

--version
Show the version and exit.

--cite

mongodb

Convert annotations from a MongoDB instance to GFF

```
json2gff mongodb [OPTIONS] [INPUT_FILE] [GFF_FILE]
```

Options

-v, --verbose

Arguments

INPUT_FILE
Optional argument

GFF_FILE
Optional argument

4.13 sampling-utils - Resampling Utilities

4.13.1 Overview

New in version 0.3.1.

Resampling Utilities

sample command

This command samples from a Fasta or FastQ file, based on a probability defined by the user (0.001 or 1 / 1000 by default, *-r* parameter), for a maximum number of sequences (100,000 by default, *-x* parameter). By default 1 sample is extracted, but as many as desired can be taken, by using the *-n* parameter.

The sequence file in input can be either be passed to the standard input or as last parameter on the command line. By defult a Fasta is expected, unless the *-q* parameter is passed.

The *-p* parameter specifies the prefix to be used, and if the output files can be gzipped using the *-z* parameter.

sample_stream command

It works in the same way as *sample*, however the file is sampled only once and the output is the stdout by default. This can be convenient if streams are a preferred way to sample the file.

sync command

Used to keep in sync forward and reverse read files in paired-end FASTQ. The scenario is that the *sample* command was used to resample a FASTQ file, usually the forward, but we need the reverse as well. In this case, the resampled file, called *master* is passed to the *-m* option and the input file is the file that is to be synced (reverse). The input file is scanned until the same header is found in the master file and when that happens, the sequence is written. The next sequence is then read from the master file and the process is repeated until all sequence in the master file are found in the input file. This implies having the 2 files sorted in the same way, which is what the *sample* command does.

Note: the old casava format is not supported by this command at the moment, as it's unusual to find it in SRA or other repositories as well.

---

rand_seq command

Generate random FastA/Q sequences, allowing the specification of GC content and number of sequences being coding or random. If the output format chosen is FastQ, qualities are generated using a decreasing model with added noise. A constant model can be specified instead with a switch. Parameters such GC, length and the type of model can be infered by passing a FastA/Q file, with the quality model fit using a LOWESS (using `mgkit.utils.sequence.extrapolate_model()`). The noise in that case is model as the a normal distribution fitted from the qualities along the sequence deviating from the fitted LOWSS and scaled back by half to avoid too drastic changes in the qualities. Also the qualities are clipped at 40 to avoid compatibility problems with FastQ readers. If inferred, the model can be saved (as a pickle file) and loaded back for analysis

Changes

Changed in version 0.3.4: using *click* instead of *argparse. Now *rand_seq* can save and reload models

Changed in version 0.3.3: added *sync*, *sample_stream* and *rand_seq* commnads

4.13.2 Options

sampling-utils

Main function

```
sampling-utils [OPTIONS] COMMAND [ARGS]...
```

Options

--version
Show the version and exit.

--cite

rand_seq

Generates random FastA/Q sequences

```
sampling-utils rand_seq [OPTIONS] [OUTPUT_FILE]
```

Options

-v, --verbose

-n, --num-seqs <num_seqs>
Number of sequences to generate [default: 1000]

-gc, --gc-content <gc_content>
GC content (defaults to .5 out of 1) [default: 0.5]

-i, --infer-params <infer_params>
Infer parameters GC content and Quality model from file

-r, --coding-prop <coding_prop>
Proportion of coding sequences [default: 0.0]

-l, --length <length>
Sequence length [default: 150]

-d, --const-model
Use a model with constant qualities + noise

-x, --dist-loc <dist_loc>
Use as the starting point quality [default: 30.0]

-q, --fastq
The output file is a FastQ file

-m, --save-model <save_model>
Save inferred qualities model to a pickle file

-a, --read-model <read_model>
Load qualities model from a pickle file

--progress
Shows Progress Bar

Arguments

OUTPUT_FILE
Optional argument

sample

Sample a FastA/Q multiple times

```
sampling-utils sample [OPTIONS] [INPUT_FILE]
```

Options

-v, --verbose

-p, --prefix <prefix>
Prefix for the file name(s) in output [default: sample]

-n, --number <number>
Number of samples to take [default: 1]

-r, --prob <prob>
Probability of picking a sequence [default: 0.001]

-x, --max-seq <max_seq>
Maximum number of sequences [default: 100000]

-q, --fastq
The input file is a fastq file

-z, --gzip
gzip output files

Arguments

INPUT_FILE
Optional argument

sample_stream

Samples a FastA/Q one time, alternative to sample if multiple sampling is not needed

```
sampling-utils sample_stream [OPTIONS] [INPUT_FILE] [OUTPUT_FILE]
```

Options

-v, --verbose

-r, --prob <prob>
Probability of picking a sequence

-x, --max-seq <max_seq>
Maximum number of sequences

-q, --fastq
The input file is a fastq file

Arguments

INPUT_FILE
Optional argument

OUTPUT_FILE
Optional argument

sync

Syncs a FastQ file generated with *sample* with the original pair of files.

```
sampling-utils sync [OPTIONS] [INPUT_FILE] [OUTPUT_FILE]
```

Options

-v, --verbose

-m, --master-file <master_file>
Resampled FastQ file that is out of sync with the original pair [required]

Arguments

INPUT_FILE
Optional argument

OUTPUT_FILE
Optional argument

CHAPTER 5

Example Notebooks

Example notebooks are included about using the library

5.1 Abundance Plots

[1]:
```
import numpy
import mgkit.plots
import mgkit.plots.abund
import seaborn as sns
import pandas as pd
```

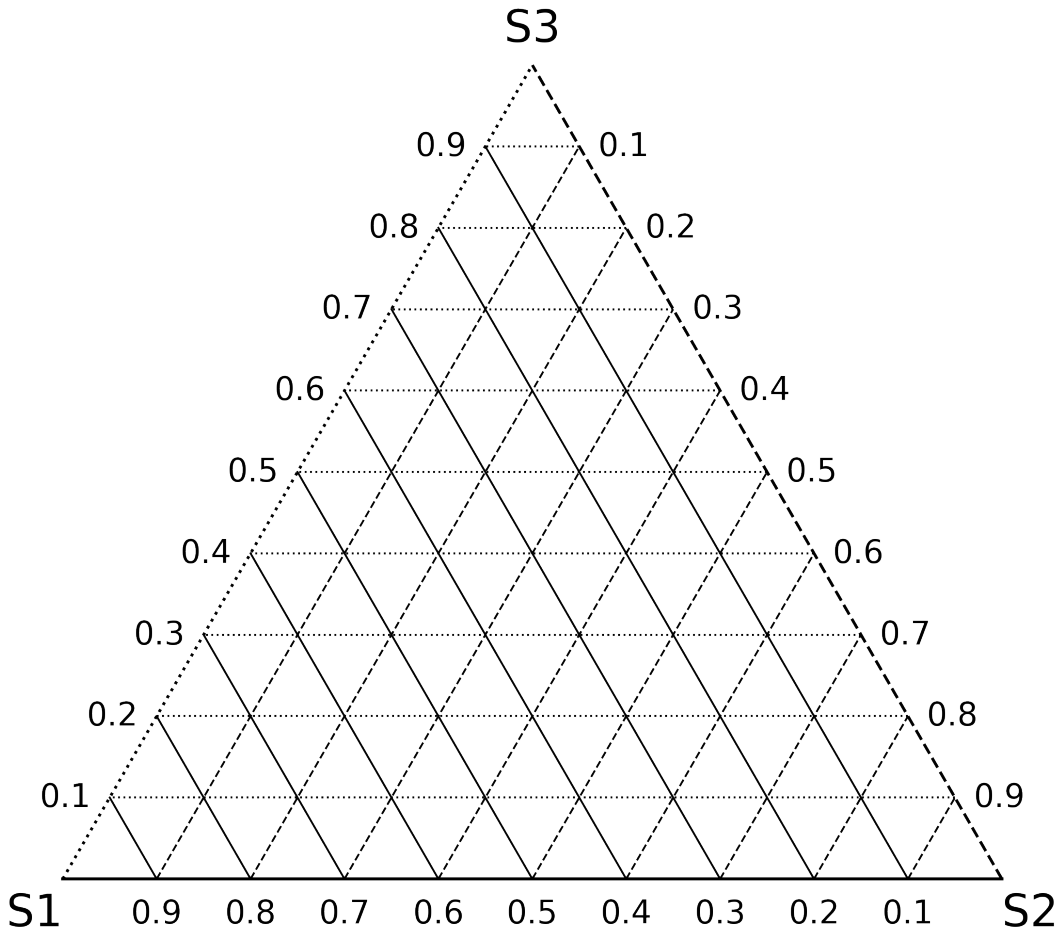
[2]:
```
n = 10
p = 0.50
size = 20
```

[3]:
```
data = pd.DataFrame({
    'S1': numpy.random.negative_binomial(n, p, size),
    'S2': numpy.random.negative_binomial(n, p + 0.1, size),
    'S3': numpy.random.negative_binomial(n, p - 0.1, size),
})
```

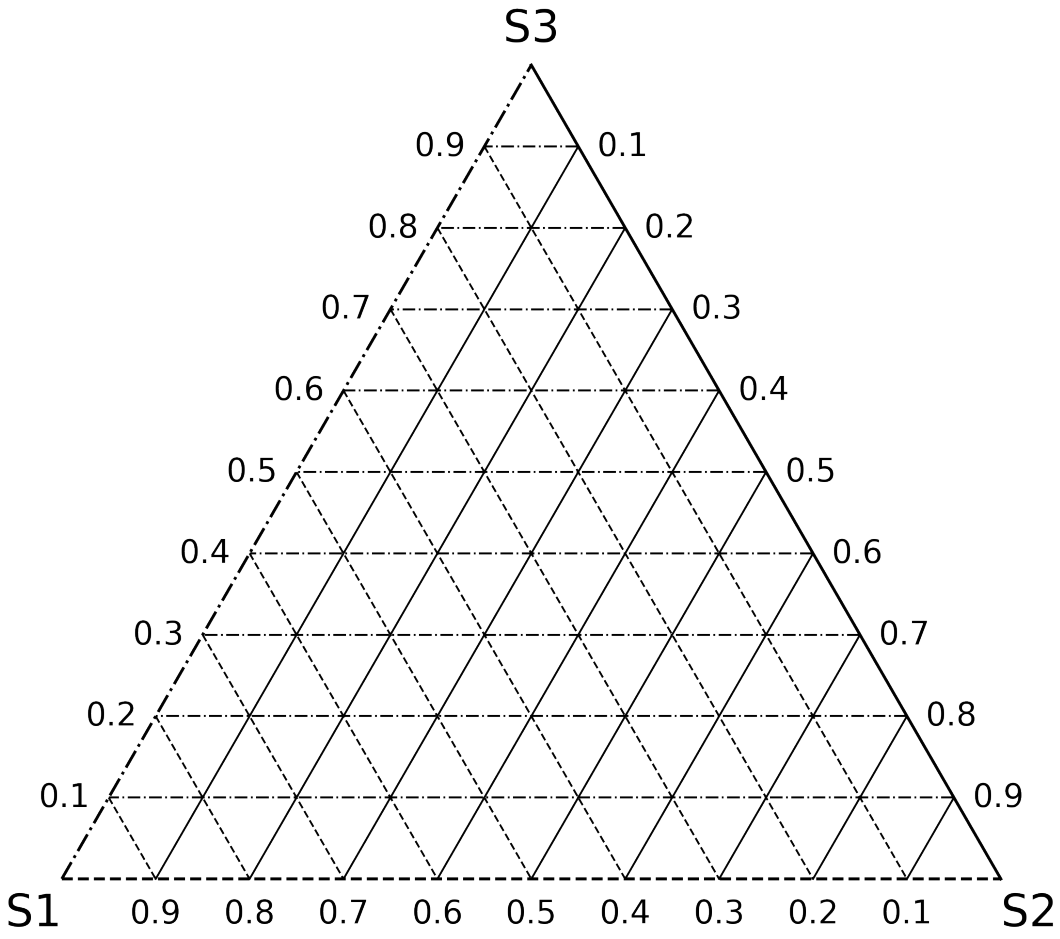
5.1.1 Triangle Plot

Grid

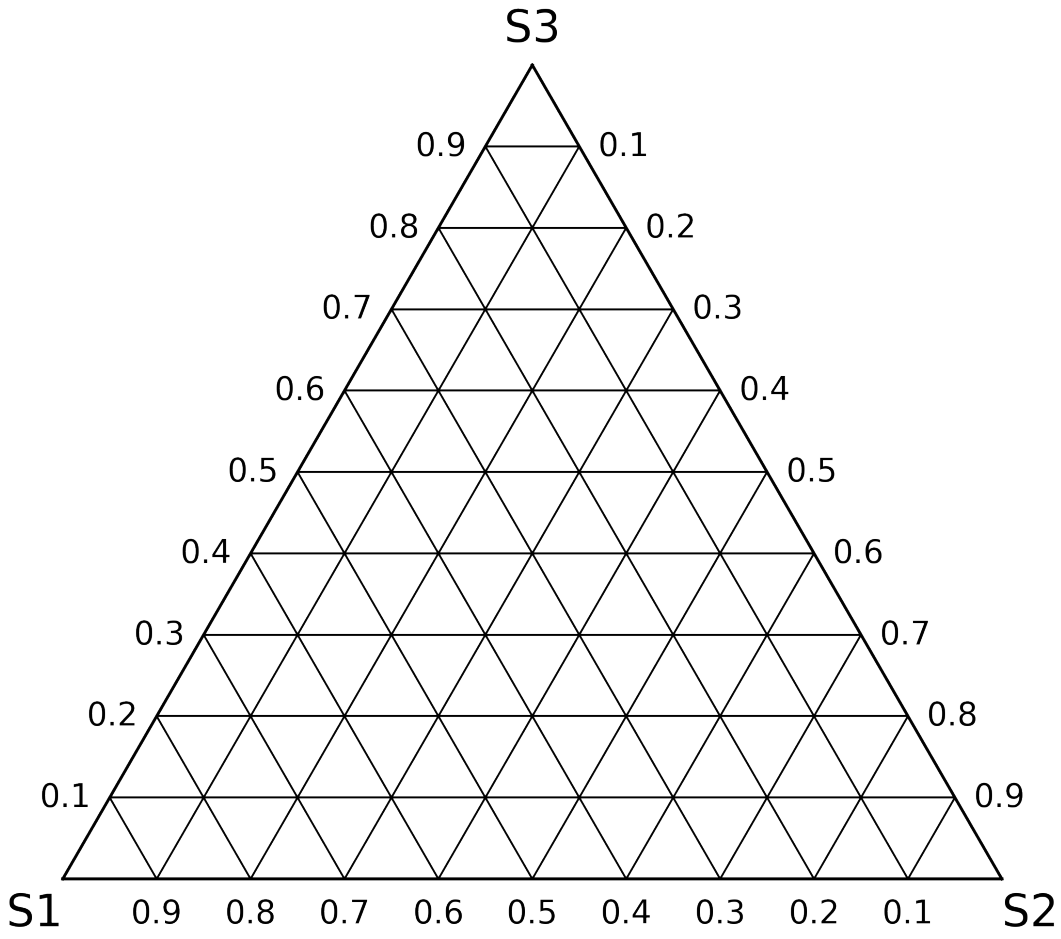
[4]:
```
# First we need to draw the triangle grid
# aspect should be equal to ensure that the triangle sides have the same length
fig, ax = mgkit.plots.get_single_figure(figsize=(10, 10), aspect='equal')
# the labels passed are first drawn from bottom-left, then bottom-right and␣
↪finally top
mgkit.plots.abund.draw_triangle_grid(ax, labels=data.columns)
```



[5]:
```
fig, ax = mgkit.plots.get_single_figure(figsize=(10, 10), aspect='equal')
# the style can be customised by passing the appropriate matplotlib line markers␣
↪with the styles parameter
mgkit.plots.abund.draw_triangle_grid(ax, labels=data.columns, styles=['--', '-.',
↪'-'])
```



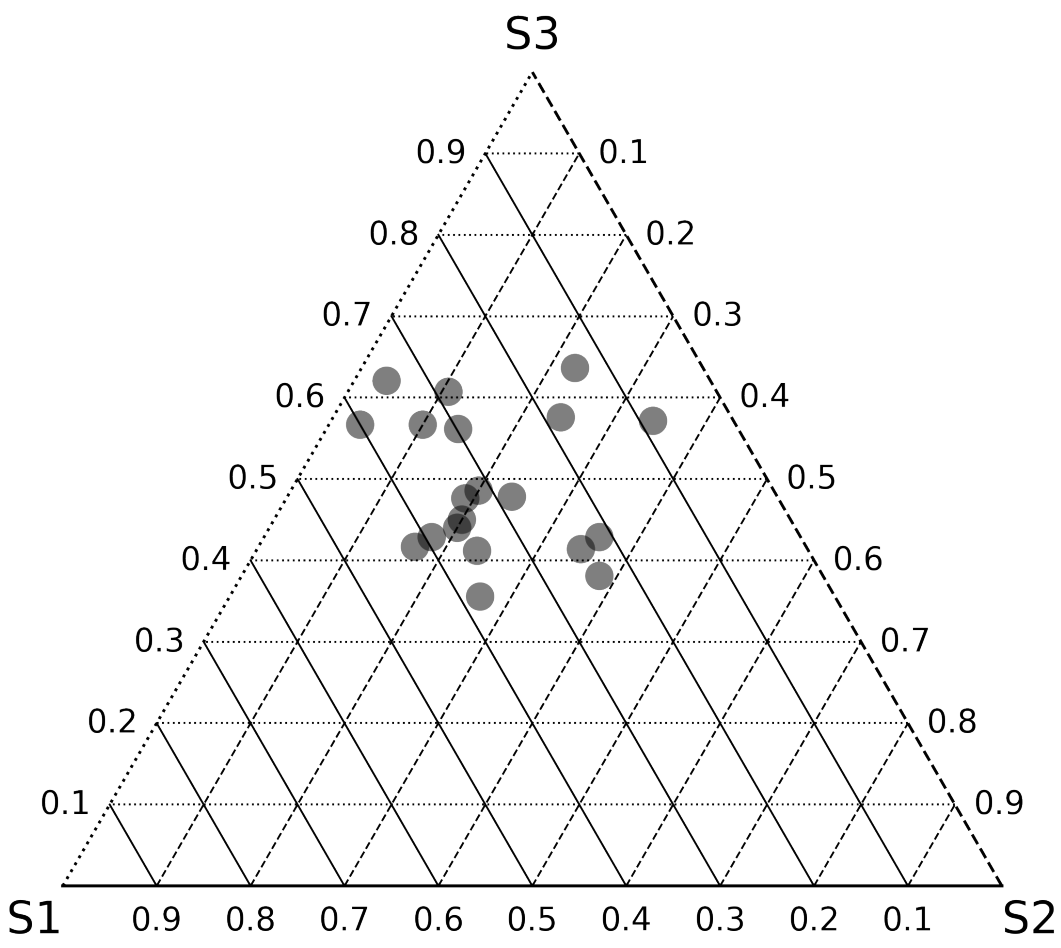
[6]:
```
fig, ax = mgkit.plots.get_single_figure(figsize=(10, 10), aspect='equal')
# The axis can be set to solid lines and the internals to dotted by passing None␣
↪as styles value
mgkit.plots.abund.draw_triangle_grid(ax, labels=data.columns, styles=None)
```



Plot

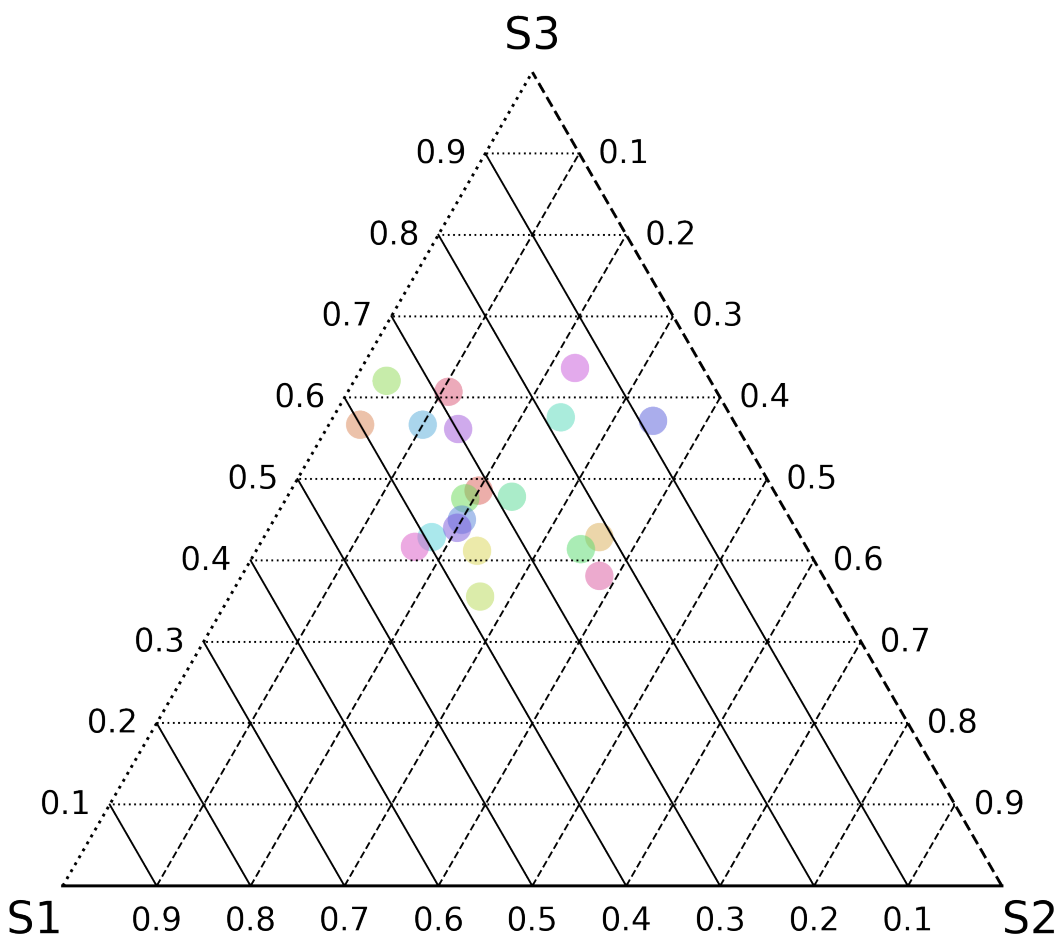
[7]:
```
fig, ax = mgkit.plots.get_single_figure(figsize=(10, 10), aspect='equal')
mgkit.plots.abund.draw_triangle_grid(ax, labels=data.columns)
# this function accept matrices with either two or three columns
mgkit.plots.abund.draw_circles(ax, data)
```

[7]:
```
<matplotlib.collections.PathCollection at 0x7f9d082be8d0>
```



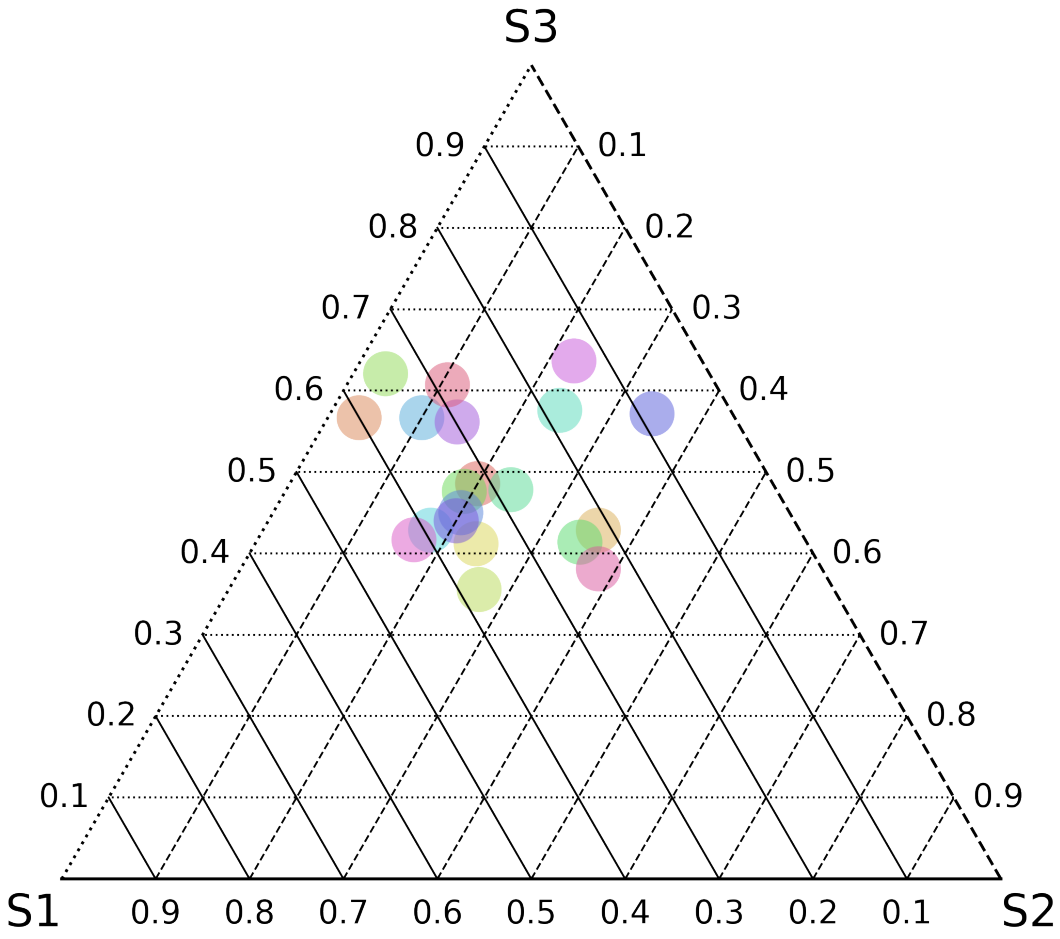
[8]:
```
fig, ax = mgkit.plots.get_single_figure(figsize=(10, 10), aspect='equal')
mgkit.plots.abund.draw_triangle_grid(ax, labels=data.columns)
# col_func is any function that accept a value (an element of data.index) and␣
↪returns a valid matplotlib color for it
col_func = lambda x: sns.color_palette('hls', len(data))[x]
mgkit.plots.abund.draw_circles(ax, data, col_func=col_func)
```

[8]:
```
<matplotlib.collections.PathCollection at 0x7f9cf6e87290>
```



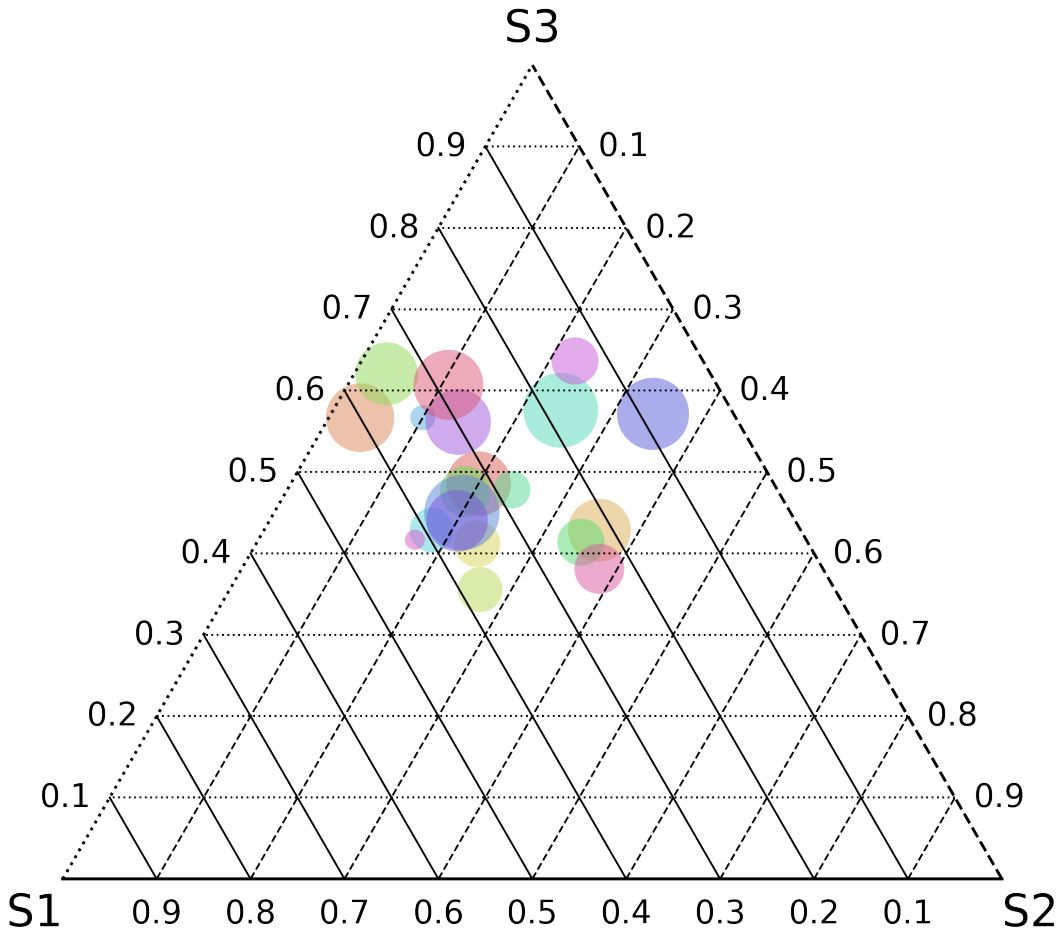
[9]:
```
fig, ax = mgkit.plots.get_single_figure(figsize=(10, 10), aspect='equal')
mgkit.plots.abund.draw_triangle_grid(ax, labels=data.columns)
# csize is the base size for the circle
mgkit.plots.abund.draw_circles(ax, data, col_func=col_func, csize=500)
```

[9]:
```
<matplotlib.collections.PathCollection at 0x7f9cf3581390>
```



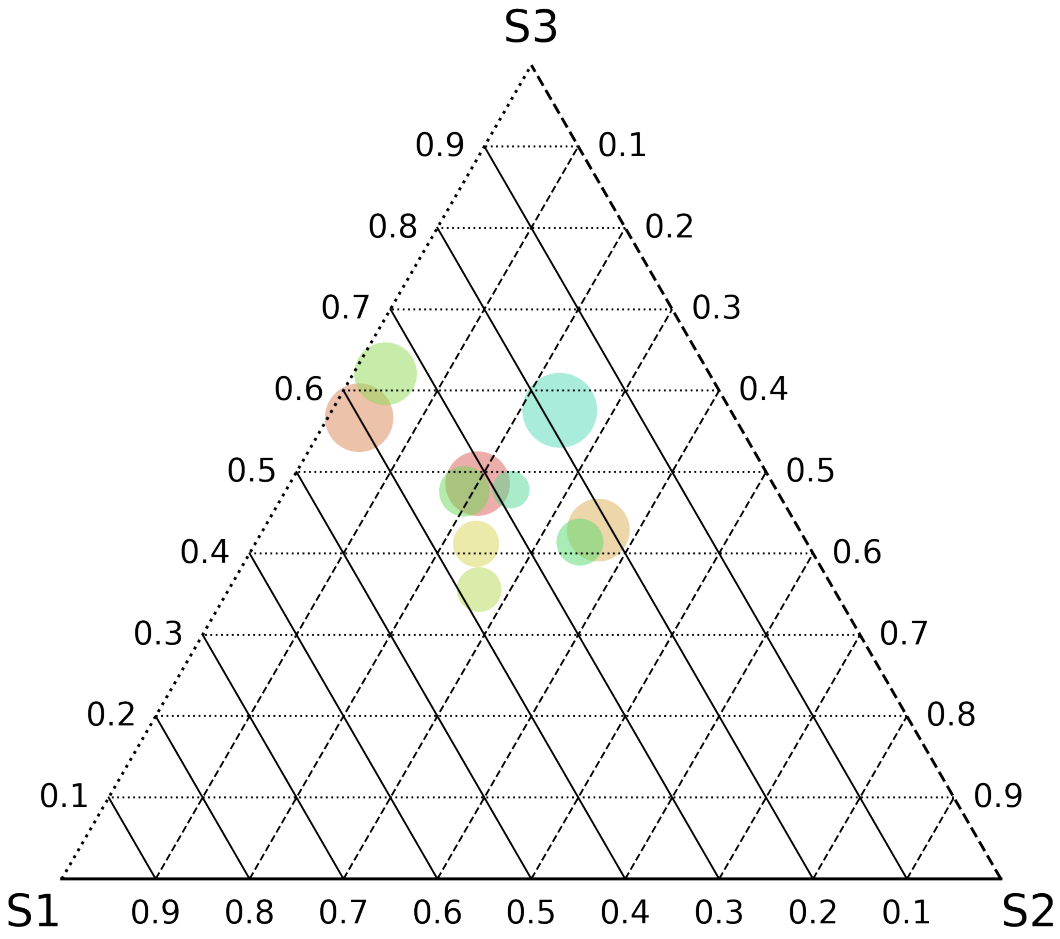
[10]:
```
fig, ax = mgkit.plots.get_single_figure(figsize=(10, 10), aspect='equal')
mgkit.plots.abund.draw_triangle_grid(ax, labels=data.columns)
# the sizescale parameter allows to specify a size factor for each row that is␣
↪multiplied to the csize parameter
sizescale = pd.Series(numpy.random.random(20) * 3)
mgkit.plots.abund.draw_circles(
    ax,
    data,
    col_func=lambda x: sns.color_palette('hls', len(data))[x],
    csize=500,
    sizescale=sizescale
)
```

[10]:
```
<matplotlib.collections.PathCollection at 0x7f9d084bed50>
```



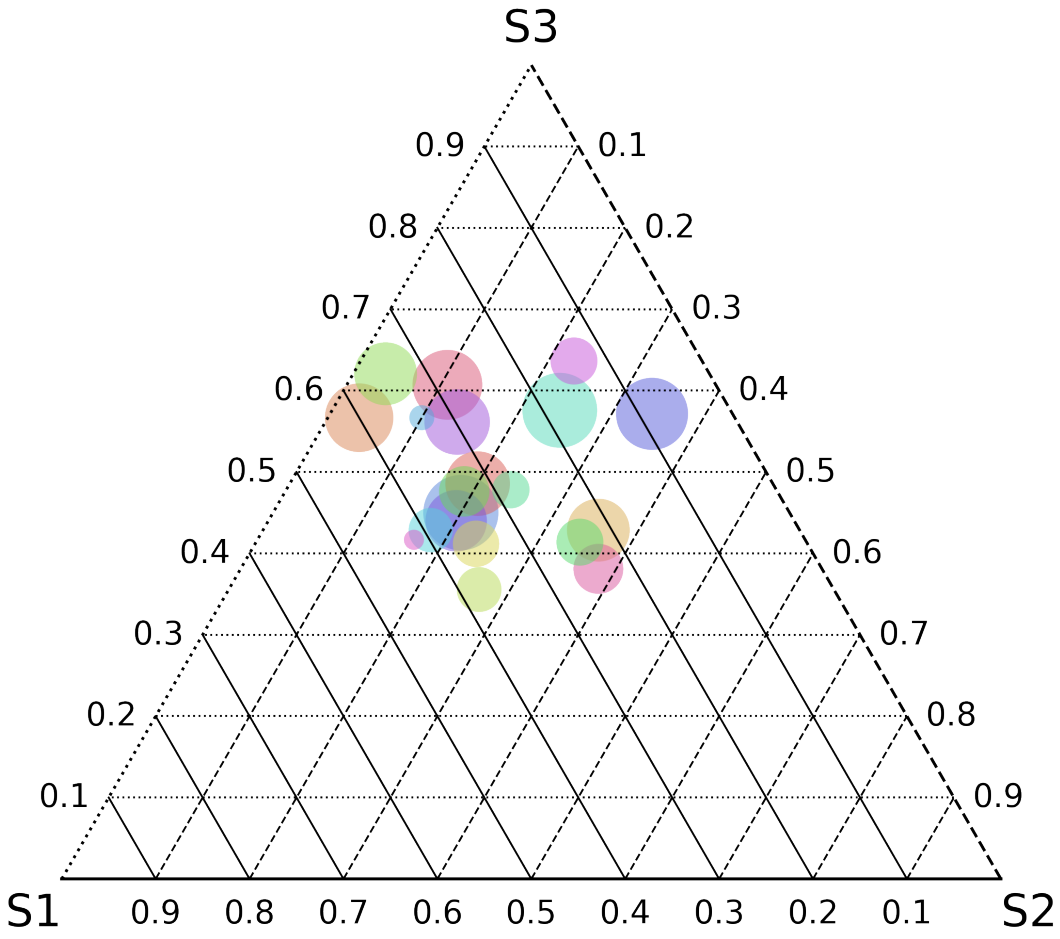
[11]:
```
fig, ax = mgkit.plots.get_single_figure(figsize=(10, 10), aspect='equal')
mgkit.plots.abund.draw_triangle_grid(ax, labels=data.columns)
# the order parameter can be used to only plot only a subset of the point
mgkit.plots.abund.draw_circles(
    ax,
    data,
    col_func=col_func,
    csize=500,
    sizescale=sizescale,
    order=data.index[:10]
)
```

[11]:
```
<matplotlib.collections.PathCollection at 0x7f9d0ce98110>
```



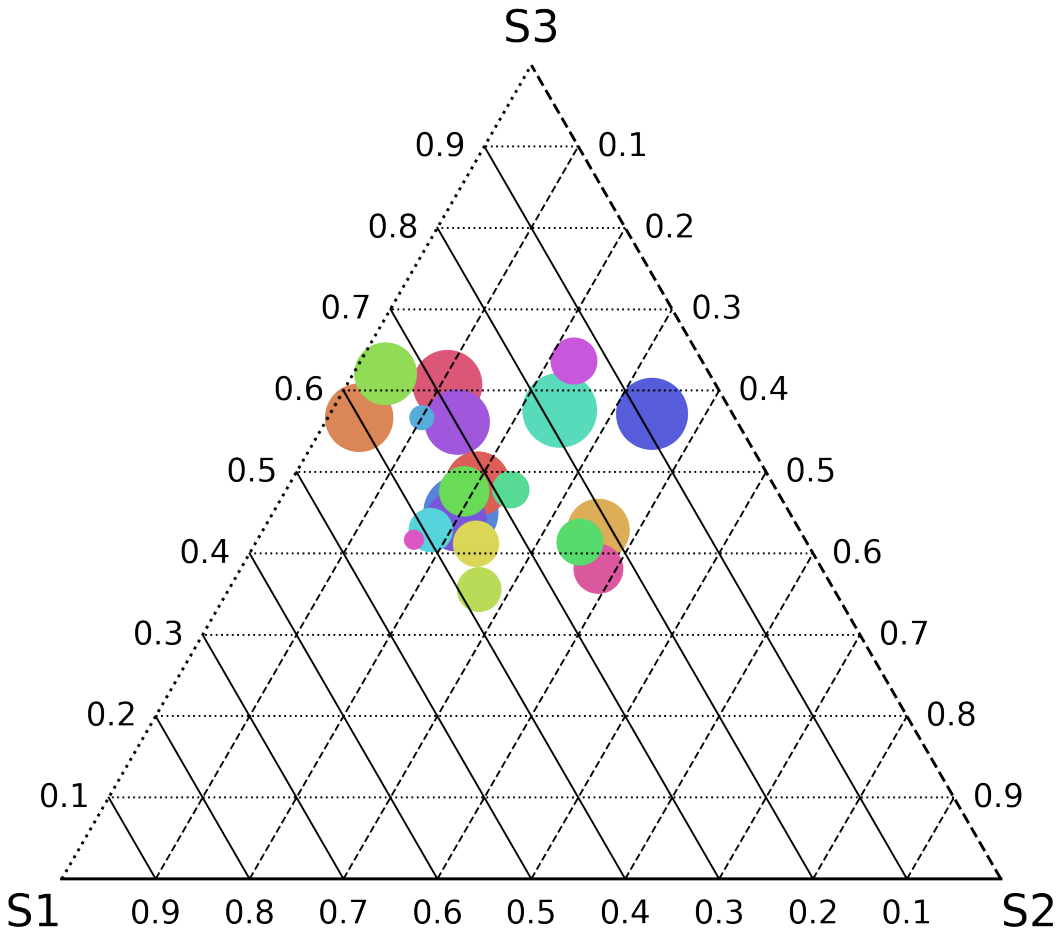
[12]:
```
fig, ax = mgkit.plots.get_single_figure(figsize=(10, 10), aspect='equal')
mgkit.plots.abund.draw_triangle_grid(ax, labels=data.columns)
# or also to make sure bigger circles are drawn first, below smaller ones
mgkit.plots.abund.draw_circles(
    ax,
    data,
    col_func=col_func,
    csize=500,
    sizescale=sizescale,
    order=sizescale.sort_values(ascending=False, inplace=False).index
)
```

[12]:
```
<matplotlib.collections.PathCollection at 0x7f9d0842f710>
```



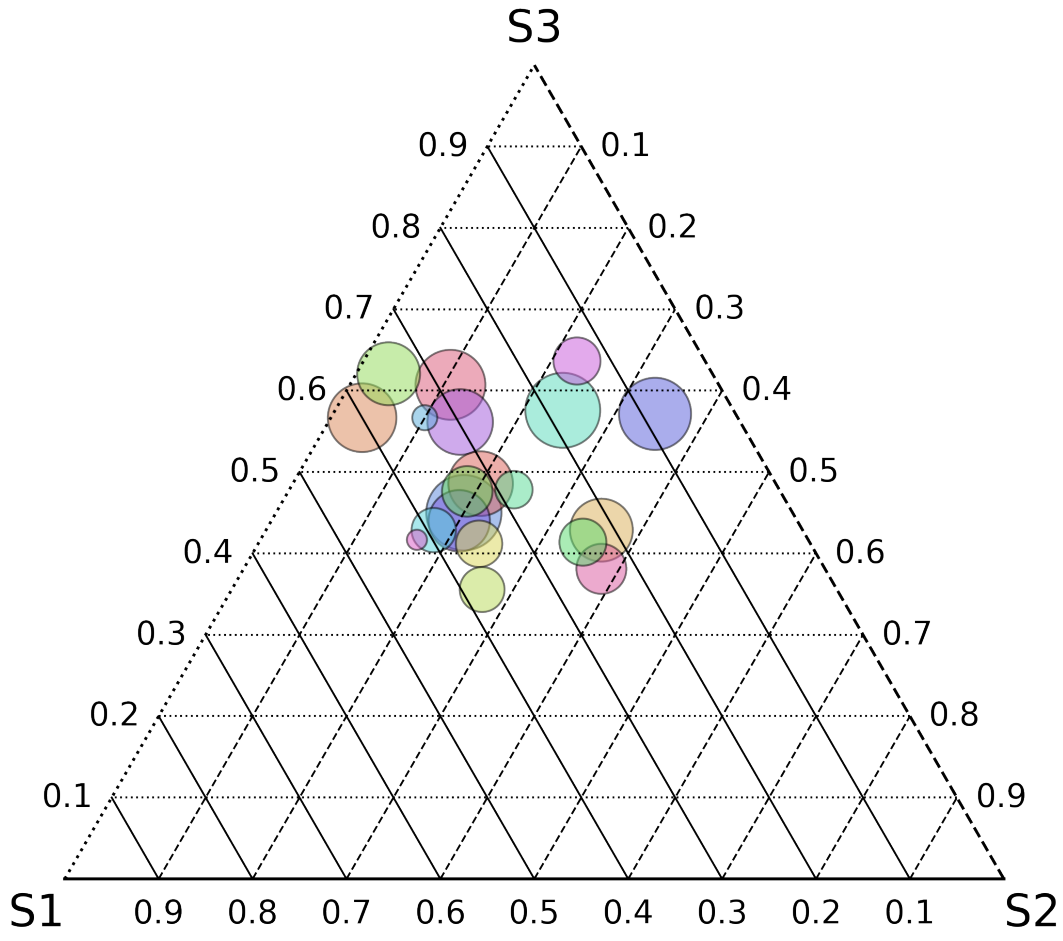
[13]:
```
fig, ax = mgkit.plots.get_single_figure(figsize=(10, 10), aspect='equal')
mgkit.plots.abund.draw_triangle_grid(ax, labels=data.columns)
# trasparency of circles can be adjusted with the alpha parameter (between 0 and 1)
mgkit.plots.abund.draw_circles(
    ax,
    data,
    col_func=col_func,
    csize=500,
    sizescale=sizescale,
    order=sizescale.sort_values(ascending=False, inplace=False).index,
    alpha=1
)
```

[13]:
```
<matplotlib.collections.PathCollection at 0x7f9d0aa5abd0>
```



[14]:
```
fig, ax = mgkit.plots.get_single_figure(figsize=(10, 10), aspect='equal')
mgkit.plots.abund.draw_triangle_grid(ax, labels=data.columns)
# if lines are required around the circles, linewidths and edgecolor can be used␣
↪to customise them
mgkit.plots.abund.draw_circles(
    ax,
    data,
    col_func=col_func,
    csize=500,
    sizescale=sizescale,
    order=sizescale.sort_values(ascending=False, inplace=False).index,
    linewidths=1,
    edgecolor='k'
)
```

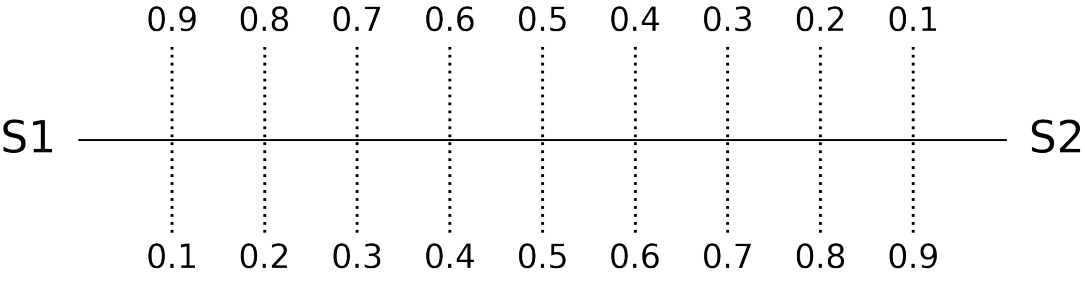
[14]:
```
<matplotlib.collections.PathCollection at 0x7f9d0a9600d0>
```



5.1.2 Abundance Plot with 2 Samples

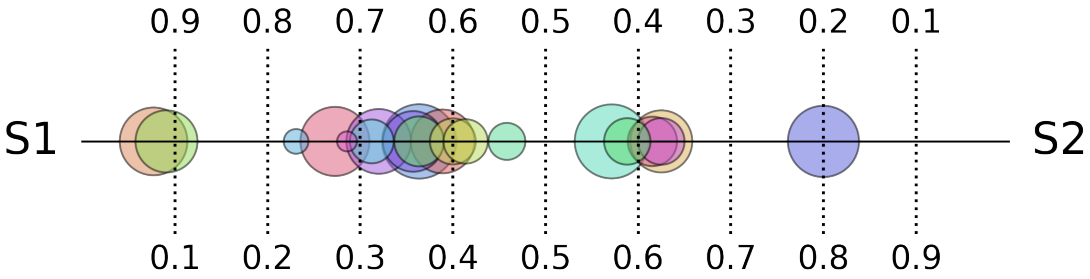
Grid

[15]:
```
# First we need to draw the 1D grid
# aspect should be set to make sure the correct aspect ratio is drawn
fig, ax = mgkit.plots.get_single_figure(figsize=(10, 3), aspect=0.1)
# the labels passed are drawn from left to right
mgkit.plots.abund.draw_1d_grid(ax, labels=data.columns[:2])
```



[16]:
```
fig, ax = mgkit.plots.get_single_figure(figsize=(10, 3), aspect=0.1)
mgkit.plots.abund.draw_1d_grid(ax, labels=data.columns[:2])
mgkit.plots.abund.draw_circles(
    ax,
    data.iloc[:, [0,1]],
    col_func=col_func,
    csize=500,
    sizescale=sizescale,
    order=sizescale.sort_values(ascending=False, inplace=False).index,
    linewidths=1,
    edgecolor='k'
)
```

[16]:
```
<matplotlib.collections.PathCollection at 0x7f9d0a837a50>
```



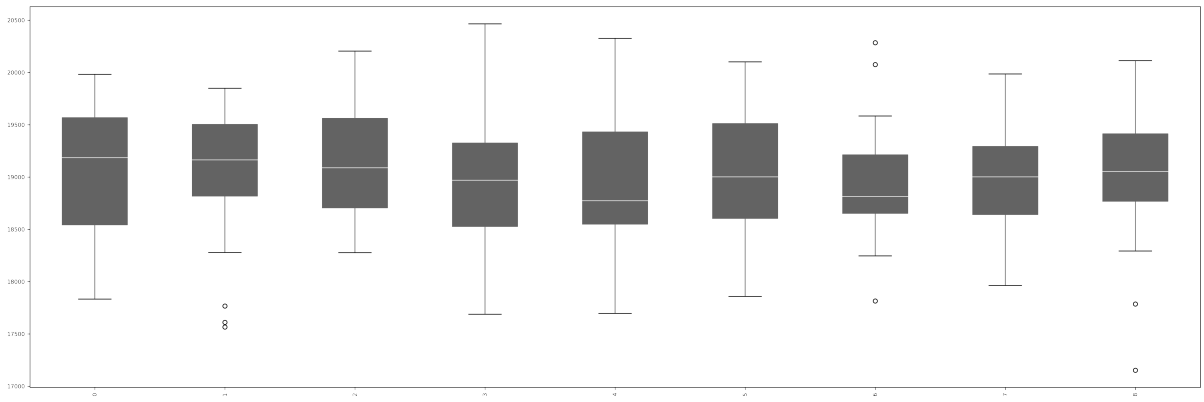
5.2 Boxplots

[1]:
```
import mgkit.plots.boxplot
import numpy
import pandas
import seaborn as sns
```

[2]:
```
nrows = 9
ncols = 30
data = pandas.DataFrame({
    x: numpy.random.negative_binomial(1000, 0.05, size=nrows)
    for x in xrange(ncols)
})
```

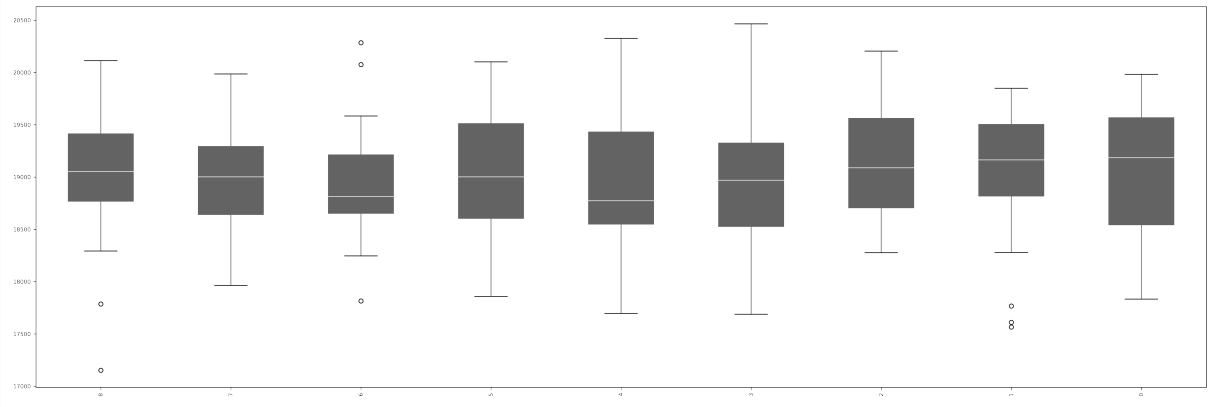
5.2.1 Simple boxplot

[3]:
```
fig, ax = mgkit.plots.get_single_figure(figsize=(30, 10))
_ = mgkit.plots.boxplot.boxplot_dataframe(data, data.index, ax)
```



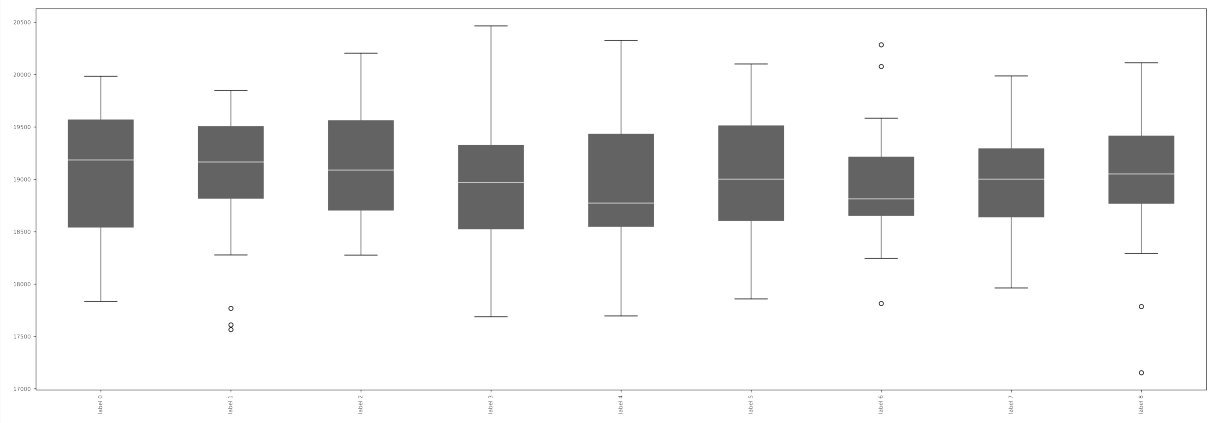
5.2.2 Change order of boxplots

[4]:
```
fig, ax = mgkit.plots.get_single_figure(figsize=(30, 10))
_ = mgkit.plots.boxplot.boxplot_dataframe(data, data.index[::-1], ax)
```



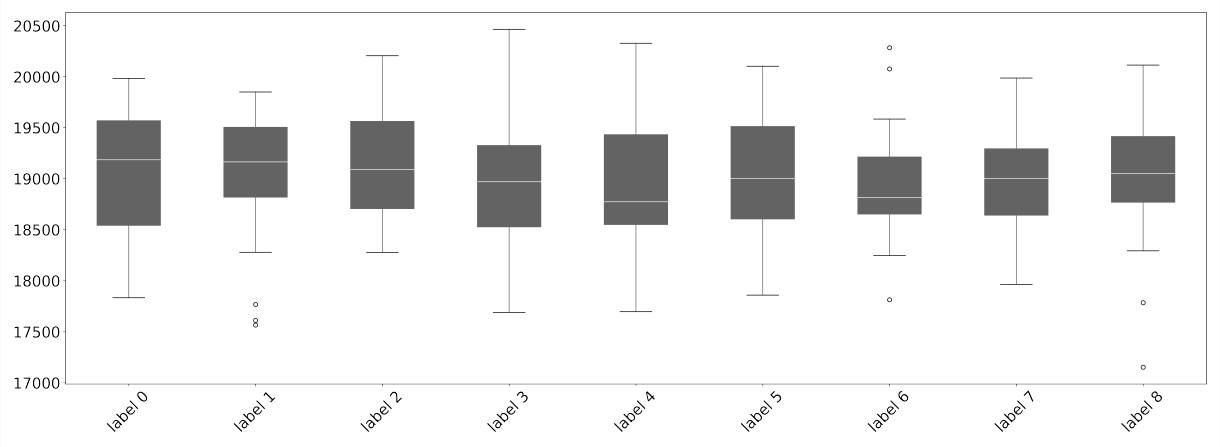
5.2.3 Change labels

[5]:
```
fig, ax = mgkit.plots.get_single_figure(figsize=(30, 10))
_ = mgkit.plots.boxplot.boxplot_dataframe(data, data.index, ax, label_map={x:
↪'label {}'.format(x) for x in data.index})
```



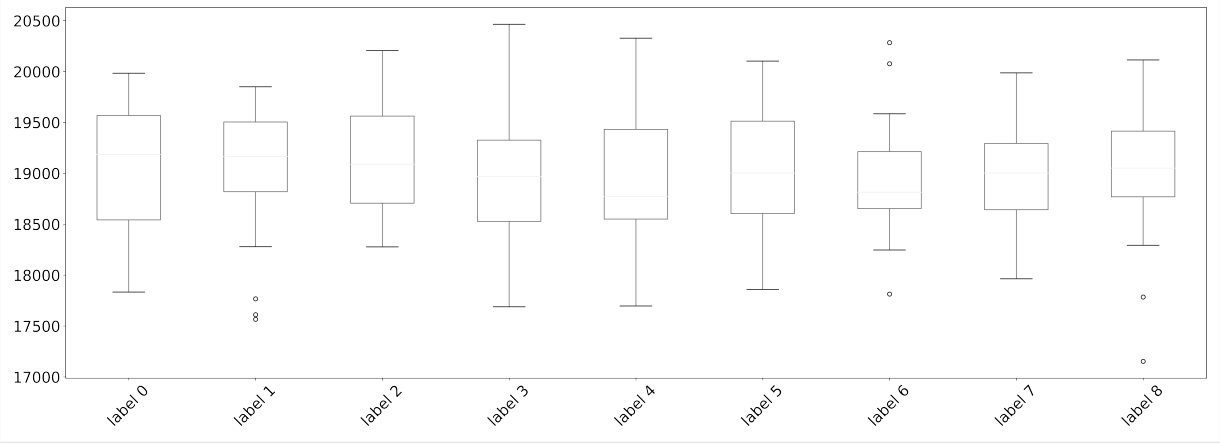
5.2.4 Change font parameters

[6]:
```
fig, ax = mgkit.plots.get_single_figure(figsize=(30, 10))
_ = mgkit.plots.boxplot.boxplot_dataframe(
    data,
    data.index,
    ax,
    label_map={x: 'label {}'.format(x) for x in data.index},
    fonts=dict(fontsize=22, rotation=45)
)
```



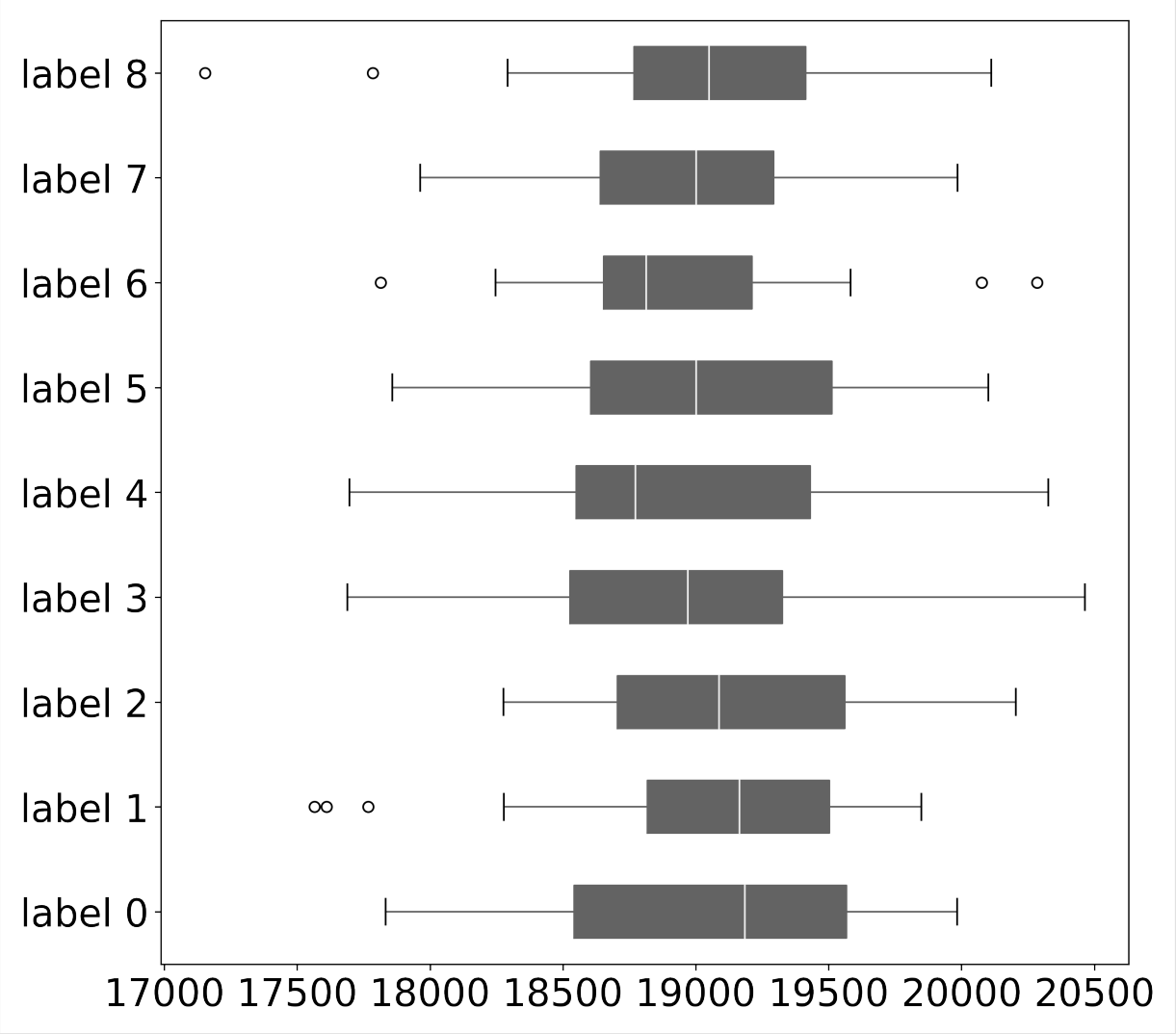
5.2.5 Empty boxplots

[7]:
```
fig, ax = mgkit.plots.get_single_figure(figsize=(30, 10))
_ = mgkit.plots.boxplot.boxplot_dataframe(
    data,
    data.index,
    ax,
    label_map={x: 'label {}'.format(x) for x in data.index},
    fonts=dict(fontsize=22, rotation=45),
    fill_box=False
)
```



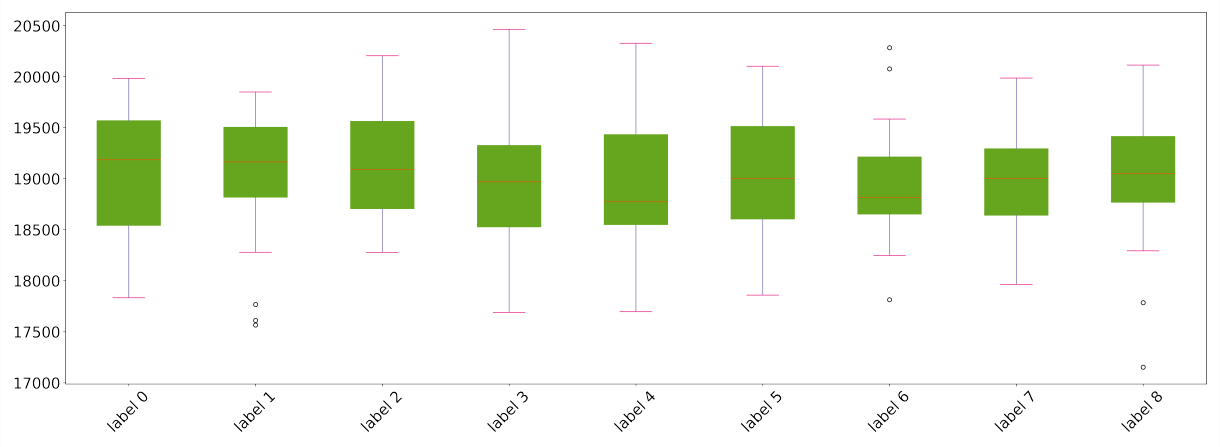
5.2.6 Vertical boxplot

[8]:
```
fig, ax = mgkit.plots.get_single_figure(figsize=(10, 10))
_ = mgkit.plots.boxplot.boxplot_dataframe(
    data,
    data.index,
    ax,
    label_map={x: 'label {}'.format(x) for x in data.index},
    fonts=dict(fontsize=22, rotation='horizontal'),
    fill_box=True,
    box_vert=False
)
```



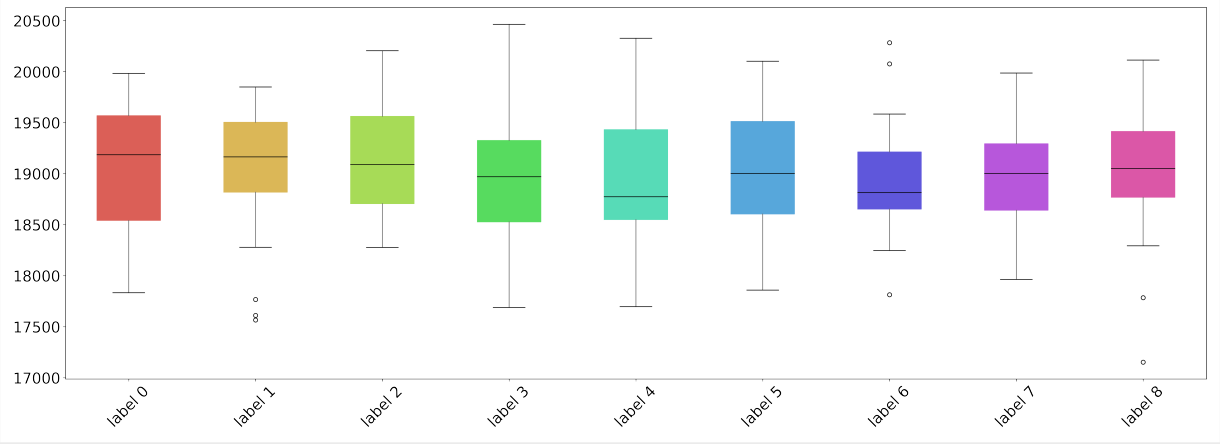
5.2.7 Change boxplot colors

[9]:
```
boxplot_colors = {
    key: col
    for key, col in zip(mgkit.plots.boxplot.DEFAULT_BOXPLOT_COLOURS, sns.color_
↪palette('Dark2', len(mgkit.plots.boxplot.DEFAULT_BOXPLOT_COLOURS)))
}
fig, ax = mgkit.plots.get_single_figure(figsize=(30, 10))
_ = mgkit.plots.boxplot.boxplot_dataframe(
    data,
    data.index,
    ax,
    label_map={x: 'label {}'.format(x) for x in data.index},
    fonts=dict(fontsize=22, rotation=45),
    fill_box=True,
    colours=boxplot_colors
)
```



5.2.8 Change data colors and the median color

[10]:
```
fig, ax = mgkit.plots.get_single_figure(figsize=(30, 10))
_ = mgkit.plots.boxplot.boxplot_dataframe(
    data,
    data.index,
    ax,
    label_map={x: 'label {}'.format(x) for x in data.index},
    fonts=dict(fontsize=22, rotation=45),
    fill_box=True,
    colours=dict(medians='k'),
    data_colours={x: y for x, y in zip(data.index, sns.color_palette('hls',␣
↪len(data.index)))}
)
```



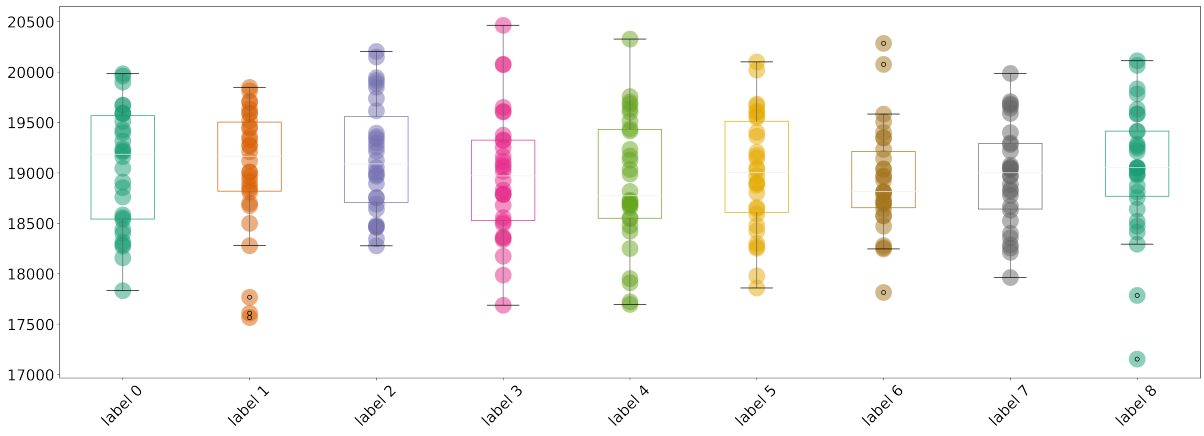
5.2.9 Adding data points

[11]:
```
reload(mgkit.plots.boxplot)
fig, ax = mgkit.plots.get_single_figure(figsize=(30, 10), dpi=300)

data_colours = {x: y for x, y in zip(data.index, sns.color_palette('Dark2',␣
↪len(data.index)))}

plot_data = mgkit.plots.boxplot.boxplot_dataframe(
    data,
    data.index,
    ax,
```

(continues on next page)

(continued from previous page)

```
    label_map={x: 'label {}'.format(x) for x in data.index},
    fonts=dict(fontsize=22, rotation=45),
    fill_box=False,
    data_colours=data_colours,
    box_vert=True
)

#note that box_vert must be the same in both boxplot_dataframe and add_values_to_
↪boxplot. Their default is the opposite, now.
mgkit.plots.boxplot.add_values_to_boxplot(
    data,
    ax,
    plot_data,
    data.index,
    data_colours=data_colours,
    s=600,
    alpha=0.5,
    box_vert=True
)
```



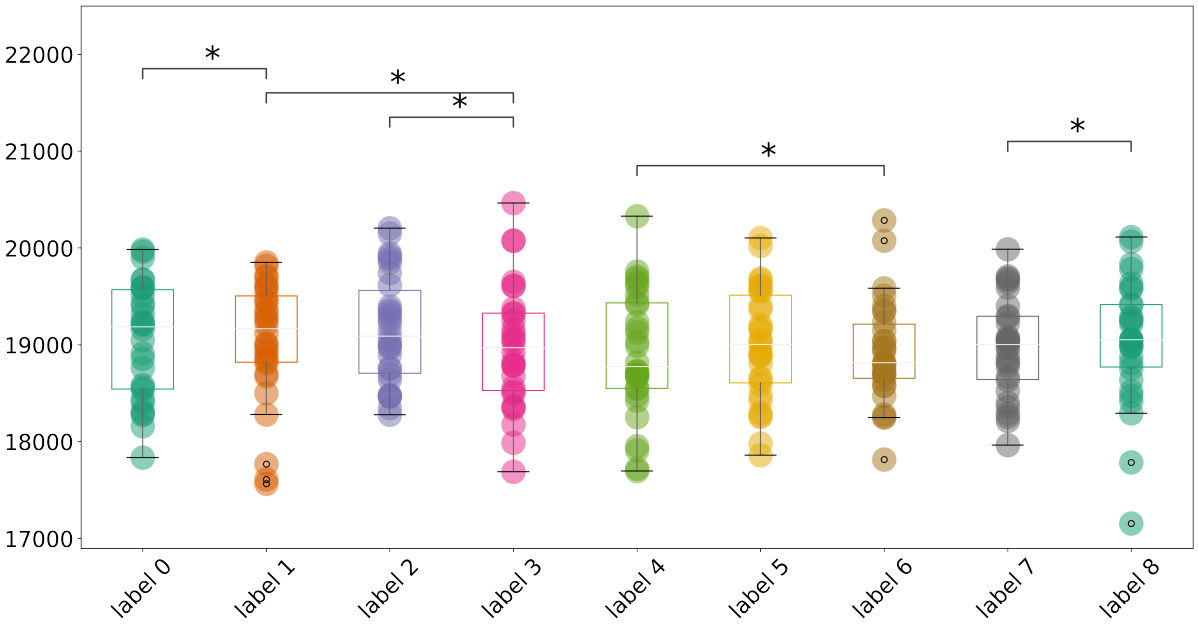
5.2.10 Adding Significance annotations

[12]:
```
reload(mgkit.plots.boxplot)
fig, ax = mgkit.plots.get_single_figure(figsize=(20, 10), dpi=300)

data_colours = {x: y for x, y in zip(data.index, sns.color_palette('Dark2',␣
↪len(data.index)))}

plot_data = mgkit.plots.boxplot.boxplot_dataframe(
    data,
    data.index,
    ax,
    label_map={x: 'label {}'.format(x) for x in data.index},
    fonts=dict(fontsize=22, rotation=45),
    fill_box=False,
    data_colours=data_colours,
    box_vert=True
)

#note that box_vert must be the same in both boxplot_dataframe and add_values_to_
↪boxplot. Their default is the opposite, now.
mgkit.plots.boxplot.add_values_to_boxplot(
    data,
    ax,
```

(continues on next page)

(continued from previous page)

```
    plot_data,
    data.index,
    data_colours=data_colours,
    s=600,
    alpha=0.5,
    box_vert=True
)
mgkit.plots.boxplot.add_significance_to_boxplot(
    [
        (0, 1),
        (1, 3),
        (2, 3),
        (7, 8),
        (4, 6)
    ],
    ax,
    (21850, 21750),
    box_vert=True,
    fontsize=32
)
_ = ax.set_ylim(top=22500)
```



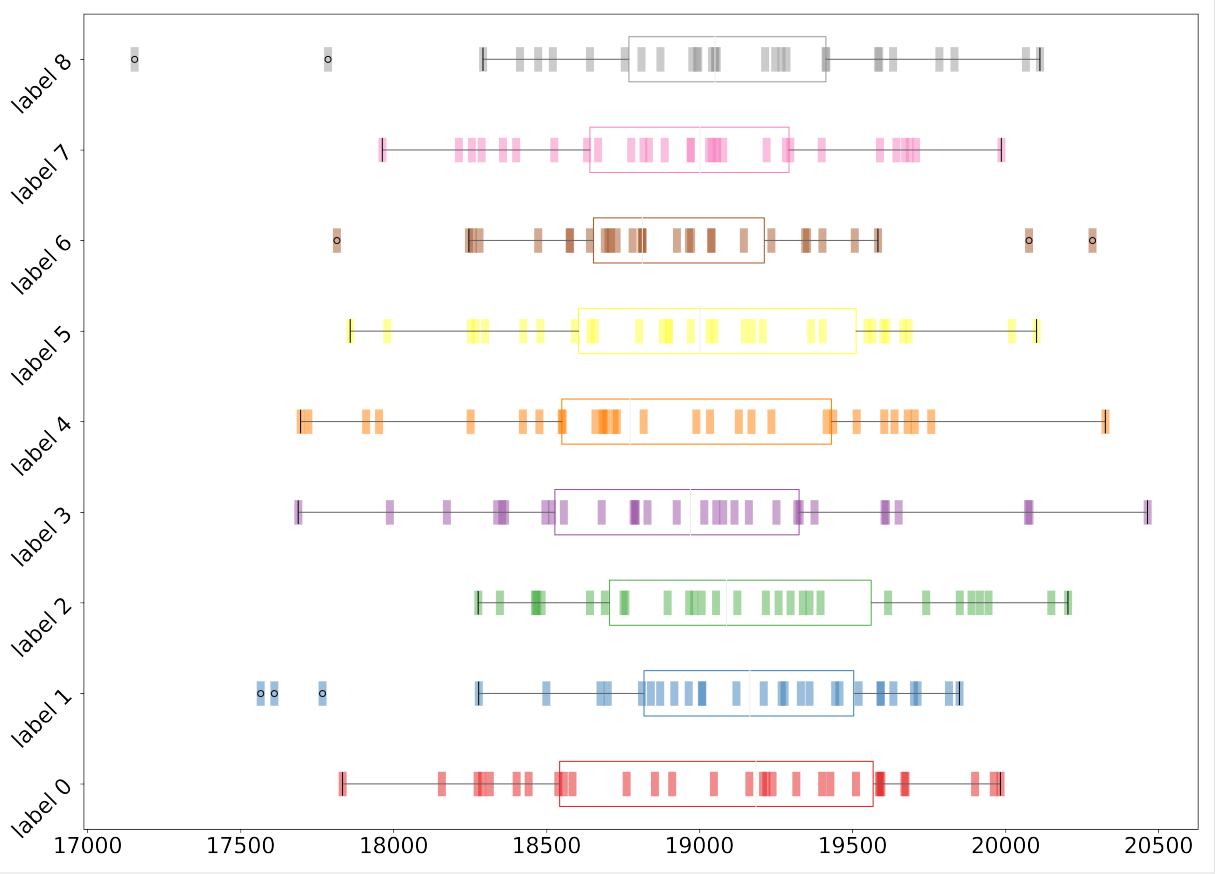
Changed direction, different palette and marker

[13]:
```
reload(mgkit.plots.boxplot)
fig, ax = mgkit.plots.get_single_figure(figsize=(20, 15), dpi=300)

data_colours = {x: y for x, y in zip(data.index, sns.color_palette('Set1',␣
↪len(data.index)))}

plot_data = mgkit.plots.boxplot.boxplot_dataframe(
    data,
    data.index,
    ax,
    label_map={x: 'label {}'.format(x) for x in data.index},
    fonts=dict(fontsize=22, rotation=45),
    fill_box=False,
```

(continues on next page)

(continued from previous page)

```
    data_colours=data_colours,
    box_vert=False
)

#note that box_vert must be the same in both boxplot_dataframe and add_values_to_
↪boxplot. Their default is the opposite, now.
mgkit.plots.boxplot.add_values_to_boxplot(
    data,
    ax,
    plot_data,
    data.index,
    data_colours=data_colours,
    s=600,
    alpha=0.5,
    marker='|',
    linewidth=8,
    box_vert=False
)
```



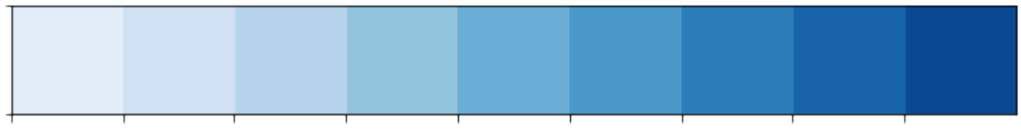
5.3 Heatmaps

[1]:
```
import mgkit.plots
import numpy
import pandas
import seaborn as sns
import matplotlib.colors
```

5.3.1 Random matrix and color map init

[2]:
```
nrow = 50
ncol = nrow

data = pandas.DataFrame(
{
    x: numpy.random.negative_binomial(500, 0.5, nrow)
    for x in xrange(ncol)
}
)
```

[3]:
```
sns.palplot(sns.color_palette('Blues', 9))
```

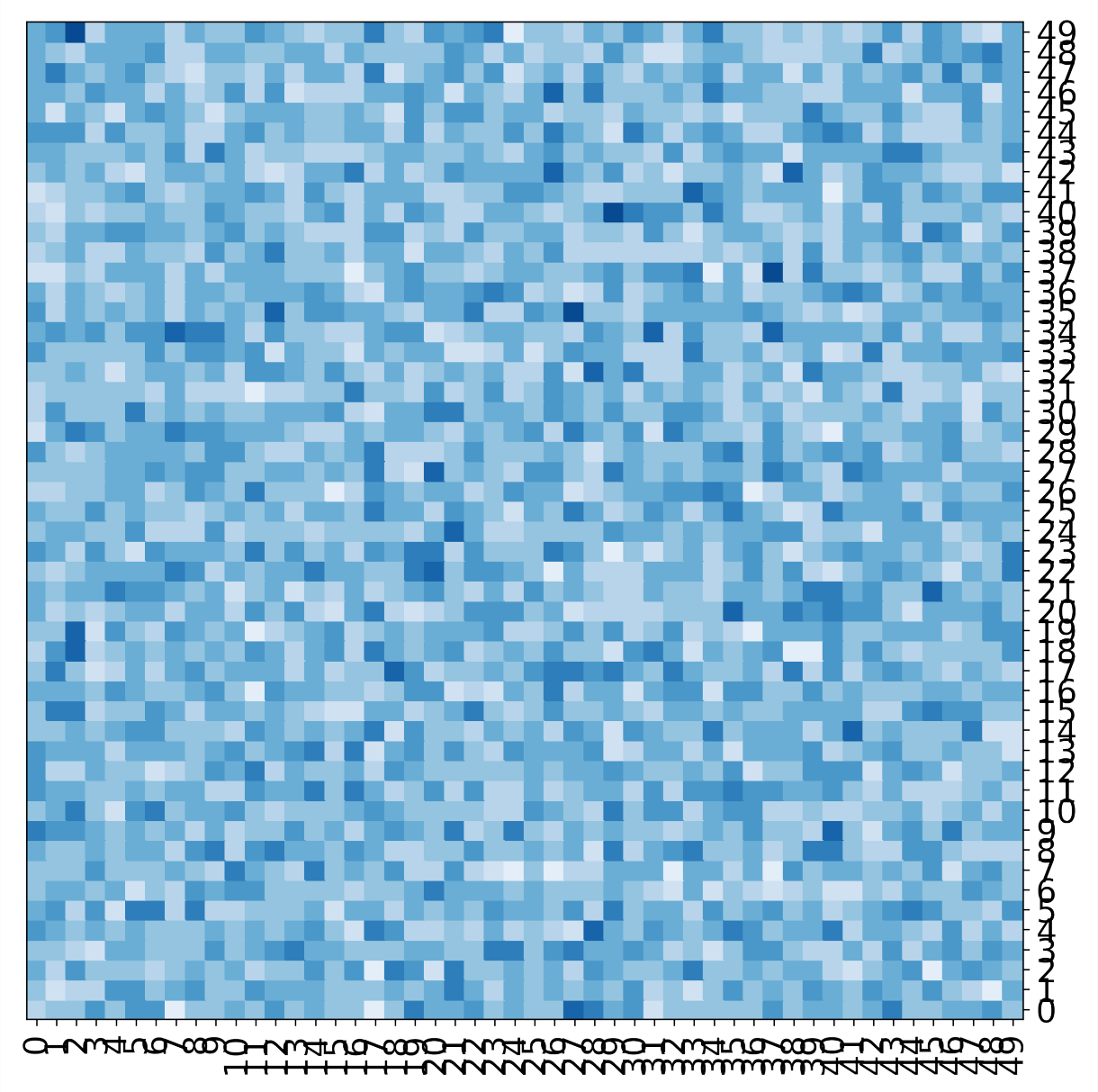


[4]:
```
cmap = matplotlib.colors.ListedColormap(sns.color_palette('Blues', 9))
```

5.3.2 Basic plot

[5]:
```
fig, ax = mgkit.plots.get_single_figure(figsize=(10,10), aspect='equal')
mgkit.plots.heatmap.baseheatmap(data, ax, cmap=cmap)
```

[5]:
```
<matplotlib.collections.QuadMesh at 0x7f83af51e850>
```

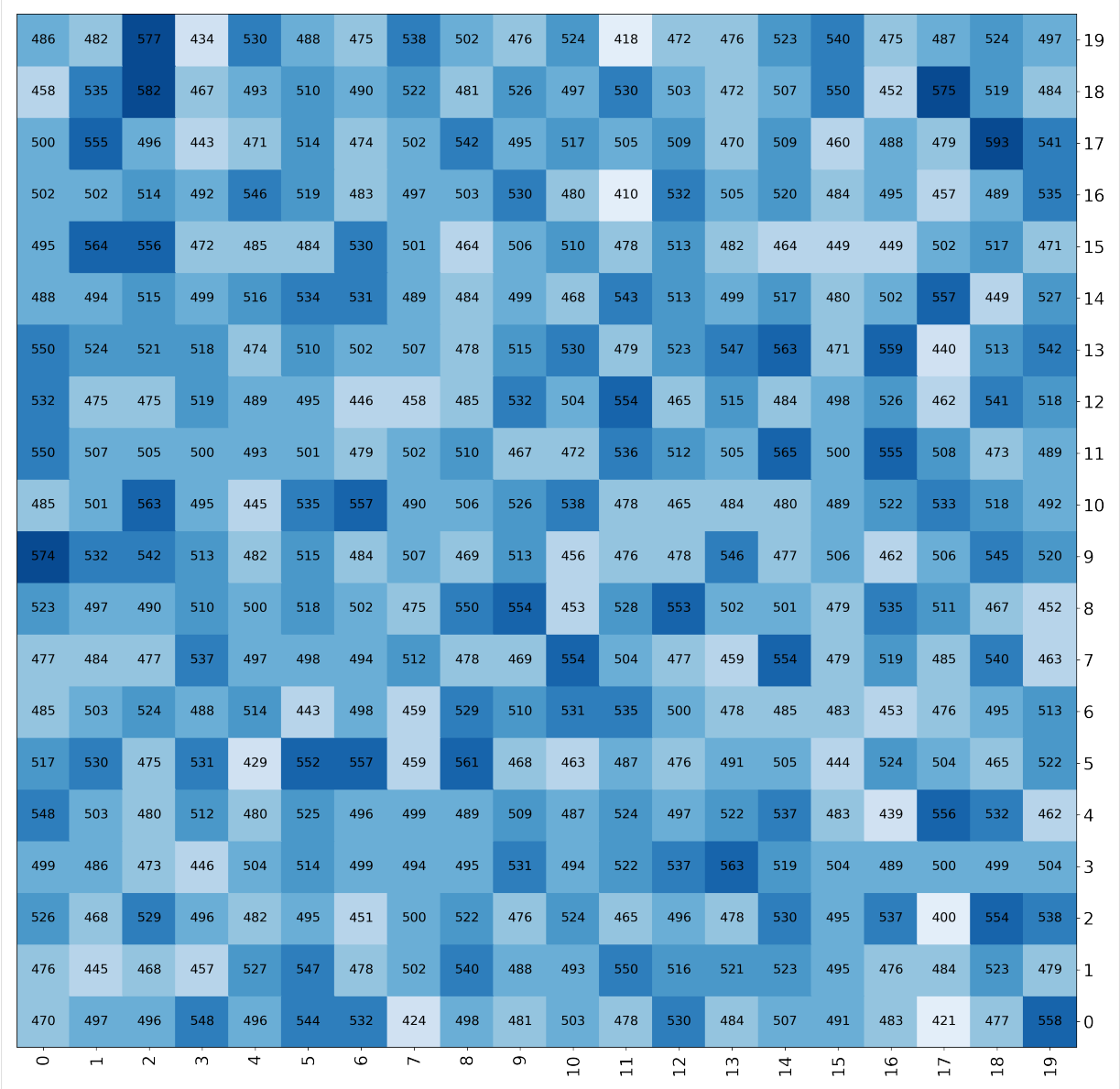


5.3.3 Add numbers to the heatmap

Default

[6]:
```
fig, ax = mgkit.plots.get_single_figure(figsize=(20,20))
mgkit.plots.heatmap.baseheatmap(data.iloc[:20, :20], ax, cmap=cmap, annot=True)
```

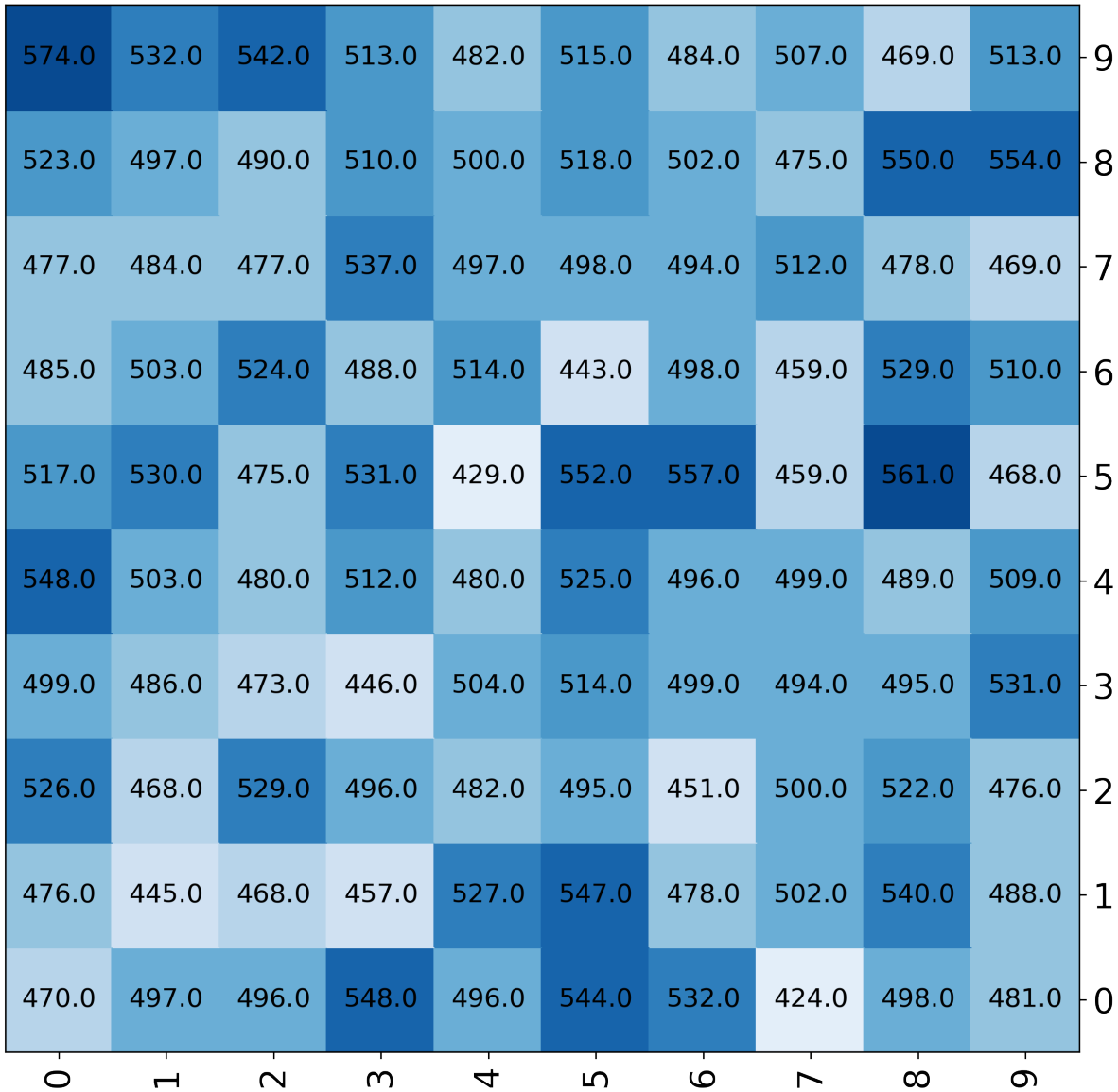
[6]:
```
<matplotlib.collections.QuadMesh at 0x7f83ab9d1110>
```



Change format of numbers

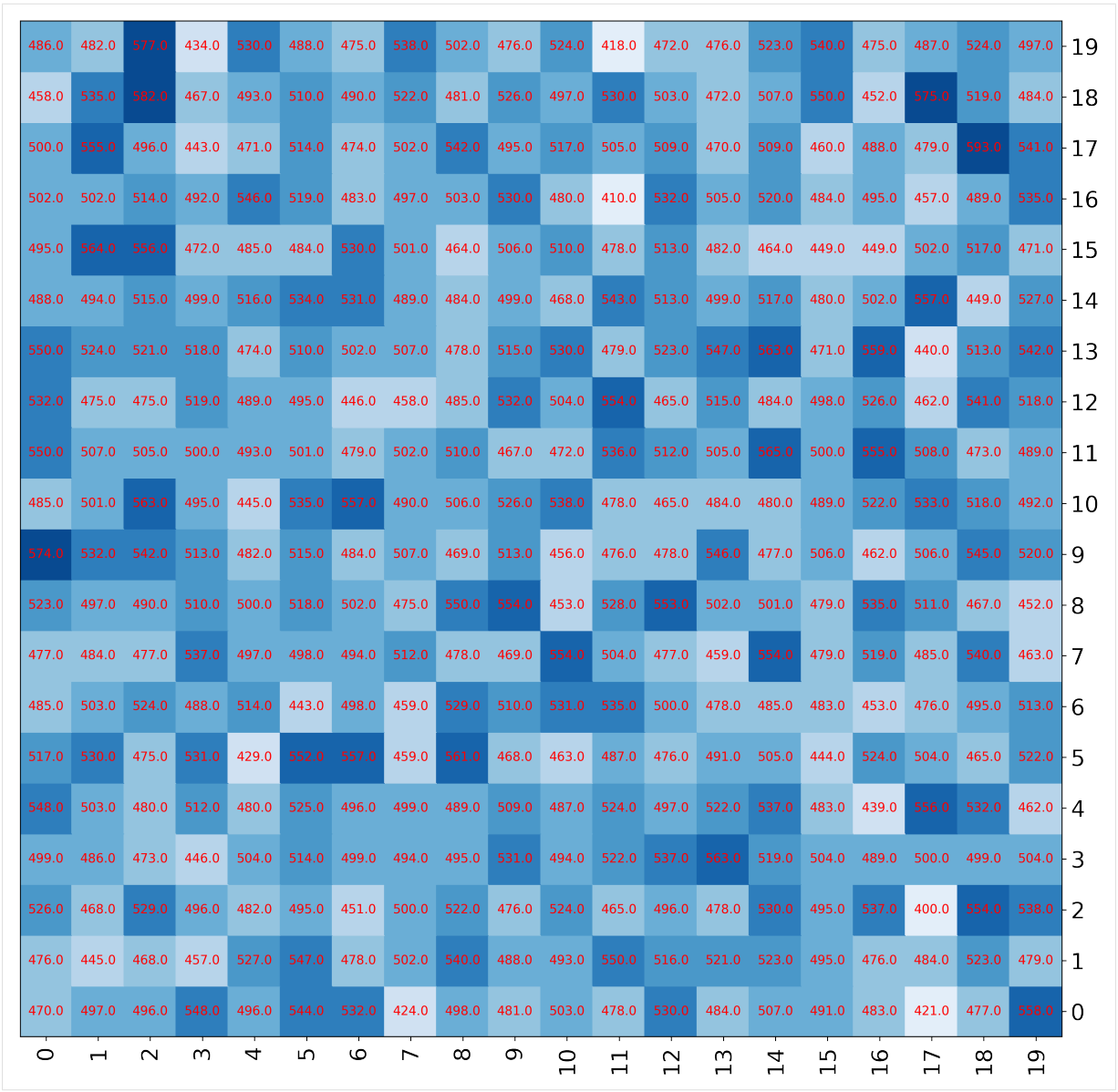
[7]:
```
fig, ax = mgkit.plots.get_single_figure(figsize=(10,10))
mgkit.plots.heatmap.baseheatmap(
    data.iloc[:10, :10],
    ax,
    cmap=cmap,
    annot=True,
    annotopts=dict(format=lambda x: "{:.1f}".format(x))
)
```

[7]:
```
<matplotlib.collections.QuadMesh at 0x7f83af5636d0>
```



[8]:
```
fig, ax = mgkit.plots.get_single_figure(figsize=(15,15))
mgkit.plots.heatmap.baseheatmap(
    data.iloc[:20, :20],
    ax,
    cmap=cmap,
    annot=True,
    annotopts=dict(
        format=lambda x: "%.1f" % x,
        fontsize=10,
        color='r'
    )
)
```

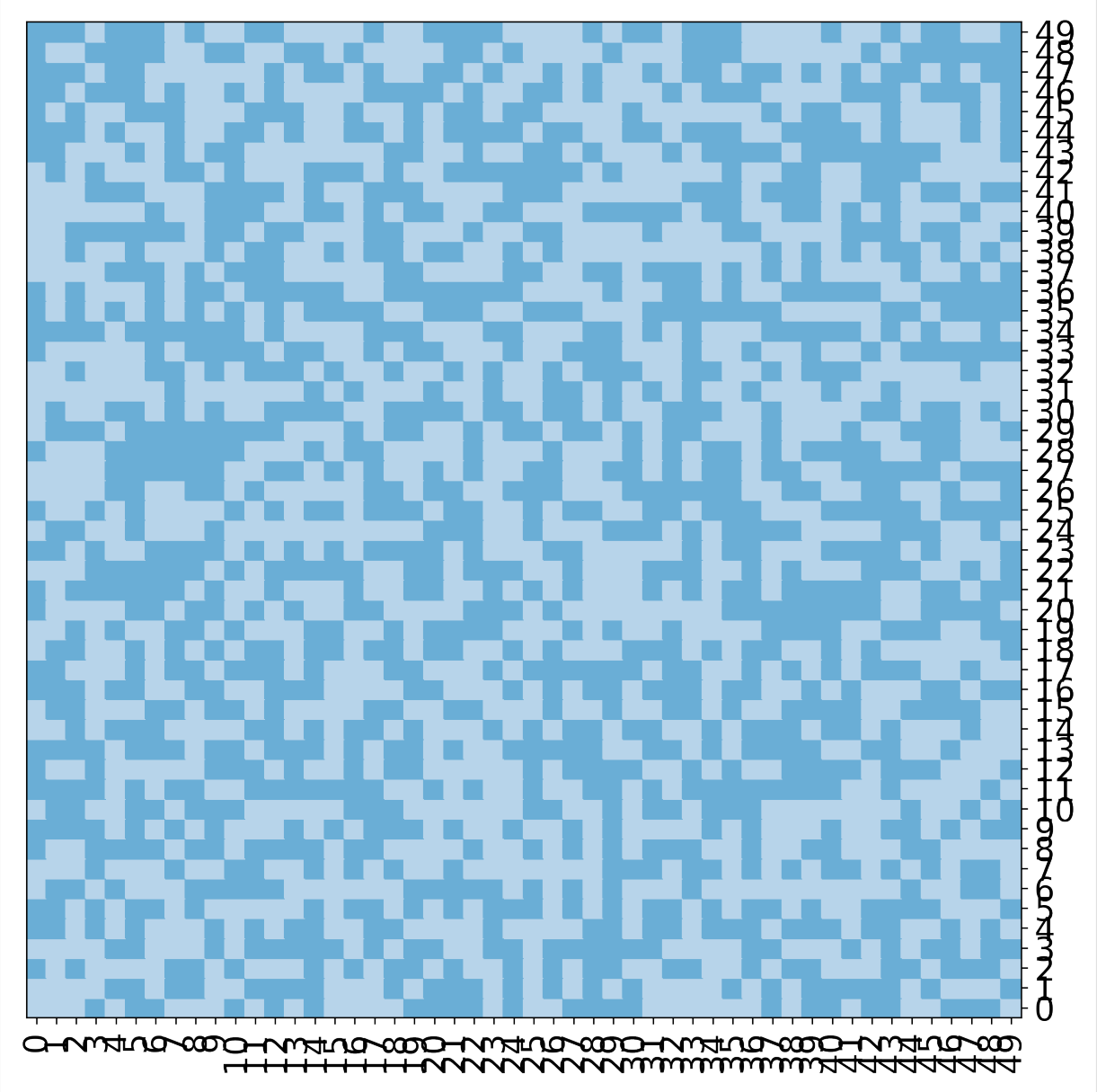
[8]:
```
<matplotlib.collections.QuadMesh at 0x7f83ad6f5310>
```



5.3.4 Using Boundaries for the colors

[9]:
```
norm = matplotlib.colors.BoundaryNorm([0, 300, 500, 700, 900, 1000], cmap.N)

fig, ax = mgkit.plots.get_single_figure(figsize=(10,10), aspect='equal')
mgkit.plots.heatmap.baseheatmap(data, ax, cmap=cmap, norm=norm)
```

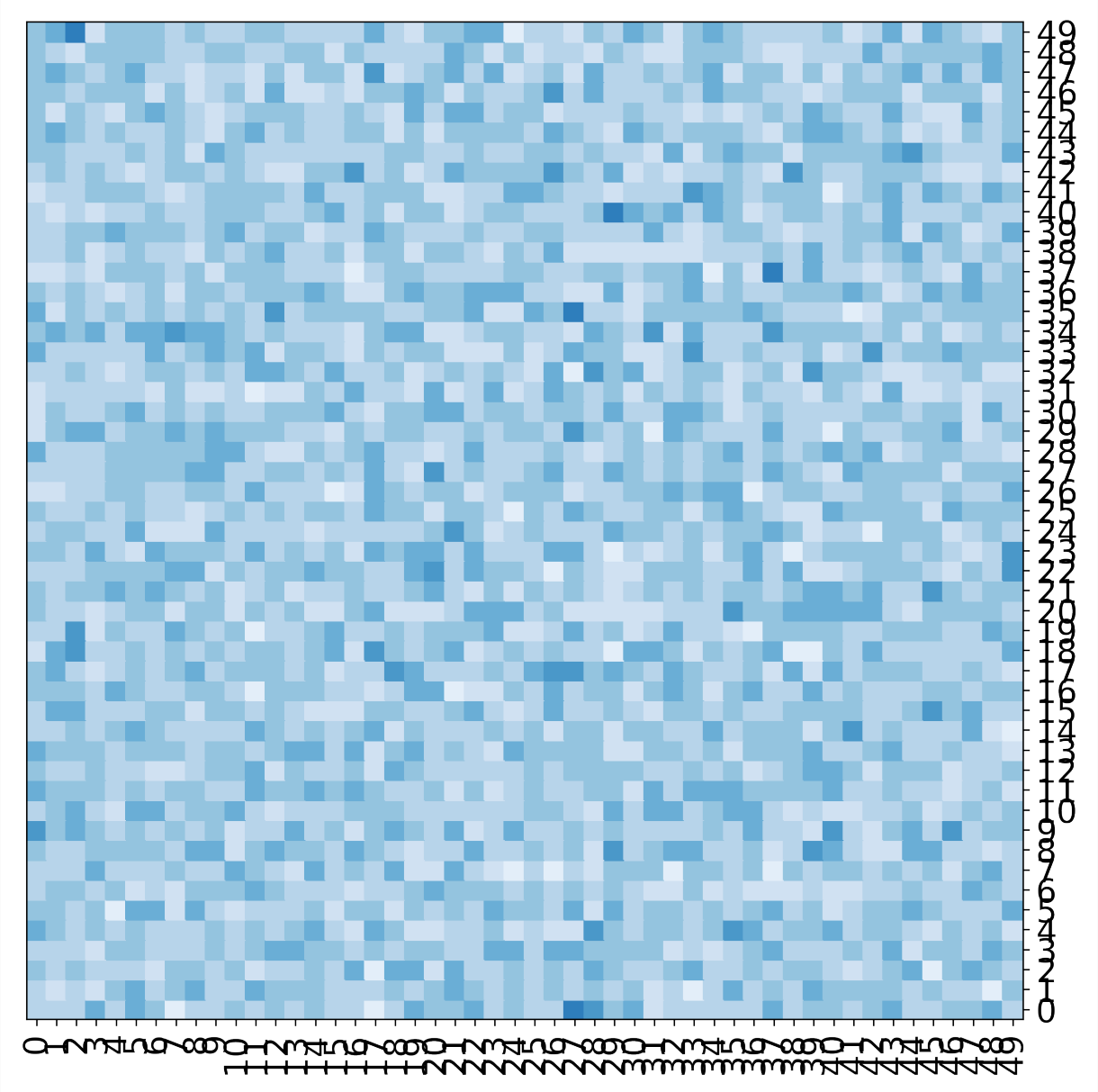
[9]:
```
<matplotlib.collections.QuadMesh at 0x7f83ad3a6a10>
```



5.3.5 Normalising the colors

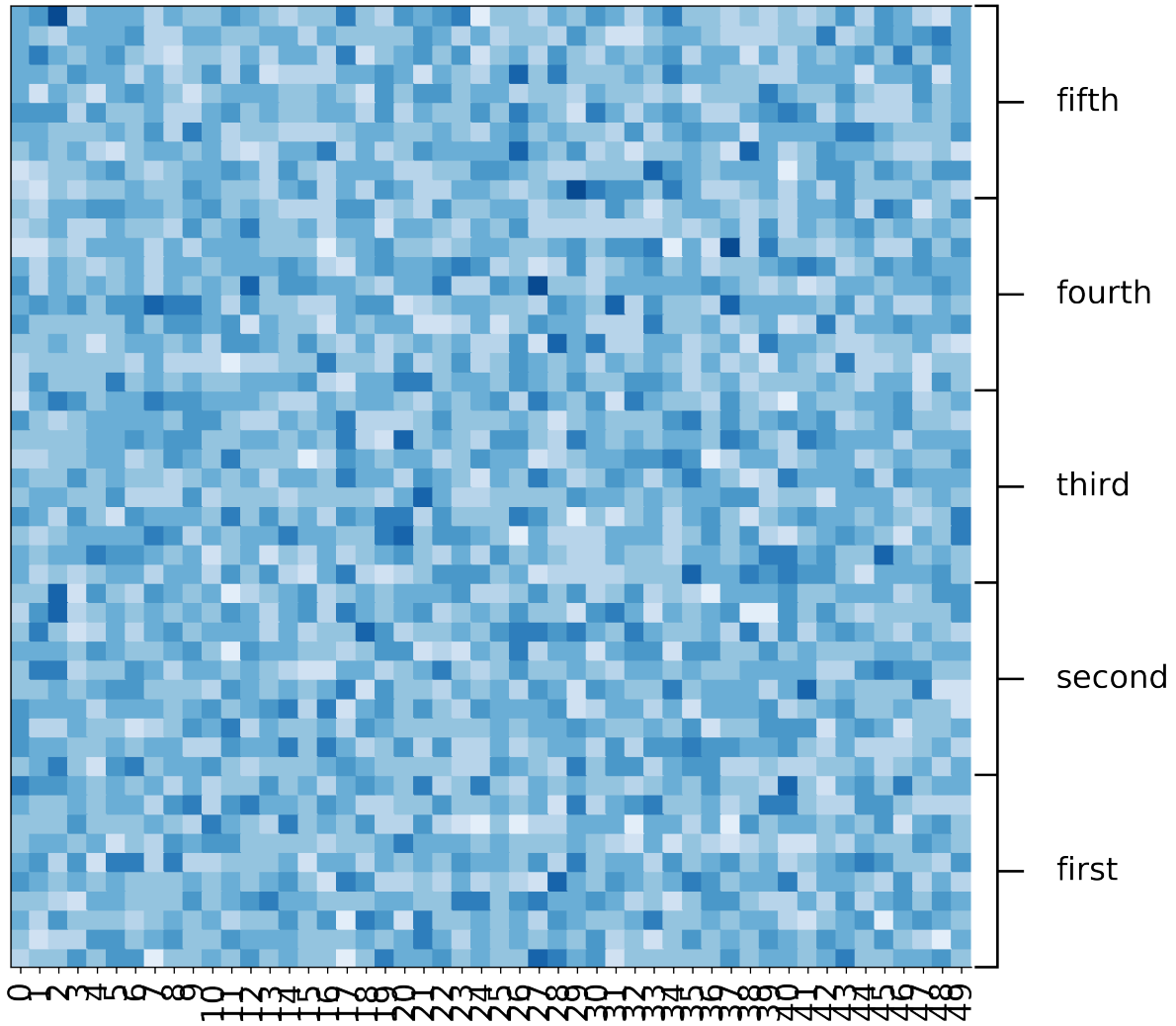
[10]:
```
norm = matplotlib.colors.Normalize(vmin=400, vmax=700, clip=True)

fig, ax = mgkit.plots.get_single_figure(figsize=(10,10), aspect='equal')
mgkit.plots.heatmap.baseheatmap(data, ax, cmap=cmap, norm=norm)
```

[10]:
```
<matplotlib.collections.QuadMesh at 0x7f83ad1fe710>
```



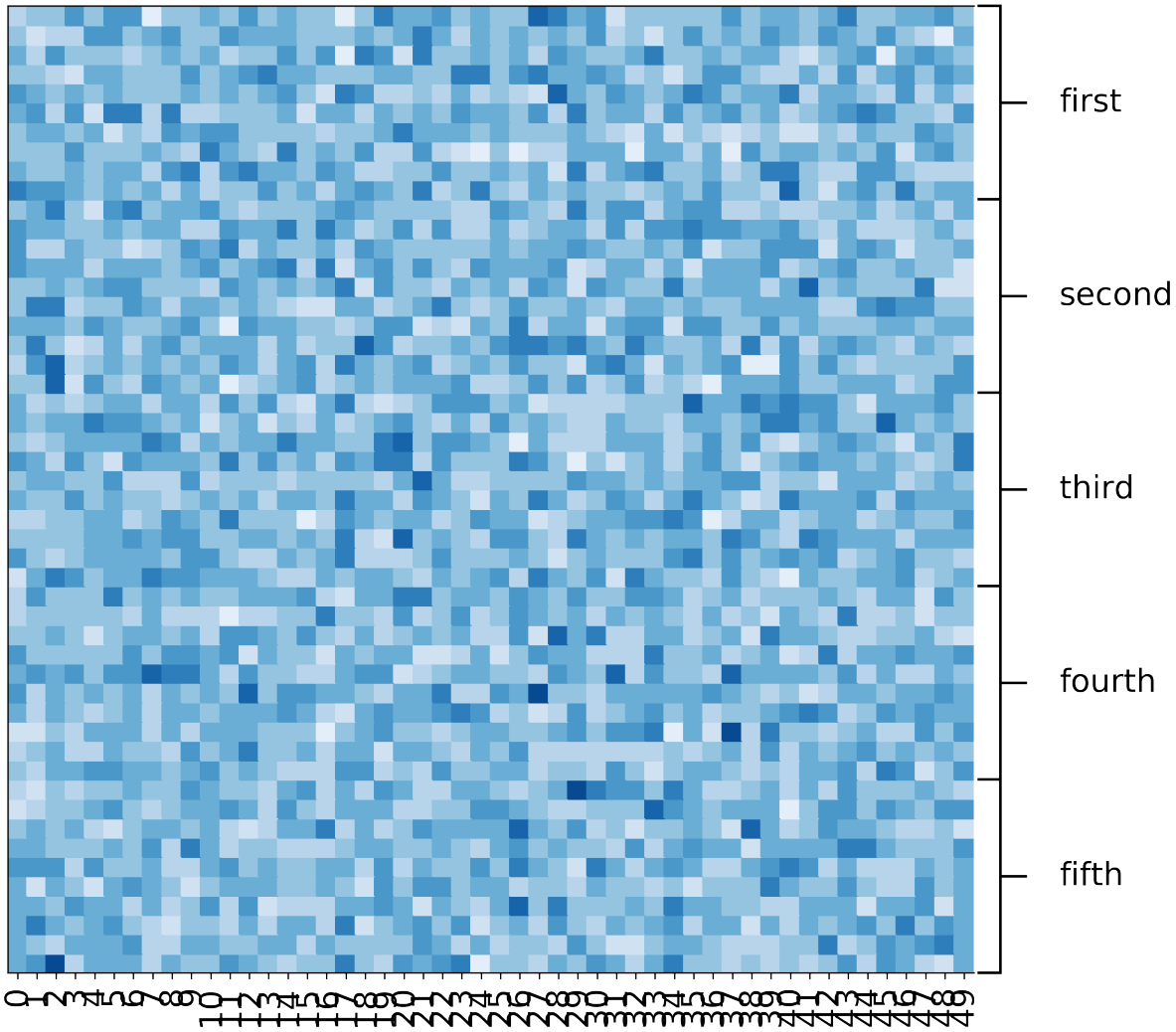
Grouping labels

[11]:
```
fig, ax = mgkit.plots.get_single_figure(figsize=(10,10), aspect='equal')
mgkit.plots.heatmap.baseheatmap(data, ax, cmap=cmap)
mgkit.plots.grouped_spine(
    [range(10), range(10, 20), range(20, 30), range(30, 40), range(40, 50)],
    ['first', 'second', 'third', 'fourth', 'fifth'],
    ax
)
```



Reversing the order of the rows

[12]:
```
fig, ax = mgkit.plots.get_single_figure(figsize=(10,10), aspect='equal')
mgkit.plots.heatmap.baseheatmap(data.loc[data.index[::-1]], ax, cmap=cmap)
mgkit.plots.grouped_spine(
    [range(10), range(10, 20), range(20, 30), range(30, 40), range(40, 50)][::-1],
    ['first', 'second', 'third', 'fourth', 'fifth'][::-1],
    ax
)
```

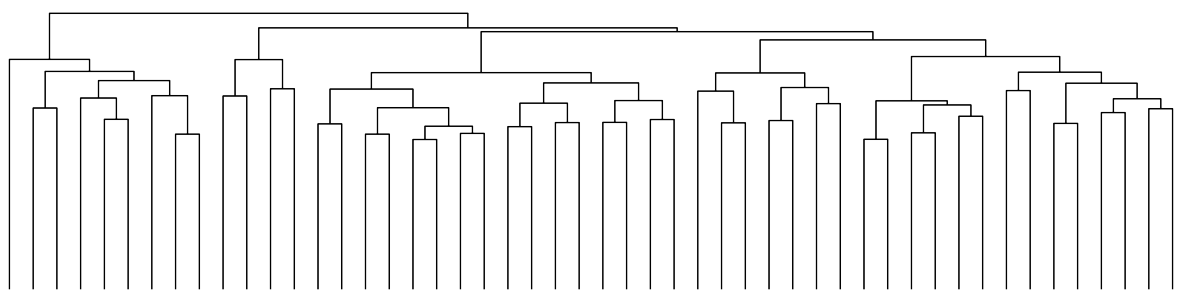


5.3.6 A dendrogram from clustering the data

Clustering rows

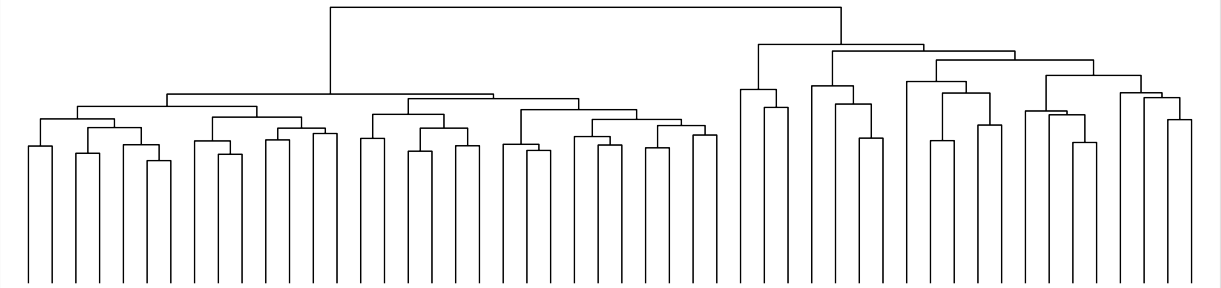
[13]:
```
fig, ax = mgkit.plots.get_single_figure(figsize=(20, 5))
_ = mgkit.plots.heatmap.dendrogram(data, ax)
```

```
/mnt/c/Users/frubino/Documents/repositories/mgkit/mgkit/plots/heatmap.py:241:␣
↪ClusterWarning: scipy.cluster: The symmetric non-negative hollow observation␣
↪matrix looks suspiciously like an uncondensed distance matrix
  clusters = hclust.linkage(pairwise_dists, method=method)
```



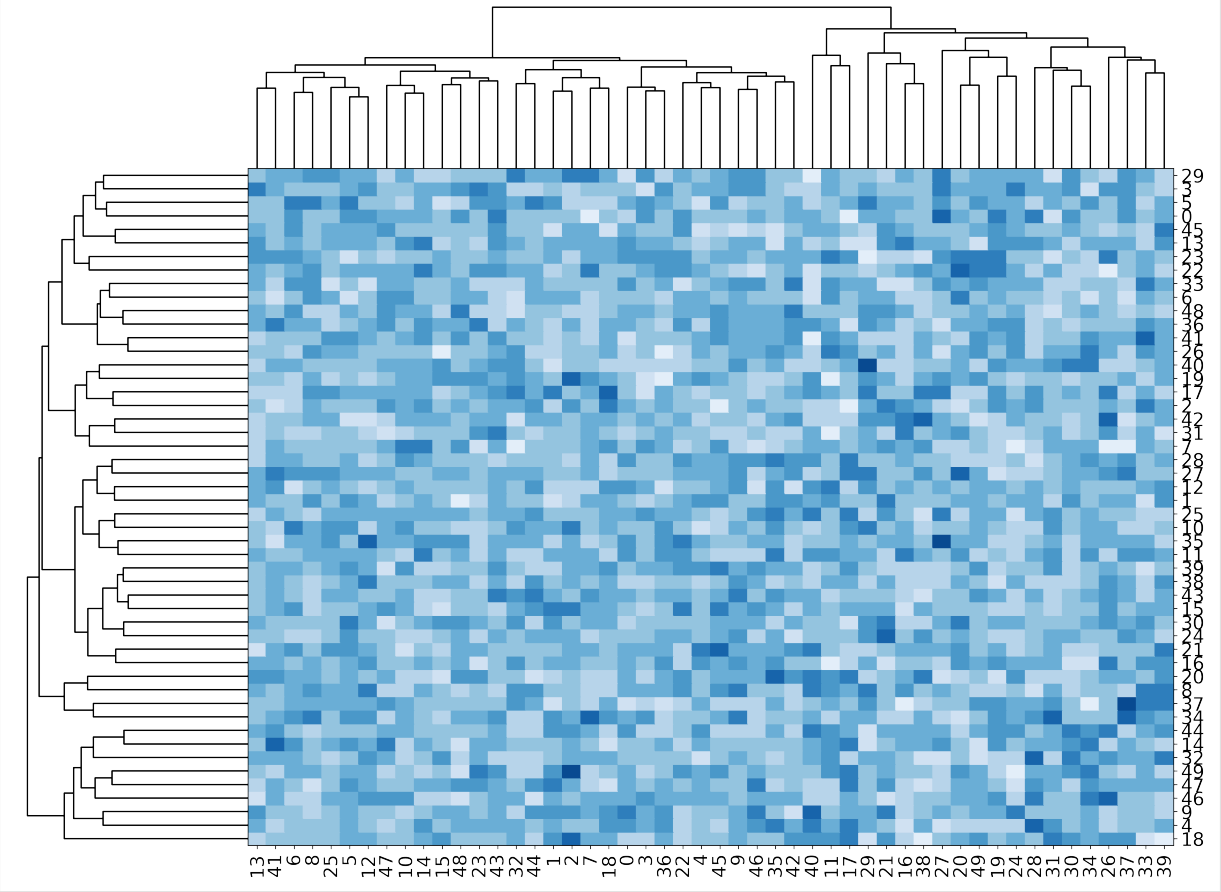
Clustering colums (You need the transposed matrix)

[14]:
```
fig, ax = mgkit.plots.get_single_figure(figsize=(20, 5))
_ = mgkit.plots.heatmap.dendrogram(data.T, ax)
```



5.3.7 A simple clustered heatmap, look at the code for customisation

[15]:
```
mgkit.plots.heatmap.heatmap_clustered(data, figsize=(20, 15), cmap=cmap)
```



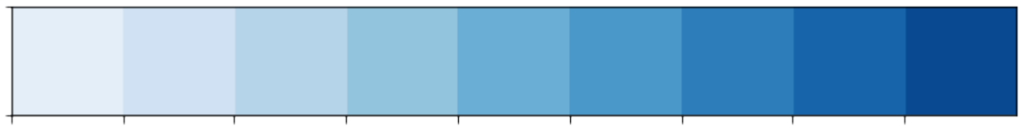
5.4 Misc. Plots Tips

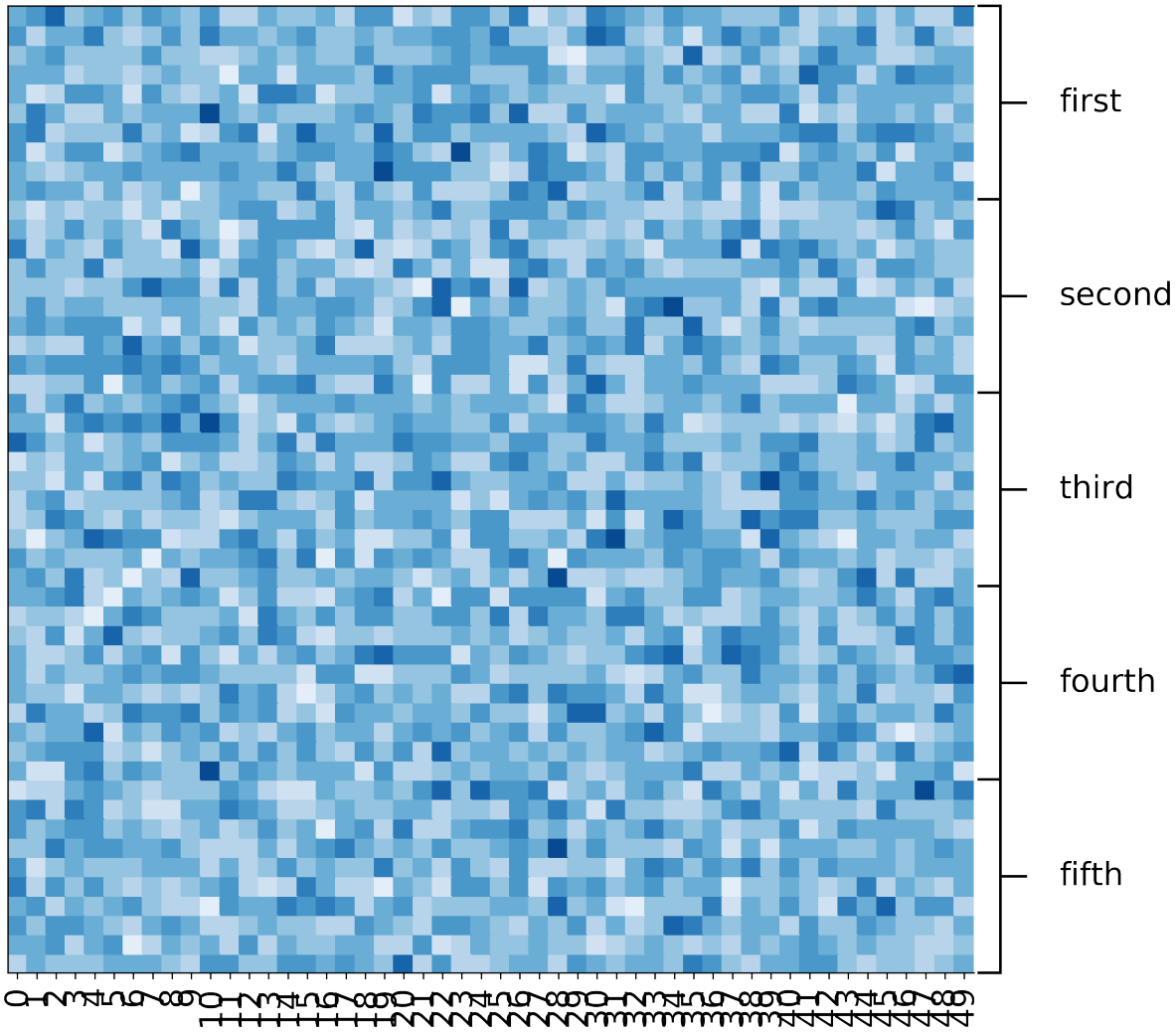
A few tips that can be useful when making plots with matplotlib

5.4.1 Trim Figure

Sometimes the plot is way smaller than the chosen figure size. When saving a figure to disk, using the *figure.savefig*, a good idea is to use *bbox_inches='tight'* to start with and adding *pad_inches=0* to remove the rest of the space.

[2]:
```
import mgkit.plots
import numpy
import pandas
import seaborn as sns
import matplotlib.colors
nrow = 50
ncol = nrow

data = pandas.DataFrame(
{
    x: numpy.random.negative_binomial(500, 0.5, nrow)
    for x in xrange(ncol)
}
)
sns.palplot(sns.color_palette('Blues', 9))
cmap = matplotlib.colors.ListedColormap(sns.color_palette('Blues', 9))

fig, ax = mgkit.plots.get_single_figure(figsize=(20,10), aspect='equal')
mgkit.plots.heatmap.baseheatmap(data.loc[data.index[::-1]], ax, cmap=cmap)
mgkit.plots.grouped_spine(
    [range(10), range(10, 20), range(20, 30), range(30, 40), range(40, 50)][::-1],
    ['first', 'second', 'third', 'fourth', 'fifth'][::-1],
    ax
)
# This will save the figure "as is"
fig.savefig('test-trim.pdf')
# This will save the figure removing most of the unused space
fig.savefig('test-trim-tight.pdf', bbox_inches='tight')
```





5.5 Examples of the *mgkit.db* package

5.5.1 Imports

[84]:
```
from mgkit.io import gff
import mgkit.net
from mgkit.db import dbm
from mgkit.db import mongo
import mgkit.taxon
import gzip
```

5.5.2 Download Example GFF

[66]:
```
# This will just load the data from the repository and save it in the same␣
↪directory as
# this notebook
data = mgkit.net.url_read('https://bitbucket.org/setsuna80/mgkit/downloads/
↪assembly.gff.gz')
# The data is compressed
open('assembly.gff.gz', 'w').write(data)
```

5.5.3 GFF Annotations

The are a few ways to load the GFF, but the result of *parse_gff* is a generator that yields one annotation at a time when it's iterate over. One way to keep the annotations in memory is building a dictionary, with the unique identifier for each annotation (called **uid**) used as key, while the annotation object is the value.

[4]:
```
# *mgkit.io.gff.parse_gff* can read compressed data, gzip, bz2 (also lzma on␣
↪python3)
annotations = {
    annotation.uid: annotation
    for annotation in gff.parse_gff('assembly.gff.gz')
}
```

Each annotation is parsed and an instance of the **mgkit.io.gff.Annotation** is created. The class contains several properties, like the unique identifier (uid), the gene identifier (gene_id) and the taxon identifier (taxon_id)

[30]:
```
annotation = annotations['d002b31c-1d78-438c-b8f9-aba791807724']
print annotation

print annotation.uid, annotation.gene_id, annotation.taxon_id
```

```
NODE_57290(-):1-87
d002b31c-1d78-438c-b8f9-aba791807724 Q72QU2 2
```

Other properties and methods can be accessed, like the *Annotation.get_mappings* to get a dictionary of all mappings, or using the *len* function on the instance to get it's length (or using the property *length*).

[11]:
```
print len(annotation), annotation.length
print annotation.get_mappings()
```

```
87 87
{'ko': ['K03695']}
```

Taxonomy and Annotations

When using metagenomics, one of the problem is associated functionality to taxonomy. MGKit contains a class that can read the taxonomy from Uniprot, which is compatible with NCBI taxonomy. The **mgkit.taxon** contains the **UniprotTaxonomy** that is use to store and in part search the taxonomy. The module contains many more functions to resolve different levels of the taxonomy. A few examples applied to the annotations loaded follow.

[90]:
```
# This will just load the data from the repository and save it in the same␣
↪directory as
# this notebook
data = mgkit.net.url_read(
    "https://bitbucket.org/setsuna80/mgkit/downloads/taxonomy.pickle.gz"
)
open('taxonomy.pickle.gz', 'w').write(data)
del data
```

[91]:
```
# Using compress taxonomy files makes it slower to load
taxonomy = mgkit.taxon.UniprotTaxonomy('taxonomy.pickle.gz')
```

[103]:
```
# to find the Bacteoidales taxon identifier
taxonomy.find_by_name('bacteroidales')
```

[103]:
```
[171549]
```

[121]:
```
# to find all the annotations that belong to the Order Bacteroidales
count = 0
for annotation in annotations.itervalues():
```

(continues on next page)

(continued from previous page)

```
    if mgkit.taxon.is_ancestor(taxonomy, annotation.taxon_id, 171549):
        count += 1
        print annotation.uid, annotation.gene_id
print "Number of annotation:", count
```

```
7233587c-b80d-4908-8ead-92734deeec81 Q7MV19
5322b316-46e5-44cf-9eb1-ef94355c7855 Q01VN6
a7308f6f-7b17-4b00-8afa-92ebecef3dd3 Q8XP14
195118b7-1236-48ad-8812-e0ec3100e7d9 Q7MV19
14b3cc41-050a-4949-b085-75db0cda12ec Q8A294
d1dad026-09ac-48e4-95fe-158e39d96a0d P49008
01b819f8-1444-4f25-a3fa-93e160fa58c2 Q7MVL1
4b3ce614-cc8a-47ea-a046-f9ca7c7ab16c Q5LI72
65bae5c6-0d23-4a08-ae3f-aec2763f4621 Q7MV19
3aef43ea-4e94-4940-bf85-743950e5ad8a Q9AGG3
16794c3c-97b8-4453-8d14-a5e37c8969b4 A6LB11
bd92adff-b8d9-411f-9488-7604eb580fd6 Q89YZ6
3441f906-f63d-45fe-a4e5-e639439d19db A6LD25
cd08ae89-1f1e-4875-851e-c0c55de8c764 A6LA51
b3bf4054-4f31-4a8a-bf19-fd0e65c56867 A6LI30
44bdfb77-1606-4194-b410-9a22c75b3b5b Q7MV19
3b5e126e-25ec-460d-a439-2520bebe0a3d A6KZH6
e908c5b1-9dec-4406-b952-009aab3fd778 A6LDS1
b425ef29-0bc0-4de7-ad96-abe5c7b75f96 Q8A9M7
2a3558c3-6f7b-49a4-a8d3-c2b0cef287d6 Q7MXZ1
376d70e0-6591-4b2b-9a06-1d9fb7fdbc66 Q7M9Y2
ff7fd5ef-9be2-404c-8137-89f368071a4e Q8A294
2665ff2c-4e9a-4c7a-9604-8433fa2ae202 A6LHY5
dd9a44d5-ed1e-4350-b05c-f0cfd510e669 A6L170
255e75a1-a59c-43fd-9396-17a3566b3063 Q8A0F5
49474358-7962-4b0c-b52a-5de935f17bfc A6LFA6
27eb1efe-ff07-401c-93db-958a38e866bc Q7MWM7
746805f5-0fdc-4499-953f-7be496b9c784 Q7MU65
e3be2158-c013-4e58-a073-ab8e3c893094 Q8A8Y4
e028b0e9-802f-4f1b-b055-f5ecca786170 Q8A1D3
f2919fc6-d8e2-4fe7-ac9f-152c46d0ebbb Q7MV19
b65468b2-d4e7-456b-871d-9cd96fa4dd48 Q02XT4
e1643d1d-12c3-4397-a6a7-d2a24f203c4a Q8A294
0d9cd52c-5969-49f7-866c-e8c5c9783b79 Q8A294
cdd362ba-448f-475f-a638-d6473b471572 A6LD68
7af092eb-20c5-46b4-b8bb-e9b0c99a8ce5 Q5LGH0
2d9172a4-fe51-4baa-a8fb-66f020ba6452 Q7MVL1
Number of annotation: 37
```

[109]:
```
# to find out the Phyla represented in the annotations
print set(
    taxonomy.get_ranked_taxon(annotation.taxon_id, rank='phylum').s_name
    for annotation in annotations.itervalues()
)
```

```
set(['arthropoda', 'microsporidia', 'korarchaeota', 'viruses', 'nematoda',␣
↪'bacteroidetes', 'nanoarchaeota', 'tenericutes', 'thermotogae', 'chlorophyta',␣
↪'cellular organisms', 'fibrobacteres', 'bacteria', 'euryarchaeota',␣
↪'verrucomicrobia', 'annelida', 'eukaryota', 'aquificae', 'ascomycota',␣
↪'actinobacteria', 'chlorobi', 'deferribacteres', 'archaea', 'bacillariophyta',␣
↪'streptophyta', 'chlamydiae', 'apicomplexa', 'dictyoglomi', 'cloacimonetes',␣
↪'gemmatimonadetes', 'thaumarchaeota', 'proteobacteria', 'acidobacteria',␣
↪'spirochaetes', 'cyanobacteria', 'firmicutes', 'chloroflexi', 'planctomycetes',␣
↪'chordata', 'euglenida', 'elusimicrobia', 'basidiomycota', 'xanthophyceae',␣
↪'nitrospirae', 'fusobacteria', 'deinococcus-thermus', 'platyhelminthes',␣
↪'crenarchaeota'])
```

[116]:
```
# to get the lineage of the first annotations
annotation = annotations['b97ead95-81a7-4caf-8d25-349ee6e276c1']
print taxonomy[annotation.taxon_id].s_name, mgkit.taxon.get_lineage(taxonomy,␣
↪annotation.taxon_id)
```

```
escherichia coli (strain k12) [131567, 2, 1224, 1236, 91347, 543, 561, 562]
```

[115]:
```
# to get the names, quickly
annotation = annotations['b97ead95-81a7-4caf-8d25-349ee6e276c1']
print taxonomy[annotation.taxon_id].s_name, mgkit.taxon.get_lineage(taxonomy,␣
↪annotation.taxon_id, names=True)
```

```
escherichia coli (strain k12) ['cellular organisms', 'bacteria', 'proteobacteria',␣
↪'gammaproteobacteria', 'enterobacteriales', 'enterobacteriaceae', 'escherichia',␣
↪'escherichia coli']
```

Issues

Keeping the annotations in memory can lead to a high memory usage, as well as a long time traversing all of them to specifically filter them. MGKit uses two solutions to interface with DBs, one is using a *dbm-like* database, *semidbm* and the other is using *MongoDB*.

5.5.4 semidbm

Packages to use *dbm* database are included with Python, but they depend on the type of OS python is installed onto. A pure Python implementation of a dbm is semidbm[60]. As other *dbm*, it works in a similar way as a dictionary, while keeping the memory usage low. To create a *semidbm* DB from annotations, the **get-gff-info** can be used, using the **dbm** command:

[21]:
```
!get-gff-info dbm -d assembly-db assembly.gff.gz
```

```
assembly-db
INFO - mgkit.db.dbm: DB "assembly-db" opened/created
INFO - mgkit.io.gff: Loading GFF from file (assembly.gff.gz)
```

or interactively, using *mgkit.db.dbm.create_gff_dbm*:

[27]:
```
db = dbm.create_gff_dbm(annotations.itervalues(), 'assembly-db')
```

```
assembly-db
```

Which also return an instance of db. *semidbm* allows the use of only strings as keys and strings as values, so for the same annotation as before, you see what MGKit stores in it, the actual GFF line:

[28]:
```
db['d002b31c-1d78-438c-b8f9-aba791807724']
```

[28]:
```
'NODE_57290\tBLAST\tCDS\t1\t87\t51.6\t-\t0\tSRR001322_cov="0";SRR001323_cov="0";
↪SRR001325_cov="3";SRR001326_cov="0";bitscore="51.6";cov="3";db="UNIPROT-SP";dbq=
↪"10";exp_nonsyn="200";exp_syn="61";gene_id="Q72QU2";identity="75.9";map_KO=
↪"K03695";taxon_db="NCBI-NT";taxon_id="2";uid="d002b31c-1d78-438c-b8f9-
↪aba791807724"\n'
```

The GFF line must then be converted back into an **Annotation** instance. To automate the process, the **mgkit.db.dbm.GFFDB** class wraps the *semidbm*. The same example as the one above:

[56]:
```
db = dbm.GFFDB('assembly-db')
db['d002b31c-1d78-438c-b8f9-aba791807724']
```

[60] https://github.com/jamesls/semidbm

[56]:
```
NODE_57290(-):1-87
```

It can also be iterated over as a dictionary (for compatibility, both *iteritems* and *items* return an iterator)

[52]:
```
for uid in db.db:
    print uid, db[uid]
    break
```

```
50dccb4d-3a49-41ed-bf8c-a1906172d8a5 NODE_49806(+):3-116
```

[55]:
```
for uid, annotation in db.iteritems():
    print uid, annotation
    break
```

```
50dccb4d-3a49-41ed-bf8c-a1906172d8a5 NODE_49806(+):3-116
```

Using this class, it is possible to use a DB as a drop-in replacement for a dictionary in a script that used annotations stored in memory in MGKit. The *examples using the taxonomy* will works in the same way, for example.

5.5.5 Using MongoDB

MongoDB[61] is Document based DB that is not based on SQL. One of the advantage of it the absence of a schema, which makes it easy to insert annotations into it. Moreover, the data in a MongoDB is easily accessible from a variety of programming languages, as well as its own shell. Another advantage is the possiblity to query the annotations and index specific values to speed up them.

In the same way as with *dbm*, the **get-gff-info** can help produce a file that can be directly loaded into a *mongod* instance.

The following example uses **pymongo** (the official client library for Python) and requires a **mongod** instance running on the same machine. The annotations will be imported into the **test** database, into the **gff** collection.

[69]:
```
!gunzip -c assembly.gff.gz | get-gff-info mongodb | mongoimport --db test --
↪collection gff --drop
```

```
2015-12-04T15:38:41.355+1000    connected to: localhost
2015-12-04T15:38:41.355+1000    dropping: test.gff
INFO - mgkit.io.gff: Loading GFF from file (<stdin>)
2015-12-04T15:38:43.830+1000    imported 9135 documents
```

You can use the **pymongo** module directly or just use the **mgkit.db.mongo.GFFDB** class to automate connection and conversion of the **JSON** documents back into **Annotation** objects.

[72]:
```
db = mongo.GFFDB('test', 'gff')
```

[74]:
```
for annotation in db.find_annotation():
    print annotation.uid, annotation.gene_id
    break
```

```
303fbf1f-8140-4f9e-9c44-ae089e67bdc3 O93746
```

The DB can be queried by passing the **GFF.find_annotation** method the same query that are explained in Pymongo documentation[62].

[76]:
```
# To look for all annotations that have the KO mapping to K01883
for annotation in db.find_annotation({'map.ko': 'K01883'}):
    print annotation
```

[61] https://www.mongodb.org/
[62] https://docs.mongodb.org/getting-started/python/client/

```
NODE_22940(-):2-97
NODE_8691(+):2-88
NODE_8691(+):5-91
NODE_30222(+):11-97
NODE_30222(+):2-82
NODE_30222(+):8-94
NODE_30222(+):5-91
NODE_36783(+):11-115
NODE_2009(-):3-104
NODE_2009(-):12-110
NODE_19876(+):3-113
NODE_35927(-):2-76
NODE_35927(-):8-163
NODE_31317(+):2-73
NODE_31317(+):5-88
NODE_29415(+):29-100
NODE_45868(-):1-96
NODE_1013(-):33-128
NODE_39238(-):1-90
NODE_39238(-):4-93
NODE_6581(-):3-116
NODE_40758(-):2-163
NODE_7805(-):1-117
NODE_28135(+):3-116
NODE_8575(+):34-123
NODE_8575(+):28-114
NODE_6979(+):1-99
NODE_35052(-):2-106
NODE_13245(-):2-94
NODE_13245(-):5-97
NODE_30508(+):1-99
NODE_19190(+):18-227
NODE_19190(+):3-113
NODE_16671(+):2-106
```

[79]:
```
# To look for all annotations that have the KO mapping to K01883 *AND*
# the taxonomy was inferred from a blast to NCBI (see refinement of
# taxonomy in theTutorial - Gene Prediction)
for annotation in db.find_annotation({'map.ko': 'K01883', 'taxon_db': 'NCBI-NT'}):
    print annotation
```

```
NODE_22940(-):2-97
NODE_30222(+):11-97
NODE_30222(+):2-82
NODE_30222(+):8-94
NODE_30222(+):5-91
NODE_40758(-):2-163
```

[117]:
```
# Finding all annotation from a specific taxon
for annotation in db.find_annotation({'taxon_id': 224911}):
    print annotation
```

```
NODE_36848(-):2-94
NODE_58432(+):8-124
NODE_48731(+):5-118
NODE_13988(+):20-190
NODE_10564(-):3-101
NODE_61599(+):8-106
NODE_58191(+):1-99
NODE_36561(+):5-115
NODE_33951(-):13-99
NODE_20537(-):6-101
```

(continues on next page)

(continued from previous page)

```
NODE_72294(-):3-95
```

Using Taxonomy

The usual approach about the taxonomy is to traverse all the annotations (those returned by one of the previous queries, even) and use the functionality in the **mgkit.taxon** module. It is possible to repeat the example that search all annotations that belong to Order *Bacteroidales*, but the records must be loaded with the lineage into the DB. This can be done having a taxonomy file, *taxonomy.pickle.gz* in our case, with the following command:

[118]:
```
!gunzip -c assembly.gff.gz | get-gff-info mongodb -t taxonomy.pickle.gz |␣
↪mongoimport --db test --collection gff --drop
```

```
2015-12-04T16:32:13.785+1000    connected to: localhost
2015-12-04T16:32:13.786+1000    dropping: test.gff
2015-12-04T16:32:16.783+1000    test.gff        0.0 B
2015-12-04T16:32:19.783+1000    test.gff        0.0 B
INFO - mgkit.taxon: Loading taxonomy from file taxonomy.pickle.gz
2015-12-04T16:32:22.785+1000    test.gff        0.0 B
2015-12-04T16:32:25.782+1000    test.gff        0.0 B
2015-12-04T16:32:28.784+1000    test.gff        0.0 B
2015-12-04T16:32:31.780+1000    test.gff        0.0 B
2015-12-04T16:32:34.783+1000    test.gff        0.0 B
2015-12-04T16:32:37.780+1000    test.gff        0.0 B
2015-12-04T16:32:40.782+1000    test.gff        0.0 B
2015-12-04T16:32:43.782+1000    test.gff        0.0 B
2015-12-04T16:32:46.785+1000    test.gff        0.0 B
2015-12-04T16:32:49.785+1000    test.gff        0.0 B
2015-12-04T16:32:52.783+1000    test.gff        0.0 B
2015-12-04T16:32:55.783+1000    test.gff        0.0 B
2015-12-04T16:32:58.781+1000    test.gff        0.0 B
2015-12-04T16:33:01.780+1000    test.gff        0.0 B
2015-12-04T16:33:04.783+1000    test.gff        0.0 B
2015-12-04T16:33:07.781+1000    test.gff        0.0 B
2015-12-04T16:33:10.781+1000    test.gff        0.0 B
2015-12-04T16:33:13.781+1000    test.gff        0.0 B
2015-12-04T16:33:16.781+1000    test.gff        0.0 B
2015-12-04T16:33:19.783+1000    test.gff        0.0 B
2015-12-04T16:33:22.781+1000    test.gff        0.0 B
2015-12-04T16:33:25.782+1000    test.gff        0.0 B
2015-12-04T16:33:28.781+1000    test.gff        0.0 B
2015-12-04T16:33:31.783+1000    test.gff        0.0 B
2015-12-04T16:33:34.785+1000    test.gff        0.0 B
2015-12-04T16:33:37.781+1000    test.gff        0.0 B
2015-12-04T16:33:40.780+1000    test.gff        0.0 B
2015-12-04T16:33:43.782+1000    test.gff        0.0 B
2015-12-04T16:33:46.780+1000    test.gff        0.0 B
2015-12-04T16:33:49.780+1000    test.gff        0.0 B
2015-12-04T16:33:52.781+1000    test.gff        0.0 B
2015-12-04T16:33:55.782+1000    test.gff        0.0 B
2015-12-04T16:33:58.785+1000    test.gff        0.0 B
2015-12-04T16:34:01.784+1000    test.gff        0.0 B
2015-12-04T16:34:04.781+1000    test.gff        0.0 B
2015-12-04T16:34:07.782+1000    test.gff        0.0 B
2015-12-04T16:34:10.785+1000    test.gff        0.0 B
2015-12-04T16:34:13.781+1000    test.gff        0.0 B
2015-12-04T16:34:16.784+1000    test.gff        0.0 B
2015-12-04T16:34:19.783+1000    test.gff        0.0 B
2015-12-04T16:34:22.780+1000    test.gff        0.0 B
2015-12-04T16:34:25.785+1000    test.gff        0.0 B
2015-12-04T16:34:28.781+1000    test.gff        0.0 B
```

(continues on next page)

(continued from previous page)

```
2015-12-04T16:34:31.780+1000    test.gff        0.0 B
2015-12-04T16:34:34.783+1000    test.gff        0.0 B
2015-12-04T16:34:37.780+1000    test.gff        0.0 B
2015-12-04T16:34:40.782+1000    test.gff        0.0 B
2015-12-04T16:34:43.781+1000    test.gff        0.0 B
2015-12-04T16:34:46.785+1000    test.gff        0.0 B
INFO - mgkit.workflow.extract_gff_info: Using cached calls to lineage
INFO - mgkit.io.gff: Loading GFF from file (<stdin>)
2015-12-04T16:34:49.783+1000    test.gff        2.7 MB
2015-12-04T16:34:51.874+1000    imported 9135 documents
```

The script will first load the taxonomy and add to each record in the database the **lineage** key. This contains an array of integers, that are the output of the **mgkit.taxon.lineage** function and can be searched using:

[123]:
```
count = 0
for annotation in db.find_annotation({'lineage': 171549}):
    count += 1
    print annotation
print "Number of annotation:", count
```

```
NODE_33533(-):2-64
NODE_18827(+):2-127
NODE_25363(+):3-95
NODE_69486(+):1-111
NODE_13380(-):3-95
NODE_8404(+):3-176
NODE_71367(+):2-106
NODE_50779(-):1-102
NODE_20694(+):129-221
NODE_38976(+):4-102
NODE_69904(+):9-110
NODE_1963(-):2-94
NODE_41194(-):18-98
NODE_47622(+):1-99
NODE_56590(+):2-103
NODE_66803(+):23-169
NODE_14043(+):4-96
NODE_35099(+):18-122
NODE_48598(-):20-97
NODE_58511(+):1-96
NODE_70185(+):2-103
NODE_56348(-):4-93
NODE_56348(-):13-102
NODE_56348(-):10-99
NODE_32336(-):1-114
NODE_59685(+):3-107
NODE_57945(+):12-134
NODE_59259(-):1-108
NODE_28794(-):5-133
NODE_72312(-):1-96
NODE_37438(+):3-107
NODE_6370(+):123-224
NODE_67647(+):2-100
NODE_28480(-):1-93
NODE_72226(+):8-103
NODE_46503(+):3-104
NODE_20236(+):1-90
Number of annotation: 37
```

And as you can see, the number of annotations is the same as the *example above*. The use of MongoDB to store the annotations can make it simplier to use richer queries, even from other languages.

CHAPTER 6

MGKit GFF Specifications

The GFF produced with MGKit follows the conventions of GFF/GTF files but it provides some additional fields in the 9th column which translate to a Python dictionary when an annotation is loaded into an `Annotation` instance.

The 9th column is a list of **key=value** item, separated by a semicolon (;); each value is also expected to be quoted with double quotes and the values to not include a semicolon or other characters that can make the parsing difficult. MGKit uses `urllib.quote()` to encode those characters and also " ()/". The *`mgkit.io.gff.from_gff()`* uses `urllib.unquote()` to set the values.

Warning: As the last column translates to a dictionary in the data structures, duplicate keys are not allowed. *`mgkit.io.gff.from_gff()`* raises an exception if any are found.

6.1 Reserved Values

Any key can be added to a GFF annotation, but MGKit expects a few key to be in the GFF annotation as summarised in the following tables.

Table 1: Reserved values, used by the scripts

Key	Value	Explanation
gene_id	any string	used to identify the gene predicted
db	any string, like UNIPROT-SP, UNIPROT-TR, NCBI-NT	identifies the database used to make the gene_id prediction
taxon_db	any string, like UNIPROT-SP, UNIPROT-TR, NCBI-NT	identifies the database used to make the taxon_id prediction
dbq	integer	identifies the quality of the database, used when filtering annotations
taxon_id	integer	identifies the annotation taxon, NCBI taxonomy is used
uid	string	unique identifier for the annotation, any string is accepted but a value is assigned by using `uuid.uuid4()`[63]
cov and {any}_cov	integer	coverage for the annotation over all samples, keys ending with *_cov* indicates coverage for each sample
exp_syn, exp_nonsyn	integer	used for expected number of synonymous and non-synonymous changes for the annotation

The following keys are added by different scripts and may be used in different scripts or annotation methods.

Table 2: Interpreted Values

Key	Value	Explanation	Used
taxon_name	string	name of the taxon	not used
lineage	string	taxon lineage	not used
EC	comma separated values	list of EC numbers associated to the annotation	used by *mgkit.io.gff.Annotation.get_ec()*
map_{any}	comma separated values	list of mapping to a specific db (e.g. eggNOG -> map_EGGNOG)	used by *mgkit.io.gff.Annotation.get_mapping()*
counts_{any}	float	Stores the count data for a sample (e.g. counts_Sample1)	used by script *add-gff-info*
fpkms_{any}	float	Stores the count data for a sample (e.g. fpkms_Sample1)	used by script *add-gff-info*

[63] https://docs.python.org/3/library/uuid.html#uuid.uuid4

CHAPTER 7

Library Reference

7.1 mgkit package

7.1.1 Subpackages

mgkit.counts package

Submodules

mgkit.counts.func module

New in version 0.1.13.

Misc functions for count data

mgkit.counts.func.**batch_load_htseq_counts**(*count_files*, *samples=None*, *cut_name=None*)

Loads a list of htseq count result files and returns a DataFrame (IDxSAMPLE)

The sample names are names are the file names if *samples* and *cut_name* are *None*, supplying a list of sample names with *samples* is the preferred way, and *cut_name* is used for backward compatibility and as an option in cases a string replace is enough.

Parameters

- **count_files** (*file or str*[64]) – file handle or string with file name
- **samples** (*iterable*) – list of sample names, in the same order as *count_files*
- **cut_name** (*str*[65]) – string to delete from the the file names to get the sample names

Returns with sample names as columns and gene_ids as index

Return type pandas.DataFrame

mgkit.counts.func.**filter_counts**(*counts_iter*, *info_func*, *gfilters=None*, *tfilters=None*)

Returns counts that pass filters for each *uid* associated *gene_id* and *taxon_id*.

Parameters

[64] https://docs.python.org/3/library/stdtypes.html#str
[65] https://docs.python.org/3/library/stdtypes.html#str

- **counts_iter** (*iterable*) – iterator that yields a tuple (uid, count)
- **info_func** (*func*) – function accepting a *uid* that returns a tuple *(gene_id, taxon_id)*
- **gfilters** (*iterable*) – list of filters to apply to each *uid* associated *gene_id*
- **tfilters** (*iterable*) – list of filters to apply to each *uid* associated *taxon_id*

Yields *tuple – (uid, count)* that pass filters

mgkit.counts.func.**from_gff**(*annotations*, *samples*, *ann_func=None*, *sample_func=None*)

New in version 0.3.1.

Loads count data from a GFF file, only for the requested samples. By default the function returns a DataFrame where the index is the *uid* of each annotation and the columns the requested samples.

This can be customised by supplying *ann_func* and *sample_func*. *sample_func* is a function that accept a sample name and is expected to return a string or a tuple. This will be used to change the columns in the DataFrame. *ann_func* must accept an *mgkit.io.gff.Annotation* instance and return an iterable, with each iteration yielding either a single element or a tuple (for a MultiIndex DataFrame), each element yielded will have the count of that annotation added to.

Parameters

- **annotation** (*iterable*) – iterable yielding annotations
- **samples** (*iterable*) – list of samples to keep
- **ann_func** (*func*) – function used to customise the output
- **sample_func** (*func*) – function to customise the column elements

Returns dataframe with the count data, columns are the samples and rows the annotation counts (unless mapped with *ann_func*)

Return type DataFrame

Exmples: Assuming we have a list of *annotations* and sample SAMPLE1 and SAMPLE2 we can obtain the count table for all annotations with this

```
>>> from_gff(annotations, ['SAMPLE1', 'SAMPLE2'])
```

Assuming we want to group the samples, for example treatment1, treatment2 and control1, control2 into a MultiIndex DataFrame column

```
>>> sample_func = lambda x: ('T' if x.startswith('t') else 'C', x)
>>> from_gff(annotations, ['treatment1', 'treatment2', 'control1',
'control2'], sample_func=sample_func)
```

Annotations can be mapped to other levels for example instead of using the *uid* that is the default, it can be mapped to the gene_id, taxon_id information that is included in the annotation, resulting in a MultiIndex index for the rows, with (gene_id, taxon_id) as key.

```
>>> ann_func = lambda x: [(x.gene_id, x.taxon_id)]
>>> from_gff(annotations, ['SAMPLE1', 'SAMPLE2'], ann_func=ann_func)
```

mgkit.counts.func.**get_uid_info**(*info_dict*, *uid*)

Simple function to get a value from a dictionary of tuples (gene_id, taxon_id)

mgkit.counts.func.**get_uid_info_ann**(*annotations*, *uid*)

Simple function to get a value from a dictionary of annotations

mgkit.counts.func.**load_counts_from_gff**(*annotations*, *elem_func=<function <lambda>>*, *sample_func=None*, *nozero=True*)

New in version 0.2.5.

Loads counts for each annotations that are stored into the annotation *counts_* attributes. Annotations with a total of 0 counts are skipped by default (nozero=True), the row index is set to the *uid* of the annotation and the column to the sample name. The functions used to transform the indices expect the annotation (for the row, *elem_func*) and the sample name (for the column, *sample_func*).

Parameters

- **annotations** (*iter*) – iterable of annotations
- **elem_func** (*func*) – function that accepts an annotation and return a str/int for a Index or a tuple for a MultiIndex, defaults to returning the *uid* of the annotation
- **sample_func** (*func, None*[66]) – function that accepts the sample name and returns tuple for a MultiIndex. Defaults to *None* so no transformation is performed
- **nozero** (*bool*[67]) – if *True*, annotations with no counts are skipped

mgkit.counts.func.**load_deseq2_results**(*file_name*, *taxon_id=None*)

New in version 0.1.14.

Reads a CSV file output with DESeq2 results, adding a taxon_id to the index for concatenating multiple results from different taxonomic groups.

Parameters **file_name** (*str*[68]) – file name of the CSV

Returns a MultiIndex DataFrame with the results

Return type pandas.DataFrame

mgkit.counts.func.**load_htseq_counts**(*file_handle*, *conv_func=<class 'int'>*)

Changed in version 0.1.15: added *conv_func* parameter

Loads an HTSeq-count result file

Parameters

- **file_handle** (*file or str*[69]) – file handle or string with file name
- **conv_func** (*func*) – function to convert the number from string, defaults to *int*, but *float* can be used as well

Yields *tuple* – first element is the gene_id and the second is the count

mgkit.counts.func.**load_sample_counts**(*info_dict*, *counts_iter*, *taxonomy*, *inc_anc=None*, *rank=None*, *gene_map=None*, *ex_anc=None*, *include_higher=True*, *cached=True*, *uid_used=None*)

Changed in version 0.1.14: added *cached* argument

Changed in version 0.1.15: added *uid_used* parameter

Changed in version 0.2.0: info_dict can be a function

Reads sample counts, filtering and mapping them if requested. It's an example of the usage of the above functions.

Parameters

- **info_dict** (*dict*[70]) – dictionary that has *uid* as key and *(gene_id, taxon_id)* as value. In alternative a function that accepts a *uid* as sole argument and returns *(gene_id, taxon_id)*
- **counts_iter** (*iterable*) – iterable that yields a *(uid, count)*
- **taxonomy** – taxonomy instance

[66] https://docs.python.org/3/library/constants.html#None
[67] https://docs.python.org/3/library/functions.html#bool
[68] https://docs.python.org/3/library/stdtypes.html#str
[69] https://docs.python.org/3/library/stdtypes.html#str
[70] https://docs.python.org/3/library/stdtypes.html#dict

- **inc_anc** (*int*[71], *list*[72]) – ancestor taxa to include
- **rank** (*str*[73]) – rank to which map the counts
- **gene_map** (*dict*[74]) – dictionary with the gene mappings
- **ex_anc** (*int*[75], *list*[76]) – ancestor taxa to exclude
- **include_higher** (*bool*[77]) – if False, any rank different than the requested one is discarded
- **cached** (*bool*[78]) – if *True*, the function will use *mgkit.simple_cache.memoize* to cache some of the functions used
- **uid_used** (*None*[79], *dict*[80]) – an empty dictionary in which to store the *uid* that were assigned to each key of the returned pandas.Series. If *None*, no information is saved

Returns array with MultiIndex *(gene_id, taxon_id)* with the filtered and mapped counts

Return type pandas.Series

mgkit.counts.func.**load_sample_counts_to_genes**(*info_func*, *counts_iter*, *taxonomy*, *inc_anc=None*, *gene_map=None*, *ex_anc=None*, *cached=True*, *uid_used=None*)

New in version 0.1.14.

Changed in version 0.1.15: added *uid_used* parameter

Reads sample counts, filtering and mapping them if requested. It's a variation of *load_sample_counts()*, with the counts being mapped only to each specific gene_id. Another difference is the absence of any assumption on the first parameter. It is expected to return a (gene_id, taxon_id) tuple.

Parameters

- **info_func** (*callable*) – any callable that accept an *uid* as the only parameter and and returns *(gene_id, taxon_id)* as value
- **counts_iter** (*iterable*) – iterable that yields a *(uid, count)*
- **taxonomy** – taxonomy instance
- **inc_anc** (*int*[81], *list*[82]) – ancestor taxa to include
- **rank** (*str*[83]) – rank to which map the counts
- **gene_map** (*dict*[84]) – dictionary with the gene mappings
- **ex_anc** (*int*[85], *list*[86]) – ancestor taxa to exclude
- **cached** (*bool*[87]) – if *True*, the function will use *mgkit.simple_cache.memoize* to cache some of the functions used

[71] https://docs.python.org/3/library/functions.html#int
[72] https://docs.python.org/3/library/stdtypes.html#list
[73] https://docs.python.org/3/library/stdtypes.html#str
[74] https://docs.python.org/3/library/stdtypes.html#dict
[75] https://docs.python.org/3/library/functions.html#int
[76] https://docs.python.org/3/library/stdtypes.html#list
[77] https://docs.python.org/3/library/functions.html#bool
[78] https://docs.python.org/3/library/functions.html#bool
[79] https://docs.python.org/3/library/constants.html#None
[80] https://docs.python.org/3/library/stdtypes.html#dict
[81] https://docs.python.org/3/library/functions.html#int
[82] https://docs.python.org/3/library/stdtypes.html#list
[83] https://docs.python.org/3/library/stdtypes.html#str
[84] https://docs.python.org/3/library/stdtypes.html#dict
[85] https://docs.python.org/3/library/functions.html#int
[86] https://docs.python.org/3/library/stdtypes.html#list
[87] https://docs.python.org/3/library/functions.html#bool

- **uid_used** (*None*[88], *dict*[89]) – an empty dictionary in which to store the *uid* that were assigned to each key of the returned pandas.Series. If *None*, no information is saved

Returns array with Index *gene_id* with the filtered and mapped counts

Return type pandas.Series

mgkit.counts.func.**load_sample_counts_to_taxon**(*info_func*, *counts_iter*, *taxonomy*, *inc_anc=None*, *rank=None*, *ex_anc=None*, *include_higher=True*, *cached=True*, *uid_used=None*)

New in version 0.1.14.

Changed in version 0.1.15: added *uid_used* parameter

Reads sample counts, filtering and mapping them if requested. It's a variation of *load_sample_counts()*, with the counts being mapped only to each specific taxon. Another difference is the absence of any assumption on the first parameter. It is expected to return a (gene_id, taxon_id) tuple.

Parameters

- **info_func** (*callable*) – any callable that accept an *uid* as the only parameter and and returns *(gene_id, taxon_id)* as value
- **counts_iter** (*iterable*) – iterable that yields a *(uid, count)*
- **taxonomy** – taxonomy instance
- **inc_anc** (*int*[90], *list*[91]) – ancestor taxa to include
- **rank** (*str*[92]) – rank to which map the counts
- **ex_anc** (*int*[93], *list*[94]) – ancestor taxa to exclude
- **include_higher** (*bool*[95]) – if False, any rank different than the requested one is discarded
- **cached** (*bool*[96]) – if *True*, the function will use *mgkit.simple_cache.memoize* to cache some of the functions used
- **uid_used** (*None*[97], *dict*[98]) – an empty dictionary in which to store the *uid* that were assigned to each key of the returned pandas.Series. If *None*, no information is saved

Returns array with Index *taxon_id* with the filtered and mapped counts

Return type pandas.Series

mgkit.counts.func.**map_counts**(*counts_iter*, *info_func*, *gmapper=None*, *tmapper=None*, *index=None*, *uid_used=None*)

Changed in version 0.1.14: added *index* parameter

Changed in version 0.1.15: added *uid_used* parameter

Maps counts according to the gmapper and tmapper functions. Each mapped gene ID count is the sum of all uid that have the same ID(s). The same is true for the taxa.

[88] https://docs.python.org/3/library/constants.html#None
[89] https://docs.python.org/3/library/stdtypes.html#dict
[90] https://docs.python.org/3/library/functions.html#int
[91] https://docs.python.org/3/library/stdtypes.html#list
[92] https://docs.python.org/3/library/stdtypes.html#str
[93] https://docs.python.org/3/library/functions.html#int
[94] https://docs.python.org/3/library/stdtypes.html#list
[95] https://docs.python.org/3/library/functions.html#bool
[96] https://docs.python.org/3/library/functions.html#bool
[97] https://docs.python.org/3/library/constants.html#None
[98] https://docs.python.org/3/library/stdtypes.html#dict

Parameters

- **counts_iter** (*iterable*) – iterator that yields a tuple (uid, count)
- **info_func** (*func*) – function accepting a *uid* that returns a tuple *(gene_id, taxon_id)*
- **gmapper** (*func*) – fucntion that accepts a *gene_id* and returns a list of mapped IDs
- **tmapper** (*func*) – fucntion that accepts a *taxon_id* and returns a new *taxon_id*
- **index** (*None*[99], *str*[100]) – if None, the index of the Series if *(gene_id, taxon_id)*, if a str, it can be either *gene* or *taxon*, to specify a single value
- **uid_used** (*None*[101], *dict*[102]) – an empty dictionary in which to store the *uid* that were assigned to each key of the returned pandas.Series. If *None*, no information is saved

Returns array with MultiIndex *(gene_id, taxon_id)* with the mapped counts

Return type pandas.Series

mgkit.counts.func.**map_counts_to_category**(*counts*, *gene_map*, *nomap=False*, *nomap_id='NOMAP'*)

Used to map the counts from a certain gene identifier to another. Genes with no mappings are not counted, unless *nomap=True*, in which case they are counted as *nomap_id*.

Parameters

- **counts** (*iterator*) – an iterator that yield a tuple, with the first value being the gene_id and the second value the count for it
- **gene_map** (*dictionary*) – a dictionary whose keys are the gene_id yield by *counts* and the values are iterable of mapping identifiers
- **nomap** (*bool*[103]) – if False, counts for genes with no mappings in *gene_map* are discarded, if True, they a counted as *nomap_id*
- **nomap_id** (*str*[104]) – name of the mapping for genes with no mappings

Returns mapped counts

Return type pandas.Series

mgkit.counts.func.**map_gene_id_to_map**(*gene_map*, *gene_id*)

Function that extract a list of gene mappings from a dictionary and returns an empty list if the *gene_id* is not found.

mgkit.counts.func.**map_taxon_id_to_rank**(*taxonomy*, *rank*, *taxon_id*, *include_higher=True*)

Maps a *taxon_id* to the request taxon rank. Returns *None* if *include_higher* is False and the found rank is not the one requested.

Internally uses *mgkit.taxon.Taxonomy.get_ranked_taxon()*

Parameters

- **taxonomy** – taxonomy instance
- **rank** (*str*[105]) – taxonomic rank requested
- **taxon_id** (*int*[106]) – taxon_id to map
- **include_higher** (*bool*[107]) – if False, any rank different than the requested one is

[99] https://docs.python.org/3/library/constants.html#None
[100] https://docs.python.org/3/library/stdtypes.html#str
[101] https://docs.python.org/3/library/constants.html#None
[102] https://docs.python.org/3/library/stdtypes.html#dict
[103] https://docs.python.org/3/library/functions.html#bool
[104] https://docs.python.org/3/library/stdtypes.html#str
[105] https://docs.python.org/3/library/stdtypes.html#str
[106] https://docs.python.org/3/library/functions.html#int
[107] https://docs.python.org/3/library/functions.html#bool

discarded

Returns if the mapping is successful, the ranked taxon_id is returned, otherwise *None* is returned

Return type (int[108], None[109])

mgkit.counts.glm module

New in version 0.3.3.

GLM models with metagenomes and metatranscriptomes. Experimental

`mgkit.counts.glm.`**`fit_lowess_interpolate`**(*endog*, *exog*, *frac=0.2*, *it=3*, *kind='slinear'*)

Fits a lowess for the passed *endog* (Y) and *exog* (X) and returns an interpolated function that describes it. The first 4 arguments are passed to `statsmodels.api.sm.nonparametric.lowess()`, while the last one is passed to `scipy.interpolate.interp1d()`

Parameters

- **`endog`** (*`array`*) – array of the dependent variable (Y)
- **`exog`** (*`array`*) – array of the indipendent variable (X)
- **`frac`** (*`float`*[110]) – fraction of the number of elements to use when fitting (0.0-1.0)
- **`it`** (*`int`*[111]) – number of iterations to fit the lowess
- **`kind`** (*`str`*[112]) – type of interpolation to use

Returns interpolated function representing the lowess fitted from the data passed

Return type func

`mgkit.counts.glm.`**`lowess_ci_bootstrap`**(*endog*, *exog*, *num=100*, *frac=0.2*, *it=3*, *alpha=0.05*, *delta=0.0*, *min_value=0.001*, *kind='slinear'*)

Bootstraps a lowess for the dependent (*endog*) and indipendent (*exog*) arguments.

Parameters

- **`endog`** (*`array`*) – indipendent variable (Y)
- **`exog`** (*`array`*) – indipendent variable (X)
- **`num`** (*`int`*[113]) – number of iterations for the bootstrap
- **`frac`** (*`float`*[114]) – fraction of the array to use when fitting
- **`it`** (*`int`*[115]) – number of iterations used to fit the lowess
- **`alpha`** (*`float`*[116]) – confidence intervals for the bootstrap
- **`delta`** (*`float`*[117]) – passed to `statsmodels.api.nonparametric.lowess()`
- **`min_value`** (*`float`*[118]) – minimum value for the function to avoid out of bounds
- **`kind`** (*`str`*[119]) – type of interpolation passed to `scipy.interpolate.`

[108] https://docs.python.org/3/library/functions.html#int
[109] https://docs.python.org/3/library/constants.html#None
[110] https://docs.python.org/3/library/functions.html#float
[111] https://docs.python.org/3/library/functions.html#int
[112] https://docs.python.org/3/library/stdtypes.html#str
[113] https://docs.python.org/3/library/functions.html#int
[114] https://docs.python.org/3/library/functions.html#float
[115] https://docs.python.org/3/library/functions.html#int
[116] https://docs.python.org/3/library/functions.html#float
[117] https://docs.python.org/3/library/functions.html#float
[118] https://docs.python.org/3/library/functions.html#float
[119] https://docs.python.org/3/library/stdtypes.html#str

interp1d()

Returns the first element is the function describing the lowest confidence interval, the second element is for the highest confidence interval and the last one for the mean

Return type tuple[120]

Note: Performance increase with the value of *delta*.

---

mgkit.counts.glm.**optimise_alpha_scipy**(*formula*, *data*, *mean_func*, *q1_func*, *q2_func*)
New in version 0.4.0.

Used to find an optimal *alpha* parameter for the Negative Binomial distribution used in *statsmodels*, using the lowess functions from *lowess_ci_bootstrap()*.

Parameters

- **formula** (*str*[121]) – the formula used for the regression
- **data** (*DataFrame*) – DataFrame for regression
- **mean_func** (*func*) – function for the mean *lowess_ci_bootstrap()*
- **q1_func** (*func*) – function for the q1 *lowess_ci_bootstrap()*
- **q2_func** (*func*) – function for the q2 *lowess_ci_bootstrap()*

Returns *alpha* value for the Negative Binomial

Return type float[122]

mgkit.counts.glm.**optimise_alpha_scipy_function**(*args*, *formula*, *data*, *criterion='aic'*)
New in version 0.4.0.

mgkit.counts.glm.**variance_to_alpha**(*mu*, *func*, *min_alpha=0.001*)
Based on the variance defined in the Negative Binomial in statsmodels

var = mu + alpha * (mu ** 2)

Parameters

- **mu** (*float*[123]) – mean to calculate the alphas for
- **func** (*func*) – function that returns the variace of the mean
- **min_alpha** (*float*[124]) – value of alpha if the *func* goes out of bounds

Returns value of alpha for the passed mean

Return type float[125]

mgkit.counts.scaling module

Scaling functions for counts

mgkit.counts.scaling.**scale_deseq**(*dataframe*)
New in version 0.1.13.

Scale a dataframe using the deseq scaling. Uses *scale_factor_deseq()*

[120] https://docs.python.org/3/library/stdtypes.html#tuple
[121] https://docs.python.org/3/library/stdtypes.html#str
[122] https://docs.python.org/3/library/functions.html#float
[123] https://docs.python.org/3/library/functions.html#float
[124] https://docs.python.org/3/library/functions.html#float
[125] https://docs.python.org/3/library/functions.html#float

mgkit.counts.scaling.**scale_factor_deseq**(*dataframe*)

New in version 0.1.13.

Returns the scale factor according to he deseq paper. The columns of the dataframe are the samples.

size factor $\hat{s}_j$ for sample *j* (from DESeq paper).

$$\hat{s}_j = median_i(\frac{k_{ij}}{(\prod_{v=1}^{m} k_{iv})^{1/m}})$$

mgkit.counts.scaling.**scale_rpkm**(*dataframe*, *gene_len*)

New in version 0.1.14.

Perform an RPKM scaling of the pandas dataframe/series supplied using the *gene_len* series containing the gene sizes for all elements of *dataframe*

$$RPKM = \frac{10^9 \cdot C}{N \cdot L}$$

Module contents

mgkit.db package

Submodules

mgkit.db.dbm module

New in version 0.2.1.

This module contains functions and classes to use for a dbm like representation of annotations using the *semidbm* package

class mgkit.db.dbm.**GFFDB**(*db=None*)

Bases: object[126]

New in version 0.2.1.

A wrapper for a semidbm instance, used to convert the GFF line stored in the DB into an *mgkit.io.gff.Annotation* instance. If a string is passed to the init method, a DB will be opened with the *c* flag.

The object behaves like a dictionary, wrapping the access to annoations using a *uid* as key and converting the line into an *mgkit.io.gff.Annotation* instance.

db = None

items()

iteritems()

itervalues()

values()

mgkit.db.dbm.**create_gff_dbm**(*annotations*, *file_name*)

New in version 0.2.1.

Creates a semidbm database, using an annotation *uid* as key and the gff line as value. The object is synced before being returned.

Note: A GFF line is used instead of a json representation because it was more compact when semidbm was tested.

---

[126] https://docs.python.org/3/library/functions.html#object

Parameters

- **annotations** (*iterable*) – iterable of annotations
- **file_name** (*str*[127]) – database file name, opened with the *c* flag.

Returns a semidbm database object

Return type object[128]

mgkit.db.mongo module

New in version 0.2.1.

This module contains functions and classes to use for a DB like representation of annotations using the *pymongo* package, a driver to **MongoDB**.

In a MongoDB document, exported from an annotation, using the *mgkit.io.gff.Annotation.to_mongodb()* method, the keys that are defined are:

```
seq_id, source, feat_type, start, end, score, strand,
phase, gene_id, taxon_id, bitscore, exp_nonsyn, exp_syn,
length, dbq, coverage, map
```

These are defined because they have values that are not strings (defined as properties in *mgkit.io.gff.Annotation*. The rest of the attributes defined are kept as well, but no ckeck for the data type is made.

Note: lineage is added as a key, whose values are taxon_id, if a function has been passed to *mgkit.io.gff.Annotation.to_mongodb()*

The exception is the **map** key in the document. It store both the EC mappings (EC attribute in the GFF), as well as all mappings whose attribute starts with *map_*. The former is usually accessed from *mgkit.io.gff.Annotation.get_ec()* while the latter from *mgkit.io.gff.Annotation.get_mapping()* or *mgkit.io.gff.Annotation.get_mappings()*.

These 3 methods return a list and this list is used in the MongoDB document. The MongoDB document will contain a **map** key where the values are the type of mappings, and the values the list of IDs the annoation maps to.

Table 1: Example for the map dictionary

Type	GFF	Annotation	MongoDB Document	MongoDB Query
EC	EC	get_ec	ec	map.ec
KO	map_KO	get_mapping('ko')	ko	map.ko
eggNOG	map_EGGNOG	get_mapping('eggnog')	eggnog	map.eggnog

class mgkit.db.mongo.**GFFDB**(*db*, *collection*, *uri=None*, *timeout=5*)

Bases: object[129]

Changed in version 0.3.4: added *timeout* parameter

Wrapper to a MongoDB connection/db. It is used to automate the convertion of MongoDB records into *mgkit.io.gff.Annotation* instances.

__getitem__(*uid*)

New in version 0.3.1.

Retrieves an annotation from the DB by its *uid*

[127] https://docs.python.org/3/library/stdtypes.html#str
[128] https://docs.python.org/3/library/functions.html#object
[129] https://docs.python.org/3/library/functions.html#object

__iter__()
New in version 0.3.1.

Iterates over all annotations

conn = None

convert_record(*record*)
Changed in version 0.3.1: removes *lineage* from the attributes

Converts the record (a dictionary instance) to an Annotation

cursor(*query=None*)
Returns a cursor for the query

db = None

find_annotation(*query=None*)
Iterate over a cursor created using *query* and yields each record after converting it to a *mgkit.io.gff.Annotation* instance, using *mgkit.db.mongo.GFFDB.convert_record()*.

insert_many(*annotations*)
New in version 0.3.4.

Inserts annotations into the DB

> **Warning:** The object must be a *mgkit.io.gff.Annotation*

insert_one(*annotation*)
New in version 0.3.4.

Inserts an annotation into the DB

Raises TypeError[130] – if the passed object is not an annotation

items()
New in version 0.3.1.

Iterates over all the annotations in the db/collection, yielding a tuple (*annotation.uid*, *annotation*)

iteritems()
New in version 0.3.1.

Alias for *GFFDB.items()*

itervalues()
New in version 0.3.1.

Alias for *GFFDB.values()*

keys()
New in version 0.3.1.

Iterates over all the *uid* in the db/collection

values()
New in version 0.3.1.

Iterates over all the annotations in the db/collection

Module contents

mgkit.filter package

[130] https://docs.python.org/3/library/exceptions.html#TypeError

Submodules

mgkit.filter.common module

Common consts/data for package filter

exception mgkit.filter.common.**FilterFails**
Bases: Exception[131]

Raised if a filter fails

mgkit.filter.gff module

GFF filtering

mgkit.filter.gff.**choose_annotation**(*ann1*, *ann2*, *overlap=100*, *choose_func=None*)
New in version 0.1.12.

Given two *mgkit.io.gff.Annotation*, if one of of the two annotations either is contained in the other or they overlap for at least a *overlap* number of bases, *choose_func* will be applied to both. The result of *choose_func* is the the annotation to be discarderd. It returns *None* if the annotations should be both kept.

No checks are made to ensure that the two annotations are on the same sequence and strand, as the *intersect* method of *mgkit.io.gff.Annotation* takes care of them.

Parameters

- **ann1** – instance of *mgkit.io.gff.Annotation*
- **ann2** – instance of *mgkit.io.gff.Annotation*
- **overlap** (*int*[132], *float*[133]) – number of bases overlap that trigger the filtering
- **choose_func** (*None*[134], *func*) – function that accepts *ann1* and *ann2* and return the one to be discarded or None if both are accepted

Returns returns either the *mgkit.io.gff.Annotation* to be discarded or None, which is the result of *choose_func*

Return type (None[135], *Annotation*)

Note: If *choose_func* is *None*, the default function is used:

```
lambda a1, a2: min(a1, a2, key=lambda el: (el.dbq, el.bitscore,
                   len(el)))
```

In order of importance the db quality, the bitscore and the length. The annotation with the lowest tuple value is the one to discard.

---

mgkit.filter.gff.**filter_annotations**(*annotations*, *choose_func=None*, *sort_func=None*, *reverse=True*)
New in version 0.1.12.

Filter an iterable of *mgkit.io.gff.Annotation* instances sorted using *sort_func* as key in *sorted* and if the order is to be *reverse*; it then applies *choose_func* on all possible pair combinations, using itertools.combinations.

[131] https://docs.python.org/3/library/exceptions.html#Exception
[132] https://docs.python.org/3/library/functions.html#int
[133] https://docs.python.org/3/library/functions.html#float
[134] https://docs.python.org/3/library/constants.html#None
[135] https://docs.python.org/3/library/constants.html#None

By default *choose_func* is `choose_annotation()` with the default values, the list of annotation is sorted by bitscore, from the highest to the lowest value.

Parameters

- **annotations** (*iterable*) – iterable of `mgkit.io.gff.Annotation` instances
- **choose_func** (*func, None*[136]) – function used to select the *losing* annotation; if *None*, it will be `choose_annotation()` with default values
- **sort_func** (*func, None*[137]) – by default the sorting key is the bitscore of the annotations
- **reverse** (*bool*[138]) – passed to *sorted*, by default is reversed

Returns a set with the annotations that pass the filtering

Return type set[139]

mgkit.filter.gff.**filter_attr_num**(*annotation*, *attr=None*, *value=None*, *greater=True*)

Checks if an annotation *attr* dictionary contains a key whose value is greater than or equal, or lower than or equal, for the requested value

Parameters

- **annotation** – `mgkit.io.gff.Annotation` instance
- **attr** (*str*[140]) – key in the `mgkit.io.gff.Annotation.attr` dictionary
- **value** (*int*[141]) – the value to which we need to compare
- **greater** (*bool*[142]) – if True the value must be equal or greater than and if False equal of lower than

Returns True if the test passes

Return type bool[143]

mgkit.filter.gff.**filter_attr_num_s**(*annotation*, *attr=None*, *value=None*, *greater=True*)

New in version 0.3.1.

Checks if an annotation *attr* dictionary contains a key whose value is greater or lower than the requested value

Parameters

- **annotation** – `mgkit.io.gff.Annotation` instance
- **attr** (*str*[144]) – key in the `mgkit.io.gff.Annotation.attr` dictionary
- **value** (*int*[145]) – the value to which we need to compare
- **greater** (*bool*[146]) – if True the value must be greater than and if False lower than

Returns True if the test passes

Return type bool[147]

[136] https://docs.python.org/3/library/constants.html#None
[137] https://docs.python.org/3/library/constants.html#None
[138] https://docs.python.org/3/library/functions.html#bool
[139] https://docs.python.org/3/library/stdtypes.html#set
[140] https://docs.python.org/3/library/stdtypes.html#str
[141] https://docs.python.org/3/library/functions.html#int
[142] https://docs.python.org/3/library/functions.html#bool
[143] https://docs.python.org/3/library/functions.html#bool
[144] https://docs.python.org/3/library/stdtypes.html#str
[145] https://docs.python.org/3/library/functions.html#int
[146] https://docs.python.org/3/library/functions.html#bool
[147] https://docs.python.org/3/library/functions.html#bool

mgkit.filter.gff.**filter_attr_str**(*annotation*, *attr=None*, *value=None*, *equal=True*)

Checks if an annotation *attr* dictionary contains a key shose value is equal to, or contains the requested value

Parameters

- **annotation** – *mgkit.io.gff.Annotation* instance
- **attr** (*str*[148]) – key in the *mgkit.io.gff.Annotation.attr* dictionary
- **value** (*int*[149]) – the value to which we need to compare
- **equal** (*bool*[150]) – if True the value must be equal and if False equal value must be contained

Returns True if the test passes

Return type bool[151]

mgkit.filter.gff.**filter_base**(*annotation*, *attr=None*, *value=None*)

Checks if an annotation attribute is equal to the requested value

Parameters

- **annotation** – *mgkit.io.gff.Annotation* instance
- **attr** (*str*[152]) – attribute of the annotation
- **value** – the value that the attribute should be equal to

Returns True if the supplied value is equal to the attribute ot False otherwise

Return type bool[153]

mgkit.filter.gff.**filter_base_num**(*annotation*, *attr=None*, *value=None*, *greater=True*)

Checks if an annotation attribute is greater, equal of lower than the requested value

Parameters

- **annotation** – *mgkit.io.gff.Annotation* instance
- **attr** (*str*[154]) – attribute of the annotation
- **value** (*int*[155]) – the value to which the attribute should be compared to
- **greater** (*bool*[156]) – if True the attribute value must be equal or greater than and if False equal of lower than

Returns True if the test passes

Return type bool[157]

mgkit.filter.gff.**filter_len**(*annotation*, *value=None*, *greater=True*)

Checks if an annotation length is longer, equal of shorter than the requested value

Parameters

- **annotation** – *mgkit.io.gff.Annotation* instance
- **value** (*int*[158]) – the length to which the attribute should be compared to

[148] https://docs.python.org/3/library/stdtypes.html#str
[149] https://docs.python.org/3/library/functions.html#int
[150] https://docs.python.org/3/library/functions.html#bool
[151] https://docs.python.org/3/library/functions.html#bool
[152] https://docs.python.org/3/library/stdtypes.html#str
[153] https://docs.python.org/3/library/functions.html#bool
[154] https://docs.python.org/3/library/stdtypes.html#str
[155] https://docs.python.org/3/library/functions.html#int
[156] https://docs.python.org/3/library/functions.html#bool
[157] https://docs.python.org/3/library/functions.html#bool
[158] https://docs.python.org/3/library/functions.html#int

- **greater** (*bool*[159]) – if True the annotation length must be equal or greater than and if False equal of lower than

Returns True if the test passes

Return type bool[160]

mgkit.filter.lists module

Module used to filter lists

mgkit.filter.lists.**aggr_filtered_list**(*val_list*, *aggr_func=<function mean>*, *filt_func=<function <lambda>>*)

Aggregate a list of values using 'aggr_func' on a list that passed the filtering in 'filt_func'.

'filt_func' is a function that returns True or False for each value in val_list. If the return value is True, the element is included in the values passed to 'aggr_func'. Internally a list comprehension is used and the result passed to 'aggr_func'

Parameters

- **val_list** (*iterable*) – list of values
- **aggr_func** (*func*) – function used to aggregate the list values
- **filt_func** (*func*) – function the return True or False

Returns the result of the applied 'aggr_func'

mgkit.filter.reads module

Some test functions to filter sequences

mgkit.filter.reads.**expected_error_rate**(*qualities*)

Calculate the expected error rate for an array of qualities (converted to probabilities).

mgkit.filter.reads.**trim_by_ee**(*qualities*, *min_length=50*, *threshold=0.5*, *chars=True*, *base=33*)

Trim a sequence based on the expected error rate.

mgkit.filter.taxon module

New in version 0.1.9.

Taxa filtering functions

mgkit.filter.taxon.**filter_by_ancestor**(*taxon_id*, *filter_list=None*, *exclude=False*, *taxonomy=None*)

New in version 0.1.13.

Convenience function for *filter_taxon_by_id_list()*, as explained in the latter example.

mgkit.filter.taxon.**filter_taxon_by_id_list**(*taxon_id*, *filter_list=None*, *exclude=False*, *func=None*)

Filter a taxon_id against a list of taxon ids. Returns True if the conditions of the filter are met.

If func is not None, a function that accepts two values is expected, it should be either a partial *is_ancestor* which only accepts taxon_id and anc_id or another function that behaves the same way.

Note: if func is None, a simple lambda is used to test identity:

[159] https://docs.python.org/3/library/functions.html#bool
[160] https://docs.python.org/3/library/functions.html#bool

```
func = lambda t_id, a_id: t_id == a_id
```

Parameters

- **taxon_id** (*int*[161]) – the taxon id to filter
- **filter_list** (*iterable*) – an iterable with taxon ids
- **exclude** (*bool*[162]) – if the filter is reversed (i.e. included if NOT found)
- **func** (*func or None*[163]) – a function that accepts taxon_id and an anc_id and returns a bool to indicated if anc_id is ancestor of taxon_id. Equivalent to *is_ancestor()*.

Returns

True if the taxon_id is in the filter list (or a descendant of it) False if it's not found. Exclude equal to True reverse the result.

Found	Exclude	Return Value
Yes	False	True
No	False	False
Yes	True	False
No	True	True

Return type bool[164]

Example

If using func and assuming that *taxonomy* is an instance of *Taxonomy* with data loaded:

```
>>> import functools
>>> import mgkit.taxon
>>> func = functools.partial(mgkit.taxon.is_ancestor, taxonomy)
>>> filter_taxon_by_id_list(1200582, [838], func=func)
True
```

Module contents

Package used to store filter functions (unless specific to a package)

mgkit.io package

Submodules

mgkit.io.blast module

Blast routines and parsers

mgkit.io.blast.**add_blast_result_to_annotation**(*annotation*, *gi_taxa_dict*, *taxonomy*, *threshold=60*)

Deprecated since version 0.4.0.

[161] https://docs.python.org/3/library/functions.html#int
[162] https://docs.python.org/3/library/functions.html#bool
[163] https://docs.python.org/3/library/constants.html#None
[164] https://docs.python.org/3/library/functions.html#bool

Adds blast information to a GFF annotation.

Parameters

- **annotation** – GFF annotation object
- **gi_taxa_dict** (*dict*[165]) – dictionary returned by `parse_gi_taxa_table()`.
- **taxonomy** – Uniprot taxonomy, used to add the taxon name to the annotation

`mgkit.io.blast.`**parse_accession_taxa_table**(*file_handle*, *acc_ids=None*, *key=1*, *value=2*, *num_lines=1000000*, *no_zero=True*)

New in version 0.2.5.

Changed in version 0.3.0: added *no_zero*

This function superseeds `parse_gi_taxa_table()`, since NCBI is deprecating the GIDs in favor of accessions like *X53318*. The new file can be found at the NCBI *ftp://ftp.ncbi.nih.gov/pub/taxonomy/accession2taxid*, for DNA sequences (*nt* DB) *nucl_gb.accession2taxid.gz*.

The file contains 4 columns, the first one is the accession without its version, the second one includes the version, the third column is the taxonomic identifier and the fourth is either the old GID or **na**.

The column used as key is the *second*, since by default the fasta headers used in NCBI DBs use the versioned identifier. To use the GID as key, the *key* parameter can be set to 3, but if no identifier is found (*na* as per the file README), the line is skipped.

Parameters

- **file_handle** (*str*[166], *file*) – file name or open file handle
- **acc_ids** (*None*[167], *list*[168]) – if it's not *None* only the keys included in the passed *acc_ids* list will be returned
- **key** (*int*[169]) – 0-based index for the column to use as accession. Defaults to the versioned accession that is used in GenBank fasta files.
- **num_lines** (*None*[170], *int*[171]) – number of which a message is logged. If None, no message is logged
- **no_zero** (*bool*[172]) – if True (default) a key with taxon_id of 0 is not yield

---

Note: GIDs are being phased out in September 2016: http://www.ncbi.nlm.nih.gov/news/03-02-2016-phase-out-of-GI-numbers/

`mgkit.io.blast.`**parse_blast_tab**(*file_handle*, *seq_id=0*, *ret_col=(0, 1, 2, 6, 7, 11)*, *key_func=None*, *value_funcs=None*)

New in version 0.1.12.

Parses blast output tab format, returning for each line a key (the query id) and the columns requested in a tuple.

Parameters

- **file_handle** (*file*) – file name or file handle for the blast ouput
- **seq_id** (*int*[173]) – index for the column which has the query id

[165] https://docs.python.org/3/library/stdtypes.html#dict
[166] https://docs.python.org/3/library/stdtypes.html#str
[167] https://docs.python.org/3/library/constants.html#None
[168] https://docs.python.org/3/library/stdtypes.html#list
[169] https://docs.python.org/3/library/functions.html#int
[170] https://docs.python.org/3/library/constants.html#None
[171] https://docs.python.org/3/library/functions.html#int
[172] https://docs.python.org/3/library/functions.html#bool
[173] https://docs.python.org/3/library/functions.html#int

- **ret_col** (*list*[174], *None*[175]) – list of indexes for the columns to be returned or *None* if all columns must be returned
- **key_func** (*None*[176], *func*) – function to transform the query id value in the key returned. If *None*, the query id is used
- **value_funcs** (*None*[177], *list*[178]) – list of functions to transform the value of all the requested columns. If *None* the values are not converted

Yields *tuple* – iterator of tuples with the first element being the query id after key_func is applied, if requested and the second element of the tuple is a tuple with the requested columns *ret_col*

Table 2: BLAST+ used with *-outfmt 6*, default columns

column index	description
0	query name
1	subject name
2	percent identities
3	aligned length
4	number of mismatched positions
5	number of gap positions
6	query sequence start
7	query sequence end
8	subject sequence start
9	subject sequence end
10	e-value
11	bit score

mgkit.io.blast.**parse_fragment_blast**(*file_handle*, *bitscore=40.0*)

New in version 0.1.13.

Parse the output of a BLAST output where the sequences are the single annotations, so the sequence names are the *uid* of the annotations.

The only returned values are the best hits, maxed by bitscore and identity.

Parameters

- **file_handle** (*str*[179], *file*) – file name or open file handle
- **bitscore** (*float*[180]) – minimum bitscore for accepting a hit

Yields *tuple* – a tuple whose first element is the *uid* (the sequence name) and the second is the a list of tuples whose first element is the GID (NCBI identifier), the second one is the identity and the third is the bitscore of the hit.

mgkit.io.blast.**parse_uniprot_blast**(*file_handle*, *bitscore=40*, *db='UNIPROT-SP'*, *dbq=10*, *name_func=None*, *feat_type='CDS'*, *seq_lengths=None*)

New in version 0.1.12.

Changed in version 0.1.13: added *name_func* argument

Changed in version 0.2.1: added *feat_type*

Changed in version 0.2.3: added *seq_lengths* and added subject *start* and *end* and *e-value*

Parses BLAST results in tabular format using *parse_blast_tab()*, applying a basic bitscore filter. Returns the annotations associated with each BLAST hit.

[174] https://docs.python.org/3/library/stdtypes.html#list
[175] https://docs.python.org/3/library/constants.html#None
[176] https://docs.python.org/3/library/constants.html#None
[177] https://docs.python.org/3/library/constants.html#None
[178] https://docs.python.org/3/library/stdtypes.html#list
[179] https://docs.python.org/3/library/stdtypes.html#str
[180] https://docs.python.org/3/library/functions.html#float

Parameters

- **file_handle** (*str*[181], *file*) – file name or open file handle
- **bitscore** (*int*[182], *float*[183]) – the minimum bitscore for an annotation to be accepted
- **db** (*str*[184]) – database used
- **dbq** (*int*[185]) – an index indicating the quality of the sequence database used; this value is used in the filtering of annotations
- **name_func** (*func*) – function to convert the name of the database sequences. Defaults to *lambda x: x.split('|')[1]*, which can be be used with fasta files provided by Uniprot
- **feat_type** (*str*[186]) – feature type in the GFF
- **seq_lengths** (*dict*[187]) – dictionary with the sequences lengths, used to deduct the frame of the '-' strand

Yields *Annotation* – instances of *mgkit.io.gff.Annotation* instance of each BLAST hit.

mgkit.io.fasta module

Simple fasta parser and a few utility functions

mgkit.io.fasta.**load_fasta**(*file_handle*)

Changed in version 0.1.13: now returns uppercase sequences

Loads a fasta file and returns a generator of tuples in which the first element is the name of the sequence and the second the sequence

Parameters **file_handle** (*str*[188], *file*) – fasta file to open; a file name or a file handle is expected

Yields *tuple* – first element is the sequence name/header, the second element is the sequence

mgkit.io.fasta.**load_fasta_files**(*files*)

New in version 0.3.4.

Loads all fasta files from a list or iterable

mgkit.io.fasta.**load_fasta_prodigal**(*file_handle*)

New in version 0.3.1.

Reads a Prodigal aminoacid fasta file and yields a dictionary with basic information about the sequences.

Parameters **file_handle** (*str*[189], *file*) – passed to *load_fasta()*

Yields *dict* – dictionary with the information contained in the header, the last of the attributes put into key *attr*, while the rest are transformed to other keys: seq_id, seq, start, end (genomic), strand, ordinal of

mgkit.io.fasta.**load_fasta_rename**(*file_handle*, *name_func=None*)

New in version 0.3.1.

[181] https://docs.python.org/3/library/stdtypes.html#str
[182] https://docs.python.org/3/library/functions.html#int
[183] https://docs.python.org/3/library/functions.html#float
[184] https://docs.python.org/3/library/stdtypes.html#str
[185] https://docs.python.org/3/library/functions.html#int
[186] https://docs.python.org/3/library/stdtypes.html#str
[187] https://docs.python.org/3/library/stdtypes.html#dict
[188] https://docs.python.org/3/library/stdtypes.html#str
[189] https://docs.python.org/3/library/stdtypes.html#str

Renames the header of the sequences using *name_func*, which is called on each header. By default, the behaviour is to keep the header to the left of the first space (BLAST behaviour).

mgkit.io.fasta.**split_fasta_file**(*file_handle*, *name_mask*, *num_files*)
New in version 0.1.13.

Splits a fasta file into a series of smaller files.

Parameters

- **file_handle** (*file, str*[190]) – fasta file with the input sequences
- **name_mask** (*str*[191]) – file name template for the splitted files, more informations are found in `mgkit.io.split_write()`
- **num_files** (*int*[192]) – number of files in which to distribute the sequences

mgkit.io.fasta.**write_fasta_sequence**(*file_handle*, *name*, *seq*, *wrap=60*, *write_mode='a'*)
Write a fasta sequence to file. If the *file_handle* is a string, the file will be opened using *write_mode*.

Parameters

- **file_handle** – file handle or string.
- **name** (*str*[193]) – header to write for the sequence
- **seq** (*str*[194]) – sequence to write
- **wrap** (*int*[195]) – int for the line wrapping. If None, the sequence will be written in a single line

mgkit.io.fastq module

Fastq utility functions

mgkit.io.fastq.**CASAVA_HEADER_NEW = '(?P<machine>[\\w-]+):\n (?P<runid>\\d+):\n (?P<celli**
New casava header regex, including indices for both forward and reverse

mgkit.io.fastq.**CASAVA_HEADER_OLD = '(?P<machine>\\w+-\\w+):\n (?P<lane>\\d):\n (?P<tile>**
Old casava header regex

mgkit.io.fastq.**check_fastq_type**(*qualities*)
Trys to guess the type of quality string used in a Fastq file

Parameters **qualities** (*str*[196]) – string with the quality scores as in the Fastq file

Return str a string with the guessed quality score

Note: Possible values are the following, classified but the values usually used in other softwares:

- ASCII33: sanger, illumina-1.8
- ASCII64: illumina-1.3, illumina-1.5, solexa-old

---

mgkit.io.fastq.**choose_header_type**(*seq_id*)
Return the guessed compiled regular expression :param str seq_id: sequence header to test

Returns compiled regular expression object or None if no match found

[190] https://docs.python.org/3/library/stdtypes.html#str
[191] https://docs.python.org/3/library/stdtypes.html#str
[192] https://docs.python.org/3/library/functions.html#int
[193] https://docs.python.org/3/library/stdtypes.html#str
[194] https://docs.python.org/3/library/stdtypes.html#str
[195] https://docs.python.org/3/library/functions.html#int
[196] https://docs.python.org/3/library/stdtypes.html#str

mgkit.io.fastq.**convert_seqid_to_new**(*seq_id*)
Convert old seq_id format for Illumina reads to the new found in Casava 1.8+

Parameters **seq_id** (*str*[197]) – seq_id of the sequence (stripped of '@')

Return str the new format seq_id

Note: Example from Wikipedia:

```
old casava seq_id:
@HWUSI-EAS100R:6:73:941:1973#0/1
new casava seq_id:
@EAS139:136:FC706VJ:2:2104:15343:197393 1:Y:18:ATCAC
```

---

mgkit.io.fastq.**convert_seqid_to_old**(*seq_id*, *index_as_seq=True*)
Deprecated since version 0.3.3.

Convert old seq_id format for Illumina reads to the new found in Casava until 1.8, which marks the new format.

Parameters

- **seq_id** (*str*[198]) – seq_id of the sequence (stripped of '@')
- **index_as_seq** (*bool*[199]) – if True, the index for the multiplex we'll be the sequence found at the end of the new format seq_id. Otherwise, 0 we'll be used

Return str the new format seq_id

mgkit.io.fastq.**load_fastq**(*file_handle*, *num_qual=False*)
New in version 0.3.1.

Loads a fastq file and returns a generator of tuples in which the first element is the name of the sequence, the second the sequence and the third the quality scores (converted in a numpy array if *num_qual* is True).

Note: this is a simple parser that assumes each sequence is on 4 lines, 1st and 3rd for the headers, 2nd for the sequence and 4th the quality scores

---

Parameters

- **file_handle** (*str*[200], *file*) – fastq file to open, can be a file name or a file handle
- **num_qual** (*bool*[201]) – if *False* (default), the quality score will be returned as ASCII character, if *True* a numpy array.

Yields *tuple* – first element is the sequence name/header, the second element is the sequence, the third is the quality score. The quality scores are kept as a string if *num_qual* is False (default) and converted to a numpy array with correct values (0-41) if *num_qual* is True

Raises

- ValueError[202] – if the headers in both sequence and quality scores are not
- valid. This implies that the sequence/qualities have carriage returns
- or the file is truncated.
- TypeError[203] – if the qualities are in a format different than sanger

[197] https://docs.python.org/3/library/stdtypes.html#str
[198] https://docs.python.org/3/library/stdtypes.html#str
[199] https://docs.python.org/3/library/functions.html#bool
[200] https://docs.python.org/3/library/stdtypes.html#str
[201] https://docs.python.org/3/library/functions.html#bool
[202] https://docs.python.org/3/library/exceptions.html#ValueError
[203] https://docs.python.org/3/library/exceptions.html#TypeError

- (min 0, max 40) or illumina-1.8 (0, 41)

mgkit.io.fastq.**load_fastq_rename**(*file_handle*, *num_qual=False*, *name_func=None*)
New in version 0.3.3.

Mirrors the same functionality in *mgkit.io.fasta.load_fasta_rename()*. Renames the header of the sequences using *name_func*, which is called on each header. By default, the behaviour is to keep the header to the left of the first space (BLAST behaviour).

mgkit.io.fastq.**write_fastq_sequence**(*file_handle*, *name*, *seq*, *qual*, *write_mode='a'*)
Changed in version 0.3.3: if *qual* is not a string it's converted to chars (phred33)

Write a fastq sequence to file. If the *file_handle* is a string, the file will be opened using *write_mode*.

Parameters

- **file_handle** – file handle or string.
- **name** (*str*[204]) – header to write for the sequence
- **seq** (*str*[205]) – sequence to write
- **qual** (*str*[206]) – quality string

mgkit.io.gff module

This modules define classes and function related to manipulation of GFF/GTF files.

class mgkit.io.gff.**Annotation**(*seq_id='None'*, *start=1*, *end=1*, *strand='+'*, *source='None'*, *feat_type='None'*, *score=0.0*, *phase=0*, *uid=None*, ***kwd*)
Bases: *mgkit.io.gff.GenomicRange*

New in version 0.1.12.

Changed in version 0.2.1: using __slots__ for better memory usage

Alternative implementation for an Annotation. When initialised, If *uid* is None, a unique id is added using *uuid.uuid4*.

add_exp_syn_count(*seq*, *syn_matrix=None*)
New in version 0.1.13.

Adds expected synonymous/non-synonymous values for an annotation.

Parameters **seq** (*str*[207]) – sequence corresponding to the annotation seq_id syn_matrix (None, dict): matrix that determines the return values. Defaults to the one defined in the called function mgkit.utils.sequnce.get_seq_expected_syn_count().

add_gc_content(*seq*)
Adds GC content information for an annotation. The formula is:

$$\frac{(G+C)}{(G+C+A+T)} \tag{7.1}$$

Modifies the instances of the annotation. gc_ratio will be added to its attributes.

Parameters **seq** (*str*[208]) – nucleotide sequence referred in the GFF

add_gc_ratio(*seq*)
Adds GC content information for an annotation. The formula is:

$$\frac{(A+T)}{(G+C)} \tag{7.2}$$

[204] https://docs.python.org/3/library/stdtypes.html#str
[205] https://docs.python.org/3/library/stdtypes.html#str
[206] https://docs.python.org/3/library/stdtypes.html#str
[207] https://docs.python.org/3/library/stdtypes.html#str
[208] https://docs.python.org/3/library/stdtypes.html#str

Modifies the instances of the annotation. gc_ratio will be added to its attributes.

Parameters **seq** (*str*[209]) – nucleotide sequence referred in the GFF

attr

bitscore
bitscore of the annotation

counts
New in version 0.2.2.

Returns the sample counts for the annotation

coverage
New in version 0.1.13.

Return the total coverage for the annotation

Return float coverage

Raises ***AttributeNotFound*** – if no coverage attribute is found

db
db used for the gene_id prediction

dbq
db quality of the annotation

exp_nonsyn
New in version 0.1.13.

Returns the expected number of non-synonymous changes

exp_syn
New in version 0.1.13.

Returns the expected number of synonymous changes

feat_type

fpkms
New in version 0.2.2.

Returns the sample fpkms for the annotation

gene_id
gene_id of the annotation, or *ko* if available

get_aa_seq(*seq*, *start=0*, *tbl=None*, *snp=None*)
New in version 0.1.16.

Returns a translated aminoacid sequence of the annotation. The snp parameter is passed to *Annotation.get_nuc_seq()*

Parameters

- **seq** (*seq*) – chromosome/contig sequence
- **start** (*int*[210]) – position (0-based) from where the correct occurs (frame). If None, the phase attribute is used
- **tbl** (*dict*[211]) – dictionary with the translation for each codon, passed to *mgkit.utils.sequence.translate_sequence()*
- **snp** (*tuple*[212]) – first element is the position of the SNP and the second element is the change

[209] https://docs.python.org/3/library/stdtypes.html#str
[210] https://docs.python.org/3/library/functions.html#int
[211] https://docs.python.org/3/library/stdtypes.html#dict
[212] https://docs.python.org/3/library/stdtypes.html#tuple

Returns aminoacid sequence

Return type str[213]

get_attr(*attr*, *conv=<class 'str'>*)

Changed in version 0.3.4: any GFF attribute can be returned

Changed in version 0.3.3: added *seq_id* as special attribute, in addition do *length*

New in version 0.1.13.

Generic method to get an attribute and convert it to a specific datatype. The order for the lookup is:

- length
- self.attr (dictionary)
- getattr(self) of the first 8 columns of a GFF (seq_id, source, ...)

get_ec(*level=4*)

New in version 0.1.13.

Changed in version 0.2.0: returns a *set* instead of a list

Returns the EC values associated with the annotation, cutting them at the desired level.

Parameters **level** (*int*[214]) – level of classification desired (between 1 and 4)

Returns list of all EC numbers associated, at the desired level, if none are found an empty set is returned

Return type set[215]

get_mapping(*db*)

New in version 0.1.13.

Returns the mappings, to a particular db, associated with the annotation.

Parameters **db** (*str*[216]) – database to which the mappings come from

Returns list of all mappings associated, to the specified db, if none are found an empty list is returned

Return type list[217]

get_mappings()

New in version 0.2.1.

Return a dictionary where the keys are the mapping DBs (lowercase) and and the values are the mapping IDs for that DB

get_nuc_seq(*seq*, *reverse=False*, *snp=None*)

New in version 0.1.13.

Changed in version 0.1.16: added *snp* parameter

Returns the nucleotidic sequence that the annotation covers. if the annotation's strand is -, and *reverse* is True, the reverse complement is returned.

Parameters

- **seq** (*seq*) – chromosome/contig sequence
- **reverse** (*bool*[218]) – if True and the strand is '-', a reverse complement is returned

[213] https://docs.python.org/3/library/stdtypes.html#str
[214] https://docs.python.org/3/library/functions.html#int
[215] https://docs.python.org/3/library/stdtypes.html#set
[216] https://docs.python.org/3/library/stdtypes.html#str
[217] https://docs.python.org/3/library/stdtypes.html#list
[218] https://docs.python.org/3/library/functions.html#bool

- **snp** (*tuple*[219]) – first element is the position of the SNP relative to the Annotation and the second element is the change

Returns nucleotide sequence with requested transformations

Return type str[220]

get_number_of_samples(*min_cov=4*)

New in version 0.1.13.

Returns the number of sample that have at least a minimum coverage of *min_cov*.

Parameters **min_cov** (*int*[221]) – minimum coverage

Return int number of samples passing the filter

Raises ***AttributeNotFound*** – if no sample coverage attribute is found

is_syn(*seq*, *pos*, *change*, *tbl=None*, *abs_pos=True*, *start=0*)

New in version 0.1.16.

Return if a SNP is synonymous or non-synonymous.

Parameters

- **seq** (*seq*) – reference sequence of the annotation
- **pos** (*int*[222]) – position of the SNP on the reference (1-based index)
- **change** (*str*[223]) – nucleotidic change
- **tbl** (*dict*[224]) – dictionary with the translation table. Defaults to the universal genetic code
- **abs_pos** (*bool*[225]) – if True the *pos* is referred to the reference and not a position relative to the annotation
- **start** (*int*[226] *or None*[227]) – phase to be used to get the start position of the codon. if None, the Annotation phase will be used

Returns True if the SNP is synonymous, false if it's non-synonymous

Return type bool[228]

length

Changed in version 0.2.0.

Length of the annotation, uses *len(self)*

phase

region

New in version 0.1.13.

Return the *region* covered by the annotation, to use in samtools

sample_coverage

New in version 0.1.13.

Returns a dictionary with the coverage for each sample, the returned dictionary has the sample id (stripped of the *_cov*) suffix and as values the coverage (converted via `int()`).

[219] https://docs.python.org/3/library/stdtypes.html#tuple
[220] https://docs.python.org/3/library/stdtypes.html#str
[221] https://docs.python.org/3/library/functions.html#int
[222] https://docs.python.org/3/library/functions.html#int
[223] https://docs.python.org/3/library/stdtypes.html#str
[224] https://docs.python.org/3/library/stdtypes.html#dict
[225] https://docs.python.org/3/library/functions.html#bool
[226] https://docs.python.org/3/library/functions.html#int
[227] https://docs.python.org/3/library/constants.html#None
[228] https://docs.python.org/3/library/functions.html#bool

Return dict dictionary with the samples' coverage

score

set_attr (*attr*, *value*)
New in version 0.1.13.

Generic method to set an attribute

set_mapping (*db*, *values*)
New in version 0.1.13.

Set mappings to a particular db, associated with the annotation.

Parameters

- **db** (*str*[229]) – database to which the mappings come from
- **mappings** (*iterable*) – iterable of mappings

source

taxon_db
db used for the taxon_id prediction

taxon_id
Changed in version 0.3.1: if taxon_id is set to "None" as a string, it's converted to *None*

taxon_id of the annotation

to_dict (*exclude_attr=None*)
New in version 0.3.1.

Return a dictionary representation of the Annotation.

Parameters **exclude_attr** (*str*[230], *list*[231]) – attributes to exclude from the dictionary, can be either a single attribute (string) or a list of strings

Returns dictionary with the annotation

Return type dict[232]

to_file (*file_handle*)
Writes the GFF annotation to *file_handle*

to_gff (*sep='='*)
Format the Annotation as a GFF string.

Parameters **sep** (*str*[233]) – separator key -> value

Returns annotation formatted as GFF

Return type str[234]

to_gtf (*gene_id_attr='uid'*, *sep=' '*)
New in version 0.1.15.

Changed in version 0.1.16: added *gene_id_attr* parameter

Changed in version 0.2.2: added *sep* argument, default to a space, now

Simple conversion to a valid GTF. gene_id and transcript_id are set to *uid* or the attribute specified using the *gene_id_attr* parameter. It's written to be used with *SNPDat*.

[229] https://docs.python.org/3/library/stdtypes.html#str
[230] https://docs.python.org/3/library/stdtypes.html#str
[231] https://docs.python.org/3/library/stdtypes.html#list
[232] https://docs.python.org/3/library/stdtypes.html#dict
[233] https://docs.python.org/3/library/stdtypes.html#str
[234] https://docs.python.org/3/library/stdtypes.html#str

to_json()
New in version 0.2.1.

Changed in version 0.3.1: now *Annotation.to_dict()* is used

Returns a json representation of the Annotation

to_mongodb(*lineage_func=None*, *indent=None*, *raw=False*)
New in version 0.2.1.

Changed in version 0.2.2: added handling of *counts_* and *fpkms_*

Changed in version 0.2.6: added *indent* parameter

Changed in version 0.3.4: added *raw*

Returns a MongoDB document that represent the Annotation.

Parameters

- **lineage** (*func*) – function used to populate the lineage key, returns a list of taxon_id
- **indent** (*int*[235]) – the amount of indent to put in the record, None (the default) is for the most compact - one line for the record
- **raw** (*bool*[236]) – if True, the method returns a string, which is the json dump, if False, the value returned is the dictionary

Returns the MongoDB document, with Annotation.uid as _id, as a string if *raw* is True, a dictionary if it is False

Return type str[237] or dict[238]

uid
New in version 0.1.13.

uid of the annotation

exception mgkit.io.gff.**AttributeNotFound**
Bases: Exception[239]

Raised if an attribute is not found in a GFF file

exception mgkit.io.gff.**DuplicateKeyError**
Bases: Exception[240]

New in version 0.1.12.

Raised if a GFF annotation contains duplicate keys

class mgkit.io.gff.**GenomicRange**(*seq_id='None'*, *start=1*, *end=1*, *strand='+'*)
Bases: object[241]

Defines a genomic range

Changed in version 0.2.1: using __slots__ for better memory usage

__contains__(*pos*)
Changed in version 0.2.3: a range or a subclass are accepted

New in version 0.1.16.

Tests if the position is inside the range of the GenomicRange

[235] https://docs.python.org/3/library/functions.html#int
[236] https://docs.python.org/3/library/functions.html#bool
[237] https://docs.python.org/3/library/stdtypes.html#str
[238] https://docs.python.org/3/library/stdtypes.html#dict
[239] https://docs.python.org/3/library/exceptions.html#Exception
[240] https://docs.python.org/3/library/exceptions.html#Exception
[241] https://docs.python.org/3/library/functions.html#object

Pos is 1-based as *GenomicRange.start* and *GenomicRange.end*

end

expand_from_list(*others*)
Expand the *GenomicRange* range instance with a list of *GenomicRange*

Parameters **others** (*iterable*) – iterable of *GenomicRange*

get_range()
New in version 0.1.13.

Returns the start and end position as a tuple

get_relative_pos(*pos*)
New in version 0.1.16.

Given an absolute position (referred to the reference), convert the position to a coordinate relative to the GenomicRange

Returns the position relative to the GenomicRange

Return type int[242]

Raises ValueError[243] – if the position is not in the range

intersect(*other*)
Return an instance of *GenomicRange* that represent the intersection of the current instance and another.

seq_id

start

strand

union(*other*)
Return the union of two *GenomicRange*

mgkit.io.gff.**annotate_sequence**(*name*, *seq*, *window=None*)

mgkit.io.gff.**annotation_coverage**(*annotations*, *seqs*, *strand=True*)
New in version 0.1.12.

Given a list of annotations and a dictionary where the keys are the sequence names referred in the annotations and the values are the sequences themselves, returns a number which indicated how much the sequence length is "covered" in annotations. If *strand* is True the coverage is strand specific.

Parameters

- **annotations** (*iterable*) – iterable of *Annotation* instances
- **seqs** (*dict*[244]) – dictionary in which the keys are the sequence names and the values are the sequences
- **strand** (*bool*[245]) – if True, the values are strand specific (the annotations) are grouped by (seq_id, strand) instead of seq_id

Yields *tuple* – the first element is the key, (seq_id, strand) if *strand* is True or seq_id if *strand* is False, and the coverage is the second value.

mgkit.io.gff.**annotation_coverage_sorted**(*annotations*, *seqs*, *strand=True*)
New in version 0.3.1.

Given a list of annotations and a dictionary where the keys are the sequence names referred in the annotations and the values are the sequences themselves, returns a number which indicated how much the sequence length is "covered" in annotations. If *strand* is True the coverage is strand specific.

[242] https://docs.python.org/3/library/functions.html#int
[243] https://docs.python.org/3/library/exceptions.html#ValueError
[244] https://docs.python.org/3/library/stdtypes.html#dict
[245] https://docs.python.org/3/library/functions.html#bool

Note: It differs from *annotation_coverage()* because it assumes the annotations are correctly sorted and in the values yielded

Parameters

- **annotations** (*iterable*) – iterable of *Annotation* instances
- **seqs** (*dict*[246]) – dictionary in which the keys are the sequence names and the values are the sequences
- **strand** (*bool*[247]) – if True, the values are strand specific (the annotations) are grouped by (seq_id, strand) instead of seq_id

Yields *tuple* – the first element is the seq_id, the second the strand (if strand is True, else it's set to *None*), and the third element is the coverage.

mgkit.io.gff.**annotation_elongation**(*ann1*, *annotations*)

New in version 0.1.12.

Given an *Annotation* instance and a list of the instances of the same class, returns the longest overlapping range that can be found and the annotations that are included in it.

Warning: annotations are not checked for seq_id and strand

Parameters

- **ann1** (Annotation) – annotation to elongate
- **annotations** (*iterable*) – iterable of *Annotation* instances

Returns the first element is the longest range found, while the the second element is a set with the annotations used

Return type tuple[248]

mgkit.io.gff.**convert_gff_to_gtf**(*file_in*, *file_out*, *gene_id_attr='uid'*)

New in version 0.1.16.

Function that uses *Annotation.to_gtf()* to convert a GFF into GTF.

Parameters

- **file_in** (*str*[249], *file*) – either file name or file handle of a GFF file
- **file_out** (*str*[250]) – file name to which write the converted annotations

mgkit.io.gff.**diff_gff**(*files*, *key_func=None*)

New in version 0.1.12.

Returns a simple diff made between a list of gff files. The annotations are grouped using *key_func*, so it depends on it to find similar annotations.

Parameters

- **files** (*iterable*) – an iterable of file handles, pointing to GFF files
- **key_func** (*func*) – function used to group annotations, defaults to this key: *(x.seq_id, x.strand, x.start, x.end, x.gene_id, x.bitscore)*

[246] https://docs.python.org/3/library/stdtypes.html#dict
[247] https://docs.python.org/3/library/functions.html#bool
[248] https://docs.python.org/3/library/stdtypes.html#tuple
[249] https://docs.python.org/3/library/stdtypes.html#str
[250] https://docs.python.org/3/library/stdtypes.html#str

Returns the returned dictionary keys are determined by key_func and as values lists. The lists elements are tuple whose first element is the index of the file, relative to *files* and the second element is the line number in which the annotation is. Can be used with the `linecache`[251] module.

Return type dict[252]

`mgkit.io.gff.`**`elongate_annotations`**(*annotations*)

New in version 0.1.12.

Given an iterable of *Annotation* instances, tries to find the all possible longest ranges and returns them.

> **Warning:** annotations are not checked for seq_id and strand

Parameters **`annotations`** (*iterable*) – iterable of *Annotation* instances

Returns set with the all ranges found

Return type set[253]

`mgkit.io.gff.`**`extract_nuc_seqs`**(*annotations*, *seqs*, *name_func=<function <lambda>>*, *reverse=False*)

New in version 0.1.13.

Extract the nucleotidic sequences from a list of annotations. Internally uses the method *Annotation.get_nuc_seq()*.

Parameters

- **`annotations`** (*iterable*) – iterable of *Annotation* instances
- **`seqs`** (*dict*[254]) – dictionary with the sequences referenced in the annotations
- **`name_func`** (*func*) – function used to extract the sequence name to be used, defaults to the uid of the annotation
- **`reverse`** (*bool*[255]) – if True the annotations on the - strand are reverse complemented

Yields *tuple* – tuple whose first element is the sequence name and the second is the sequence to which the annotation refers.

`mgkit.io.gff.`**`from_aa_blast_frag`**(*hit*, *parent_ann*, *aa_seqs*)

`mgkit.io.gff.`**`from_gff`**(*line*, *strict=True*, *encoding='ascii'*)

New in version 0.1.12.

Changed in version 0.2.6: added *strict* parameter

Changed in version 0.4.0: added *encoding* parameter

Parse GFF line and returns an *Annotation* instance

Parameters

- **`line`** (*str*[256]) – GFF line
- **`strict`** (*bool*[257]) – if True duplicate keys raise an exception

Returns instance of *Annotation* for the line

Return type *Annotation*

[251] https://docs.python.org/3/library/linecache.html#module-linecache
[252] https://docs.python.org/3/library/stdtypes.html#dict
[253] https://docs.python.org/3/library/stdtypes.html#set
[254] https://docs.python.org/3/library/stdtypes.html#dict
[255] https://docs.python.org/3/library/functions.html#bool
[256] https://docs.python.org/3/library/stdtypes.html#str
[257] https://docs.python.org/3/library/functions.html#bool

Raises *DuplicateKeyError* – if the attribute column has duplicate keys

mgkit.io.gff.**from_glimmer3**(*header*, *line*, *feat_type='CDS'*)

New in version 0.1.12.

Parses the line of a GLIMMER3 ouput and returns an instance of a GFF annotation.

Parameters

- **header** (*str*[258]) – the seq_id to which the ORF belongs
- **line** (*str*[259]) – the prediction line for the orf
- **feat_type** (*str*[260]) – the feature type to use

Returns instance of annotation

Return type *Annotation*

Example

Assuming a GLIMMER3 output like this:

```
>sequence0001
orf00001        66       611  +3      6.08
```

The code used is:

```
>>> header = 'sequence0001'
>>> line = 'orf00001        66       611  +3      6.08'
>>> from_glimmer3(header, line)
```

mgkit.io.gff.**from_hmmer**(*line*, *aa_seqs*, *feat_type='gene'*, *source='HMMER'*, *db='CUSTOM'*, *custom_profiles=True*, *noframe=False*)

New in version 0.1.15: first implementation to move old scripts to new GFF specs

Changed in version 0.2.1: removed compatibility with old scripts

Changed in version 0.2.2: taxon_id and taxon_name are not saved for non-custom profiles

Changed in version 0.3.1: added support for non mgkit-translated sequences (*noframe*)

Parse HMMER results (one line), it won't parse commented lines (starting with #)

Parameters

- **line** (*str*[261]) – HMMER domain table line
- **aa_seqs** (*dict*[262]) – dictionary with amino-acid sequences (name->seq), used to get the correct nucleotide positions
- **feat_type** (*str*[263]) – string to be used in the 'feature type' column
- **source** (*str*[264]) – string to be used in the 'source' column
- **custom_profiles** (*bool*[265]) – if True, the profile name contains gene, taxonomy and reviewed information in the form KOID_TAXONID_TAXON-NAME(-nr)
- **noframe** (*bool*[266]) – if True, the sequence is assumed to be in frame f0

[258] https://docs.python.org/3/library/stdtypes.html#str
[259] https://docs.python.org/3/library/stdtypes.html#str
[260] https://docs.python.org/3/library/stdtypes.html#str
[261] https://docs.python.org/3/library/stdtypes.html#str
[262] https://docs.python.org/3/library/stdtypes.html#dict
[263] https://docs.python.org/3/library/stdtypes.html#str
[264] https://docs.python.org/3/library/stdtypes.html#str
[265] https://docs.python.org/3/library/functions.html#bool
[266] https://docs.python.org/3/library/functions.html#bool

Returns A *Annotation* instance

Note: if *custom_profiles* is False, gene_id, taxon_id and taxon_name will be equal to the profile name

mgkit.io.gff.**from_json**(*line*)
New in version 0.2.1.

Returns an Annotation from a json representation

mgkit.io.gff.**from_mongodb**(*record*, *lineage=True*)
New in version 0.2.1.

Changed in version 0.2.2: added handling of *counts_* and *fpkms_*

Changed in version 0.2.6: better handling of missing attributes and added *lineage* parameter

Returns a *Annotation* instance from a MongoDB record (created) using *Annotation.to_mongodb()*. The actual record returned by pymongo is a dictionary that is copied, manipulated and passed to the `Annotation.__init__()`.

Parameters

- **record** (*dict*[267]) – a dictionary with the full record from a MongoDB query
- **lineage** (*bool*[268]) – indicates if the lineage information in the record should be kept in the annotation

Returns instance of *Annotation* object

Return type *Annotation*

mgkit.io.gff.**from_nuc_blast**(*hit*, *db*, *feat_type='CDS'*, *seq_len=None*, *to_nuc=False*, ***kwd*)
New in version 0.1.12.

Changed in version 0.1.16: added *to_nuc* parameter

Changed in version 0.2.3: removed *to_nuc*, the hit can include the subject end/start and evalue

Returns an instance of *Annotation*

Parameters

- **hit** (*tuple*[269]) – a BLAST hit, from *mgkit.io.blast.parse_blast_tab()*
- **db** (*str*[270]) – db used with BLAST

Keyword Arguments

- **feat_type** (*str*[271]) – feature type in the GFF
- **seq_len** (*int*[272]) – sequence length, if supplied, the phase for strand '-' can be assigned, otherwise is assigned a 0
- ****kwd** – any additional column

Returns instance of *Annotation*

Return type *Annotation*

mgkit.io.gff.**from_nuc_blast_frag**(*hit*, *parent_ann*, *db='NCBI-NT'*)

[267] https://docs.python.org/3/library/stdtypes.html#dict
[268] https://docs.python.org/3/library/functions.html#bool
[269] https://docs.python.org/3/library/stdtypes.html#tuple
[270] https://docs.python.org/3/library/stdtypes.html#str
[271] https://docs.python.org/3/library/stdtypes.html#str
[272] https://docs.python.org/3/library/functions.html#int

mgkit.io.gff.**from_prodigal_frag**(*main_gff*, *blast_gff*, *attr='ID'*, *split_func=None*)

Changed in version 0.3.3: fixed a bug for the strand, also the code is tested

New in version 0.2.6: *experimental*

Reads the GFF given in output by PRODIGAL and the resulting GFF from using BLAST (or other software) on the aa or nucleotide file output by PRODIGAL.

It then integrates the two outputs, so to the PRODIGAL GFF is added the information from the the output of the gene prediction software used.

Parameters

- **main_gff** (*file*) – GFF file from PRODIGAL
- **blast_gff** (*file*) – GFF with the returned annotations
- **attr** (*str*[273]) – attribute in the PRODIGAL GFF that is used to identify an annotation
- **split_func** (*func*) – function to rename the headers from the predicted sequences back to their parent sequence

Yields *annotation* – annotation for each *blast_gff* back translated

mgkit.io.gff.**from_sequence**(*name*, *seq*, *feat_type='SEQUENCE'*, ***kwd*)

New in version 0.1.12.

Returns an instance of *Annotation* for the full length of a sequence

Parameters

- **name** (*str*[274]) – name of the sequence
- **seq** (*str*[275]) – sequence, to get the length of the annotation

Keyword Arguments

- **feat_type** (*str*[276]) – feature type in the GFF
- ****kwd** – any additional column

Returns instance of *Annotation*

Return type *Annotation*

mgkit.io.gff.**get_annotation_map**(*annotations*, *key_func*, *value_func*)

New in version 0.1.15.

Applies two functions to an iterable of annotations with an iterator returned with the applied functions. Useful to build a dictionary

Parameters

- **annotations** (*iterable*) – iterable of annotations
- **key_func** (*func*) – function that accept an annotation as argument and returns one value, the first of the returned tuple
- **value_func** (*func*) – function that accept an annotation as argument and returns one value, the second of the returned tuple

Yields *tuple* – a tuple where the first value is the result of *key_func* on the passed annotation and the second is the value returned by *value_func* on the same annotation

mgkit.io.gff.**group_annotations**(*annotations*, *key_func=<function <lambda>>*)

New in version 0.1.12.

[273] https://docs.python.org/3/library/stdtypes.html#str
[274] https://docs.python.org/3/library/stdtypes.html#str
[275] https://docs.python.org/3/library/stdtypes.html#str
[276] https://docs.python.org/3/library/stdtypes.html#str

Group *Annotation* instances in a dictionary by using a key function that returns the key to be used in the dictionary.

Parameters

- **annotations** (*iterable*) – iterable with *Annotation* instances
- **key_func** (*func*) – function used to extract the key used in the dictionary, defaults to a function that returns (ann.seq_id, ann.strand)

Returns dictionary whose keys are returned by *key_func* and the values are lists of annotations

Return type dict[277]

Example

```
>>> ann = [Annotation(seq_id='seq1', strand='+', start=10, end=15),
... Annotation(seq_id='seq1', strand='+', start=1, end=5),
... Annotation(seq_id='seq1', strand='-', start=30, end=100)]
>>> group_annotations(ann)
{('seq1', '+'): [seq1(+):10-15, seq1(+):1-5], ('seq1', '-'): [seq1(-):30-100]}
```

mgkit.io.gff.**group_annotations_by_ancestor**(*annotations*, *ancestors*, *taxonomy*)

New in version 0.1.13.

Group annotations by the ancestors provided.

Parameters

- **annotations** (*iterable*) – annotations to group
- **ancestors** (*iterable*) – list of ancestors accepted
- **taxonomy** – taxonomy class

Returns grouped annotations

Return type dict[278]

mgkit.io.gff.**group_annotations_sorted**(*annotations*, *key_func=<function <lambda>>*)

New in version 0.1.13.

Group *Annotation* instances by using a key function that returns a key. Assumes that the annotations are already sorted to return an iterator and save memory. One way to sort them is using: *sort -s -k 1,1 -k 7,7* on the file.

Parameters

- **annotations** (*iterable*) – iterable with *Annotation* instances
- **key_func** (*func*) – function used to extract the key used in the dictionary, defaults to a function that returns (ann.seq_id, ann.strand)

Yields *list* – a list of the grouped annotations by *key_func* values

mgkit.io.gff.**load_gff_base_info**(*files*, *taxonomy=None*, *exclude_ids=None*, *include_taxa=None*, *encoding='ascii'*)

This function is useful if the number of annotations in a GFF is high or there are memory constraints on the system. It returns a dictionary that can be used with functions like *mgkit.counts.func.load_sample_counts()*.

Parameters

- **files** (*iterable, str*[279]) – file name or list of paths of GFF files
- **taxonomy** – taxonomy pickle file, needed if include_taxa is not None

[277] https://docs.python.org/3/library/stdtypes.html#dict
[278] https://docs.python.org/3/library/stdtypes.html#dict
[279] https://docs.python.org/3/library/stdtypes.html#str

- **exclude_ids** (*set*[280], *list*[281]) – a list of gene_id to exclude from the dictionary
- **include_taxa** (*int*[282], *iterable*) – a taxon_id or list thereof to be passed to `mgkit.taxon.taxonomy.is_ancestor()`, so only the taxa that have the those taxon_id(s) as ancestor(s) are kept
- **encoding** (*str*[283]) – passed to *parse_gff()*

Returns dictionary where the key is *Annotation.uid* and the value is a tuple (*Annotation.gene_id*, *Annotation.taxon_id*)

Return type dict[284]

mgkit.io.gff.**load_gff_mappings**(*files*, *map_db*, *taxonomy=None*, *exclude_ids=None*, *include_taxa=None*, *encoding='ascii'*)

This function is useful if the number of annotations in a GFF is high or there are memory constraints on the system. It returns a dictionary that can be used with functions like *mgkit.counts.func.load_sample_counts()*.

Parameters

- **files** (*iterable*, *str*[285]) – file name or list of paths of GFF files
- **map_db** (*str*[286]) – any kind mapping in the GFF, as passed to *Annotation.get_mapping()*
- **taxonomy** – taxonomy pickle file, needed if include_taxa is not None
- **exclude_ids** (*set*[287], *list*[288]) – a list of gene_id to exclude from the dictionary
- **include_taxa** (*int*[289], *iterable*) – a taxon_id or list thereof to be passed to `mgkit.taxon.taxonomy.is_ancestor()`, so only the taxa that have the those taxon_id(s) as ancestor(s) are kept
- **encoding** (*str*[290]) – passed to *parse_gff()*

Returns dictionary where the key is *Annotation.gene_id* and the value is a list of mappings, as returned by *Annotation.get_mapping()*

Return type dict[291]

mgkit.io.gff.**parse_gff**(*file_handle*, *gff_type=<function from_gff>*, *strict=True*, *encoding='ascii'*)

Changed in version 0.4.0: In some cases ASCII decoding is not enough, so it is parametrised now

Changed in version 0.3.4: added decoding from binary for compatibility with Python3

Changed in version 0.2.6: added *strict* parameter

Changed in version 0.2.3: correctly handling of GFF with comments of appended sequences

Changed in version 0.1.12: added *gff_type* parameter

Parse a GFF file and returns generator of `GFFKegg` instances

Accepts a file handle or a string with the file name

Parameters

[280] https://docs.python.org/3/library/stdtypes.html#set
[281] https://docs.python.org/3/library/stdtypes.html#list
[282] https://docs.python.org/3/library/functions.html#int
[283] https://docs.python.org/3/library/stdtypes.html#str
[284] https://docs.python.org/3/library/stdtypes.html#dict
[285] https://docs.python.org/3/library/stdtypes.html#str
[286] https://docs.python.org/3/library/stdtypes.html#str
[287] https://docs.python.org/3/library/stdtypes.html#set
[288] https://docs.python.org/3/library/stdtypes.html#list
[289] https://docs.python.org/3/library/functions.html#int
[290] https://docs.python.org/3/library/stdtypes.html#str
[291] https://docs.python.org/3/library/stdtypes.html#dict

- **file_handle** (*str*[292], *file*) – file name or file handle to read from
- **gff_type** (*class*) – class/function used to parse a GFF annotation
- **strict** (*bool*[293]) – if True duplicate keys raise an exception
- **encoding** (*str*[294]) – encoding of the file, if ascii fails, use utf8

Yields *Annotation* – an iterator of *Annotation* instances

mgkit.io.gff.**parse_gff_files**(*files*, *strict=True*)

New in version 0.1.15.

Changed in version 0.2.6: added *strict* parameter

Function that returns an iterator of annotations from multiple GFF files.

Parameters

- **files** (*iterable*, *str*[295]) – iterable of file names of GFF files, or a single file name
- **strict** (*bool*[296]) – if True duplicate keys raise an exception

Yields *Annotation* – iterator of annotations

mgkit.io.gff.**split_gff_file**(*file_handle*, *name_mask*, *num_files=2*, *encoding='ascii'*)

New in version 0.1.14.

Changed in version 0.2.6: now accept a file object as sole input

Changed in version 0.4.0: added *encoding* parameter

Splits a GFF, or a list of them, into a number of files. It is assured that annotations for the same sequence are kept in the same file, which is useful for cases like filtering, even when the annotations are from different GFF files.

Internally, a structure is kept to check if a sequence ID is already been stored to a file, in which case the annotation is written to that file, otherwise a random file handles (among the open ones) is chosen.

Parameters

- **file_handle** (*str*[297], *list*[298]) – a single or list of file handles (or file names), from which the GFF annotations are read
- **name_mask** (*str*[299]) – a string used as template for the output file names on which the function applies `string.format()`
- **num_files** (*int*[300]) – the number of files to split the records

Example

```
>>> import glob
>>> files = glob.glob('*.gff')
>>> name_mask = 'split-file-{0}.gff'
>>> split_gff_file(files, name_mask, 5)
```

[292] https://docs.python.org/3/library/stdtypes.html#str
[293] https://docs.python.org/3/library/functions.html#bool
[294] https://docs.python.org/3/library/stdtypes.html#str
[295] https://docs.python.org/3/library/stdtypes.html#str
[296] https://docs.python.org/3/library/functions.html#bool
[297] https://docs.python.org/3/library/stdtypes.html#str
[298] https://docs.python.org/3/library/stdtypes.html#list
[299] https://docs.python.org/3/library/stdtypes.html#str
[300] https://docs.python.org/3/library/functions.html#int

mgkit.io.gff.**write_gff**(*annotations*, *file_handle*, *verbose=True*)

Changed in version 0.1.12: added *verbose* argument

Write a GFF to file

Parameters

- **annotations** (*iterable*) – iterable that returns GFFKegg or *Annotation* instances
- **file_handle** (*str*[301], *file*) – file name or file handle to write to
- **verbose** (*bool*[302]) – if True, a message is logged

mgkit.io.glimmer module

mgkit.io.glimmer.**parse_glimmer3**(*file_handle*)

Parses an ouput file from glimmer3 and yields the header and prediction lines. Used to feed the *mgkit.io.gff.from_glimmer3()* function.

Parameters **file_handle** (*str*[303], *file*) – file name or file handle to read from

Yields *tuple* – first element is the sequence of the predicted gene and the second is the prediction line

mgkit.io.snpdat module

SNPDat reader

class mgkit.io.snpdat.**SNPDatRow**(*line=None*, *rev_comp=None*)

Bases: object[304]

Class containing information ouputted by SNPDat in its result file. One instance contains information about a row in the file.

chr_name
the queried SNPs chromosome ID

Type str[305]

chr_pos
queried SNPs genomic location

Type int[306]

in_feat
Whether or not the queried SNP was within a feature

Type bool[307]

region
Region containing the SNP; either exonic, intronic or intergenic

Type str[308]

feat_dist
Distance to nearest feature

[301] https://docs.python.org/3/library/stdtypes.html#str
[302] https://docs.python.org/3/library/functions.html#bool
[303] https://docs.python.org/3/library/stdtypes.html#str
[304] https://docs.python.org/3/library/functions.html#object
[305] https://docs.python.org/3/library/stdtypes.html#str
[306] https://docs.python.org/3/library/functions.html#int
[307] https://docs.python.org/3/library/functions.html#bool
[308] https://docs.python.org/3/library/stdtypes.html#str

Type int[309]

feature
Either the closest feature to the SNP or the feature containing the SNP

Type str[310]

num_features
number of different features that the SNP is annotated to

Type int[311]

feat_num
number of annotations of the current feature

Type int[312]

feat_start
Start of feature (bp)

Type int[313]

feat_end
End of feature (bp)

Type int[314]

gene_id
gene ID for the current feature

Type str[315]

gene_name
gene name for the current feature

Type str[316]

transcript_id
transcript ID for the current feature

Type str[317]

transcript_name
transcript name for the current feature

Type str[318]

exon
exon that contains the current feature and the total number of annotated exons for the gene containing the feature

Type tuple[319]

strand
strand sense of the feature

Type str[320]

[309] https://docs.python.org/3/library/functions.html#int
[310] https://docs.python.org/3/library/stdtypes.html#str
[311] https://docs.python.org/3/library/functions.html#int
[312] https://docs.python.org/3/library/functions.html#int
[313] https://docs.python.org/3/library/functions.html#int
[314] https://docs.python.org/3/library/functions.html#int
[315] https://docs.python.org/3/library/stdtypes.html#str
[316] https://docs.python.org/3/library/stdtypes.html#str
[317] https://docs.python.org/3/library/stdtypes.html#str
[318] https://docs.python.org/3/library/stdtypes.html#str
[319] https://docs.python.org/3/library/stdtypes.html#tuple
[320] https://docs.python.org/3/library/stdtypes.html#str

ann_frame
annotated reading frame (when contained in the GTF)

Type str[321]

frame
reading frame estimated by SNPdat

Type str[322]

num_stops
estimated number of stop codons in the estimated reading frame

Type int[323]

codon
codon containing the SNP, position in the codon and reference base and mutation

Type tuple[324]

nuc_change
amino acid for the reference codon and new amino acid with the mutation in place

Type tuple[325]

nuc_ref
reference nucleotide

Type str[326], None[327]

aa_change
amino acid for the reference codon and new amino acid with the mutation in place

Type str[328]

synonymous
Whether or not the mutation is synonymous

Type bool[329]

protein_id
protein ID for the current feature

Type str[330]

messages
messages in the SNPDat line

Type str[331]

aa_change

ann_frame

chr_name

chr_pos

codon

exon

[321] https://docs.python.org/3/library/stdtypes.html#str
[322] https://docs.python.org/3/library/stdtypes.html#str
[323] https://docs.python.org/3/library/functions.html#int
[324] https://docs.python.org/3/library/stdtypes.html#tuple
[325] https://docs.python.org/3/library/stdtypes.html#tuple
[326] https://docs.python.org/3/library/stdtypes.html#str
[327] https://docs.python.org/3/library/constants.html#None
[328] https://docs.python.org/3/library/stdtypes.html#str
[329] https://docs.python.org/3/library/functions.html#bool
[330] https://docs.python.org/3/library/stdtypes.html#str
[331] https://docs.python.org/3/library/stdtypes.html#str

feat_dist

feat_end

feat_num

feat_start

feature

frame

gene_id

gene_name

in_feat

messages

nuc_change

nuc_ref

num_features

num_stops

protein_id

region

strand

synonymous

transcript_id

transcript_name

mgkit.io.snpdat.**snpdat_reader**(*f_handle*)
Simple SNPDat reader.

f_handle: file handle or string for the SNPDat result file

Returns generator of SNPDatRow instances

mgkit.io.uniprot module

New in version 0.1.13.

Uniprot file formats

mgkit.io.uniprot.**MAPPINGS = {'biocyc': 'BioCyc', 'eggnog': 'eggNOG', 'embl': 'EMBL',**
Some of the mappings contained in the idmapping.dat.gz

mgkit.io.uniprot.**parse_uniprot_mappings**(*file_handle*, *gene_ids=None*, *mappings=None*, *num_lines=10000000*)
Parses a Uniprot mapping file[332], returning a generator with the mappings.

Parameters

- **file_handle** (*str*[333], *file*) – file name or open file handle
- **gene_ids** (*None*[334], *set*[335]) – if not None, the returned mappings are for the gene IDs specified

[332] ftp://ftp.uniprot.org/pub/databases/uniprot/current_release/knowledgebase/idmapping/idmapping.dat.gz
[333] https://docs.python.org/3/library/stdtypes.html#str
[334] https://docs.python.org/3/library/constants.html#None
[335] https://docs.python.org/3/library/stdtypes.html#set

- **mappings** (*None*[336], *set*[337]) – mappings to be returned
- **num_lines** (*None*[338], *int*[339]) – number of which a message is logged. If None, no message is logged

Yields *tuple* – the first element is the gene ID, the second is the mapping type and third element is the mapped ID

mgkit.io.uniprot.**uniprot_mappings_to_dict**(*file_handle*, *gene_ids*, *mappings*, *num_lines=None*)

Changed in version 0.3.4: added *num_lines*

Parses a Uniprot mapping file[340], returning a generator of dictionaries with the mappings requested.

Parameters

- **file_handle** (*str*[341], *file*) – file name or open file handle
- **gene_ids** (*None*[342], *set*[343]) – if not None, the returned mappings are for the gene IDs specified
- **mappings** (*None*[344], *set*[345]) – mappings to be returned
- **num_lines** (*int*[346], *None*[347]) – passed to *parse_uniprot_mappings()*

Yields *tuple* – the first element is the gene ID, the second is a dictionary with all the mappings found, the key is the mapping type and the value is a list of all mapped IDs

mgkit.io.utils module

Various utilities to help read and process files

exception mgkit.io.utils.**UnsupportedFormat**

Bases: OSError[348]

Raised if the a file can't be opened with the correct module

mgkit.io.utils.**compressed_handle**(*file_handle*)

New in version 0.1.13.

Tries to wrap a file handle in the appropriate compressed file class.

Parameters **file_handle** (*str*[349]) – file handle

Returns the same file handle if no suitable compressed file class is found or the new file_handle which supports the compression

Return type file

Raises *UnsupportedFormat* – if the module to open the file is not available

mgkit.io.utils.**group_tuples_by_key**(*iterator*, *key_func=None*, *skip_elements=0*)

New in version 0.3.1.

[336] https://docs.python.org/3/library/constants.html#None
[337] https://docs.python.org/3/library/stdtypes.html#set
[338] https://docs.python.org/3/library/constants.html#None
[339] https://docs.python.org/3/library/functions.html#int
[340] ftp://ftp.uniprot.org/pub/databases/uniprot/current_release/knowledgebase/idmapping/idmapping.dat.gz
[341] https://docs.python.org/3/library/stdtypes.html#str
[342] https://docs.python.org/3/library/constants.html#None
[343] https://docs.python.org/3/library/stdtypes.html#set
[344] https://docs.python.org/3/library/constants.html#None
[345] https://docs.python.org/3/library/stdtypes.html#set
[346] https://docs.python.org/3/library/functions.html#int
[347] https://docs.python.org/3/library/constants.html#None
[348] https://docs.python.org/3/library/exceptions.html#OSError
[349] https://docs.python.org/3/library/stdtypes.html#str

Group the elements of an iterator by a key and yields the grouped elements. The elements yielded by the iterator are assumed to be a list or tuple, with the default key (when *key_func* is None) being the first of the of the objects inside that element. This behaviour can be customised by passing to *key_func* a function that accept an element and returns the key to be used.

Note: the iterable assumen that the elements are already sorted by their keys

---

Parameters

- **iterator** (*iterable*) – iterator to be grouped
- **key_func** (*func*) – function that accepts a element and returns its associated key
- **skip_elements** (*int*[350]) – number of elements to skip at the start

Yields *list* – a list of the grouped elements by key

mgkit.io.utils.**open_file**(*file_name*, *mode='r'*)

New in version 0.1.12.

Changed in version 0.3.4: using *io.open*, always in binary mode

Opens a file using the extension as a guide to which module to use.

Note: Unicode makes for a slower *.translate* method in Python2, so it's best to use the *open* builtin.

Parameters

- **file_name** (*str*[351]) – file name
- **mode** (*str*[352]) – mode used to open the file

Returns file handle

Return type file

Raises *UnsupportedFormat* – if the module to open the file is not available

mgkit.io.utils.**split_write**(*records*, *name_mask*, *write_func*, *num_files=2*)

New in version 0.1.13.

Splits the writing of a number of records in a series of files. The *name_mask* is used as template for the file names. A string like "split-files-{0}" can be specified and the function applies format with the index of the pieces.

Parameters

- **records** (*iterable*) – an iterable that returns a object to be saved
- **name_mask** (*str*[353]) – a string used as template for the output file names on which the function applies `string.format()`
- **write_func** (*func*) – a function that is called to write to the files. It needs to accept a file handles as first argument and the record returned by *records* as the second argument
- **num_files** (*int*[354]) – the number of files to split the records

[350] https://docs.python.org/3/library/functions.html#int
[351] https://docs.python.org/3/library/stdtypes.html#str
[352] https://docs.python.org/3/library/stdtypes.html#str
[353] https://docs.python.org/3/library/stdtypes.html#str
[354] https://docs.python.org/3/library/functions.html#int

Module contents

Package used to contain code related to I/O operations

mgkit.mappings package

Submodules

mgkit.mappings.cazy module

Module containing classes and functions to deal with CaZy data

mgkit.mappings.cazy.**CAZY_FAMILIES = {'CBM': 'Carbohydrate-Binding Module', 'CE': 'Carboh**
CaZy families

mgkit.mappings.eggnog module

Module containing classes and functions to deal with eggNOG data

---

Todo:

- unify download of data from web

---

mgkit.mappings.eggnog.**EGGNOG_CAT = {'A': 'RNA processing and modification', 'B': 'Chroma**
Single letter functional categories

mgkit.mappings.eggnog.**EGGNOG_CAT_KEYS = (('J', 'A', 'K', 'L', 'B'), ('D', 'Y', 'V', 'T',**
Used to build map of broader categories (*EGGNOG_CAT_NAMES*) to more specific ones

mgkit.mappings.eggnog.**EGGNOG_CAT_MAP = {'CELLULAR PROCESSES AND SIGNALING': ('D', 'Y', '**
Functional categories (broader, *EGGNOG_CAT_NAMES*) mappings to more specific one (*EGGNOG_CAT*).

mgkit.mappings.eggnog.**EGGNOG_CAT_NAMES = ('INFORMATION STORAGE AND PROCESSING', 'CELLULA**
Functional categories (broader)

class mgkit.mappings.eggnog.**NOGInfo**(*members=None*, *funccat=None*, *description=None*)
Bases: object[355]

New in version 0.1.14.

Changed in version 0.4.0: made file reading compatible with Python 3

Mappings from Uniprot to eggNOG

..note:

```
load_description is optional
```

get_gene_funccat(*gene_id*)
Returns the functional category (one letter, *EGGNOG_CAT* keys) for the requested eggNOG gene ID

get_gene_nog(*gene_id*)
Returns the COG/NOG ID of the requested eggNOG gene ID

get_nog_funccat(*nog_id*)
Returns the functional category (one letter, *EGGNOG_CAT* keys) for the requested eggNOG COG/NOG ID

[355] https://docs.python.org/3/library/functions.html#object

get_nog_gencat(*nog_id*)
Returns the functional category (*EGGNOG_CAT_NAMES* keys) for the requested eggNOG COG/NOG IDs

get_nogs_funccat(*nog_ids*)
Returns the functional categories for a list of COG/NOG IDs. Uses *NOGInfo.get_nog_funccat()*

load_description(*file_handle*)
Loads data from *NOG.description.txt.gz*

file_handle can either an open file or a path

load_funccat(*file_handle*)
Loads data from *NOG.funccat.txt.gz*

file_handle can either an open file or a path

load_members(*file_handle*)
Loads data from *NOG.members.txt.gz*

file_handle can either an open file or a path

mgkit.mappings.eggnog.**get_general_eggnog_cat**(*category*)
New in version 0.1.14.

Returns the functional category (*EGGNOG_CAT_NAMES* keys) for the requested single letter functional category (*EGGNOG_CAT* keys)

mgkit.mappings.enzyme module

New in version 0.1.14.

EC mappings

mgkit.mappings.enzyme.**ENZCLASS_REGEX = '^(\\d)\\. ?([\\d-]+)\\. ?([\\d-]+)\\. ?([\\d-]+)**
Used to get the description for the higher level enzyme classes from the file *enzclass.txt* on expasy[356]

mgkit.mappings.enzyme.**LEVEL1_NAMES = {1: 'oxidoreductases', 2: 'transferases', 3: 'hy**
Top level classification names

mgkit.mappings.enzyme.**change_mapping_level**(*ec_map*, *level=3*)
New in version 0.1.14.

Given a dictionary, whose values are dictionaries, in which a key is named *ec* and its value is an iterable of EC numbers, returns an iterator that can be used to build a dictionary with the same top level keys and the values are sets of the transformed EC numbers.

Parameters

- **ec_map** (*dict*[357]) – dictionary generated by *mgkit.net.uniprot.get_gene_info()*
- **level** (*int*[358]) – number from 1 to 4, to specify the level of the mapping, passed to *get_enzyme_level()*

Yields *tuple* – a tuple (gene_id, set(ECs)), which can be passed to *dict* to make a dictionary

Example

[356] http://expasy.org
[357] https://docs.python.org/3/library/stdtypes.html#dict
[358] https://docs.python.org/3/library/functions.html#int

```
>>> from mgkit.net.uniprot import get_gene_info
>>> from mgkit.mappings.enzyme import change_mapping_level
>>> ec_map = get_gene_info('Q9HFQ1', columns='ec')
{'Q9HFQ1': {'ec': '1.1.3.4'}}
>>> dict(change_mapping_level(ec_map, level=2))
{'Q9HFQ1': {'1.1'}}
```

mgkit.mappings.enzyme.**get_enzyme_full_name**(*ec_id*, *ec_names*, *sep=', '*)

New in version 0.2.1.

From a EC identifiers and a dictionary of names builds a comma separated name (by default) that identifies the function of the enzyme.

Parameters

- **ec_id** (*str*[359]) – EC identifier
- **ec_names** (*dict*[360]) – a dictionary of names that can be produced using *parse_expasy_file()*
- **sep** (*str*[361]) – string used to join the names

Returns the enzyme classification name

Return type str[362]

mgkit.mappings.enzyme.**get_enzyme_level**(*ec*, *level=4*)

New in version 0.1.14.

Returns an enzyme class at a specific level , between 1 and 4 (by default the most specific, 4)

Parameters

- **ec** (*str*[363]) – a string representing an EC number (e.g. 1.2.4.10)
- **level** (*int*[364]) – from 1 to 4, to get a different level specificity of in the enzyme classification

Returns the EC number at the requested specificity

Return type str[365]

Example

```
>>> from mgkit.mappings.enzyme import get_enzyme_level
>>> get_enzyme_level('1.1.3.4', 1)
'1'
>>> get_enzyme_level('1.1.3.4', 2)
'1.1'
>>> get_enzyme_level('1.1.3.4', 3)
'1.1.3'
>>> get_enzyme_level('1.1.3.4', 4)
'1.1.3.4'
```

mgkit.mappings.enzyme.**get_mapping_level**(*ec_map*, *level=3*)

New in version 0.3.0.

Given a dictionary, whose values are iterable of EC numbers, returns an iterator that can be used to build a dictionary with the same top level keys and the values are sets of the transformed EC numbers.

[359] https://docs.python.org/3/library/stdtypes.html#str
[360] https://docs.python.org/3/library/stdtypes.html#dict
[361] https://docs.python.org/3/library/stdtypes.html#str
[362] https://docs.python.org/3/library/stdtypes.html#str
[363] https://docs.python.org/3/library/stdtypes.html#str
[364] https://docs.python.org/3/library/functions.html#int
[365] https://docs.python.org/3/library/stdtypes.html#str

Parameters

- **ec_map** (*dict*[366]) – dictionary genes to EC
- **level** (*int*[367]) – number from 1 to 4, to specify the level of the mapping, passed to *get_enzyme_level()*

Yields *tuple* – a tuple (gene_id, set(ECs)), which can be passed to *dict* to make a dictionary

mgkit.mappings.enzyme.**parse_expasy_file**(*file_name*)

Used to load enzyme descriptions from the file *enzclass.txt* on expasy[368].

The FTP url for enzclass.txt is: ftp://ftp.expasy.org/databases/enzyme/enzclass.txt

mgkit.mappings.go module

Module containing classes and functions to deal with Gene Ontology data

mgkit.mappings.pandas_map module

Module that contains mapping operations on pandas data structures

mgkit.mappings.pandas_map.**calc_coefficient_of_variation**(*dataframe*)

Calculate coefficient of variation for a DataFrame. Uses formula from Wikipedia[369]

The formula used is $\left(1 + \frac{1}{4n}\right) * c_v$ where $c_v = \frac{s}{\bar{x}}$

mgkit.mappings.pandas_map.**concatenate_and_rename_tables**(*dataframes*, *roots*)

Concatenates a list of `pandas.DataFrame` instances and renames the columns prepending a string to each column in each table from a list of prefixes.

Parameters

- **dataframes** (*iterable*) – iterable of DataFrame instances
- **roots** (*iterable*) – list of prefixes to append to the column names of each DataFrame

Return DataFrame returns a DataFrame instance

Todo:

- move to pandas_utils?

---

mgkit.mappings.pandas_map.**group_dataframe_by_mapping**(*dataframe*, *mapping*, *root_taxon*, *name_dict=None*)

Return a `pandas.DataFrame` filtered by mapping and root taxon, the values for each column is averaged over all genes mapping to a category.

Parameters

- **dataframe** (*DataFrame*) – DataFrame with multindex gene-root
- **mapping** (*dict*[370]) – dictionary of category->genes
- **root_taxon** (*str*[371]) – root taxon to group genes

[366] https://docs.python.org/3/library/stdtypes.html#dict
[367] https://docs.python.org/3/library/functions.html#int
[368] http://expasy.org
[369] http://en.wikipedia.org/wiki/Coefficient_of_variation
[370] https://docs.python.org/3/library/stdtypes.html#dict
[371] https://docs.python.org/3/library/stdtypes.html#str

- **name_dict** (*dict*[372]) – dictionary of category->name

Return DataFrame DataFrame filtered

mgkit.mappings.pandas_map.**make_stat_table**(*dataframes*, *roots*)

Produces a pandas.DataFrame that summarise the supplied DataFrames. The stats include mean, stdev and coefficient of variation for each root taxon.

Parameters

- **dataframes** (*iterable*) – iterable of DataFrame instances
- **roots** (*iterable*) – list of root taxa to which each table belongs

Return DataFrame returns a DataFrame instance

mgkit.mappings.taxon module

Module used to map taxon_id to different levels in the taxonomy.

mgkit.mappings.taxon.**map_taxon_by_id_list**(*taxon_id*, *map_ids*, *func*)

Maps a taxon_id to a list of taxon IDs, using the function supplied.

Parameters

- **taxon_id** (*int*[373]) – taxon ID to map
- **map_ids** (*iterable*) – list of taxon IDs to which the taxon_id will be mapped.
- **func** (*func*) – function used to map the IDs, accepts two taxon IDs

Results:

generator: generator expression of all IDs in map_ids to which taxon_id can be mapped.

Example

If mapping a taxon (Prevotella ruminicola) to Prevotella or Clostridium, using as *func* *mgkit.taxon.is_ancestor()* and taxonomy is an instance of *mgkit.taxon.Taxonomy*.

```
>>> import functools
>>> from mgkit.taxon import is_ancestor
>>> func = functools.partial(is_ancestor, taxonomy)
>>> list(map_taxon_by_id_list(839, [838, 1485], func))
[838]
```

mgkit.mappings.utils module

Utilities to map genes

mgkit.mappings.utils.**count_genes_in_mapping**(*gene_lists*, *labels*, *mapping*, *normalise=False*)

Maps lists of ids to a mapping dictionary, returning a pandas.DataFrame in which the rows are the labels provided and the columns the categories to which the ids map. Each element of the matrix label-category is the sum of all ids in the relative gene list that maps to the specific category.

Parameters

- **gene_lists** (*iterable*) – an iterable in which each element is a iterable of ids that can be mapped to mapping

[372] https://docs.python.org/3/library/stdtypes.html#dict
[373] https://docs.python.org/3/library/functions.html#int

- **labels** (*iterable*) – an iterable of strings that defines the labels to be used in the resulting rows in the `pandas.DataFrame`; must have the same length as gene_lists
- **mapping** (*dict*[374]) – a dictionary in the form: gene_id->[cat1, cat2, .., catN]
- **normalise** (*bool*[375]) – if True the counts are normalised over the total for each row.

Returns a `pandas.DataFrame` instance

`mgkit.mappings.utils.`**group_annotation_by_mapping**(*annotations*, *mapping*, *attr='ko'*)

Group annotations by mapping dictionary

Parameters

- **annotations** (*iterable*) – iterable of `gff.GFFKeg` instances
- **mapping** (*dict*[376]) – dictionary with mappings for the attribute requested
- **attr** (*str*[377]) – attribute of the annotation to be used as key in mapping

Return dict dictionary category->annotations

Module contents

mgkit.net package

Submodules

mgkit.net.embl module

Access EMBL Services

`mgkit.net.embl.`**EMBL_DBID = 'embl_cds'**
Default database id

exception `mgkit.net.embl.`**EntryNotFound**
Bases: `Exception`[378]

Raised if at least one entry was not found by *get_sequences_by_ids()*. *NOT_FOUND* is used to check if any entry wasn't downloaded.

`mgkit.net.embl.`**LOG = <Logger mgkit.net.embl (WARNING)>**
Log instance for this module

`mgkit.net.embl.`**NONE_FOUND = 'ERROR 12.+?.\\n?'**
Regular expression to check if no entry was found, used by *NoEntryFound*

`mgkit.net.embl.`**NOT_FOUND = 'Entry: .+? not found.\\n'**
Appears in the resulting fasta (not tried on other formats) in the case that at least one entry wasn't found.

exception `mgkit.net.embl.`**NoEntryFound**
Bases: `Exception`[379]

Raised if no sequences where found by *get_sequences_by_ids()*, the check is based on the *NONE_FOUND* variable.

`mgkit.net.embl.`**URL_REST = 'http://www.ebi.ac.uk/Tools/dbfetch/dbfetch/'**
Base URL for EMBL DBFetch REST API

[374] https://docs.python.org/3/library/stdtypes.html#dict
[375] https://docs.python.org/3/library/functions.html#bool
[376] https://docs.python.org/3/library/stdtypes.html#dict
[377] https://docs.python.org/3/library/stdtypes.html#str
[378] https://docs.python.org/3/library/exceptions.html#Exception
[379] https://docs.python.org/3/library/exceptions.html#Exception

mgkit.net.embl.**datawarehouse_search**(*query*, *domain='sequence'*, *result='sequence_release'*, *display='fasta'*, *offset=0*, *length=100000*, *contact=None*, *download='gzip'*, *url='http://www.ebi.ac.uk/ena/data/warehouse/search?'*, *fields=None*)

Changed in version 0.2.3: added *fields* parameter to retrieve tab separated information

New in version 0.1.13.

Perform a datawarehouse search on EMBL dbs. Instructions on the query language used to query the datawarehouse are available at this page[380] with more details about the databases domains at this page[381]

Parameters

- **query** (*str*[382]) – query for the search enging
- **domain** (*str*[383]) – database domain to search
- **result** (*str*[384]) – domain result requested
- **display** (*str*[385]) – display option (format to retrieve the entries)
- **offset** (*int*[386]) – the offset of the search results, defaults to the first
- **length** (*int*[387]) – number of results to retrieve at the specified offset and the limit is automatically set a 100,000 records for query
- **contact** (*str*[388]) – email of the user
- **download** (*str*[389]) – type of response. Gzip responses are automatically decompressed
- **url** (*str*[390]) – base URL for the resource
- **fields** (*None*[391], *iterable*) – must be an iterable of fields to be returned if display is set to *report*

Returns the raw request

Return type str[392]

Examples

Querying EMBL for all sequences of type rRNA of the *Clostridium* genus. Only from the EMBL release database in fasta format:

```
>>> query = 'tax_tree(1485) AND mol_type="rRNA"'
>>> result = 'sequence_release'
>>> display = 'fasta'
>>> data = embl.datawarehouse_search(query, result=result,
... display=display)
>>> len(data)
35919
```

[380] http://www.ebi.ac.uk/ena/about/browser#data_warehouse
[381] http://www.ebi.ac.uk/ena/data/warehouse/usage
[382] https://docs.python.org/3/library/stdtypes.html#str
[383] https://docs.python.org/3/library/stdtypes.html#str
[384] https://docs.python.org/3/library/stdtypes.html#str
[385] https://docs.python.org/3/library/stdtypes.html#str
[386] https://docs.python.org/3/library/functions.html#int
[387] https://docs.python.org/3/library/functions.html#int
[388] https://docs.python.org/3/library/stdtypes.html#str
[389] https://docs.python.org/3/library/stdtypes.html#str
[390] https://docs.python.org/3/library/stdtypes.html#str
[391] https://docs.python.org/3/library/constants.html#None
[392] https://docs.python.org/3/library/stdtypes.html#str

Each entry taxon_id from the same data can be retrieved using *report* as the *display* option and *fields* an iterable of fields to just ('accession', tax_id'):

```
>>> query = 'tax_tree(1485) AND mol_type="rRNA"'
>>> result = 'sequence_release'
>>> display = 'report'
>>> fields = ('accession', 'tax_id')
>>> data = embl.datawarehouse_search(query, result=result,
    display=display, fields=fields)
```

mgkit.net.embl.**dbfetch**(*embl_ids*, *db='embl'*, *contact=None*, *out_format='seqxml'*, *num_req=10*)

New in version 0.1.12.

Function that allows to use dbfetch service (REST). More information on the output formats and the database available at the service page[393]

Parameters

- **embl_ids** (*str*[394], *iterable*) – list or single sequence id to retrieve
- **db** (*str*[395]) – database from which retrieve the sequence data
- **contact** (*str*[396]) – email contact to use as per EMBL guidlines
- **out_format** (*str*[397]) – output format, depends on database
- **num_req** (*int*[398]) – number of ids per request

Returns a list with the results from each request sent. Each request sent has a maximum number *num_req* of ids, so the number of items in the list depends by the number of ids in *embl_ids* and the value of *num_req*.

Return type list[399]

mgkit.net.embl.**get_sequences_by_ids**(*embl_ids*, *contact=None*, *out_format='fasta'*, *num_req=10*, *embl_db='embl_cds'*, *strict=False*)

Changed in version 0.3.4: removed *compress* as it's bases on the *requests* package

Downloads entries using EBI REST API. It can download one entry at a time or accept an iterable and all sequences will be downloaded in batches of at most num_req.

It's fairly general, so can be customised, from the DB used to the output format: all batches are simply concatenate.

Note: There are some checks on the some errors reported by the EMBL api, but not documented, in particular two errors, which are just reported as text lines in the fasta file (the only one tested at this time).

The are two possible cases:

- if no entry was found *NoEntryFound* will be raised.
- if at least one entry wasn't found:
  - if strict is False (the default) the error will be just logged as a debug message
  - if strict is True *EntryNotFound* is raised

Parameters

[393] http://www.ebi.ac.uk/Tools/dbfetch/syntax.jsp
[394] https://docs.python.org/3/library/stdtypes.html#str
[395] https://docs.python.org/3/library/stdtypes.html#str
[396] https://docs.python.org/3/library/stdtypes.html#str
[397] https://docs.python.org/3/library/stdtypes.html#str
[398] https://docs.python.org/3/library/functions.html#int
[399] https://docs.python.org/3/library/stdtypes.html#list

- **embl_ids** (*iterable, str*[400]) – list of ids to download
- **contact** (*str*[401]) – email address to be passed in the query
- **format** (*str*[402]) – format of the entry
- **num_req** (*int*[403]) – number of entries to download with each request
- **embl_db** (*str*[404]) – db to which the ids refer to
- **strict** (*bool*[405]) – if True, a check on the number of entries retrieved is performed

Returns the entries requested

Return type str[406]

Raises

- *EntryNotFound* – if at least an entry was not found
- *NoEntryFound* – if NO entry were found

Warning: The number of sequences that can be downloaded at a time is 11, it seems, since the returned sequences for each request was at most 11. I didn't find any mention of this in the API docs, but it may be a restriction that's temporary.

mgkit.net.pfam module

New in version 0.2.3.

This module defines routine to access Pfam information using a network connection

mgkit.net.pfam.**get_pfam_families**(*key='id'*)

New in version 0.2.3.

Gets a dictionary with the accession/id/description of Pfam families from Pfam. This list can be accessed using the URL: http://pfam.xfam.org/families?output=text

The output is a tab separated file where the fields are:

- ACCESSION
- ID
- DESCRIPTION

Parameters **key** (*str*[407]) – if the value is *id*, the key of the dictionary is the ID, otherwise ID swaps position with ACCESSION (the new key)

Returns by default the function returns a dictionary that uses the ID as key, while the value is a tuple (ACCESSION, DESCRIPTION). ID is the default because the *hmmer2gff - Convert HMMER output to GFF* script output uses ID as *gene_id* value when using the HMM provided by Pfam

Return type dict[408]

[400] https://docs.python.org/3/library/stdtypes.html#str
[401] https://docs.python.org/3/library/stdtypes.html#str
[402] https://docs.python.org/3/library/stdtypes.html#str
[403] https://docs.python.org/3/library/functions.html#int
[404] https://docs.python.org/3/library/stdtypes.html#str
[405] https://docs.python.org/3/library/functions.html#bool
[406] https://docs.python.org/3/library/stdtypes.html#str
[407] https://docs.python.org/3/library/stdtypes.html#str
[408] https://docs.python.org/3/library/stdtypes.html#dict

mgkit.net.uniprot module

Contains function and constants for Uniprot access

mgkit.net.uniprot.**UNIPROT_GET = 'http://www.uniprot.org/uniprot/'**
URL to Uniprot REST API

mgkit.net.uniprot.**UNIPROT_MAP = 'http://www.uniprot.org/mapping/'**
URL to Uniprot mapping REST API

mgkit.net.uniprot.**UNIPROT_TAXONOMY = 'http://www.uniprot.org/taxonomy/'**
URL to Uniprot REST API - Taxonomy

mgkit.net.uniprot.**get_gene_info**(*gene_ids*, *columns*, *max_req=50*, *contact=None*)
New in version 0.1.12.

Get informations about a list of genes. it uses *query_uniprot()* to send the request and format the response in a dictionary.

Parameters

- **gene_ids** (*iterable, str*[409]) – gene id(s) to get informations for
- **columns** (*list*[410]) – list of columns
- **max_req** (*int*[411]) – number of maximum *gene_ids* per request
- **contact** (*str*[412]) – email address to be passed in the query (requested Uniprot API)

Returns dictionary where the keys are the *gene_ids* requested and the values are dictionaries with the names of the *columns* requested as keys and the corresponding values, which can be lists if the values are are semicolon separated strings.

Return type dict[413]

Example

To get the taxonomy ids for some genes:

```
>>> uniprot.get_gene_info(['Q09575', 'Q8DQI6'], ['organism-id'])
{'Q09575': {'organism-id': '6239'}, 'Q8DQI6': {'organism-id': '171101'}}
```

mgkit.net.uniprot.**get_gene_info_iter**(*gene_ids*, *columns*, *contact=None*, *max_req=50*)
New in version 0.3.3.

Alternative function to *get_gene_info()*, returning an iterator to avoid connections timeouts when updating a dictionary

This funciton's parameters are the same as *get_gene_info()*

mgkit.net.uniprot.**get_ko_to_eggnog_mappings**(*ko_ids*, *contact=None*)
New in version 0.1.14.

It's not possible to map in one go KO IDs to eggNOG IDs via the API in Uniprot. This function uses *query_uniprot()* to get all Uniprot IDs requested and the return a dictionary with all their eggNOG IDs they map to.

Parameters

- **ko_ids** (*iterable*) – an iterable of KO IDs
- **contact** (*str*[414]) – email address to be passed in the query (requested Uniprot API)

[409] https://docs.python.org/3/library/stdtypes.html#str
[410] https://docs.python.org/3/library/stdtypes.html#list
[411] https://docs.python.org/3/library/functions.html#int
[412] https://docs.python.org/3/library/stdtypes.html#str
[413] https://docs.python.org/3/library/stdtypes.html#dict
[414] https://docs.python.org/3/library/stdtypes.html#str

Returns The format of the resulting dictionary is ko_id -> {eggnog_id1, ..}

Return type dict[415]

mgkit.net.uniprot.**get_mappings**(*entry_ids*, *db_from='ID'*, *db_to='EMBL'*, *out_format='tab'*, *contact=None*)

Gets mapping of genes using Uniprot REST API. The db_from and db_to values are the ones accepted by Uniprot API. The same applies to out_format, the only processed formats are 'list', which returns a list of the mappings (should be used with one gene only) and 'tab', which returns a dictionary with the mapping. All other values returns a string with the newline stripped.

Parameters

- **entry_ids** (*iterable*) – iterable of ids to be mapped (there's a limit) to the maximum length of a HTTP request, so it should be less than 50
- **db_from** (*str*[416]) – string that identify the DB for elements in entry_ids
- **db_to** (*str*[417]) – string that identify the DB to which map entry_ids
- **out_format** (*str*[418]) – format of the mapping; 'list' and 'tab' are processed
- **contact** (*str*[419]) – email address to be passed in the query (requested Uniprot API)

Returns tuple, dict or str depending on out_format value

mgkit.net.uniprot.**get_sequences_by_ko**(*ko_id*, *taxonomy*, *contact=None*, *reviewed=True*)

Gets sequences from Uniprot, restricting to the taxon id passed.

Parameters

- **ko_id** (*str*[420]) – KO id of the sequences to download
- **taxonomy** (*int*[421]) – id of the taxon
- **contact** (*str*[422]) – email address to be passed in the query (requested by Uniprot API)
- **reviewed** (*bool*[423]) – if the sequences requested must be reviewed

Returns string with the fasta file downloaded

mgkit.net.uniprot.**get_uniprot_ec_mappings**(*gene_ids*, *contact=None*)

New in version 0.1.14.

Shortcut to download EC mapping of Uniprot IDs. Uses *get_gene_info()* passing the correct column (*ec*).

mgkit.net.uniprot.**ko_to_mapping**(*ko_id*, *query*, *columns*, *contact=None*)

Returns the mappings to the supplied KO. Can be used for any id, the query format is free as well as the columns returned. The only restriction is using a tab format, that is parsed.

Parameters

- **ko_id** (*str*[424]) – id used in the query
- **query** (*str*[425]) – query passed to the Uniprot API, ko_id is replaced using `str.format()`

[415] https://docs.python.org/3/library/stdtypes.html#dict
[416] https://docs.python.org/3/library/stdtypes.html#str
[417] https://docs.python.org/3/library/stdtypes.html#str
[418] https://docs.python.org/3/library/stdtypes.html#str
[419] https://docs.python.org/3/library/stdtypes.html#str
[420] https://docs.python.org/3/library/stdtypes.html#str
[421] https://docs.python.org/3/library/functions.html#int
[422] https://docs.python.org/3/library/stdtypes.html#str
[423] https://docs.python.org/3/library/functions.html#bool
[424] https://docs.python.org/3/library/stdtypes.html#str
[425] https://docs.python.org/3/library/stdtypes.html#str

- **column** (*str*[426]) – column used in the results table used to map the ids
- **contact** (*str*[427]) – email address to be passed in the query (requested Uniprot API)

Note: each mapping in the column is separated by a ;

mgkit.net.uniprot.**parse_uniprot_response**(*data*, *simple=True*)

New in version 0.1.12.

Parses raw response from a Uniprot query (tab format only) from functions like *query_uniprot()* into a dictionary. It requires that the first column is the entry id (or any other unique id).

Parameters

- **data** (*str*[428]) – string response from Uniprot
- **simple** (*bool*[429]) – if True and the number of columns is 1, the dictionary returned has a simplified structure

Returns The format of the resulting dictionary is entry_id -> {column1 -> value, column2 -> value, ..} unless there's only one column and *simple* is True, in which case the value is equal to the value of the only column.

Return type dict[430]

mgkit.net.uniprot.**query_uniprot**(*query*, *columns=None*, *format='tab'*, *limit=None*, *contact=None*, *baseurl='http://www.uniprot.org/uniprot/'*)

New in version 0.1.12.

Changed in version 0.1.13: added *baseurl* and made *columns* a default argument

Queries Uniprot, returning the raw response in tbe format specified. More informations at the page[431]

Parameters

- **query** (*str*[432]) – query to submit, as put in the input box
- **columns** (*None*[433], *iterable*) – list of columns to return
- **format** (*str*[434]) – response format
- **limit** (*int*[435], *None*[436]) – number of entries to return or *None* to request all entries
- **contact** (*str*[437]) – email address to be passed in the query (requested Uniprot API)
- **baseurl** (*str*[438]) – base url for the REST API, can be either *UNIPROT_GET* or *UNIPROT_TAXONOMY*

Returns raw response from the query

Return type str[439]

[426] https://docs.python.org/3/library/stdtypes.html#str
[427] https://docs.python.org/3/library/stdtypes.html#str
[428] https://docs.python.org/3/library/stdtypes.html#str
[429] https://docs.python.org/3/library/functions.html#bool
[430] https://docs.python.org/3/library/stdtypes.html#dict
[431] http://www.uniprot.org/faq/28
[432] https://docs.python.org/3/library/stdtypes.html#str
[433] https://docs.python.org/3/library/constants.html#None
[434] https://docs.python.org/3/library/stdtypes.html#str
[435] https://docs.python.org/3/library/functions.html#int
[436] https://docs.python.org/3/library/constants.html#None
[437] https://docs.python.org/3/library/stdtypes.html#str
[438] https://docs.python.org/3/library/stdtypes.html#str
[439] https://docs.python.org/3/library/stdtypes.html#str

Example

To get the taxonomy ids for some genes:

```
>>> uniprot.query_uniprot('Q09575 OR Q8DQI6', ['id', 'organism-id'])
'Entry\tOrganism ID\nQ8DQI6\t171101\nQ09575\t6239\n'
```

Warning: because of limits in the length of URLs, it's advised to limit the length of the query string.

mgkit.net.utils module

Utility functions for the network package

mgkit.net.utils.**url_open**(*url*, *data=None*, *headers=None*, *agent=None*, *get=True*, *stream=False*)

Changed in version 0.3.4: now uses *requests*

Parameters

- **url** (*str*[440]) – url to request
- **data** (*str*[441]) – parameters to pass to the request
- **headers** (*dict*[442]) – any additional headers
- **agent** (*str*[443]) – user agent to use
- **get** (*bool*[444]) – True if the request is a GET, False for POST
- **stream** (*bool*[445]) – returns an iterator to stream over
- **url** – url to request
- **data** – data to add to the request
- **compress** (*bool*[446]) – if the response should be compressed
- **agent** – if supplied, the 'User-Agent' header we'll be added to the request

Returns the response handle

mgkit.net.utils.**url_read**(*url*, *data=None*, *agent=None*, *headers=None*, *get=True*)

Changed in version 0.3.4: now uses *requests*, removed *compressed* and added *headers*, *get*

Opens an URL and reads the

Wrapper of *url_open()* which reads the full response

Parameters

- **url** (*str*[447]) – url to request
- **data** (*dict*[448] *or None*[449]) – data to add to the request
- **headers** (*dict*[450] *or None*[451]) – additional headers

[440] https://docs.python.org/3/library/stdtypes.html#str
[441] https://docs.python.org/3/library/stdtypes.html#str
[442] https://docs.python.org/3/library/stdtypes.html#dict
[443] https://docs.python.org/3/library/stdtypes.html#str
[444] https://docs.python.org/3/library/functions.html#bool
[445] https://docs.python.org/3/library/functions.html#bool
[446] https://docs.python.org/3/library/functions.html#bool
[447] https://docs.python.org/3/library/stdtypes.html#str
[448] https://docs.python.org/3/library/stdtypes.html#dict
[449] https://docs.python.org/3/library/constants.html#None
[450] https://docs.python.org/3/library/stdtypes.html#dict
[451] https://docs.python.org/3/library/constants.html#None

- **agent** (*str*[452] *or None*[453]) – if supplied, the 'User-Agent' header we'll be added to the request
- **get** (*bool*[454]) – uses a GET operation if True, POST if False

Returns the response data

Return type str[455]

Module contents

Package with functions/classes used in accessing network resources

mgkit.plots package

Submodules

mgkit.plots.abund module

New in version 0.1.15.

Module to plot relative abundances in a 1D or 3D projection

mgkit.plots.abund.**col_func_firstel**(*key*, *colors=None*)

mgkit.plots.abund.**col_func_name**(*key*, *func=None*, *colors=None*)

mgkit.plots.abund.**col_func_taxon**(*taxon_id*, *taxonomy*, *anc_ids*, *colpal*)

mgkit.plots.abund.**draw_1d_grid**(*ax, labels=['LAM', 'SAM'], fontsize=22*)

Changed in version 0.2.0: reworked internals and changed defaults

Draws a 1D axis, to display propotions.

Parameters

- **ax** – an axis instance
- **labels** (*iterable*) – list of string to be put for the axes
- **fontsize** (*float*[456]) – font size for the labels, the tick font size is equal to *0.75 * fontsize*

mgkit.plots.abund.**draw_axis_internal_triangle**(*ax*, *color='r'*, *linewidth=2.0*)

New in version 0.2.5.

Draws a triangle that indicates the 50% limit for all 3 samples

Parameters

- **ax** – axis to use
- **color** (*str*[457], *float*[458], *tuple*[459]) – color used to draw the triangle
- **linewidth** (*float*[460]) – line width

[452] https://docs.python.org/3/library/stdtypes.html#str
[453] https://docs.python.org/3/library/constants.html#None
[454] https://docs.python.org/3/library/functions.html#bool
[455] https://docs.python.org/3/library/stdtypes.html#str
[456] https://docs.python.org/3/library/functions.html#float
[457] https://docs.python.org/3/library/stdtypes.html#str
[458] https://docs.python.org/3/library/functions.html#float
[459] https://docs.python.org/3/library/stdtypes.html#tuple
[460] https://docs.python.org/3/library/functions.html#float

mgkit.plots.abund.**draw_circles**(*ax*, *data*, *col_func=<function col_func_name>*, *csize=200*, *alpha=0.5*, *sizescale=None*, *order=None*, *linewidths=0.0*, *edgecolor='none'*)

Changed in version 0.2.0: changed internals and added return value

Draws a scatter plot over either a planar-simplex projection, if the number of coordinates is 3, or in a 1D axis.

If the number of coordinates is 3, `project_point()` is used to project the point in 2 coordinates. The coordinates are converted in proportions internally.

Parameters

- **ax** – axis to plot on
- **data** (*pandas.DataFrame*) – a DataFrame with 2 for a 1D plot or 3 columns for a planar-simplex
- **col_func** (*func*) – a function that accept a parameter, an element of the DataFrame index and returns a colour for it
- **csize** (*int*[461]) – the base size of the circles
- **alpha** (*float*[462]) – transparency of the circles, between 0 and 1 included
- **sizescale** (*None*[463], *pandas.Series*) – a Series or dictionary with the same elements as the Index of *data*, whose values are the size factors that are multiplied to *csize*. If **None**, the size of the circles is equal to *csize*
- **order** (*None*[464], *iterable*) – iterable with the elements of *data* Index, to specify the order in which the circles must be plotted. If None, the order is the same as *data.index*
- **linewidths** (*float*[465]) – width of the circle line
- **edgecolor** (*str*[466]) – color of the circle line

Returns the return value of matplotlib *scatter*

Return type PathCollection

Note: To **not** have circle lines, *edgecolor* must be *'none'* and *linewidths* equal *0*

---

mgkit.plots.abund.**draw_triangle_grid**(*ax, labels=['LAM', 'SAM', 'EAM'], linewidth=1.0, styles=['-', ':', '–'], fontsize=22*)

Changed in version 0.2.0: reworked internals and changed defaults

Draws a triangle as axes, for a planar-simplex projection.

Parameters

- **ax** – an axis instance
- **labels** (*iterable*) – list of string to be put for the axes
- **styles** (*None*[467], *iterable*) – either *None* for solid lines or matplotlib line markers. These are in sync between the internal lines and the axes.
- **linewidth** (*float*[468]) – line width for the axes, the internal lines are equal to *0.75 * linewidth*

[461] https://docs.python.org/3/library/functions.html#int
[462] https://docs.python.org/3/library/functions.html#float
[463] https://docs.python.org/3/library/constants.html#None
[464] https://docs.python.org/3/library/constants.html#None
[465] https://docs.python.org/3/library/functions.html#float
[466] https://docs.python.org/3/library/stdtypes.html#str
[467] https://docs.python.org/3/library/constants.html#None
[468] https://docs.python.org/3/library/functions.html#float

- **fontsize** (*float*[469]) – font size for the labels, the tick font size is equal to *0.75 * fontsize*

mgkit.plots.abund.**project_point**(*point*)

Project a tuple containing coordinates (i.e. x, y, z) to planar-simplex.

Parameters **point** (*tuple*[470]) – contains the three coordinates to project

Returns the projected point in a planar-simplex

Return type tuple[471]

mgkit.plots.boxplot module

New in version 0.1.14.

Code related to boxplots

mgkit.plots.boxplot.**add_values_to_boxplot**(*dataframe*, *ax*, *plot_data*, *plot_order*, *data_colours=None*, *alpha=0.5*, *s=80*, *marker='o'*, *linewidth=0.01*, *box_vert=False*)

New in version 0.1.13.

Changed in version 0.1.14: added *box_vert* parameter

Changed in version 0.1.16: changed default value for *linewidth*

Adds the values of a dataframe used in *boxplot_dataframe()* to the plot. *linewidth* must be higher than 0 if a marker like | is used.

A list of markers is available at this page[472]

> **Warning:** Contrary to *boxplot_dataframe()*, the boxplot default is horizontal (*box_vert*). The default will change in a later version.

Parameters

- **dataframe** – dataframe with the values to plot
- **ax** – an axis instance
- **plot_data** – return value from *boxplot_dataframe()*
- **plot_order** (*iterable*) – row order used to plot the boxes
- **data_colours** (*dict*[473]) – colors used for the values
- **alpha** (*float*[474]) – alpha value for the colour
- **s** (*int*[475]) – size of the marker drawn
- **marker** (*str*[476]) – one of the accepted matplotlib markers
- **linewidth** (*float*[477]) – width of the line used to draw the marker
- **box_vert** (*bool*[478]) – specify if the original boxplot is vertical or not

[469] https://docs.python.org/3/library/functions.html#float
[470] https://docs.python.org/3/library/stdtypes.html#tuple
[471] https://docs.python.org/3/library/stdtypes.html#tuple
[472] http://matplotlib.org/api/markers_api.html
[473] https://docs.python.org/3/library/stdtypes.html#dict
[474] https://docs.python.org/3/library/functions.html#float
[475] https://docs.python.org/3/library/functions.html#int
[476] https://docs.python.org/3/library/stdtypes.html#str
[477] https://docs.python.org/3/library/functions.html#float
[478] https://docs.python.org/3/library/functions.html#bool

mgkit.plots.boxplot.**add_significance_to_boxplot**(*sign_indices*, *ax*, *pos*, *box_vert=True*, *fontsize=16*)

New in version 0.1.16.

Add significance groups to boxplots

Parameters

- **sign_indices** (*iterable*) – iterable in which each element is a tuple; each element of the tuple is the numerical index of the position of the significant boxplot
- **ax** – an axis instance
- **pos** (*tuple*[479]) – the 2 values are the coordinates for the top line, and the the lowest bound for the whisker
- **box_vert** (*bool*[480]) – if the boxplot is vertical
- **fontsize** (*float*[481]) – size for the * (star)

mgkit.plots.boxplot.**boxplot_dataframe_multindex**(*dataframe*, *axes*, *plot_order=None*, *label_map=None*, *fonts=None*, *fill_box=True*, *colours=None*, *data_colours=None*, *box_vert=True*)

New in version 0.1.13.

Todo: documentation

The function draws a series of boxplots from a DataFrame object, whose order is directed by the iterable plot_order. The columns of each DataFrame row contains the values for each boxplot. An axes object is needed.

Parameters

- **dataframe** – dataframe to plot
- **plot_order** (*iterable*) – row order used to plot the boxes
- **axes** – an axes instance
- **label_map** (*dict*[482]) – a map that converts the items in plot_order to a label used on the plot X axes
- **fonts** (*dict*[483]) – dictionary with properties for x axis labels, DEFAULT_BOXPLOT_FONTCONF is used by default
- **fill_box** (*bool*[484]) – if True each box is filled with the same colour of its outline
- **colours** (*dict*[485]) – dictionary with properties for each boxplot if data_colours is None, whi overrides box, whiskers and fliers. Defaults to DEFAULT_BOXPLOT_COLOURS
- **data_colours** (*dict*[486]) – dictionary of colours for each boxplot, a set of colours can be obtained using func:*map_taxon_to_colours*

Returns the plot data same as matplotlib boxplot function

[479] https://docs.python.org/3/library/stdtypes.html#tuple
[480] https://docs.python.org/3/library/functions.html#bool
[481] https://docs.python.org/3/library/functions.html#float
[482] https://docs.python.org/3/library/stdtypes.html#dict
[483] https://docs.python.org/3/library/stdtypes.html#dict
[484] https://docs.python.org/3/library/functions.html#bool
[485] https://docs.python.org/3/library/stdtypes.html#dict
[486] https://docs.python.org/3/library/stdtypes.html#dict

mgkit.plots.boxplot.**boxplot_dataframe**(*dataframe*, *plot_order*, *ax*, *label_map=None*, *fonts=None*, *fill_box=True*, *colours=None*, *data_colours=None*, *box_vert=True*, *widths=0.5*)

New in version 0.1.7: To move from an all-in-one drawing to a more modular one.

Changed in version 0.1.13: added box_vert parameter

Changed in version 0.1.16: added *widths* parameter

The function draws a series of boxplots from a DataFrame object, whose order is directed by the iterable plot_order. The columns of each DataFrame row contains the values for each boxplot. An ax object is needed.

Parameters

- **dataframe** – dataframe to plot
- **plot_order** (*iterable*) – row order used to plot the boxes
- **ax** – an axis instance
- **label_map** (*dict*[487]) – a map that converts the items in plot_order to a label used on the plot X ax
- **fonts** (*dict*[488]) – dictionary with properties for x axis labels, DEFAULT_BOXPLOT_FONTCONF is used by default
- **fill_box** (*bool*[489]) – if True each box is filled with the same colour of its outline
- **colours** (*dict*[490]) – dictionary with properties for each boxplot if data_colours is None, whi overrides box, whiskers and fliers. Defaults to DEFAULT_BOXPLOT_COLOURS
- **data_colours** (*dict*[491]) – dictionary of colours for each boxplot, a set of colours can be obtained using func:*map_taxon_to_colours*
- **box_vert** (*bool*[492]) – if False the boxplots are drawn horizontally
- **widths** (*float*[493]) – width (scalar or array) of the boxplots width(s)

Returns the plot data; same as matplotlib boxplot function

mgkit.plots.colors module

New in version 0.1.14.

Contains code related to colour

mgkit.plots.colors.**float_to_hex_color**(*r*, *g*, *b*)

New in version 0.1.14.

Converts RGB float values to Hexadecimal value string

mgkit.plots.colors.**palette_float_to_hex**(*palette*)

New in version 0.1.16.

Applies *float_to_hex_color()* to an iterable of colors

[487] https://docs.python.org/3/library/stdtypes.html#dict
[488] https://docs.python.org/3/library/stdtypes.html#dict
[489] https://docs.python.org/3/library/functions.html#bool
[490] https://docs.python.org/3/library/stdtypes.html#dict
[491] https://docs.python.org/3/library/stdtypes.html#dict
[492] https://docs.python.org/3/library/functions.html#bool
[493] https://docs.python.org/3/library/functions.html#float

mgkit.plots.heatmap module

New in version 0.1.14.

Code related to heatmaps.

mgkit.plots.heatmap.**baseheatmap**(*data*, *ax*, *norm=None*, *cmap=None*, *xticks=None*, *yticks=None*, *fontsize=18*, *meshopts=None*, *annot=False*, *annotopts=None*)

Changed in version 0.2.3: added *annot* and *annot_args* arguments

A basic heatmap using `matplotlib.pyplot.pcolormesh()`. It expect a `pandas.DataFrame`.

Note: Rows a plot bottom to up, while the columns left to right. Change the order of the DataFrame if needed.

Parameters

- **data** (*pandas.DataFrame*) – matrix to plot. The DataFrame labels are used
- **ax** – axes to use
- **norm** – if needed, `matplotlib.colors.BoundaryNorm` or `matplotlib.colors.Normalize` can be used to fine tune the colors
- **cmap** (*None*[494], *matplotlib.colors.ListedColormap*) – color map to use
- **xticks** (*None*[495], *dict*[496]) – dictionary with additional options to pass to *set_xticklabels*
- **yticks** (*None*[497], *dict*[498]) – dictionary with additional options to pass to *set_yticklabels*
- **fontsize** (*int*[499]) – font size to use for the labels
- **meshopts** (*None*[500], *dict*[501]) – additional options to pass to `matplotlib.pyplot.pcolormesh()`
- **annot** (*bool*[502]) – if True the values of the matrix will be added
- **annot_args** (*None*[503], *dict*[504]) – dictionary with the options for the annotations. The option *format* is a function that returns the formatted number, defaults to a number with no decimal part

Returns the return value of `matplotlib.pyplot.pcolormesh()`

Return type matplotlib.collections.QuadMesh

mgkit.plots.heatmap.**grouped_spine**(*groups*, *labels*, *ax*, *which='y'*, *spine='right'*, *spine_opts=None*, *start=0*)

Changed in version 0.2.0: added *va*, *ha* keys to *spine_opts*, changed the label positioning

Changed in version 0.2.5: added *start* parameter

Changes the spine of an heatmap axis given the groups of labels.

[494] https://docs.python.org/3/library/constants.html#None
[495] https://docs.python.org/3/library/constants.html#None
[496] https://docs.python.org/3/library/stdtypes.html#dict
[497] https://docs.python.org/3/library/constants.html#None
[498] https://docs.python.org/3/library/stdtypes.html#dict
[499] https://docs.python.org/3/library/functions.html#int
[500] https://docs.python.org/3/library/constants.html#None
[501] https://docs.python.org/3/library/stdtypes.html#dict
[502] https://docs.python.org/3/library/functions.html#bool
[503] https://docs.python.org/3/library/constants.html#None
[504] https://docs.python.org/3/library/stdtypes.html#dict

Note: It should work for any plot, but was not tested

Parameters

- **groups** (*iterable*) – a nested list where each is element is a list containing the labels that belong to that group.
- **labels** (*iterable*) – an iterable with the labels of the groups. Needs to be in the same order as groups
- **ax** – axis to use (same as heatmap)
- **which** (*str*[505]) – to specify the axis, either *x* or *y*
- **spine** (*str*[506]) – position of the spine. if *which* is **x** accepted values are *top* and *bottom*, if which is **y** *left* and *right* are accepted
- **spine_opts** (*dict*[507]) – additional options to pass to the spine class
- **start** (*int*[508]) – the start coordinate for the grouped spine. Defaults to 0

mgkit.plots.heatmap.**dendrogram**(*data*, *ax*, *method='complete'*, *orientation='top'*, *use_dist=True*, *dist_func=<function pdist>*)

Changed in version 0.1.16: added *use_dist* and *dist_func* parameters

Plots a dendrogram of the clustered rows of the given matrix; if the columns are to be clustered, the transposed matrix needs to be passed.

Parameters

- **data** (*pandas.DataFrame*) – matrix to plot. The DataFrame labels are used
- **ax** – axes to use
- **method** (*str*[509]) – clustering method used, internally `scipy.cluster.hierarchy.linkage()` is used.
- **orientation** (*str*[510]) – direction for the plot. *top*, *bottom*, *left* and *right* are accepted; *top* will draw the leaves at the bottom.
- **use_dist** (*bool*[511]) – if True, the function *dist_func* will be applied to *data* to get a distance matrix
- **dist_func** (*func*) – distance function to be used

Returns The dendrogram plotted, as returned by `scipy.cluster.hierarchy.dendrogram()`

mgkit.plots.heatmap.**heatmap_clustered**(*data*, *figsize=(10, 5)*, *cmap=None*, *norm=None*)

Plots a heatmap clustered on both rows and columns.

Parameters

- **data** (*pandas.DataFrame*) – matrix to plot. The DataFrame labels are used
- **figsize** (*tuple*[512]) – passed to *mgkit.plots.utils.get_grid_figure()*
- **cmap** (*None*[513], *matplotlib.colors.ListedColormap*) – color map to use

[505] https://docs.python.org/3/library/stdtypes.html#str
[506] https://docs.python.org/3/library/stdtypes.html#str
[507] https://docs.python.org/3/library/stdtypes.html#dict
[508] https://docs.python.org/3/library/functions.html#int
[509] https://docs.python.org/3/library/stdtypes.html#str
[510] https://docs.python.org/3/library/stdtypes.html#str
[511] https://docs.python.org/3/library/functions.html#bool
[512] https://docs.python.org/3/library/stdtypes.html#tuple
[513] https://docs.python.org/3/library/constants.html#None

- **norm** – if needed, `matplotlib.colors.BoundaryNorm` or `matplotlib.colors.Normalize` can be used to fine tune the colors

mgkit.plots.unused module

New in version 0.1.14.

Code deprecated or untested

mgkit.plots.unused.**TAXON_COLOURS = {'archaea': '#4daf4a', 'bacteria': '#377eb8', 'cili**
Default colours for root taxa

mgkit.plots.unused.**barchart_categories**(*data*, *colours=None*, *title=''*, *tickfont='small'*, *xlabel_dict=None*, *barlabel_dict=None*, *width=0.9*, *rotation='vertical'*, *file_name=None*, *fig_size=None*, *fig_aspect=None*)

Parameters

- **data** – DataFrame where the number of rows indicates how many bars will plotted per column
- **colours** – must be equal the number of data rows if supplied or it will be blue by default
- **title** – chart title
- **tickfont** – font size for ticks (only for column axis)
- **xlabel_dict** – a mapping to the acual labels to use for the columns. Defaults to columns' names
- **barlabel_dict** – a mapping to the acual labels to use for the row. Defaults to rows' names
- **width** – bar width

Returns axis instance

mgkit.plots.unused.**get_taxon_colors_new**(*taxa*, *taxonomy*, *default_colour='#ffff33'*)
Returns a dictionary of taxa and their assigned colours based on *TAXON_COLOURS* and the taxonomy provided. Uses the `taxon.Taxonomy.get_taxon_root()` to determine the root of a taxon.

Parameters

- **taxa** (*iterable*) – iterable of taxon ids
- **taxonomy** (Taxonomy) – taxonomy instance
- **default_colour** – colour used in case there's no known root for the taxon

Return dict dictionary mapping taxon_id to colour

mgkit.plots.unused.**lineplot_values_on_second_axis**(*gene_num*, *axis*, *colour='c'*, *ylabel=''*)
Deprecated since version 0.1.13.

Adds a lineplot on a second axis using twinx

mgkit.plots.unused.**map_taxon_to_colours**(*taxa*, *taxonomy*, *default_colour='#ffff33'*)
Returns a dictionary of taxa and their assigned colours based on *TAXON_COLOURS* and the taxonomy provided. Uses the `taxon.Taxonomy.get_taxon_root()` to determine the root of a taxon.

Parameters

- **taxa** (*iterable*) – iterable of taxon ids
- **taxonomy** (Taxonomy) – taxonomy instance

- **default_colour** – colour used in case there's no known root for the taxon

Return dict dictionary mapping taxon_id to colour

mgkit.plots.unused.**plot_contig_assignment_bar**(*series*, *taxon_colours=None*, *log_scale=False*, *index=None*, *file_name=None*, *fig_aspect=None*, *xlabels_size=8*)

Plots barchart for contig assignment

Parameters

- **series** – `pandas.Series` instance with the data
- **taxon_colours** (*dict*[514]) – colour of the bars for each taxon
- **log_scale** (*bool*[515]) – if True the y axis is log scaled
- **fig_aspect** (*tuple*[516]) – tuple with figure size
- **xlabels_size** (*int*[517]) – size of the taxon labels
- **index** – optional `pandas.Index` used to reindex the series
- **file_name** (*str*[518]) – name of the file to write the graph to

mgkit.plots.unused.**plot_scatter_2d**(*data*, *labels*, *colours=None*, *pointsize=10.0*, *title=''*, *xlabel=''*, *ylabel=''*, *centers=None*, *marker='*'*, *marker_colour=None*, *marker-size=None*, *hull_points=True*, *linewidth=0.2*, *legend=True*, *anno_center=True*)

Scatter plot in 2d. Used for cluster results

Parameters

- **data** (*array*) – `numpy.array` with shape n, 2
- **labels** (*array*) – labels to categorise samples
- **colours** (*dict*[519]) – dictionary whose keys are the labels and the values are valid `matplotlib` colours
- **pointsize** (*int*[520]) – point size
- **title** (*str*[521]) – plot title
- **xlabel** (*str*[522]) – label for x axis
- **ylabel** (*str*[523]) – label for y axis

Returns axis instance

mgkit.plots.unused.**plot_scatter_3d**(*data*, *labels*, *colours=None*, *pointsize=10.0*, *title=''*, *xlabel=''*, *ylabel=''*, *zlabel=''*)

Scatter plot in 3d. Used for cluster results

Parameters

- **data** (*array*) – `numpy.array` with shape n, 3
- **labels** (*array*) – labels to categorise samples

[514] https://docs.python.org/3/library/stdtypes.html#dict
[515] https://docs.python.org/3/library/functions.html#bool
[516] https://docs.python.org/3/library/stdtypes.html#tuple
[517] https://docs.python.org/3/library/functions.html#int
[518] https://docs.python.org/3/library/stdtypes.html#str
[519] https://docs.python.org/3/library/stdtypes.html#dict
[520] https://docs.python.org/3/library/functions.html#int
[521] https://docs.python.org/3/library/stdtypes.html#str
[522] https://docs.python.org/3/library/stdtypes.html#str
[523] https://docs.python.org/3/library/stdtypes.html#str

- **colours** (*dict*[524]) – dictionary whose keys are the labels and the values are valid `matplotlib` colours
- **pointsize** (*int*[525]) – point size
- **title** (*str*[526]) – plot title
- **xlabel** (*str*[527]) – label for x axis
- **ylabel** (*str*[528]) – label for y axis
- **zlabel** (*str*[529]) – label for z axis

Returns axis instance

`mgkit.plots.unused.`**scatter_gene_values**(*gene_dict*, *xlabel='Profile pN/pS'*, *ylabel='Rumen pN/pS'*, *title=''*, *colours=None*, *file_name=None*, *plot_order=None*, *line_colour='r'*, *max_limit=None*, *axes=None*)

Plots gene-taxon pN/pS from profiles against their observed values.

Parameters

- **gene_dict** (*dict*[530]) – dictionary that contains the data
- **xlabel** (*str*[531]) – label for x axis
- **ylabel** (*str*[532]) – label for y axis
- **title** (*str*[533]) – graph title
- **colours** – colours used in for the different datasets; defaults to TAXON_COLOURS
- **file_name** (*str*[534]) – path to which the graph is to be saved (by default) it doesn't write to disk
- **plot_order** (*iterable*) – the order used in plotting the data points; default to the order of the gene_dict dictionary keys
- **coloUr** – valid colour for the lines in the plot
- **max_limit** (*float*[535]) – used to put a limit on the plot
- **axes** – optional axes used to draw the scatter plot

Returns the axis object used for the plot

mgkit.plots.utils module

New in version 0.1.14.

Misc code

`mgkit.plots.utils.`**get_grid_figure**(*rows*, *cols*, *dpi=300*, *figsize=(10, 20)*, ***kwd*)

New in version 0.1.13.

Simple wrapper to init a GridSpec figure

[524] https://docs.python.org/3/library/stdtypes.html#dict
[525] https://docs.python.org/3/library/functions.html#int
[526] https://docs.python.org/3/library/stdtypes.html#str
[527] https://docs.python.org/3/library/stdtypes.html#str
[528] https://docs.python.org/3/library/stdtypes.html#str
[529] https://docs.python.org/3/library/stdtypes.html#str
[530] https://docs.python.org/3/library/stdtypes.html#dict
[531] https://docs.python.org/3/library/stdtypes.html#str
[532] https://docs.python.org/3/library/stdtypes.html#str
[533] https://docs.python.org/3/library/stdtypes.html#str
[534] https://docs.python.org/3/library/stdtypes.html#str
[535] https://docs.python.org/3/library/functions.html#float

Parameters

- **rows** (*int*[536]) – number of rows
- **columns** (*int*[537]) – number of columns
- **dpi** (*int*[538]) – dpi used for the figure
- **figsize** (*tuple*[539]) – size of the figure in inches

Returns the figure and axes objects

Return type tuple[540]

mgkit.plots.utils.**get_single_figure**(*dpi=300*, *figsize=(10*, *20)*, *aspect='auto'*)

Changed in version 0.1.14: added *aspect* parameter

Simple wrapper to init a single figure

Parameters

- **dpi** (*int*[541]) – dpi used for the figure
- **figsize** (*tuple*[542]) – size of the figure in inches
- **aspect** (*str*[543], *float*[544]) – aspect ratio to be passed to figure.add_subplot

Returns the figure and axes objects

Return type tuple[545]

mgkit.plots.utils.**legend_patches**(*labels*, *colors*)

New in version 0.3.1.

Makes handles (using matplotlib Patch) that can be passed to the legend method of a matplotlib axes instance

Parameters

- **labels** (*iterable*) – iterable that yields a label
- **colors** (*iterable*) – iterable that yields a valid matplotlib color

Returns list of patches that can be passed to the *handles* parameter in the *ax.legend* method

Return type list[546]

Module contents

New in version 0.1.14.

mgkit.snps package

Submodules

mgkit.snps.classes module

Manage SNP data.

[536] https://docs.python.org/3/library/functions.html#int
[537] https://docs.python.org/3/library/functions.html#int
[538] https://docs.python.org/3/library/functions.html#int
[539] https://docs.python.org/3/library/stdtypes.html#tuple
[540] https://docs.python.org/3/library/stdtypes.html#tuple
[541] https://docs.python.org/3/library/functions.html#int
[542] https://docs.python.org/3/library/stdtypes.html#tuple
[543] https://docs.python.org/3/library/stdtypes.html#str
[544] https://docs.python.org/3/library/functions.html#float
[545] https://docs.python.org/3/library/stdtypes.html#tuple
[546] https://docs.python.org/3/library/stdtypes.html#list

class mgkit.snps.classes.**GeneSNP**(*gene_id=''*, *taxon_id=0*, *exp_syn=0*, *exp_nonsyn=0*, *coverage=None*, *snps=None*, *uid=None*, *json_data=None*)

Bases: *mgkit.snps.classes.RatioMixIn*

New in version 0.1.13.

Class defining gene and synonymous/non-synonymous SNPs.

It defines background synonymous/non-synonymous attributes and only has a method right now, which calculate pN/pS ratio. The method is added through a mixin object, so the ratio can be customised and be shared with the old implementation.

uid
unique id for the isoform (to be referenced in a GFF file)

Type str[547]

gene_id
gene id

Type str[548]

taxon_id
gene taxon

Type int[549]

exp_syn
expected synonymous changes

Type int[550]

exp_nonsyn
expected non-synonymous changes

Type int[551]

coverage
gene coverage

Type int[552]

snps
list of SNPs associated with the gene, each element is a tuple with the position (relative to the gene start), the second is the nucleotidic change and the third is the aa SNP type as defined by *SNPType*.

Type list[553]

Note: The main difference with the `GeneSyn` is that all snps are kept and *syn* and *nonsyn* are not attributes but properties that return the count of synonymous and non-synonymous SNPs in the *snps* list.

Warning: This class uses more memory than `GeneSyn` because it doesn't use __slots__, it may be changed in later versions.

add(*other*)
Inplace addition of another instance values. No check for them being the same gene/taxon, it's up to the user to check that they can be added together.

[547] https://docs.python.org/3/library/stdtypes.html#str
[548] https://docs.python.org/3/library/stdtypes.html#str
[549] https://docs.python.org/3/library/functions.html#int
[550] https://docs.python.org/3/library/functions.html#int
[551] https://docs.python.org/3/library/functions.html#int
[552] https://docs.python.org/3/library/functions.html#int
[553] https://docs.python.org/3/library/stdtypes.html#list

Parameters **other** – instance of GeneSyn to add

add_snp(*position*, *change*, *snp_type=<SNPType.unknown: 0>*)
Adds a SNP to the list

Parameters

- **position** (*int*[554]) – SNP position, relative to the gene start
- **change** (*str*[555]) – nucleotidic change
- **snp_type** (*enum*) – one of the values defined in *SNPType*

coverage = None

exp_nonsyn = None

exp_syn = None

from_json(*data*)
Instantiate the instance with values from a json definition

Parameters **data** (*str*[556]) – json representation, as returned by *GeneSNP.to_json()*

gene_id = None

nonsyn
Returns the expected non-synonymous changes

snps = None

syn
Returns the expected synonymous changes

taxon_id = None

to_json()
Returns a json definition of the instance

Returns json representation of the instance

Return type str[557]

uid = None

class mgkit.snps.classes.**RatioMixIn**
Bases: object[558]

calc_ratio(*haplotypes=False*)
Changed in version 0.2.2: split the function to handle *flag_value* in another method

Calculate $\frac{pN}{pS}$ for the gene.

$$\frac{pN}{pS} = \frac{{}^{oN}/_{eN}}{{}^{oS}/_{eS}} \tag{7.3}$$

WHere:

- oN (number of non-synonymous - **nonsyn**)
- eN (expected number of non-synonymous - **exp_nonsyn**)
- oS (number of synonymous - **syn**)
- eS (expected number of synonymous - **exp_syn**)

Parameters

[554] https://docs.python.org/3/library/functions.html#int
[555] https://docs.python.org/3/library/stdtypes.html#str
[556] https://docs.python.org/3/library/stdtypes.html#str
[557] https://docs.python.org/3/library/stdtypes.html#str
[558] https://docs.python.org/3/library/functions.html#object

- **flag_value** (*bool*[559]) – when there's no way to calculate the ratio, the possible cases will be flagged with a negative number. This allows to make substitutions for these values
- **haplotypes** (*bool*[560]) – if true, coverage information is not used, because the SNPs are assumed to come from an alignment that has sequences having haplotypes

Returns

the $\frac{pN}{pS}$ for the gene.

Note: Because pN or pS can be 0, and the return value would be NaN, we take in account some special cases. The default return value in this cases is `numpy.nan`.

- Both synonymous and non-synonymous values are 0:
  - if both the syn and nonsyn attributes are 0 but there's coverage for this gene, we return a 0, as there's no evolution in this gene. Before, the coverage was checked by this method against either the passed *min_cov* parameter that was equal to `MIN_COV`. Now the case is for the user to check the coverage and functions in *mgkit.snps.conv_func* do that. If enough coverage was achieved, the *haplotypes* parameter can be used to return a 0

All other cases return a NaN value

Return type float[561]

calc_ratio_flag()

New in version 0.2.2.

Handles cases where it's important to flag the returned value, as explained in `GeneSNP.calc_ratio()`, and when the both the number of synonymous and non-synonymous is greater than 0, the pN/pS value is returned.

- **The number of non-synonymous is greater than 0 but the number of**

  synonymous is 0:

  - if **flag_value** is **True**, the returned value is **-1**

  - The number of synonymous is greater than 0 but the number of non-synonymous is 0:
    - if **flag_value** is **True**, the returned value is **-2**

oS	oN	return value
>0	>0	**pN/pS**
0	0	**-3**
>0	0	**-1**
0	>0	**-2**

class mgkit.snps.classes.**SNPType**

Bases: `enum.Enum`[562]

New in version 0.1.13.

Enum that defines SNP types. Supported at the moment:

- unknown = 0

[559] https://docs.python.org/3/library/functions.html#bool
[560] https://docs.python.org/3/library/functions.html#bool
[561] https://docs.python.org/3/library/functions.html#float
[562] https://docs.python.org/3/library/enum.html#enum.Enum

- syn (synonymous) = 1
- nonsyn (non-synonymous) = 2

Note: No support is planned at the moment to support indel mutations

```
nonsyn = 2
syn = 1
unknown = 0
```

mgkit.snps.conv_func module

Wappers to use some of the general function of the snps package in a simpler way.

mgkit.snps.conv_func.**get_full_dataframe**(*snp_data*, *taxonomy*, *min_num=3*, *index_type=None*, *filters=None*)

New in version 0.1.12.

Changed in version 0.2.2: added *filters* argument

Returns a `DataFrame` with the pN/pS of the given SNPs data.

Shortcut for using *combine_sample_snps()*, using filters from *get_default_filters()*.

Parameters

- **snp_data** (*dict*[563]) – dictionary sample->GeneSyn of SNPs data
- **taxonomy** – Uniprot Taxonomy
- **min_num** (*int*[564]) – minimum number of samples in which a valid pN/pS is found
- **index_type** (*str*[565], *None*[566]) – type of index to return
- **filters** (*iterable*) – list of filters to apply, otherwise uses the default filters

Returns `pandas.DataFrame` of pN/pS values. The index type is None (gene-taxon)

Return type DataFrame

mgkit.snps.conv_func.**get_gene_map_dataframe**(*snp_data*, *taxonomy*, *gene_map*, *min_num=3*, *index_type='gene'*, *filters=None*)

New in version 0.1.11.

Changed in version 0.2.2: added *filters* argument

Returns a `DataFrame` with the pN/pS of the given SNPs data, mapping all taxa to the gene map.

Shortcut for using *combine_sample_snps()*, using filters from *get_default_filters()* and as *gene_func* parameter *map_gene_id()*.

Parameters

- **snp_data** (*dict*[567]) – dictionary sample->GeneSyn of SNPs data
- **taxonomy** – Uniprot Taxonomy
- **min_num** (*int*[568]) – minimum number of samples in which a valid pN/pS is found

[563] https://docs.python.org/3/library/stdtypes.html#dict
[564] https://docs.python.org/3/library/functions.html#int
[565] https://docs.python.org/3/library/stdtypes.html#str
[566] https://docs.python.org/3/library/constants.html#None
[567] https://docs.python.org/3/library/stdtypes.html#dict
[568] https://docs.python.org/3/library/functions.html#int

- **gene_map** (*dict*[569]) – dictionary of mapping for the gene_ids in in SNPs data
- **index_type** (*str*[570], *None*[571]) – type of index to return
- **filters** (*iterable*) – list of filters to apply, otherwise uses the default filters

Returns pandas.DataFrame of pN/pS values. The index type is 'gene'

Return type DataFrame

mgkit.snps.conv_func.**get_gene_taxon_dataframe**(*snp_data*, *taxonomy*, *gene_map*, *min_num=3*, *rank='genus'*, *index_type=None*, *filters=None*)

New in version 0.1.12.

Changed in version 0.2.2: added *filters* argument

Todo: edit docstring

Returns a DataFrame with the pN/pS of the given SNPs data, mapping all taxa to the gene map.

Shortcut for using *combine_sample_snps()*, using filters from *get_default_filters()* and as *gene_func* parameter *map_gene_id()*.

Parameters

- **snp_data** (*dict*[572]) – dictionary sample->GeneSyn of SNPs data
- **taxonomy** – Uniprot Taxonomy
- **min_num** (*int*[573]) – minimum number of samples in which a valid pN/pS is found
- **gene_map** (*dict*[574]) – dictionary of mapping for the gene_ids in in SNPs data
- **index_type** (*str*[575], *None*[576]) – type of index to return
- **filters** (*iterable*) – list of filters to apply, otherwise uses the default filters

Returns pandas.DataFrame of pN/pS values. The index type is 'gene'

Return type DataFrame

mgkit.snps.conv_func.**get_rank_dataframe**(*snp_data*, *taxonomy*, *min_num=3*, *rank='order'*, *index_type='taxon'*, *filters=None*)

New in version 0.1.11.

Changed in version 0.2.2: added *filters* argument

Returns a DataFrame with the pN/pS of the given SNPs data, mapping all taxa to the specified rank. Higher taxa won't be included.

Shortcut for using *combine_sample_snps()*, using filters from *get_default_filters()* and as *taxon_func* parameter *map_taxon_id_to_rank()*, with include_higher equals to False

Parameters

- **snp_data** (*dict*[577]) – dictionary sample->GeneSyn of SNPs data
- **taxonomy** – Uniprot Taxonomy

[569] https://docs.python.org/3/library/stdtypes.html#dict
[570] https://docs.python.org/3/library/stdtypes.html#str
[571] https://docs.python.org/3/library/constants.html#None
[572] https://docs.python.org/3/library/stdtypes.html#dict
[573] https://docs.python.org/3/library/functions.html#int
[574] https://docs.python.org/3/library/stdtypes.html#dict
[575] https://docs.python.org/3/library/stdtypes.html#str
[576] https://docs.python.org/3/library/constants.html#None
[577] https://docs.python.org/3/library/stdtypes.html#dict

- **min_num** (*int*[578]) – minimum number of samples in which a valid pN/pS is found
- **rank** (*str*[579]) – taxon rank to map. Valid ranks are found in *mgkit.taxon.TAXON_RANKS*
- **index_type** (*str*[580], *None*[581]) – type of index to return
- **filters** (*iterable*) – list of filters to apply, otherwise uses the default filters

Returns `pandas.DataFrame` of pN/pS values. The index type is 'taxon'

Return type DataFrame

mgkit.snps.filter module

SNPs filtering functions

`mgkit.snps.filter.`**filter_genesyn_by_coverage**(*gene_syn*, *min_cov=None*)

Checks if the coverage of the provided *gene_syn* is at least *min_cov*

Parameters

- **gene_syn** – `GeneSyn` instance
- **min_cov** (*int*[582]) – minimum coverage allowed (included)

Returns True if the gene has enough coverage

Return type bool[583]

Raises `FilterFails` – if min_cov is None

`mgkit.snps.filter.`**filter_genesyn_by_gene_id**(*gene_syn*, *gene_ids=None*, *exclude=False*, *id_func=None*)

Checks if the gene_id is listed in the filter_list.

Parameters

- **gene_syn** – `GeneSyn` instance
- **gene_ids** (*iterable*) – list of gene IDs to include/exclude
- **exclude** (*bool*[584]) – if the filter is reversed

Returns if the exclude is True, the gene_id must appear in the gene_ids, if False, returns True only if gene_id is NOT in gene_ids.

Return type bool[585]

Raises `FilterFails` – if gene_ids is None

`mgkit.snps.filter.`**filter_genesyn_by_taxon_id**(*gene_syn*, *taxonomy=None*, *filter_list=None*, *exclude=False*, *func=None*)

Checks if the *taxon_id* attribute of *gene_syn* is the *filter_list*. Excelude reverses the result. If func is supplied, it's used to traverse the *taxonomy*.

Parameters

- **gene_syn** – `GeneSyn` instance
- **taxonomy** – a valid taxonomy (instance of *Taxonomy*)

[578] https://docs.python.org/3/library/functions.html#int
[579] https://docs.python.org/3/library/stdtypes.html#str
[580] https://docs.python.org/3/library/stdtypes.html#str
[581] https://docs.python.org/3/library/constants.html#None
[582] https://docs.python.org/3/library/functions.html#int
[583] https://docs.python.org/3/library/functions.html#bool
[584] https://docs.python.org/3/library/functions.html#bool
[585] https://docs.python.org/3/library/functions.html#bool

- **filter_list** (*iterable*) – list of taxon IDs to include/exclude
- **exclude** (*bool*[586]) – if the filter is reversed
- **func** (*func*) – *is_ancestor()*

Returns if the exclude is True, the gene_id must appear in the gene_ids, if False, returns True only if gene_id is NOT in gene_ids.

Return type bool[587]

Raises `FilterFails` – if filter_list is None or taxonomy is None and func is not None

`mgkit.snps.filter.`**get_default_filters**(*taxonomy*, ***kwargs*)

Retuns a list of filters that are used by default. it needs a valid taxonomy and gets the default arguments from *mgkit.consts.DEFAULT_SNP_FILTER*.

`mgkit.snps.filter.`**pipe_filters**(*iterable*, **funcs*)

Pipes a list of filter to iterable, using the python ifilter function in the itertools module. Now using *builtins.filter*

mgkit.snps.funcs module

Functions used in SNPs manipulation

`mgkit.snps.funcs.`**build_rank_matrix**(*dataframe*, *taxonomy=None*, *taxon_rank=None*)

Make a rank matrix from a `pandas.Series` with the pN/pS values of a dataset.

Parameters

- **dataframe** – `pandas.Series` instance with a MultiIndex (gene-taxon)
- **taxonomy** – `taxon.Taxonomy` instance with the full taxonomy
- **taxon_rank** (*str*[588]) – taxon rank to limit the specifity of the taxa included

Returns `pandas.DataFrame` instance

`mgkit.snps.funcs.`**combine_sample_snps**(*snps_data*, *min_num*, *filters*, *index_type=None*, *gene_func=None*, *taxon_func=None*, *use_uid=False*, *flag_values=False*, *haplotypes=True*, *store_uids=False*)

Changed in version 0.2.2: added *use_uid* argument

Changed in version 0.3.1: added *haplotypes*

Changed in version 0.4.0: added *store_uids*

Combine a dictionary sample->gene_index->GeneSyn into a `pandas.DataFrame`. The dictionary is first filtered with the functions in *filters*, mapped to different taxa and genes using *taxon_func* and *gene_func* respectively. The returned DataFrame is also filtered for each row having at least a *min_num* of not NaN values.

Parameters

- **snps_data** (*dict*[589]) – dictionary with the *GeneSNP* instances
- **min_num** (*int*[590]) – the minimum number of not NaN values necessary in a row to be returned
- **filters** (*iterable*) – iterable containing filter functions, a list can be found in *mgkit.snps.filter*

[586] https://docs.python.org/3/library/functions.html#bool
[587] https://docs.python.org/3/library/functions.html#bool
[588] https://docs.python.org/3/library/stdtypes.html#str
[589] https://docs.python.org/3/library/stdtypes.html#dict
[590] https://docs.python.org/3/library/functions.html#int

- **index_type** (*str*[591], *None*[592]) – if *None*, each row index for the DataFrame will be a MultiIndex with *gene* and *taxon* as elements. If the equals 'gene', the row index will be gene based and if 'taxon' will be taxon based
- **gene_func** (*func*) – a function to map a gene_id to a gene_map. See *mapper.map_gene_id()* for an example
- **taxon_func** (*func*) – a function to map a taxon_id to a list of IDs. See `mapper.map_taxon_id_to_rank` or `mapper.map_taxon_id_to_ancestor` for examples
- **use_uid** (*bool*[593]) – if True, uses the *GeneSNP.uid* instead of *GeneSNP.gene_id*
- **flag_values** (*bool*[594]) – if True, `mgkit.snps.classes.GeneSNP.calc_ratio_flag()` will be used, instead of `mgkit.snps.classes.GeneSNP.calc_ratio()`
- **haplotypes** (*bool*[595]) – if *flag_values* is False, and *haplotypes* is True, the 0/0 case will be returned as 0 instead of NaN
- **store_uids** (*bool*[596]) – if True a dictionary with the uid used for each cell (e.g. gene/taxon/sample)

Returns `pandas.DataFrame` with the pN/pS values for the input SNPs, with the columns being the samples. if *store_uids* is True the return value is a tuple (DataFrame, dict)

Return type DataFrame

`mgkit.snps.funcs.`**flat_sample_snps**(*snps_data*, *min_cov*)

New in version 0.1.11.

Adds all the values of a gene across all samples into one instance of `classes.GeneSNP`, giving the average gene among all samples.

Parameters

- **snps_data** (*dict*[597]) – dictionary with the instances of `classes.GeneSNP`
- **min_cov** (*int*[598]) – minimum coverage required for the each instance to be added

Returns the dictionary with only one key (*all_samples*), which can be used with *combine_sample_snps()*

Return type dict[599]

`mgkit.snps.funcs.`**group_rank_matrix**(*dataframe*, *gene_map*)

Group a rank matrix using a mapping, in the form map_id->ko_ids.

Parameters

- **dataframe** – instance of a rank matrix from *build_rank_matrix()*
- **gene_map** (*dict*[600]) – dictionary with the mapping

Returns `pandas.DataFrame` instance

`mgkit.snps.funcs.`**order_ratios**(*ratios*, *aggr_func=<function median>*, *reverse=False*, *key_filter=None*)

Given a dictionary of id->iterable where iterable contains the values of interest, the function uses aggr_func to sort (ascending by default) it and return a list with the key in the sorted order.

[591] https://docs.python.org/3/library/stdtypes.html#str
[592] https://docs.python.org/3/library/constants.html#None
[593] https://docs.python.org/3/library/functions.html#bool
[594] https://docs.python.org/3/library/functions.html#bool
[595] https://docs.python.org/3/library/functions.html#bool
[596] https://docs.python.org/3/library/functions.html#bool
[597] https://docs.python.org/3/library/stdtypes.html#dict
[598] https://docs.python.org/3/library/functions.html#int
[599] https://docs.python.org/3/library/stdtypes.html#dict
[600] https://docs.python.org/3/library/stdtypes.html#dict

Parameters

- **ratios** (*dict*[601]) – dictionary instance id->iterable
- **aggr_func** (*function*) – any function returning a value that can be used as a key in sorting
- **reverse** (*bool*[602]) – the default is ascending sorting (False), set to True to reverse key_filter: list of keys to use for ordering, if None, every key is used

Returns iterable with the sort order

mgkit.snps.funcs.**significance_test**(*dataframe*, *taxon_id1*, *taxon_id2*, *test_func=<function ks_2samp>*)

New in version 0.1.11.

Perform a statistical test on each gene distribution in two different taxa.

For each gene common to the two taxa, the distribution of values in all samples (columns) between the two specified taxa is tested.

Parameters

- **dataframe** – `pandas.DataFrame` instance
- **taxon_id1** – the first taxon ID
- **taxon_id2** – the second taxon ID
- **test_func** – function used to test, defaults to `scipy.stats.ks_2samp()`

Returns with all pvalues from the tests

Return type pandas.Series

mgkit.snps.funcs.**write_sign_genes_table**(*out_file*, *dataframe*, *sign_genes*, *taxonomy*, *gene_names=None*)

Write a table with the list of significant genes found in a dataframe, the significant gene list is the result of `wilcoxon_pairwise_test_dataframe()`.

Out_file the file name or file object to write the file

Dataframe the dataframe which was tested for significant genes

Sign_genes gene list that are significant

Taxonomy taxonomy object

Gene_names dictionary with the name of the the genes. Optional

mgkit.snps.mapper module

Mapping functions for SNPs - Should be move into an 'iterator' package to be shared with other modules?

mgkit.snps.mapper.**map_gene_id**(*gene_id*, *gene_map=None*)

Returns an iterator for all the values of a dictionary. if gene_id is not found in the gene_map, an empty iterator is returned.

Parameters

- **gene_id** (*immutable*) – gene_id or any other dictionary key.
- **gene_map** (*dict*[603]) – a dictionary in the form key->[v1, v2, .. vN]

Returns iterator (empty if gene_id is not in gene_map) with the values

Return type generator

[601] https://docs.python.org/3/library/stdtypes.html#dict
[602] https://docs.python.org/3/library/functions.html#bool
[603] https://docs.python.org/3/library/stdtypes.html#dict

mgkit.snps.mapper.**map_taxon_id_to_ancestor**(*taxon_id*, *anc_ids=None*, *func=None*)
Given a taxon_id and a list of ancestors IDs, returns an iterator with the IDs that are ancestors of taxon_id.

Parameters

- **taxon_id** (*int*[604]) – taxon ID to be mapped
- **anc_ids** (*iterable*) – taxon IDs to check for ancestry
- **func** – function used to check for ancestry - partial function for *mgkit.taxon.is_ancestor()* that accepts taxon_id and anc_id

Returns iterator with the values or empty

Return type generator

Note: check *mgkit.filter.taxon.filter_taxon_by_id_list()* for examples on using func

mgkit.snps.mapper.**map_taxon_id_to_rank**(*taxon_id*, *rank=None*, *taxonomy=None*, *include_higher=False*)
Given a taxon_id, returns an iterator with only the element that correspond to the requested rank. If the taxon returned by *mgkit.taxon.Taxonomy.get_ranked_taxon* has a different rank than requested, the iterator will be empty if *include_higher* is False and the returned taxon ID if True.

Parameters

- **taxon_id** (*int*[605]) – taxon ID to be mapped
- **rank** (*str*[606]) – taxon rank used (*mgkit.taxon.TAXON_RANKS*)
- **include_higher** (*bool*[607]) – determines if a rank higher than the one requested is to be returned

Returns iterator with the values or empty

Return type generator

Module contents

SNPs data package

mgkit.utils package

Submodules

mgkit.utils.common module

Utility functions

mgkit.utils.common.**apply_func_window**(*func*, *data*, *window*, *step=0*)

mgkit.utils.common.**average_length**(*a1s*, *a1e*, *a2s*, *a2e*)
Given two sets of coordinates, a1 and a2, returns the average length.

Parameters

- **a1s** (*int*[608]) – a1 leftmost number

[604] https://docs.python.org/3/library/functions.html#int
[605] https://docs.python.org/3/library/functions.html#int
[606] https://docs.python.org/3/library/stdtypes.html#str
[607] https://docs.python.org/3/library/functions.html#bool
[608] https://docs.python.org/3/library/functions.html#int

- **a1e** (*int*[609]) – a1 rightmost number
- **a2s** (*int*[610]) – a2 leftmost number
- **a2e** (*int*[611]) – a2 rightmost number

Return float the average length

mgkit.utils.common.**between**(*pos*, *start*, *end*)

Tests if a number is between two others

Parameters

- **pos** (*int*[612]) – number to test
- **start** (*int*[613]) – leftmost number
- **end** (*int*[614]) – rightmost number

Return bool if the number is between start and end

mgkit.utils.common.**complement_ranges**(*intervals*, *end=None*)

New in version 0.3.1.

Perform a complement operation of the list of intervals, i.e. returning the ranges (tuples) that are not included in the list of intervals. *union_ranges()* is first called on the intervals.

Note: the *end* parameter is there for cases where the ranges passed don't cover the whole space. Assuming a list of ranges from annotations on a nucleotidic sequence, if the last range doesn't include the last position of the sequence, passing end equal to the length of the sequence will make the function include a last range that includes it

---

Parameters

- **intervals** (*intervals*) – iterable where each element is a closed range (tuple)
- **end** (*int*[615]) – if the end of the complement intervals is supposed to be outside the last range.

Returns the list of intervals that complement the ones passed.

Return type list[616]

Examples

```
>>> complement_ranges([(1, 10), (11, 20), (25, 30)], end=100)
[(21, 24), (31, 100)]
>>> complement_ranges([(1, 10), (11, 20), (25, 30)])
[(21, 24)]
>>> complement_ranges([(0, 2), (3, 17), (18, 20)])
[]
>>> complement_ranges([(0, 2), (3, 17), (18, 20)], end=100)
[(21, 100)]
```

mgkit.utils.common.**deprecated**(*func*)

This is a decorator which can be used to mark functions as deprecated. It will result in a warning being emitted when the function is used.

[609] https://docs.python.org/3/library/functions.html#int
[610] https://docs.python.org/3/library/functions.html#int
[611] https://docs.python.org/3/library/functions.html#int
[612] https://docs.python.org/3/library/functions.html#int
[613] https://docs.python.org/3/library/functions.html#int
[614] https://docs.python.org/3/library/functions.html#int
[615] https://docs.python.org/3/library/functions.html#int
[616] https://docs.python.org/3/library/stdtypes.html#list

from https://wiki.python.org/moin/PythonDecoratorLibrary

mgkit.utils.common.**range_intersect**(*start1*, *end1*, *start2*, *end2*)

New in version 0.1.13.

Given two ranges in the form *(start, end)*, it returns the range that is the intersection of the two.

Parameters

- **start1** (*int*[617]) – start position for the first range
- **end1** (*int*[618]) – end position for the first range
- **start2** (*int*[619]) – start position for the second range
- **end2** (*int*[620]) – end position for the second range

Returns returns a tuple with the start and end position for the intersection of the two ranges, or *None* if the intersection is empty

Return type (None[621], tuple[622])

mgkit.utils.common.**range_substract**(*start1*, *end1*, *start2*, *end2*)

mgkit.utils.common.**ranges_length**(*ranges*)

New in version 0.1.12.

Given an iterable where each element is a range, a tuple whose elements are numbers with the first being less than or equal to the second, the function sums the lengths of all ranges.

> **Warning:** it's supposed to be used on intervals that were first passed to functions like *union_ranges()*. If values overlap, there the sum will be wrong

Parameters **ranges** (*iterable*) – each element is a tuple like *(1, 10)*

Returns sum of all ranges lengths

Return type int[623]

mgkit.utils.common.**union_range**(*start1*, *end1*, *start2*, *end2*)

New in version 0.1.12.

Changed in version 0.3.1: changed behaviour, since the intervals are meant to be closed

If two numeric ranges overlap, it returns the new range, otherwise None is returned. Works on both int and float numbers, even mixed.

Parameters

- **start1** (*numeric*) – start of range 1
- **end1** (*numeric*) – end of range 1
- **start2** (*numeric*) – start of range 2
- **end2** (*numeric*) – end of range 2

Returns union of the ranges or None if the ranges don't overlap

Return type (tuple[624] or None[625])

[617] https://docs.python.org/3/library/functions.html#int
[618] https://docs.python.org/3/library/functions.html#int
[619] https://docs.python.org/3/library/functions.html#int
[620] https://docs.python.org/3/library/functions.html#int
[621] https://docs.python.org/3/library/constants.html#None
[622] https://docs.python.org/3/library/stdtypes.html#tuple
[623] https://docs.python.org/3/library/functions.html#int
[624] https://docs.python.org/3/library/stdtypes.html#tuple
[625] https://docs.python.org/3/library/constants.html#None

Example

```
>>> union_range(10, 13, 1, 10)
(1, 13)
>>> union_range(1, 10, 11, 13)
(1, 13)
>>> union_range(1, 10, 12, 13)
None
```

mgkit.utils.common.**union_ranges**(*intervals*)

New in version 0.3.1.

From a list of ranges, assumed to be closed, performs a union of all elements.

Parameters **intervals** (*intervals*) – iterable where each element is a closed range (tuple)

Returns the list of ranges that are the union of all elements passed

Return type list[626]

Examples

```
>>> union_ranges([(1, 2), (3, 7), (6, 12), (9, 17), (18, 20)])
[(1, 20)]
>>> union_ranges([(1, 2), (3, 7), (6, 12), (9, 14), (18, 20)])
[(1, 14), (18, 20)]
```

mgkit.utils.dictionary module

Dictionary utils

class mgkit.utils.dictionary.**HDFDict**(*file_name*, *table*, *cast=<class 'int'>*, *cache=True*)

Bases: object[627]

Changed in version 0.3.3: added *cache* in __init__

New in version 0.3.1.

Used a table in a HDFStore (from pandas) as a dictionary. The table must be indexed to perform well. Read only.

Note: the dictionary cannot be modified and exception:*ValueError* will be raised if the table is not in the file

---

mgkit.utils.dictionary.**apply_func_to_values**(*dictionary*, *func*)

New in version 0.1.12.

Assuming a dictionary whose values are iterables, *func* is applied to each element of the iterable, retuning a *set* of all transformed elements.

Parameters

- **dictionary** (*dict*[628]) – dictionary whose values are iterables
- **func** (*func*) – function to apply to the dictionary values

Returns dictionary with transformed values

[626] https://docs.python.org/3/library/stdtypes.html#list
[627] https://docs.python.org/3/library/functions.html#object
[628] https://docs.python.org/3/library/stdtypes.html#dict

Return type dict[629]

class mgkit.utils.dictionary.**cache_dict_file**(*iterator*, *skip_lines=0*)

Bases: object[630]

New in version 0.3.0.

Used to cache the result of a function that yields a tuple (key and value). If the value is found in the internal dictionary (as the class behave), the correspondent value is returned, otherwise the iterator is advanced until the key is found.

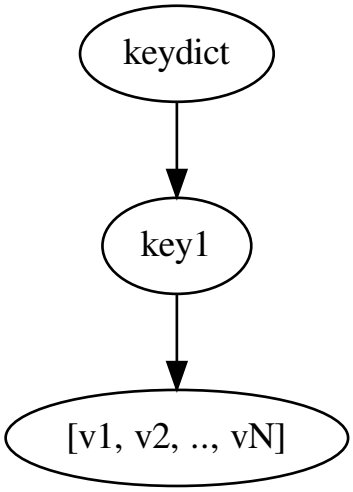
Example

```
>>> from mgkit.io.blast import parse_accession_taxa_table
>>> i = parse_accession_taxa_table('nucl_gb.accession2taxid.gz', key=0)
>>> d = cache_dict_file(i)
>>> d['AH001684']
4400
```

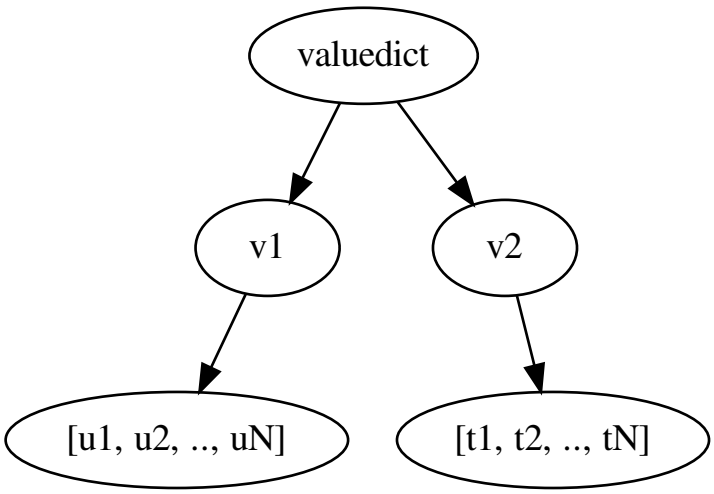
next()

mgkit.utils.dictionary.**combine_dict**(*keydict*, *valuedict*)

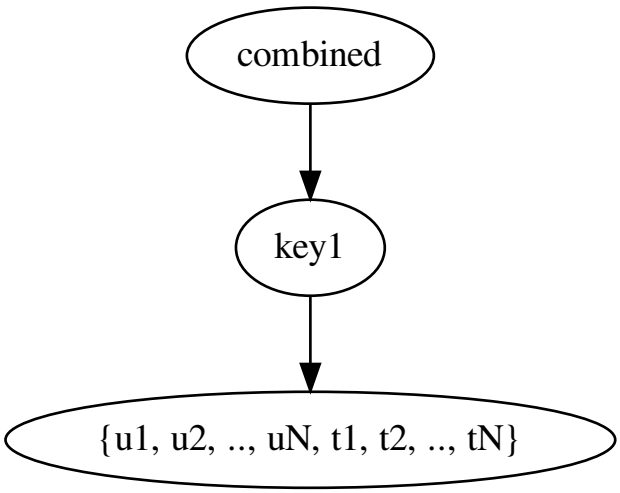
Combine two dictionaries when the values of keydict are iterables. The combined dictionary has the same keys as keydict and the its values are sets containing all the values associated to keydict values in valuedict.



[629] https://docs.python.org/3/library/stdtypes.html#dict
[630] https://docs.python.org/3/library/functions.html#object



Resulting dictionary will be



Parameters

- **keydict** (*dict*[631]) – dictionary whose keys are the same as the returned dictionary
- **valuedict** (*dict*[632]) – dictionary whose values are the same as the returned dictionary

Return dict combined dictionary

mgkit.utils.dictionary.**combine_dict_one_value**(*keydict*, *valuedict*)

Combine two dictionaries by the value of the keydict is used as a key in valuedict and the resulting dictionary

[631] https://docs.python.org/3/library/stdtypes.html#dict
[632] https://docs.python.org/3/library/stdtypes.html#dict

is composed of keydict keys and valuedict values.

Same as `comb_dict()`, but each value in keydict is a single element that is key in valuedict.

Parameters

- **keydict** (*dict*[633]) – dictionary whose keys are the same as the returned dictionary
- **valuedict** (*dict*[634]) – dictionary whose values are the same as the returned dictionary

Return dict combined dictionary

`mgkit.utils.dictionary.`**filter_nan**(*ratios*)
Returns a dictionary with the NaN values taken out

`mgkit.utils.dictionary.`**filter_ratios_by_numbers**(*ratios*, *min_num*)
Returns from a dictionary only the items for which the length of the iterables that is the value of the item, is equal or greater of min_num.

Parameters

- **ratios** (*dict*[635]) – dictionary key->list
- **min_num** (*int*[636]) – minimum number of elements in the value iterable

Return dict filtered dictionary

`mgkit.utils.dictionary.`**find_id_in_dict**(*s_id*, *s_dict*)
Finds a value 's_id' in a dictionary in which the values are iterables. Returns a list of keys that contain the value.

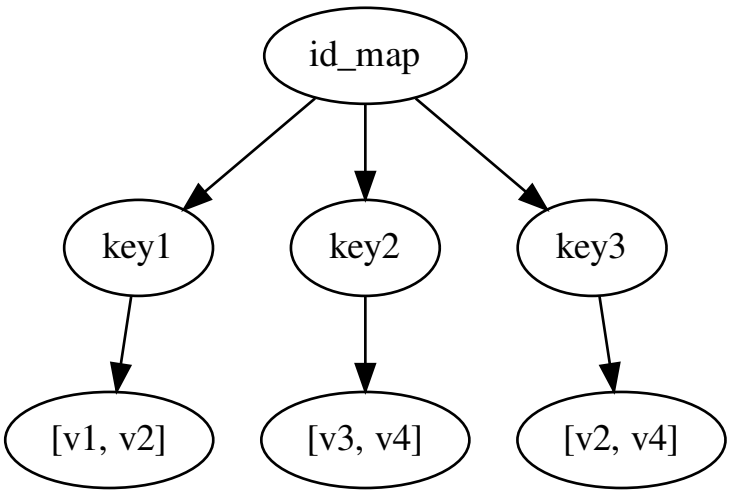
Parameters

- **s_id** (*dict*[637]) – element to look for in the dictionary's values
- **d** (*object*[638]) – dictionary to search in

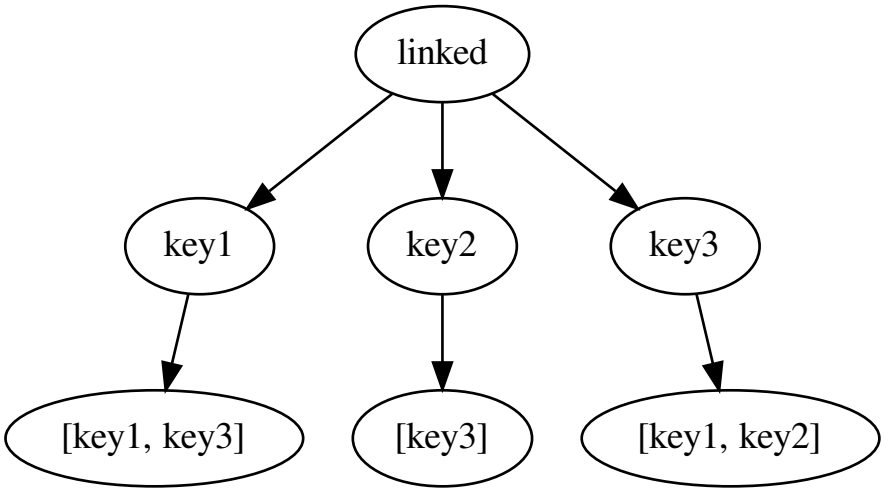
Return list list of keys in which d was found

`mgkit.utils.dictionary.`**link_ids**(*id_map*, *black_list=None*)
Given a dictionary whose values (iterables) can be linked back to other keys, it returns a dictionary in which the keys are the original keys and the values are sets of keys to which they can be linked.

[633] https://docs.python.org/3/library/stdtypes.html#dict
[634] https://docs.python.org/3/library/stdtypes.html#dict
[635] https://docs.python.org/3/library/stdtypes.html#dict
[636] https://docs.python.org/3/library/functions.html#int
[637] https://docs.python.org/3/library/stdtypes.html#dict
[638] https://docs.python.org/3/library/functions.html#object



Becomes:



Parameters

- **id_map** (*dict*[639]) – dictionary of keys to link
- **black_list** (*iterable*) – iterable of values to skip in making the links

Return dict linked dictionary

mgkit.utils.dictionary.**merge_dictionaries**(*dicts*)

New in version 0.3.1.

Merges keys and values from a list/iterable of dictionaries. The resulting dictionary's values are converted into sets, with the assumption that the values are one of the following: float, str, int, bool

[639] https://docs.python.org/3/library/stdtypes.html#dict

mgkit.utils.dictionary.**reverse_mapping**(*map_dict*)

Given a dictionary in the form: key->[v1, v2, .., vN], returns a dictionary in the form: v1->[key1, key2, .., keyN]

Parameters **map_dict** (*dict*[640]) – dictionary to reverse

Return dict reversed dictionary

mgkit.utils.dictionary.**split_dictionary_by_value**(*value_dict*, *threshold*, *aggr_func=<function median>*, *key_filter=None*)

Splits a dictionary, whose values are iterables, based on a threshold:

- one in which the result of aggr_func is lower than the threshold (first)
- one in which the result of aggr_func is equal or greater than the threshold (second)

Parameters

- **valuedict** (*dict*[641]) – dictionary to be splitted
- **threshold** (*number*) – must be comparable to threshold
- **aggr_func** (*func*) – function used to aggregate the dictionary values
- **key_filter** (*iterable*) – if specified, only these key will be in the resulting dictionary

Returns two dictionaries

mgkit.utils.sequence module

Module containing functions related to sequence data

Note: For those functions without a docstring, look at the same with a underscore ('_') prepended.

class mgkit.utils.sequence.**Alignment**(*seqs=None*)

Bases: object[642]

Simple alignment class

add_seq(*name*, *seq*)

Add a sequence to the alignment

Parameters

- **name** (*str*[643]) – name of the sequence
- **seq** (*str*[644]) – sequence

add_seqs(*seqs*)

Add sequences to the alignment

Parameters **seqs** (*iterable*) – iterable that returns (name, seq)

get_consensus(*nucl=True*)

Changed in version 0.1.16: added *nucl* parameter

The consensus sequence is constructed by checking the nucleotide that has the maximum number of counts for each position in the alignment.

[640] https://docs.python.org/3/library/stdtypes.html#dict
[641] https://docs.python.org/3/library/stdtypes.html#dict
[642] https://docs.python.org/3/library/functions.html#object
[643] https://docs.python.org/3/library/stdtypes.html#str
[644] https://docs.python.org/3/library/stdtypes.html#str

Parameters **nucl** (*bool*[645]) – specify if the alignment is nucleotidic

Returns consensus sequence

Return type str[646]

get_position(*pos*)
Get all characters at a position

Parameters **pos** (*int*[647]) – position to return (0-based)

Return str all characters occuring at the position

get_seq_len()
Get the length of the alignment

get_snps(*ref_seq=None*, *full_size=False*)
A SNP is called for the nucleotide that has the most counts among the ones that differ in the each site of the alignment. If two nucleotides have the same maximum count, one is randomly chosen.

Parameters

- **ref_seq** (*str*[648]) – a reference sequence can be provided, if None, a consensus sequence is produced for the alignment
- **full_size** (*bool*[649]) – if True a tuple is returned for each position in the alignment. If there is no SNP at a position the value for the SNP is None

Return list a list of tuples (position, SNP)

mgkit.utils.sequence.**REV_COMP = {'A': 'T', 'C': 'G', 'G': 'C', 'T': 'A'}**
Dictionary containing the complement of each nucleotide sequence

mgkit.utils.sequence.**TRANS_TABLE = {'AAA': 'K', 'AAC': 'N', 'AAG': 'K', 'AAT': 'N', 'ACA**
Translation table - Universal genetic code

mgkit.utils.sequence.**_get_kmers**(*seq*, *k*)
New in version 0.2.6.

Returns a generator, with every iteration yielding a kmer of size *k*

Parameters

- **seq** (*str*[650]) – sequence
- **k** (*int*[651]) – kmer size

Yields *str* – a portion of *seq*, of size *k* with a step of *1*

mgkit.utils.sequence.**_sequence_signature**(*seq*, *w_size*, *k_size=4*, *step=None*)
New in version 0.2.6.

Returns the signature of a sequence, based on a kmer length, over a sliding window. Each sliding window signature is placed in order into a list, with each element being a collections.Counter[652] instance whose keys are the kmer found in that window.

Parameters

- **seq** (*str*[653]) – sequence for which to get the signature
- **w_size** (*int*[654]) – size of the sliding window size

[645] https://docs.python.org/3/library/functions.html#bool
[646] https://docs.python.org/3/library/stdtypes.html#str
[647] https://docs.python.org/3/library/functions.html#int
[648] https://docs.python.org/3/library/stdtypes.html#str
[649] https://docs.python.org/3/library/functions.html#bool
[650] https://docs.python.org/3/library/stdtypes.html#str
[651] https://docs.python.org/3/library/functions.html#int
[652] https://docs.python.org/3/library/collections.html#collections.Counter
[653] https://docs.python.org/3/library/stdtypes.html#str
[654] https://docs.python.org/3/library/functions.html#int

- **k_size** (*int*[655]) – size of the kmer to use `get_kmers()`
- **step** (*int*[656]) – step to use in `sliding_window()`

Returns a list of `collections.Counter`[657] instances, for each window used

Return type list[658]

`mgkit.utils.sequence._signatures_matrix`(*seqs*, *w_size*, *k_size=4*, *step=None*)

New in version 0.2.6.

Return a matrix (pandas.DataFrame) where the columns are the kmer found in all sequences *seqs* and the rows are the a MultiIndex with the first level being the sequnce name and the second the index of the sliding window for which a signature was computed.

Parameters

- **seqs** (*iterable*) – iterable that yields a tuple, with the first element being the sequence name and the second the sequence itself
- **w_size** (*int*[659]) – size of the sliding window size
- **k_size** (*int*[660]) – size of the kmer to use `get_kmers()`
- **step** (*int*[661]) – step to use in `sliding_window()`, defaults to half of the window size

Returns a DataFrame where the columns are the kmers and the rows are the signatures of each contigs/windows.

Return type pandas.DataFrame

`mgkit.utils.sequence._sliding_window`(*seq*, *size*, *step=None*)

New in version 0.2.6.

Returns a generator, with every iteration yielding a subsequence of size *size*, with a step of *step*.

Parameters

- **seq** (*str*[662]) – sequnece
- **size** (*int*[663]) – size of the sliding window
- **step** (*int*[664], *None*[665]) – the step to use in the sliding window. If *None*, half of the sequence length is used

Yields *str* – a subsequence of size *size* and step *step*

`mgkit.utils.sequence.calc_n50`(*seq_lengths*)

Calculate the N50 statistics for a `numpy.array` of sequence lengths.

The algorithm finds in the supplied array the element (contig length) for which the sum all contig lengths equal or greater than it is equal to half of all assembled base pairs.

Parameters **seq_lengths** (*array*) – an instance of a numpy array containing the sequence lengths

Return int the N50 statistics value

655 https://docs.python.org/3/library/functions.html#int
656 https://docs.python.org/3/library/functions.html#int
657 https://docs.python.org/3/library/collections.html#collections.Counter
658 https://docs.python.org/3/library/stdtypes.html#list
659 https://docs.python.org/3/library/functions.html#int
660 https://docs.python.org/3/library/functions.html#int
661 https://docs.python.org/3/library/functions.html#int
662 https://docs.python.org/3/library/stdtypes.html#str
663 https://docs.python.org/3/library/functions.html#int
664 https://docs.python.org/3/library/functions.html#int
665 https://docs.python.org/3/library/constants.html#None

mgkit.utils.sequence.**check_snp_in_seq**(*ref_seq*, *pos*, *change*, *start=0*, *trans_table=None*)
Check a SNP in a reference sequence if it is a synonymous or non-synonymous change.

Parameters

- **ref_seq** (*str*[666]) – reference sequence
- **pos** (*int*[667]) – SNP position - it is expected to be a 1 based index
- **change** (*str*[668]) – nucleotide change occuring at *pos*
- **start** (*int*[669]) – the starting position for the coding region - 0 based index
- **trans_table** (*dict*[670]) – translation table used - codon->AA

Return bool True if it is a synonymous change, False if non-synonymous

mgkit.utils.sequence.**convert_aa_to_nuc_coord**(*start*, *end*, *frame=0*)
Converts aa coordinates to nucleotidic ones. The coordinates must be from '+' strand. For the '-' strand, use *reverse_aa_coord()* first.

Parameters

- **start** (*int*[671]) – start of the annotation (lowest number)
- **end** (*int*[672]) – end of the annotation (highest number)
- **frame** (*int*[673]) – frame of the AA translation (0, 1 or 2)

Returns the first element is the converted *start* and the second element is the converted *end*

Return type tuple[674]

Note: the coordinates are assumed to be 1-based indices

---

mgkit.utils.sequence.**extrapolate_model**(*quals*, *frac=0.5*, *scale_adj=0.5*)
New in version 0.3.3.

Extrapolate a quality model from a list of qualities. It uses internally a LOWESS as the base, which is used to estimate the noise as a normal distribution.

Parameters

- **quals** (*list*[675]) – list of arrays of qualities, sorted by position in the corresponding sequence
- **frac** (*float*[676]) – fraction of the data used for the LOWESS fit (uses statsmodels)
- **scale_adj** (*float*[677]) – value by which the scale of the normal distribution will be multiplied. Defaults to halving the scale

Returns the first element is the qualities fit with a LOWESS, the second element is the distribution

Return type tuple[678]

[666] https://docs.python.org/3/library/stdtypes.html#str
[667] https://docs.python.org/3/library/functions.html#int
[668] https://docs.python.org/3/library/stdtypes.html#str
[669] https://docs.python.org/3/library/functions.html#int
[670] https://docs.python.org/3/library/stdtypes.html#dict
[671] https://docs.python.org/3/library/functions.html#int
[672] https://docs.python.org/3/library/functions.html#int
[673] https://docs.python.org/3/library/functions.html#int
[674] https://docs.python.org/3/library/stdtypes.html#tuple
[675] https://docs.python.org/3/library/stdtypes.html#list
[676] https://docs.python.org/3/library/functions.html#float
[677] https://docs.python.org/3/library/functions.html#float
[678] https://docs.python.org/3/library/stdtypes.html#tuple

mgkit.utils.sequence.**get_contigs_info**(*file_name*, *pp=False*)

Changed in version 0.2.4: file_name can be a *dict* name->seq or a list of sequences

New in version 0.2.1.

Given a file name for a fasta file with sequences, a dictionary of name->seq, or a list of sequences, returns the following information in a tuple, or a string if *pp* is True:

- number of sequences
- total base pairs
- max length
- min length
- average length
- N50 statistic

Parameters

- **file_name** (*str*[679]) – fasta file to open
- **pp** (*bool*[680]) – if True, a formatted string is returned

Returns the returned value depends on the value of *pp*, if True a formatted string is returned, otherwise the tuple with all values is.

Return type str[681], tuple[682]

mgkit.utils.sequence.**get_seq_expected_syn_count**(*seq*, *start=0*, *syn_matrix=None*)

Calculate the expected number of synonymous and non-synonymous changes in a nucleotide sequence. Assumes that the sequence is already in the correct frame and its length is a multiple of 3.

Parameters

- **seq** (*iterable*) – nucleotide sequence (uppercase chars)
- **start** (*int*[683]) – frame of the sequence
- **syn_matrix** (*dict*[684]) – dictionary that contains the expected number of changes for a codon, as returned by *get_syn_matrix()*

Return tuple tuple with counts of expected counts (syn, nonsyn)

mgkit.utils.sequence.**get_seq_number_of_syn**(*ref_seq*, *snps*, *start=0*, *trans_table=None*)

Given a reference sequence and a list of SNPs, calculates the number of synonymous and non-synonymous SNP.

Parameters

- **ref_seq** (*str*[685]) – reference sequence
- **snps** (*iterable*) – list of tuples (position, SNP) - zero based index
- **start** (*int*[686]) – the frame used for the reference {0, 1, 2}
- **trans_table** (*dict*[687]) – translation table used - codon->AA

Return tuple synonymous and non-synonymous counts

[679] https://docs.python.org/3/library/stdtypes.html#str
[680] https://docs.python.org/3/library/functions.html#bool
[681] https://docs.python.org/3/library/stdtypes.html#str
[682] https://docs.python.org/3/library/stdtypes.html#tuple
[683] https://docs.python.org/3/library/functions.html#int
[684] https://docs.python.org/3/library/stdtypes.html#dict
[685] https://docs.python.org/3/library/stdtypes.html#str
[686] https://docs.python.org/3/library/functions.html#int
[687] https://docs.python.org/3/library/stdtypes.html#dict

mgkit.utils.sequence.**get_syn_matrix**(*trans_table=None*, *nuc_list=None*)
Returns a dictionary containing the expected count of synonymous and non-synonymous changes that a codon can have if one base is allowed to change at a time.

There are 9 possible changes per codon.

Parameters

- **trans_table** (*dict*[688]) – a tranlation table, defaults to `seq_utils.TRANS_TABLE`
- **nuc_list** (*iterable*) – a list of nucleotides in which a base can change, default to the keys of `seq_utils.REV_COMP`

Return dict returns a dictionary in which for each codon a dictionary {'syn': 0, 'nonsyn': 0} holds the number of expected changes

mgkit.utils.sequence.**get_syn_matrix_all**(*trans_table=None*)
Same as *get_syn_matrix()* but a codon can change in any of the ones included in trans_table.

There are 63 possible changes per codon.

mgkit.utils.sequence.**get_variant_sequence**(*seq*, **snps*)
New in version 0.1.16.

Return a sequence changed in the positions requested.

Parameters

- **seq** (*str*[689]) – a sequence
- ***snps** (*tuple*[690]) – each argument passed is a tuple with the first element as a position in the sequence (1-based index) and the second element is the character to substitute in the sequence

Returns string with the changed characters

Return type str[691]

Example

```
>>> get_variant_sequence('ACTGATATATGCGCGCATCT', (1, 'C'))
'CCTGNTGTATGCGCGCATCT'
```

Note: It is used for nucleotide sequences, but it is valid to use any string

mgkit.utils.sequence.**make_reverse_table**(*tbl=None*)
Makes table to reverse complement a sequence by *reverse_complement()*. The table used is the complement for each nucleotide, defaulting to *REV_COMP*

mgkit.utils.sequence.**put_gaps_in_nuc_seq**(*nuc_seq*, *aa_seq*, *trim=True*)
Match the gaps in an amino-acid aligned sequence to its original nucleotide sequence. If the nucleotide sequence is not a multiple of 3, the trim option by default trim those bases from the output.

Parameters

- **nuc_seq** (*str*[692]) – original nucleotide sequence
- **aa_seq** (*str*[693]) – aligned amino-acid sequence

[688] https://docs.python.org/3/library/stdtypes.html#dict
[689] https://docs.python.org/3/library/stdtypes.html#str
[690] https://docs.python.org/3/library/stdtypes.html#tuple
[691] https://docs.python.org/3/library/stdtypes.html#str
[692] https://docs.python.org/3/library/stdtypes.html#str
[693] https://docs.python.org/3/library/stdtypes.html#str

- **trim** (*bool*[694]) – if True trim last nucleotide(s)

Return str gapped nucleotide sequence

mgkit.utils.sequence.**qualities_model_constant**(*length=150*, *scale=1*, *loc=35*)
New in version 0.3.3.

Model with constant quality

Parameters

- **length** (*int*[695]) – length of the qualities
- **scale** (*float*[696]) – base level of the qualities
- **loc** (*float*[697]) – loc parameter of the normal distribution

Returns first element is sequence qualities, the second element contains the distribution used to randomise them

Return type tuple[698]

mgkit.utils.sequence.**qualities_model_decrease**(*length=150*, *scale=None*, *loc=35*)
New in version 0.3.3.

The model is a decreasing one, from 35 and depends on the length of the sequence.

Parameters

- **length** (*int*[699]) – length of the qualities
- **scale** (*float*[700]) – base level of the qualities
- **loc** (*float*[701]) – loc parameter of the normal distribution

Returns first element is sequence qualities, the second element contains the distribution used to randomise them

Return type tuple[702]

mgkit.utils.sequence.**random_qualities**(*n=1*, *length=150*, *model=None*)
New in version 0.3.3.

Parameters

- **n** (*int*[703]) – number of quality arrays to yield
- **length** (*int*[704]) – length of the quality array
- **model** (*tuple*[705]) – a tuple specifying the qualities and error distribution, if *None* *qualities_model_decrease()* is used

Yields *numpy.array* – numpy array of qualities, with the maximum value of 40

mgkit.utils.sequence.**random_sequences**(*n=1*, *length=150*, *p=None*)
New in version 0.3.3.

Returns an iterator of random squences, where each nucleotide probability can be customised in the order (A, C, T, G)

Parameters

[694] https://docs.python.org/3/library/functions.html#bool
[695] https://docs.python.org/3/library/functions.html#int
[696] https://docs.python.org/3/library/functions.html#float
[697] https://docs.python.org/3/library/functions.html#float
[698] https://docs.python.org/3/library/stdtypes.html#tuple
[699] https://docs.python.org/3/library/functions.html#int
[700] https://docs.python.org/3/library/functions.html#float
[701] https://docs.python.org/3/library/functions.html#float
[702] https://docs.python.org/3/library/stdtypes.html#tuple
[703] https://docs.python.org/3/library/functions.html#int
[704] https://docs.python.org/3/library/functions.html#int
[705] https://docs.python.org/3/library/stdtypes.html#tuple

- **n** (*int*[706]) – number of sequences to yield
- **length** (*int*[707]) – length of each sequence
- **p** (*tuple*[708]) – tuple with the probability of a nucleotide to occur, in the order A, C, T, G

Yields *str* – string representing a nucleotidic sequence

mgkit.utils.sequence.**random_sequences_codon**(*n=1, length=150, codons=['TTT', 'TCT', 'TAT', 'TGT', 'TTC', 'TCC', 'TAC', 'TGC', 'TTA', 'TCA', 'TAA', 'TGA', 'TTG', 'TCG', 'TAG', 'TGG', 'CTT', 'CCT', 'CAT', 'CGT', 'CTC', 'CCC', 'CAC', 'CGC', 'CTA', 'CCA', 'CAA', 'CGA', 'CTG', 'CCG', 'CAG', 'CGG', 'ATT', 'ACT', 'AAT', 'AGT', 'ATC', 'ACC', 'AAC', 'AGC', 'ATA', 'ACA', 'AAA', 'AGA', 'ATG', 'ACG', 'AAG', 'AGG', 'GTT', 'GCT', 'GAT', 'GGT', 'GTC', 'GCC', 'GAC', 'GGC', 'GTA', 'GCA', 'GAA', 'GGA', 'GTG', 'GCG', 'GAG', 'GGG'], p=None, frame=None*)

New in version 0.3.3.

Returns an iterator of nucleotidic sequences, based on a defined genetic code (passed as parameter, defaults to the universal one). The sequence if first sampled with replacement from the codon list, with a number of codons that covers the length chosen plus an additional one to allow a frame shift as set by *frame*

Note: If the probability (for each codon) are supplied, the number of sequences required to match those probabilities within a 10% margin of error is of at least 10.000 sequences, for 5% at leas 100.000

Parameters

- **n** (*int*[709]) – number of sequences to yield
- **length** (*int*[710]) – length of the sequences
- **codons** (*iterable*) – codons used when generating the sequences
- **p** (*tuple*[711]) – probability of each codon occurence, in the same order as *codons*
- **frame** (*int*[712] *or None*[713]) – used to define a specific frame shift occuring in the sequence (0 to 2) or a random one (if *None*)

Yields *str* – string representing a nucleotidic sequence

mgkit.utils.sequence.**reverse_aa_coord**(*start*, *end*, *seq_len*)

Used to reverse amino-acid coordinates when parsing an AA annotation on the - strand. Used when the BLAST or HMMER annotations use AA sequences.

Parameters

- **start** (*int*[714]) – start of the annotation

[706] https://docs.python.org/3/library/functions.html#int
[707] https://docs.python.org/3/library/functions.html#int
[708] https://docs.python.org/3/library/stdtypes.html#tuple
[709] https://docs.python.org/3/library/functions.html#int
[710] https://docs.python.org/3/library/functions.html#int
[711] https://docs.python.org/3/library/stdtypes.html#tuple
[712] https://docs.python.org/3/library/functions.html#int
[713] https://docs.python.org/3/library/constants.html#None
[714] https://docs.python.org/3/library/functions.html#int

- **end** (*int*[715]) – end of the annotation
- **seq_len** (*int*[716]) – aa sequence length

Returns reversed (from strand - to strand +) coordinates. The first element is the converted *start* and the second element is the converted *end*

Return type tuple[717]

Note:

- start and end are 1-based indices

---

mgkit.utils.sequence.**reverse_complement**(*seq*, *tbl={65: 'T', 67: 'G', 71: 'C', 84: 'A'}*)
Returns the reverse complement of a nucleotide sequence

Parameters

- **seq** (*str*[718]) – nucleotide sequence with uppercase characters
- **tbl** (*dict*[719]) – translation table returned by *make_reverse_table()*

Return str returns the reverse complement of a nucleotide sequence

mgkit.utils.sequence.**reverse_complement_old**(*seq*, *tbl=None*)
Returns the reverse complement of a nucleotide sequence

Parameters

- **seq** (*str*[720]) – nucleotide sequence with uppercase characters
- **tbl** (*dict*[721]) – dictionary of complement bases, like *REV_COMP*

Return str returns the reverse complement of a nucleotide sequence

mgkit.utils.sequence.**sequence_composition**(*sequence*, *chars=('A', 'T', 'C', 'G')*)
New in version 0.1.13.

Returns the number of occurences of each unique character in the sequence

Parameters

- **sequence** (*str*[722]) – sequence
- **chars** (*iterable, None*[723]) – iterable of the chars to test, default to (A, C, T, G). if None checks all unique characters in the sequencce

Yields *tuple* – the first element is the nucleotide and the second is the number of occurences in *sequence*

mgkit.utils.sequence.**sequence_gc_content**(*sequence*)
Changed in version 0.3.3: in case of *ZeroDivisionError* returns .5

New in version 0.1.13.

Calculate GC content information for an annotation. The formula is:

$$\frac{(G+C)}{(G+C+A+T)} \tag{7.4}$$

Parameters **sequence** (*str*[724]) – sequence

[715] https://docs.python.org/3/library/functions.html#int
[716] https://docs.python.org/3/library/functions.html#int
[717] https://docs.python.org/3/library/stdtypes.html#tuple
[718] https://docs.python.org/3/library/stdtypes.html#str
[719] https://docs.python.org/3/library/stdtypes.html#dict
[720] https://docs.python.org/3/library/stdtypes.html#str
[721] https://docs.python.org/3/library/stdtypes.html#dict
[722] https://docs.python.org/3/library/stdtypes.html#str
[723] https://docs.python.org/3/library/constants.html#None
[724] https://docs.python.org/3/library/stdtypes.html#str

Returns GC content

Return type float[725]

mgkit.utils.sequence.**sequence_gc_ratio**(*sequence*)
New in version 0.1.13.

Calculate GC ratio information for a sequence. The formula is:

$$\frac{(A+T)}{(G+C)} \tag{7.5}$$

Parameters **sequence** (*str*[726]) – sequence

Returns GC ratio, or *numpy.nan* if G = C = 0

Return type float[727]

mgkit.utils.sequence.**translate_sequence**(*sequence*, *start=0*, *tbl=None*, *reverse=False*)
Translate a nucleotide sequence in an amino acid one.

Parameters

- **sequence** (*str*[728]) – sequence to translate, it's expected to be all caps
- **start** (*int*[729]) – 0-based index for the translation to start
- **tbl** (*dict*[730]) – dictionary with the translation for each codon
- **reverse** (*bool*[731]) – if True, *reverse_complement()* will be called and the returned sequence translated

Return str the translated sequence

mgkit.utils.trans_tables module

The module contains translation tables

Not all genetic codes are included, taken from: http://www.ncbi.nlm.nih.gov/Taxonomy/Utils/wprintgc.cgi?mode=t#SG2

mgkit.utils.trans_tables.**BAC_PLT = {'AAA': 'K', 'AAC': 'N', 'AAG': 'K', 'AAT': 'N', 'ACA**
The Bacterial and Plant Plastid Code transl_table=11

mgkit.utils.trans_tables.**DRS_MIT = {'AAA': 'K', 'AAC': 'N', 'AAG': 'K', 'AAT': 'N', 'ACA**
Drosophila Mithocondrion genome lacks a codon, compare to vertebrates

mgkit.utils.trans_tables.**INV_MIT = {'AAA': 'K', 'AAC': 'N', 'AAG': 'K', 'AAT': 'N', 'ACA**
The Invertebrate Mitochondrial Code transl_table=5

mgkit.utils.trans_tables.**PRT_MIT = {'AAA': 'K', 'AAC': 'N', 'AAG': 'K', 'AAT': 'N', 'ACA**
The Mold, Protozoan, and Coelenterate Mitochondrial Code and the Mycoplasma/Spiroplasma Code transl_table=4

mgkit.utils.trans_tables.**UNIVERSAL = {'AAA': 'K', 'AAC': 'N', 'AAG': 'K', 'AAT': 'N', 'A**
Universal genetic code transl_table=1

mgkit.utils.trans_tables.**YST_ALT = {'AAA': 'K', 'AAC': 'N', 'AAG': 'K', 'AAT': 'N', 'ACA**
The Alternative Yeast Nuclear Code transl_table=12

[725] https://docs.python.org/3/library/functions.html#float
[726] https://docs.python.org/3/library/stdtypes.html#str
[727] https://docs.python.org/3/library/functions.html#float
[728] https://docs.python.org/3/library/stdtypes.html#str
[729] https://docs.python.org/3/library/functions.html#int
[730] https://docs.python.org/3/library/stdtypes.html#dict
[731] https://docs.python.org/3/library/functions.html#bool

Module contents

Package that contains utility functions/classes

mgkit.workflow package

Submodules

mgkit.workflow.add_gff_info module

Add more information to GFF annotations: gene mappings, coverage, taxonomy, etc..

Uniprot Command

If the *gene_id* of an annotation is a Uniprot ID, the script queries Uniprot for the requested information. At the moment the information that can be added is the taxon_id, taxon_name, lineage and mapping to EC, KO, eggNOG IDs.

It's also possible to add mappings to other databases using the *-m* option with the correct identifier for the mapping, which can be found at this page[732]; for example if it's we want to add the mappings of uniprot IDs to *BioCyc*, in the *abbreviation* column of the mappings we find that it's identifier is *REACTOME_ID*, so we pass *-m REACTOME* to the script (leaving *_ID* out). Mapped IDs are separated by commas.

The taxonomy IDs are not overwritten if they are found in the annotations, the *-f* is provided to force the overwriting of those values.

See also *MGKit GFF Specifications* for more informations about the GFF specifications used.

Note: As the script needs to query Uniprot a lot, it is recommended to split the GFF in several files, so an error in the connection doesn't waste time.

However, a cache is kept to reduce the number of connections

Coverage Command

Adds coverage information from BAM alignment files to a GFF file, using the function `mgkit.align.add_coverage_info()`, the user needs to supply for each sample a BAM file, using the *-a* option, whose parameter is in the form *sample,samplealg.bam*. More samples can be supplied adding more *-a* arguments.

Hint: As an example, to add coverage for *sample1*, *sample2* the command line is:

```
add-gff-info coverage -a sample1,sample1.bam -a sample2,sample2.bam \
inputgff outputgff
```

A total coverage for the annotation is also calculated and stored in the *cov* attribute, while each sample coverage is stored into *sample_cov* as per *MGKit GFF Specifications*.

Adding Coverage from samtools depth

The *cov_samtools* allows the use of the output of *samtools* **depth** command. The *-aa* options must be used to pass information about all base pairs and sequences coverage in the BAM/SAM file. The command work similarly to

[732] http://www.uniprot.org/faq/28

coverage, accepting compressed *depth* files as well. If only one *depth* file is passed and no sample is passed, the attribute in the GFF will be *cov*, otherwise the attribute will be *sample1_cov*, *sample2_cov*, etc.

To create samtools *depth* files, this command must be used:

```
$ samtools depth -aa bam_file
```

Uniprot Offline Mappings

Similar to the *uniprot* command, it uses the idmapping[733] file provided by Uniprot, which speeds up the process of adding mappings and taxonomy IDs from Uniprot gene IDs. It's not possible tough to add *EC* mappings with this command, as those are not included in the file.

Kegg Information

The *kegg* command allows to add information to each annotation. Right now the information that can be added is restricted to the pathway(s) (reference KO) a KO is part of and both the KO and pathway(s) descriptions. This information is stored in keys starting with **ko_**.

Expected Aminoacidic Changes

Some scripts, like *snp_parser - SNPs analysis*, require information about the expected number of synonymous and non-synonymous changes of an annotation. This can be done using `mgkit.io.gff.Annotation.add_exp_syn_count()` by the user of the command *exp_syn* of this script. The attributes added to each annotation are explained in the *MGKit GFF Specifications*

Adding Count Data

Count data on a per-sample basis can be added with the *counts* command. The accepted inputs are from HTSeq-count and featureCounts. The ouput produced by featureCounts, is the one from using its **-f** option must be used.

This script accept by default a tab separated file, with a uid in the first column and the other columns are the counts for each sample, in the same order as they are passed to the **-s** option. To use the featureCounts file format, this script **-e** option must be used.

The sample names must be provided in the same order as the columns in the input files. If the counts are FPKMS the *-f* option can be used.

Adding Taxonomy from a Table

There are cases where it may needed or preferred to add the taxonomy from a *gene_id* already provided in the GFF file. For such cases the *addtaxa* command can be used. It works in a similar way to the *taxonomy* command, only it expect three different type of inputs:

- *GI-Taxa* table from NCBI (e.g. gi_taxid_nucl.dmp,)
- tab separated table
- dictionary
- HDF5

The first two are tab separated files, where on each line, the first column is the *gene_id* that is found in the first column, while the second if the *taxon_id*.

[733] ftp://ftp.uniprot.org/pub/databases/uniprot/current_release/knowledgebase/idmapping/idmapping.dat.gz

The third option is a serialised Python *dict*/hash table, whose keys are the *gene_id* and the value is that gene corresponding *taxon_id*. The serialised formats accepted are msgpack, json and pickle. The *msgpack* module must be importable. The option to use json and msgpack allow to integrate this script with other languages without resorting to a text file.

The last option is a HDF5 created using the *to_hdf* command in *taxon-utils - Taxonomy Utilities*. This requires *pandas* installed and *pytables* and it provides faster lookup of IDs in the table.

While the default is to look for the *gene_id* attribute in the GFF annotation, another attribute can be specified, using the **-gene-attr** option.

Note: the dictionary content is loaded after the table files and its keys and corresponding values takes precedence over the text files.

Warning: from September 2016 NCBI will retire the GI. In that case the same kind of table can be built from the *nucl_gb.accession2taxid.gz* file The format is different, but some information can be found in `mgkit.io.blast.parse_accession_taxa_table()`

Adding information from Pfam

Adds the Pfam description for the annotation, by downloading the list from Pfam.

The options allow to specify in which attribute the ID/ACCESSION is stored (defaults to *gene_id*) and which one between ID/ACCESSION is the value of that attribute (defaults to *ID*). if no description is found for the family, a warning message is logged.

Changes

Changed in version 0.3.4: removed the *taxonomy* command, since a similar result can be obtained with *taxon-utils lca* and *add-gff-info addtaxa*. Removed *eggnog* command and added option to verbose the logging in *cov_samtools* (now is quiet), also changed the interface

Changed in version 0.3.3: changed how *addtaxa -a* works, to allow the use of *seq_id* as key to add the taxon_id

Changed in version 0.3.0: added *cov_samtools* command, *–split* option to *exp_syn*, *-c* option to *addtaxa*. *kegg* now does not skip annotations when the attribute is not found.

Changed in version 0.2.6: added *skip-no-taxon* option to *addtaxa*

Changed in version 0.2.5: if a dictionary is supplied to *addtaxa*, the GFF is not preloaded

Changed in version 0.2.3: added *pfam* command, renamed *gitaxa* to *addtaxa* and made it general

Changed in version 0.2.2: added *eggnog*, *gitaxa* and *counts* command

Changed in version 0.2.1.

- added *-d* to *uniprot* command
- added cache to *uniprot* command
- added *kegg* command (cached)

Changed in version 0.1.16: added *exp_syn* command

Changed in version 0.1.15: *taxonomy* command *-b* option changed

Changed in version 0.1.13.

- added *–force-taxon-id* option to the *uniprot* command
- added *coverage* command

- added *taxonomy* command
- added *unipfile* command

New in version 0.1.12.

mgkit.workflow.add_gff_info.**add_uniprot_info**(*annotations*, *email*, *force_taxon_id*, *taxon_id*, *lineage*, *eggnog*, *enzymes*, *kegg_orthologs*, *protein_names*, *mapping*, *info_cache*)

mgkit.workflow.add_gff_info.**load_featurecounts_files**(*count_files*, *samples*)

mgkit.workflow.add_gff_info.**load_htseq_count_files**(*count_files*, *samples*)

mgkit.workflow.add_gff_info.**parse_hdf5_arg**(*ctx*, *param*, *values*)

mgkit.workflow.add_gff_info.**split_sample_alg**(*ctx*, *param*, *values*)
Split sample/alignment option

mgkit.workflow.blast2gff module

Blast output conversion in GFF requires a BLAST+ tabular format which can be obtained by using the *–outfmt 6* option with the default columns, as specified in *mgkit.io.blast.parse_blast_tab()*. The script can get data from the standard in and ouputs GFF lines on the standard output by default.

Uniprot

The Function *mgkit.io.blast.parse_uniprot_blast()* is used, which filters BLAST hits based on bitscore and adds by default a *db* attribute to the annotation with the value *UNIPROT-SP*, indicating that the SwissProt db is used and a *dbq* attribute with the value 10. The feature type used in the GFF is CDS.

BlastDB

If a BlastDB, such as *nt* or *nr* was used, the **blastdb** command offers some quick defaults to parse BLAST results.

It now includes options to control the way the sequence header are formatted. Options to change the separator used, as well as the column used as *gene_id*. This was added because at the moment the GI identifier (the second column in the header) is used, but it's being phased out in favour of the embl/gb/dbj (right now the fourth column in the header). This should easy the transition to the new format and makes it easier to adapt an older pipeline/blastdb to newer files (like the ID to TAXA files).

The header from the a *ncbi-nt* header looks like this:

```
gi|160361034|gb|CP000884.1
```

This is the default output accepted by the *blastdb* command. The fields are separated by | (pipe) and the GI is used (–gene-index 1, since internally the string is split by the separator and the second element is take - lists indices are 0-based in Python). This output uses the following options:

```
--header-sep '|' --gene-index 1
```

Notice the single quotes to pass the pipe symbol, since *bash* would interpret it as pipeing to the next coommand otherwise. This is the default.

In case, for the same header, we want to use the *gb* identifier, the only option to be specified is:

```
--gene-index 3
```

This will get the fourth element of the header (since we're splitting by pipe).

As in the *uniprot* command, the *gene_id* can be set to use the whole header, using the *-n* option. Useful in case the *BLAST* db that was used was custom made. While pipe is used in major databases, it was made the default, by if the db used has different conventions the separator can be changed. There's also the options of later changing the *gene_id* in the output GFF if necessary.

Changes

Changed in version 0.3.4: using *click* instead of *argparse*

Changed in version 0.2.6: added *-r* option to *blastdb*

Changed in version 0.2.5: added more options to give user control to the *blastdb* command

New in version 0.2.3: added *–fasta-file* option, added more data from a blsat hit

New in version 0.2.2: added *blastdb* command

Changed in version 0.2.1: added *-ft* option

Changed in version 0.1.13: added *-n* and *-k* parameters to *uniprot* command

New in version 0.1.12.

`mgkit.workflow.blast2gff.`**`load_fasta_file`**(*file_name*)

`mgkit.workflow.blast2gff.`**`validate_params`**(*ctx*, *param*, *values*)

mgkit.workflow.extract_gff_info module

Extract information from GFF files

sequence command

Used to extract the nucleotidic sequences from GFF annotations. It requires the *fasta* file containing the sequences referenced in the GFF *seq_id* attribute (first column of the raw GFF).

The sequnces extract have as identifier the *uid* stored in the GFF file and by default the sequnece is not reverse complemented if the annotation is on the **-** strand, but this can be changed by using the *-r* option.

The sequences are wrapped at 60 characters, as per FASTA specs, but this behavior can be disabled by specifing the *-w* option.

> **Warning:** The reference file is loaded in memory

dbm command

Creates a dbm DB using the *semidbm* package. The database can then be loaded using *mgkit.db.dbm.GFFDB*

mongodb command

Outputs annotations in a format supported by MongoDB. More information about it can be found in *mgkit.db.mongo*

gtf command

Outputs annotations in the GTF format

split command

Splits a GFF file into smaller chunks, ensuring that all of a sequence annotations are in the same file.

cov command

Calculate annotation coverage for each contig in a GFF file. The command can be run as strand specific (not by default) and requires the reference file to which the annotation refer to. The output file is a tab separated one, with the first column being the sequence name, the second is the strand (+, -, or NA if not strand specific) and the third is the percentage of the sequence covered by annotations.

Warning: The GFF file is assumed to be sorted, by sequence or sequence-strand if wanted. The GFF file can be sorted using *sort -s -k 1,1 -k 7,7* for strand specific, or *sort -s -k 1,1* if not strand specific.

Changes

Changed in version 0.3.4: using *click* instead of *argparse*, renamed *split* command *–json* to *–json-out*

Changed in version 0.3.1: added *cov* command

Changed in version 0.3.0: added *–split* option to *sequence* command

Changed in version 0.2.6: added *split* command, *–indent* option to *mongodb*

Changed in version 0.2.3: added *–gene-id* option to *gtf* command

New in version 0.2.2: added *gtf* command

New in version 0.2.1: *dbm* and *mongodb* commands

New in version 0.1.15.

mgkit.workflow.fasta_utils module

New in version 0.3.0.

Scripts that includes some functionality to help use FASTA files with the framework

split command

Used to split a fasta file into smaller fragments

translate command

Used to translate nucleotide sequences into amino acids.

uid command

Used to change a FASTA file headers to a unique ID. A table (tab separated) with the changes made can be kept, using the *–table* option.

Changes

New in version 0.3.0.

Changed in version 0.3.1: added *translate* and *uid* command

Changed in version 0.3.4: ported to *click*

`mgkit.workflow.fasta_utils.`**`load_trans_table`**(*table_name*)
Loads translation table

`mgkit.workflow.fasta_utils.`**`translate_seq`**(*name*, *seq*, *trans_table*)
Tranlates sequence into the 6 frames

mgkit.workflow.fastq_utils module

Commands

- Interleave/deinterleave paired-end fastq files.
- Converts to FASTA
- sort 2 files to sync the headers

Changes

Changed in version 0.3.4: moved to use click, internal fastq parsing, removed *rand* command

Changed in version 0.3.1: added stdin/stdout defaults for some commands

Changed in version 0.3.0: added *convert* command to FASTA

`mgkit.workflow.fastq_utils.`**`report_counts`**(*count*, *wcount*, *counter=None*)
Logs the status

mgkit.workflow.filter_gff module

Filters GFF annotations in different ways.

Value Filtering

Enables filtering of GFF annotations based on the the values of attributes of a GFF annotation. The filters are based on equality of numbers (internally converted into float) and strings, a string contained in the value of an attribute less or greater than are included as well. The length of annotation has the attribute *length* and can be tested.

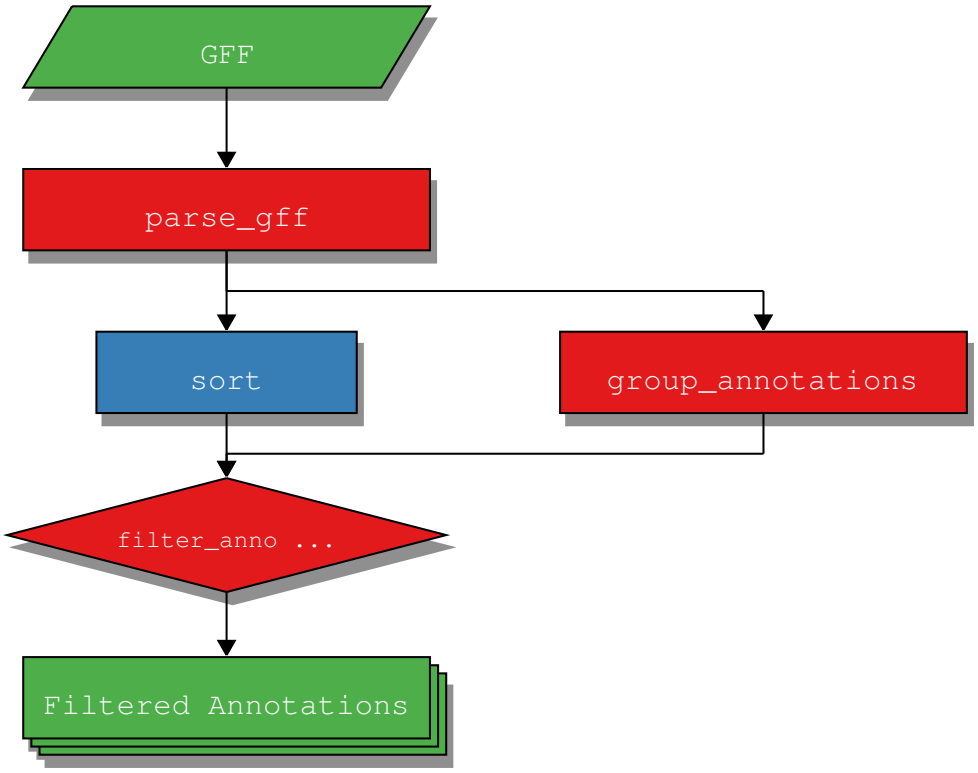
Overlap Filtering

Filters overlapping annotations using the functions *mgkit.filter.gff.choose_annotation()* and *mgkit.filter.gff.filter_annotations()*, after the annotations are grouped by both sequence and strand. If the GFF is sorted by sequence name and strand, the *-t* can be used to make the filtering use less memory. It can be sorted in Unix using *sort -s -k 1,1 -k 7,7 gff_file*, which applies a stable sort using the sequence name as the first key and the strand as the second key.

Note: It is also recommended to use:

```
export LC_ALL=C
```

To speed up the sorting



The above digram describes the internals of the script.

The annotations needs first to be grouped by seq_id and strand, forming a group that can be then be passed to *mgkit.filter.gff.filter_annotations()*. This function:

1. sort annotations by bit score, from the highest to the lowest
2. loop over all combination of N=2 annotations:
    1. choose which of the two annotations to discard if they overlap for a the required amount of bp (defaults to 100bp)
    2. in which case, the preference is given to the db quality first, than the bit score and finally the lenght of annotation, the one with the highest values is kept

While the default behaviour is the same, now it is posible to decided the function used to discard one the two annotations. It is possible to use the *-c* argument to pass a string that defines the function. The string passed must

start with or without a **+**. Using **+** translates into the builtin function *max* while no **+** translates into *min* from the second character on, any number of attributes can be used, separated by commas. The attributes, however, must be one of the properties defined in `mgkit.io.gff.Annotation`, *bitscore* that returns the value converted in a *float*. Internally the attributes are stored as strings, so for attributes that have no properties in the class, such as *evalue*, the *float* builtin is applied.

The tuples built for both annotations are then passed to the comparison function to be selected and the value returned by it is **discarded**. The order of the elements in the string is important to define the priority given to each element in the comparison and the leftmost one has the highesst priority.

Examples of function strings:

- *-dbq,bitscore,length* becomes max((ann1.dbq, ann1.bitscore, ann1.length), (ann2.dbq, ann2.bitscore, ann2.length) - This is default and previously only choice
- *-bitscore,length,dbq* uses the same elements but gives lowest priority to *dbq*
- *+evalue*: will discard the annotation with the highest *evalue*

Per Sequence Values

The *sequence* command allows to filter on a per sequence basis, using functions such as the median, quantile and mean on attributes like evalue, bitscore and identity. The file can be passed as sorted already, saving memory (like in the *overlap* command), but it's not needed to sort the file by strand, only by the first column.

Coverage Filtering

The *cov* command calculates the coverage of annotations as a measure of the percentage of each reference sequence length. A minimum coverage percentage can be used to keep the annotations of sequences that have a greater or equal coverage than the specified one.

Changes

New in version 0.1.12.

Changed in version 0.1.13: added *–sorted* option

Changed in version 0.2.0: changed option *-c* to accept a string to filter overlap

Changed in version 0.2.5: added *sequence* command

Changed in version 0.2.6: added *length* as attribute and *min*/*max*, and *ge* is the default comparison for command *sequence*, *–sort-attr* to *overlap*

Changed in version 0.3.1: added *–num-gt* and *–num-lt* to *values* command, added *cov* command

Changed in version 0.3.4: moved to use *click* for argument parsing reworked the *values*, *sequence* commands

`mgkit.workflow.filter_gff.`**`filter_eq`**(*annotation*, *attr=None*, *value=None*, *conv=None*)

`mgkit.workflow.filter_gff.`**`filter_gt`**(*annotation*, *attr=None*, *value=None*, *conv=None*, *equal=None*)

`mgkit.workflow.filter_gff.`**`filter_in`**(*annotation*, *attr=None*, *value=None*, *conv=None*)

`mgkit.workflow.filter_gff.`**`filter_lt`**(*annotation*, *attr=None*, *value=None*, *conv=None*, *equal=None*)

`mgkit.workflow.filter_gff.`**`find_comparison`**(*comparison*)

`mgkit.workflow.filter_gff.`**`make_choose_func`**(*values*)
Builds the function used to choose between two annotations.

`mgkit.workflow.filter_gff.`**`perseq_calc_threshold`**(*annotations*, *attribute*, *function*, *func_arg=None*)

mgkit.workflow.filter_gff.**setup_filters**(*str_eq*, *str_in*, *num_eq*, *num_ge*, *num_le*, *num_gt*, *num_lt*)

mgkit.workflow.filter_gff.**validate_params**(*ctx*, *param*, *values*, *convert=<class 'str'>*)

mgkit.workflow.hmmer2gff module

Script to convert HMMER results files (domain table) to a GFF file, the name of the profiles are expected to be now in the form *GENEID_TAXONID_TAXON-NAME(-nr)* by default, but any other profile name is accepted.

The profiles tested are those made from Kegg Orthologs, from the *download_profiles* script. If the *–no-custom-profiles* options is used, the script can be used with any profile name. The profile name will be used for *gene_id*, *taxon_id* and *taxon_name* in the GFF file.

It is possible to use seuqnces not translated using mgkit, no information on the frame is assumed, so this script can be used against a protein DB. For example Uniprot can be searched for profiles, in which case the **–no-frame** options must be used.

Note: for GENEID, old documentation points to KOID, it is the same

Warning: The compatibility with old data has been **removed**, meaning that old experiments must use the scripts from those versions. It is possible to use multiple environments, with *virtualenv* for this purpose. An examples is given in *Installation*.

Changes

Changed in version 0.1.15: adapted to new GFF module and specs

Changed in version 0.2.1: added options to customise output and filters and old restrictions

Changed in version 0.3.1: added *–no-frame* option for non mgkit-translated proteins, sequence headers are handled the same way as HMMER (truncated at the first space)

mgkit.workflow.hmmer2gff.**get_aa_data**(*f_handle*)
Load aminoacid seuqnces used by HMMER.

mgkit.workflow.hmmer2gff.**main**()
Main loop

mgkit.workflow.hmmer2gff.**parse_domain_table_contigs**(*options*)
Parse the HMMER result file

mgkit.workflow.hmmer2gff.**set_parser**()
Setup command line options

mgkit.workflow.json2gff module

Changed in version 0.3.4: using *click* instead of *argparse*

New in version 0.2.6.

This script converts annotations in JSON format that were created using MGKit back into GFF annotations.

mongodb command

Annotations converted into MongoDB records with *get-gff-info mongodb* can be converted back into a GFF file using this command. It can be useful to get a GFF file as output from a query to a MongoDB instance on the command line.

For example:

mongoexport -d db -c test | json2gff mongodb

will convert all the annotations in the database *db*, collection *test* to the standard out.

mgkit.workflow.sampling_utils module

New in version 0.3.1.

Resampling Utilities

sample command

This command samples from a Fasta or FastQ file, based on a probability defined by the user (0.001 or 1 / 1000 by default, *-r* parameter), for a maximum number of sequences (100,000 by default, *-x* parameter). By default 1 sample is extracted, but as many as desired can be taken, by using the *-n* parameter.

The sequence file in input can be either be passed to the standard input or as last parameter on the command line. By defult a Fasta is expected, unless the *-q* parameter is passed.

The *-p* parameter specifies the prefix to be used, and if the output files can be gzipped using the *-z* parameter.

sample_stream command

It works in the same way as *sample*, however the file is sampled only once and the output is the stdout by default. This can be convenient if streams are a preferred way to sample the file.

sync command

Used to keep in sync forward and reverse read files in paired-end FASTQ. The scenario is that the *sample* command was used to resample a FASTQ file, usually the forward, but we need the reverse as well. In this case, the resampled file, called *master* is passed to the *-m* option and the input file is the file that is to be synced (reverse). The input file is scanned until the same header is found in the master file and when that happens, the sequence is written. The next sequence is then read from the master file and the process is repeated until all sequence in the master file are found in the input file. This implies having the 2 files sorted in the same way, which is what the *sample* command does.

Note: the old casava format is not supported by this command at the moment, as it's unusual to find it in SRA or other repositories as well.

---

rand_seq command

Generate random FastA/Q sequences, allowing the specification of GC content and number of sequences being coding or random. If the output format chosen is FastQ, qualities are generated using a decreasing model with added noise. A constant model can be specified instead with a switch. Parameters such GC, length and the type of model can be infered by passing a FastA/Q file, with the quality model fit using a LOWESS (using `mgkit.utils.sequence.extrapolate_model()`). The noise in that case is model as the a normal distribution

fitted from the qualities along the sequence deviating from the fitted LOWSS and scaled back by half to avoid too drastic changes in the qualities. Also the qualities are clipped at 40 to avoid compatibility problems with FastQ readers. If inferred, the model can be saved (as a pickle file) and loaded back for analysis

Changes

Changed in version 0.3.4: using *click* instead of *argparse. Now *rand_seq* can save and reload models

Changed in version 0.3.3: added *sync*, *sample_stream* and *rand_seq* commnads

mgkit.workflow.sampling_utils.**compare_header**(*header1*, *header2*, *header_type=None*)

mgkit.workflow.sampling_utils.**infer_parameters**(*file_handle*, *fastq_bool*, *progress*)

mgkit.workflow.snp_parser module

This script parses results of SNPs analysis from any tool for SNP calling[734] and integrates them into a format that can be later used for other scripts in the pipeline.

It integrates coverage and expected number of syn/nonsyn change and taxonomy from a GFF file, SNP data from a VCF file.

Note: The script accept gzipped VCF files

Changes

Changed in version 0.2.1: added *-s* option for VCF files generated using bcftools

Changed in version 0.1.16: reworkked internals and removed SNPDat, syn/nonsyn evaluation is internal

Changed in version 0.1.13: reworked the internals and the classes used, including options -m and -s

mgkit.workflow.snp_parser.**check_snp_in_set**(*samples*, *snp_data*, *pos*, *change*, *annotations*, *seq*)

Used by *parse_vcf()* to check if a SNP

Parameters

- **samples** (*iterable*) – list of samples that contain the SNP
- **snp_data** (*dict*[735]) – dictionary from *init_count_set()* with per sample SNPs information

mgkit.workflow.snp_parser.**init_count_set**(*annotations*)

mgkit.workflow.snp_parser.**main**()

Main function

mgkit.workflow.snp_parser.**parse_vcf**(*vcf_file*, *snp_data*, *min_reads*, *min_af*, *min_qual*, *annotations*, *seqs*, *options*, *line_num=100000*)

Parse VCF file counts synonymous and non-synonymous SNPs

Parameters

- **vcf_file** (*file*) – file handle to a VCF file
- **snp_data** (*dict*[736]) – dictionary from *init_count_set()* with per sample SNPs information

[734] GATK pipeline was tested, but it is possible to use samtools and bcftools
[735] https://docs.python.org/3/library/stdtypes.html#dict
[736] https://docs.python.org/3/library/stdtypes.html#dict

- **min_reads** (*int*[737]) – minimum number of reads to accept a SNP
- **min_af** (*float*[738]) – minimum allele frequency to accept a SNP
- **min_qual** (*int*[739]) – minimum quality (Phred score) to accept a SNP
- **annotations** (*dict*[740]) – annotations grouped by their reference sequence
- **seqs** (*dict*[741]) – reference sequences
- **line_num** (*int*[742]) – the interval in number of lines at which progress will be printed

mgkit.workflow.snp_parser.**save_data**(*output_file*, *snp_data*)
Pickle data structures to the disk.

Parameters

- **output_file** (*str*[743]) – base name for pickle files
- **snp_data** (*dict*[744]) – dictionary from *init_count_set()* with per sample SNPs information

mgkit.workflow.snp_parser.**set_parser**()
Sets command line arguments parser

mgkit.workflow.taxon_utils module

The script contains commands used to access functionality related to taxonomy, without the need to write ad-hoc code for functionality that can be part of a workflow. One example is access to the the last common ancestor function contained in the *mgkit.taxon*.

Last Common Ancestor (lca and lca_line)

These commands expose the functionality of `last_common_ancestor_multiple()`, making it accessible via the command line. They differ in the input file format and the choice of output files.

the *lca* command can be used to define the last common ancestor of contigs from the annotation in a GFF file. The command uses the *taxon_ids* from all annotations belonging to a contig/sequence, if they have a **bitscore** higher or equal to the one passed (50 by default). The default output of the command is a tab separated file where the first column is the contig/sequence name, the taxon_id of the last common ancestor, its scientific/common name and its lineage.

For example:

```
contig_21   172788  uncultured phototrophic eukaryote   cellular organisms,
↪environmental samples
```

If the *-r* is used, by passing the fasta file containing the nucleotide sequences the output file is a GFF where for each an annotation for the full contig length contains the same information of the tab separated file format.

The **lca_line** command accept as input a file where each line consist of a list of taxon_ids. The separator for the list can be changed and it defaults to TAB. The last common ancestor for all taxa on a line is searched. The ouput of this command is the same as the tab separated file of the **lca** command, with the difference that instead of the first column, which in this command becames a list of all *taxon_ids* that were used to find the last common ancestor for that line. The list of *taxon_ids* is separated by semicolon ";".

[737] https://docs.python.org/3/library/functions.html#int
[738] https://docs.python.org/3/library/functions.html#float
[739] https://docs.python.org/3/library/functions.html#int
[740] https://docs.python.org/3/library/stdtypes.html#dict
[741] https://docs.python.org/3/library/stdtypes.html#dict
[742] https://docs.python.org/3/library/functions.html#int
[743] https://docs.python.org/3/library/stdtypes.html#str
[744] https://docs.python.org/3/library/stdtypes.html#dict

Note: Both also accept the *-n* option, to report the config/line and the taxon_ids that had no common ancestors. These are treated as errors and do not appear in the output file.

Krona Output

New in version 0.3.0.

The *lca* command supports the writing of a file compatible with Krona. The output file can be used with the *ktImportText/ImportText.pl* script included with KronaTools[745]. Specifically, the output from *taxon_utils* will be a file with all the lineages found (tab separated), that can be used with:

```
$ ktImportText -q taxon_utils_ouput
```

Note the use of *-q* to make the script count the lineages. Sequences with no LCA found will be marked as *No LCA* in the graph, the *-n* is not required.

Note: Please note that the output won't include any sequence that didn't have a hit with the software used. If that's important, the **-kt** option can be used to add a number of *Unknown* lines at the end, to read the total supplied.

Filter by Taxon

The **filter** command of this script allows to filter a GFF file using the *taxon_id* attribute to include only some annotations, or exclude some. The filter is based on the *mgkit.taxon.is_ancestor* function, and the *mgkit.filter.taxon.filter_taxon_by_id_list*. It can also filter a table (tab separated values) when the first element is an ID and the second is a taxon_id. An example of a table of this sort is the output of the *download-ncbi-taxa.sh* and *download-uniprot-taxa.sh*, where each accession of a database is associated to a taxon_id.

Multiple taxon_id can be passed, either for inclusion or exclusion. If both exclusion and inclusion is used, the first check is on the inclusion and then on the exclusion. In alternative to passing taxon_id, taxon_names can be passed, with values such as 'cellular organisms' that needs to be quoted. Example:

```
$ taxon-utils filter -i 2 -in archaea -en prevotella -t taxonomy.pickle in.gff out.
↪gff
```

Which will keep only line that are from Bacteria (taxon_id=2) and exclude those from the genus *Prevotella*. It will be also include Archaea.

Multiple inclusion and exclusion flags can be put:

```
$ taxon-utils filter -i 2 -i 2172 -t taxonomy in.gff out.gff
```

In particular, the inclusion flag is tested first and then the exclusion is tested. So a line like this one:

```
printf "TEST\t838\nTEST\t1485" | taxon-utils filter -p -t taxonomy.pickle -i 2 -i␣
↪1485 -e 838
```

Will produce **TEST 1485**, because both Prevotella (838) and Clostridium (1485) are Bacteria (2) OR Prevotella, but Prevotella must be excluded according to the exclusion option. This line also illustrate that a tab-separated file, where the second column contains taxon IDs, can be filtered. In particular it can be applied to files produced by *download-ncbi-taxa.sh* or *download-uniprot-taxa.sh* (see *Download Taxonomy*).

Warning: Annotations with no taxon_id are not included in the output of both filters

[745] https://github.com/marbl/Krona/wiki

Convert Taxa Tables to HDF5

This command is used to convert the taxa tables download from Uniprot and NCBI, using the scripts mentioned in download-data, *download-uniprot-taxa.sh* and *download-ncbi-taxa* into a HDF5 file that can be used with the *addtaxa* command in *add-gff-info - Add informations to GFF annotations*.

The advantage is a faster lookup of the IDs. The other is a smaller memory footprint when a great number of annotations are kept in memory.

Changes

Changed in version 0.3.4: changed interface and behaviour for *filter*, also now can filter tables; *lca* has changed the interface and allows the output of a 2 column table

Changed in version 0.3.1: added *to_hdf* command

Changed in version 0.3.1: added *-j* option to *lca*, which outputs a JSON file with the LCA results

Changed in version 0.3.0: added *-k* and *-kt* options for Krona output, lineage now includes the LCA also added *-a* option to select between lineages with only ranked taxa. Now it defaults to all components.

Changed in version 0.2.6: added *feat-type* option to *lca* command, added phylum output to nolca

New in version 0.2.5.

mgkit.workflow.taxon_utils.**get_taxon_info**(*taxonomy*, *taxon_id*, *only_ranked*)

mgkit.workflow.taxon_utils.**validate_taxon_ids**(*taxon_ids*, *taxonomy*)

mgkit.workflow.taxon_utils.**validate_taxon_names**(*taxon_names*, *taxonomy*)

mgkit.workflow.taxon_utils.**write_json**(*lca_dict*, *seq_id*, *taxonomy*, *taxon_id*, *only_ranked*)

mgkit.workflow.taxon_utils.**write_krona**(*file_handle*, *taxonomy*, *taxon_id*, *only_ranked*)

mgkit.workflow.taxon_utils.**write_lca_gff**(*file_handle*, *seq_id*, *seq*, *taxon_id*, *taxon_name*, *lineage*, *feat_type*)

mgkit.workflow.taxon_utils.**write_lca_tab**(*file_handle*, *seq_id*, *taxon_id*, *taxon_name*, *rank*, *lineage*)

mgkit.workflow.taxon_utils.**write_lca_tab_simple**(*file_handle*, *seq_id*, *taxon_id*)

mgkit.workflow.taxon_utils.**write_no_lca**(*file_handle*, *seq_id*, *taxon_ids*, *extra=None*)

mgkit.workflow.utils module

Utility functions for workflows

class mgkit.workflow.utils.**CiteAction**(*option_strings*, *dest='==SUPPRESS=='*, *default='==SUPPRESS=='*, *help='Show citation for the framework'*)

Bases: argparse.Action[746]

Argparse action to print the citation, using the *mgkit.cite()* function.

class mgkit.workflow.utils.**PrintManAction**(*option_strings*, *dest='==SUPPRESS=='*, *default='==SUPPRESS=='*, *help='Show the script manual'*, *manual=''*)

Bases: argparse.Action[747]

New in version 0.2.6.

[746] https://docs.python.org/3/library/argparse.html#argparse.Action
[747] https://docs.python.org/3/library/argparse.html#argparse.Action

Argparse action to print the manual

mgkit.workflow.utils.**add_basic_options**(*parser*, *manual=''*)
Changed in version 0.2.6: added *quiet* option

Adds verbose and version options to the option parser

mgkit.workflow.utils.**cite_callback**(*ctx*, *param*, *value*)

mgkit.workflow.utils.**exit_script**(*message*, *ret_value*)
Used to exit the script with a return value

Module contents

Workflows used to script the library - execute bits of the pipelines supported

7.1.2 Submodules

mgkit.align module

Module dealing with BAM/SAM files

class mgkit.align.**SamtoolsDepth**(*file_handle*, *num_seqs=10000*, *max_size=1000000*, *max_size_dict=None*)
Bases: object[748]

Changed in version 0.4.0: uses pandas.SparseArray now. It should use less memory, but needs pandas version > 0.24

New in version 0.3.0.

A class used to cache the results of *read_samtools_depth()*, while reading only the necessary data from a'samtools depth -aa' file.

data = None

file_handle = None

max_size = None

max_size_dict = None

region_coverage(*seq_id*, *start*, *end*)
Returns the mean coverage of a region. The *start* and *end* parameters are expected to be 1-based coordinates, like the correspondent attributes in *mgkit.io.gff.Annotation* or *mgkit.io.gff.GenomicRange*.

If the sequence for which the coverage is requested is not found, the *depth* file is read (and cached) until it is found.

Parameters

- **seq_id** (*str*[749]) – sequence for which to return mean coverage
- **start** (*int*[750]) – start of the region
- **end** (*int*[751]) – end of the region

Returns mean coverage of the requested region

Return type float[752]

[748] https://docs.python.org/3/library/functions.html#object
[749] https://docs.python.org/3/library/stdtypes.html#str
[750] https://docs.python.org/3/library/functions.html#int
[751] https://docs.python.org/3/library/functions.html#int
[752] https://docs.python.org/3/library/functions.html#float

mgkit.align.**add_coverage_info**(*annotations*, *bam_files*, *samples*, *attr_suff='_cov'*)
Changed in version 0.3.4: the coverage now is returned as floats instead of int

Adds coverage information to annotations, using BAM files.

The coverage information is added for each sample as a 'sample_cov' and the total coverage as as 'cov' attribute in the annotations.

Note: The bam_files and sample variables must have the same order

Parameters

- **annotations** (*iterable*) – iterable of annotations
- **bam_files** (*iterable*) – iterable of `pysam.Samfile` instances
- **sample** (*iterable*) – names of the samples for the BAM files

mgkit.align.**covered_annotation_bp**(*files*, *annotations*, *min_cov=1*, *progress=False*)
New in version 0.1.14.

Returns the number of base pairs covered of annotations over multiple samples.

Parameters

- **files** (*iterable*) – an iterable that returns the alignment file names
- **annotations** (*iterable*) – an iterable that returns annotations
- **min_cov** (*int*[753]) – minumum coverage for a base to counted
- **progress** (*bool*[754]) – if *True*, a progress bar is used

Returns a dictionary whose keys are the uid and the values the number of bases that are covered by reads among all samples

Return type dict[755]

mgkit.align.**get_region_coverage**(*bam_file*, *seq_id*, *feat_from*, *feat_to*)
Return coverage for an annotation.

Note: feat_from and feat_to are 1-based indexes

Parameters

- **bam_file** (*Samfile*) – instance of `pysam.Samfile`
- **seq_id** (*str*[756]) – sequence id
- **feat_from** (*int*[757]) – start position of feature
- **feat_to** (*int*[758]) – end position of feature

Return int coverage array for the annotation

mgkit.align.**read_samtools_depth**(*file_handle*, *num_seqs=10000*, *seq_ids=None*)
Changed in version 0.4.0: now returns 3 array, instead of 2. Also added *seq_ids* to skip lines

Changed in version 0.3.4: *num_seqs* can be None to avoid a log message

[753] https://docs.python.org/3/library/functions.html#int
[754] https://docs.python.org/3/library/functions.html#bool
[755] https://docs.python.org/3/library/stdtypes.html#dict
[756] https://docs.python.org/3/library/stdtypes.html#str
[757] https://docs.python.org/3/library/functions.html#int
[758] https://docs.python.org/3/library/functions.html#int

New in version 0.3.0.

Reads a samtools *depth* file, returning a generator that yields the array of each base coverage on a per-sequence base.

Note: The information on position is not used, to use numpy and save memory. samtools *depth* should be called with the *-aa* option:

```
`samtools depth -aa bamfile`
```

This options will output both base position with 0 coverage and sequneces with no aligned reads

---

Parameters

- **file_handle** (*file*) – file handle of the coverage file
- **num_seqs** (*int*[759] *or None*[760]) – number of sequence that fires a log message. If None, no message is triggered
- **seq_ids** (*dict*[761], *set*[762]) – a hashed container like a dictionary or set with the sequences to return

Yields *tuple* – the first element is the sequence identifier, the second one is the *numpy* array with the positions, the third element is the *numpy* array with the coverages

mgkit.consts module

Module containing constants for the filter package

mgkit.consts.**BLACK_LIST = ['bos', 'pecora', 'lolium', 'streptophyta', 'oryza', 'fabales'**
Default taxa black list, includes all taxa names that are to be excluded from some analysis.

mgkit.consts.**DEFAULT_SNP_FILTER = {'black_list': [903, 35500, 4520, 35493, 4527, 72025,**
Default filter options for filtering `mgkit.snps.GeneSyn`

mgkit.consts.**MIN_COV = 4**
Minumum coverage required in some functions.

mgkit.consts.**MIN_NUM = 10**
Used to set the minimum number of replicates for some functions

mgkit.graphs module

New in version 0.1.12.

Graph module

mgkit.graphs.**EDGE_LINKS = [(<function <lambda>>, 'CP_LINK', 0.0), (<function <lambda>>,**
Sample edge_links for *link_graph()*

class mgkit.graphs.**Reaction**(*kegg_id*, *substrates*, *products*, *reversible*, *orthologs*, *pathway*)
Bases: object[763]

New in version 0.4.0.

Object used to hold information about a reaction entry in Kegg

[759] https://docs.python.org/3/library/functions.html#int
[760] https://docs.python.org/3/library/constants.html#None
[761] https://docs.python.org/3/library/stdtypes.html#dict
[762] https://docs.python.org/3/library/stdtypes.html#set
[763] https://docs.python.org/3/library/functions.html#object

__eq__(*other*)
Tests equality by comparing the IDs and the compounds

cmp_compounds(*other*)
Compares the substrates and products of the current instance with those of another one, using information about the reversibility of the reaction.

irreversible_paths = None

kegg_id = None

orthologs = None

pathways
Set which includes all the pathways in which the reaction was found

products = None

reversible
Property that returns the reversibility of the reaction according to the information in the pathways. Returns True if the number of pathways in which the reaction was observed as reversible is greater or equal than the number of pathwaysin which the reaction was observerd as irreversible.

reversible_paths = None

substrates = None

to_edges()
Returns a generator of edges to be used when building a graph, along with an attribute that specify if the reaction is reversible.

to_edges_compounds()

to_nodes()
Returns a generator that returns the nodes associated with reaction, to be used in a graph, along with attributes about the type of node (reaction or compound).

update(*other*)
Updates the current instance with information from another instance. the underlining sets that hold the information are update with those from the *other* instance.

Raises ValueError[764] – if the ID of the reaction is different

mgkit.graphs.**add_module_compounds**(*graph*, *rn_defs*)
New in version 0.3.1.

Modify in-place a graph, by adding additional compounds from a dictionary of definitions. It uses the reversible/irreversible information for each reaction to add the correct number of edges to the graph.

Parameters

- **graph** (*graph*) – a graph to update with additional compounds
- **rn_defs** (*dict*[765]) – a dictionary, whose keys are reactions IDs and the values are instances of mgkit.kegg.KeggReaction

mgkit.graphs.**annotate_graph_nodes**(*graph*, *attr*, *id_map*, *default=None*, *conv=None*)
New in version 0.1.12.

Changed in version 0.4.0: added *conv* parameter and reworked internals

Add/Changes nodes attribute *attr* using a dictionary of ids->values.

Note: If the id is not found in *id_map*:

- default is None: no value added for that node

[764] https://docs.python.org/3/library/exceptions.html#ValueError
[765] https://docs.python.org/3/library/stdtypes.html#dict

- default is not None: the node attribute will be set to *default*

Parameters

- **graph** – the graph to annotate
- **attr** (*str*[766]) – the attribute to annotate
- **id_map** (*dict*[767]) – the dictionary with the values for each node
- **default** – the value used in case an *id* is not found in *id_map*, if None, the attribute is not set for missing values
- **conv** (*func*) – function to convert the value to another type

mgkit.graphs.**build_graph**(*id_links*, *name*, *edge_type=''*, *weight=0.5*)

New in version 0.1.12.

Builds a networkx graph from a dictionary of nodes, as outputted by *mgkit.kegg.KeggClientRest.get_pathway_links()*. The graph is undirected, and all edges weight are the same.

Parameters

- **id_links** (*dict*[768]) – dictionary with the links
- **name** (*str*[769]) – name of the graph
- **edge_type** (*str*[770]) – an optional name for the *edge_type* attribute set for each edge
- **weight** (*float*[771]) – the weight assigned to each edge in the graph

Returns an instance of `networkx.Graph`

Return type graph

mgkit.graphs.**build_weighted_graph**(*id_links*, *name*, *weights*, *edge_type=''*)

New in version 0.1.14.

Builds a networkx graph from a dictionary of nodes, as outputted by *mgkit.kegg.KeggClientRest.get_pathway_links()*. The graph is undirected, and all edges weight are the same.

Parameters

- **id_links** (*dict*[772]) – dictionary with the links
- **name** (*str*[773]) – name of the graph
- **edge_type** (*str*[774]) – an optional name for the *edge_type* attribute set for each edge
- **weight** (*float*[775]) – the weight assigned to each edge in the graph

Returns an instance of `networkx.Graph`

Return type graph

mgkit.graphs.**copy_edges**(*g*, *graph1*, *name=None*, ***kwd*)

New in version 0.1.12.

Used by *link_nodes()* to copy edges

[766] https://docs.python.org/3/library/stdtypes.html#str
[767] https://docs.python.org/3/library/stdtypes.html#dict
[768] https://docs.python.org/3/library/stdtypes.html#dict
[769] https://docs.python.org/3/library/stdtypes.html#str
[770] https://docs.python.org/3/library/stdtypes.html#str
[771] https://docs.python.org/3/library/functions.html#float
[772] https://docs.python.org/3/library/stdtypes.html#dict
[773] https://docs.python.org/3/library/stdtypes.html#str
[774] https://docs.python.org/3/library/stdtypes.html#str
[775] https://docs.python.org/3/library/functions.html#float

mgkit.graphs.**copy_nodes**(*g*, *graph1*, *name=None*, *id_attr=None*, ***kwd*)
New in version 0.1.12.

Used by *link_nodes()* to copy nodes

mgkit.graphs.**filter_graph**(*graph*, *id_list*, *filter_func=<function <lambda>>*)
New in version 0.1.12.

Filter a graph based on the *id_list* provided and the filter function used to test the id attribute of each node.

A node is removed if *filter_func* returns True on a node and its id attribute is not in *id_list*

Parameters

- **graph** – the graph to filter
- **id_list** (*iterable*) – the list of nodes that are to remain in the graph
- **filter_func** (*func*) – function which accept a single parameter and return a boolean

Returns an instance of networkx.Graph

Return type graph

mgkit.graphs.**from_kgml**(*entry*, *graph=None*, *rn_ids=None*)
New in version 0.3.1.

Given a KGML file (as string), representing a pathway in Kegg, returns a networkx DiGraph, using reaction directionality included in the KGML. If a reaction is reversible, 2 edges (from and to) for each compound/reaction pair are added, giving the bidirectionality.

Note: substrate and products included in a KGML don't represent the complete reaction, excluding in general cofactors or more general terms. Those can be added using *add_module_compounds()*, which may be more useful when used with a restricted number of reactions (e.g. a module)

Parameters

- **entry** (*str*[776]) – KGML file as a string, or anything that can be passed to ElementTree
- **graph** (*graph*) – an instance of a networkx DiGraph if the network is to be updated with a new KGML, if *None* a new one is created
- **rn_ids** (*set*[777]) – a set/list of reaction IDs that are to be included, if *None* all reactions are used

Returns a networkx DiGraph with the reaction/compounds

Return type graph

mgkit.graphs.**link_graph**(*graphs*, *edge_links*)
New in version 0.1.12.

Link nodes of a set of graphs using the specifics in edge_links. The resulting graph nodes are renamed, and the nodes that are shared between the graphs linked.

Parameters

- **graphs** – iterable of graphs
- **edge_links** – iterable with function, edge_type and weight for the links between graphs

Returns an instance of networkx.Graph

Return type graph

[776] https://docs.python.org/3/library/stdtypes.html#str
[777] https://docs.python.org/3/library/stdtypes.html#set

mgkit.graphs.**link_nodes**(*g*, *graph1*, *graph2*, *id_filter*, *link_type*, *weight*)
New in version 0.1.12.

Used by *link_graph()* to link nodes with the same *id*

mgkit.graphs.**merge_kgmls**(*kgmls*)
New in version 0.4.0.

Parses multiple KGMLs and merges the reactions from them.

Parameters **kgmls** (*iterable*) – iterable of KGML files (content) to be passed to *parse_kgml_reactions()*

Returns dictionary with the reactions from amm te KGML files

Return type dict[778]

mgkit.graphs.**parse_kgml_reactions**(*kgml*)
New in version 0.4.0.

Parses a KGML for reactions, returning a dictionary with instances of *Reaction* as values and the IDs as keys.

Parameters **kgml** (*str*[779]) – the KGML file content as a string (to be passed)

Returns dictionary of ID->Reaction

Return type dict[780]

mgkit.graphs.**rename_graph_nodes**(*graph*, *name_func=None*, *exclude_ids=None*)

mgkit.kegg module

Module containing classes and functions to access Kegg data

class mgkit.kegg.**KeggClientRest**(*cache=None*)
Bases: object[781]

Changed in version 0.3.1: added a *cache* attribute for some methods

Kegg REST client

The class includes methods and data to use the REST API provided by Kegg. At the moment it provides methods to for 'link', 'list' and 'get' operations,

Kegg REST API[782]

api_url = 'http://rest.kegg.jp/'

cache = None

contact = None

conv(*target_db*, *source_db*, *strip=True*)
New in version 0.3.1.

Kegg Help:

http://rest.kegg.jp/conv/<target_db>/<source_db>

(<target_db> <source_db>) = (<kegg_db> <outside_db>) | (<outside_db> <kegg_db>)

For gene identifiers: <kegg_db> = <org> <org> = KEGG organism code or T number <outside_db> = ncbi-proteinid | ncbi-geneid | uniprot

[778] https://docs.python.org/3/library/stdtypes.html#dict
[779] https://docs.python.org/3/library/stdtypes.html#str
[780] https://docs.python.org/3/library/stdtypes.html#dict
[781] https://docs.python.org/3/library/functions.html#object
[782] http://www.kegg.jp/kegg/rest/keggapi.html

For chemical substance identifiers: <kegg_db> = drug | compound | glycan <outside_db> = pubchem | chebi http://rest.kegg.jp/conv/<target_db>/<dbentries>

For gene identifiers: <dbentries> = database entries involving the following <database> <database> = <org> | genes | ncbi-proteinid | ncbi-geneid | uniprot <org> = KEGG organism code or T number

For chemical substance identifiers: <dbentries> = database entries involving the following <database> <database> = drug | compound | glycan | pubchem | chebi

Examples

```
>>> kc = KeggClientRest()
>>> kc.conv('ncbi-geneid', 'eco')
{'eco:b0217': {'ncbi-geneid:949009'},
 'eco:b0216': {'ncbi-geneid:947541'},
 'eco:b0215': {'ncbi-geneid:946441'},
 'eco:b0214': {'ncbi-geneid:946955'},
 'eco:b0213': {'ncbi-geneid:944903'},
...
>>> kc.conv('ncbi-proteinid', 'hsa:10458+ece:Z5100')
{'10458': {'NP_059345'}, 'Z5100': {'AAG58814'}}
```

cpd_desc_re = re.compile("cpd:(C\\d{5})\\t([\\w+, ()\\[\\]'*.-]+);?\\n?")

cpd_re = re.compile('ENTRY\\s+(C\\d{5})\\s+Compound\\nNAME\\s+([,.\\w+ ()-]+);?')

empty_cache(*methods=None*)

New in version 0.3.1.

Empties the cache completely or for a specific method(s)

Parameters **methods** (*iterable, str*[783]) – string or iterable of strings that are part of the cache. If None the cache is fully emptied

find(*query*, *database*, *options=None*, *strip=True*)

New in version 0.3.1.

Kegg Help:

http://rest.kegg.jp/find/<database>/<query>

<database> = pathway | module | ko | genome | <org> | compound | glycan | reaction | rclass | enzyme | disease | drug | dgroup | environ | genes | ligand

<org> = KEGG organism code or T number

http://rest.kegg.jp/find/<database>/<query>/<option>

<database> = compound | drug <option> = formula | exact_mass | mol_weight

Examples

```
>>> kc = KeggClientRest()
>>> kc.find('CH4', 'compound')
{'C01438': 'Methane; CH4'}
>>> kc.find('K00844', 'genes', strip=False)
{'tped:TPE_0072': 'hexokinase; K00844 hexokinase [EC:2.7.1.1]',
...
>>> kc.find('174.05', 'compound', options='exact_mass')
{'C00493': '174.052823',
 'C04236': '174.052823',
 'C16588': '174.052823',
```

(continues on next page)

[783] https://docs.python.org/3/library/stdtypes.html#str

(continued from previous page)

```
 'C17696': '174.052823',
 'C18307': '174.052823',
 'C18312': '174.052823',
 'C21281': '174.052823'}
```

get_entry(*k_id*, *option=None*)
Changed in version 0.3.1: this is now cached

The method abstract the use of the 'get' operation in the Kegg API

Parameters

- **k_id** (*str*[784]) – kegg id of the resource to get
- **option** (*str*[785]) – optional, to specify a format

get_ids_names(*target='ko'*, *strip=True*)
New in version 0.1.13.

Changed in version 0.3.1: the call is now cached

Returns a dictionary with the names/description of all the id of a specific target, (ko, path, cpd, etc.)

If strip=True the id will stripped of the module abbreviation (e.g. md:M00002->M00002)

get_ortholog_pathways()
Gets ortholog pathways, replace 'map' with 'ko' in the id

get_pathway_links(*pathway*)
Returns a dictionary with the mappings KO->compounds for a specific Pathway or module

get_reaction_equations(*ids*, *max_len=10*)
Get the equation for the reactions

id_prefix = {'C': 'cpd', 'K': 'ko', 'R': 'rn', 'k': 'map', 'm': 'path'}

ko_desc_re = re.compile("ko:(K\\d{5})\\t.+?;\\s+([\\w+, ()/:'\\[\\]-]+)(\\[EC:)?\\r

link(*target*, *source*, *options=None*)
New in version 0.2.0.

Implements "link" operation in Kegg REST

http://www.genome.jp/linkdb/

link_ids(*target*, *kegg_ids*, *max_len=50*)
Changed in version 0.3.1: removed *strip* and cached the results

The method abstract the use of the 'link' operation in the Kegg API

The target parameter can be one of the following:

```
pathway | brite | module | disease | drug | environ | ko | genome |
<org> | compound | glycan | reaction | rpair | rclass | enzyme

<org> = KEGG organism code or T number
```

Parameters

- **target** (*str*[786]) – the target db
- **ids** – can be either a single id as a string or a list of ids
- **strip** (*bool*[787]) – if the prefix (e.g. ko:K00601) should be stripped

[784] https://docs.python.org/3/library/stdtypes.html#str
[785] https://docs.python.org/3/library/stdtypes.html#str
[786] https://docs.python.org/3/library/stdtypes.html#str
[787] https://docs.python.org/3/library/functions.html#bool

- **max_len** (*int*[788]) – the maximum number of ids to retrieve with each request, should not exceed 50

Return dict dictionary mapping requested id to target id(s)

list_ids(*k_id*)

The method abstract the use of the 'list' operation in the Kegg API

The k_id parameter can be one of the following:

```
pathway | brite | module | disease | drug | environ | ko | genome |
<org> | compound | glycan | reaction | rpair | rclass | enzyme

<org> = KEGG organism code or T number
```

Parameters **k_id** (*str*[789]) – kegg database to get list of ids

Return list list of ids in the specified database

load_cache(*file_handle*)

New in version 0.3.1.

Loads the cache from file

rn_eq_re = re.compile('C\\d{5}')

rn_name_re = re.compile('R\\d{5}')

write_cache(*file_handle*)

New in version 0.3.1.

Write the cache to file

class mgkit.kegg.**KeggModule**(*entry=None*, *old=False*)

Bases: object[790]

New in version 0.1.13.

Used to extract information from a pathway module entry in Kegg

The entry, as a string, can be either passed at instance creation or with *KeggModule.parse_entry()*

classes = None

compounds = None

entry = ''

find_submodules()

New in version 0.3.0.

Returns the possible submodules, as a list of tuples where the elements are the first and last compounds in a submodule

first_cp

Returns the first compound in the module

last_cp

Returns the first compound in the module

name = ''

parse_entry(*entry*)

Parses a Kegg module entry and change the instance values. By default the reactions IDs are substituted with the KO IDs

[788] https://docs.python.org/3/library/functions.html#int
[789] https://docs.python.org/3/library/stdtypes.html#str
[790] https://docs.python.org/3/library/functions.html#object

parse_entry2(*entry*)
New in version 0.3.0.

Parses a Kegg module entry and change the instance values. By default the reactions IDs are NOT substituted with the KO IDs.

static parse_reaction(*line*, *ko_ids=None*)
Changed in version 0.3.0: cleaned the parsing

parses the lines with the reactions and substitute reaction IDs with the corresponding KO IDs if provided

reactions = None

to_edges(*id_only=None*)
Changed in version 0.3.0: added id_only and changed to reflect changes in *reactions*

Returns the reactions as edges that can be supplied to make graph.

Parameters **id_only** (*None*[791], *iterable*) – if None the returned edges are for the whole module, if an iterable (converted to a set[792]), only edges for those reactions are returned

Yields *tuple* – the elements are the compounds and reactions in the module

mgkit.kegg.**parse_reaction**(*line*, *prefix=('C', 'G')*)
New in version 0.3.1.

Parses a reaction equation from Kegg, returning the left and right components. Needs testing

Parameters **line** (*str*[793]) – reaction string

Returns left and right components as *sets*

Return type tuple[794]

Raises ValueError[795] – if the

mgkit.logger module

Module configuring log information

class mgkit.logger.**ColorFormatter**(*fmt=None*, *datefmt=None*, *style='%'*)
Bases: logging.Formatter[796]

colors = {'CRITICAL': 'red', 'DEBUG': 'blue', 'ERROR': 'magenta', 'INFO': 'green',

format(*record*)
Format the specified record as text.

The record's attribute dictionary is used as the operand to a string formatting operation which yields the returned string. Before formatting the dictionary, a couple of preparatory steps are carried out. The message attribute of the record is computed using LogRecord.getMessage(). If the formatting string uses the time (as determined by a call to usesTime(), formatTime() is called to format the event time. If there is exception information, it is formatted using formatException() and appended to the message.

mgkit.logger.**config_log**(*level=10*, *output=<_io.TextIOWrapper name='<stderr>' mode='w' encoding='UTF-8'>*)
Minimal configuration of :mod'logging' module, default to debug level and the output is printed to standard error

Parameters

[791] https://docs.python.org/3/library/constants.html#None
[792] https://docs.python.org/3/library/stdtypes.html#set
[793] https://docs.python.org/3/library/stdtypes.html#str
[794] https://docs.python.org/3/library/stdtypes.html#tuple
[795] https://docs.python.org/3/library/exceptions.html#ValueError
[796] https://docs.python.org/3/library/logging.html#logging.Formatter

- **level** (*int*[797]) – logging level
- **output** (*file*) – file to which write the log

mgkit.logger.**config_log_to_file**(*level=10*, *output=None*)
New in version 0.1.14.

Minimal configuration of :mod‘logging‘ module, default to debug level and the output is printed to script name, using *sys.argv[0]*.

Parameters

- **level** (*int*[798]) – logging level
- **output** (*file*) – file to which write the log

mgkit.simple_cache module

class mgkit.simple_cache.**memoize**(*func*)
Bases: dict[799]

a cache found on the PythonDecoratorLibrary[800]

Not sure about the license for it.

mgkit.taxon module

This module gives access to Uniprot taxonomy data. It also defines classes to filter, order and group data by taxa

exception mgkit.taxon.**NoLcaFound**
Bases: Exception[801]

New in version 0.1.13.

Raised if no lowest common ancestor can be found in the taxonomy

mgkit.taxon.**TAXON_RANKS = ('superkingdom', 'kingdom', 'phylum', 'class', 'subclass', 'or**
Taxonomy ranks included in the pickled data

mgkit.taxon.**TAXON_ROOTS = ('archaea', 'bacteria', 'fungi', 'metazoa', 'environmental sam**
Root taxa used in analysis and filtering

mgkit.taxon.**TaxonTuple**
A representation of a Uniprot Taxon

alias of *mgkit.taxon.UniprotTaxonTuple*

class mgkit.taxon.**Taxonomy**(*fname=None*)
Bases: object[802]

Class that contains the whole Uniprot taxonomy. Defines some methods to easy access of taxonomy. Follows the conventions of NCBI Taxonomy.

Defines:

- methods to load taxonomy from a pickle file or a generic file handle
- can be iterated over and returns a generator its UniprotTaxon instances
- can be used as a dictionary, in which the key is a taxon_id and the value is its UniprotTaxon instance

[797] https://docs.python.org/3/library/functions.html#int
[798] https://docs.python.org/3/library/functions.html#int
[799] https://docs.python.org/3/library/stdtypes.html#dict
[800] https://wiki.python.org/moin/PythonDecoratorLibrary#Alternate_memoize_as_dict_subclass
[801] https://docs.python.org/3/library/exceptions.html#Exception
[802] https://docs.python.org/3/library/functions.html#object

__contains__(*taxon*)
Returns True if the taxon is in the taxonomy

Accepts an int (check for taxon_id) or an instance of UniprotTaxon

__getitem__(*taxon_id*)
Defines dictionary behavior. Key is a taxon_id, the returned value is a UniprotTaxon instance

__iter__()
Defines iterable behavior. Returns a generator for UniprotTaxon instances

__len__()
Returns the number of taxa contained

__repr__()
New in version 0.2.5.

add_lineage(***lineage*)
New in version 0.3.1.

Adds a lineage to the taxonomy. It's passed by keyword arguments, where each key is a value in the *TAXON_RANKS* rankes and the value is the scientific name. Appended underscores '_' will be stripped from the rank name. This is for cases such as *class* where the key is a reserved word in Python. Also one extra node can be added, such as strain/cultivar/subspecies and so on, but one only is expected to be passed.

Parameters **lineage** (*dict*[803]) – the lineage as a keyword arguments

Returns the taxon_id of the last element in the lineage

Return type int[804]

Raises

- ValueError[805] – if more than a keyword argument is not contained in
- *TAXON_RANKS*

add_taxon(*taxon_name*, *common_name=''*, *rank='no rank'*, *parent_id=None*)
New in version 0.3.1.

Adds a taxon to the taxonomy. If a taxon with the same name and rank is found, its taxon_id is returned, otherwise a new taxon_id is returned.

Parameters

- **taxon_name** (*str*[806]) – scientific name of the taxon
- **common_name** (*str*[807]) – common name
- **rank** (*str*[808]) – rank, defaults to 'no rank'
- **parent_id** (*int*[809]) – taxon_id of the parent, defaults to *None*, which is the taxonomy root

Returns the taxon_id of the added taxon (if new), or the taxon_id of the taxon with the same name and rank found in the taxonomy

Return type int[810]

Raises

[803] https://docs.python.org/3/library/stdtypes.html#dict
[804] https://docs.python.org/3/library/functions.html#int
[805] https://docs.python.org/3/library/exceptions.html#ValueError
[806] https://docs.python.org/3/library/stdtypes.html#str
[807] https://docs.python.org/3/library/stdtypes.html#str
[808] https://docs.python.org/3/library/stdtypes.html#str
[809] https://docs.python.org/3/library/functions.html#int
[810] https://docs.python.org/3/library/functions.html#int

- KeyError[811] – if more than one taxon has already the passed name and
- rank and it can't be resolved by looking at the parent_id passed,
- the exception is raised.

drop_taxon(*taxon_id*)
New in version 0.3.1.

Drops a taxon and all taxa below it in the taxonomy. Also reset the name map for conistency.

Parameters **taxon_id** (*int*[812]) – taxon_id to drop from the taxonomy

find_by_name(*s_name*, *rank=None*, *strict=True*)
Changed in version 0.2.3: the search is now case insensitive

Changed in version 0.3.1: added *rank* and *strict* parameter

Returns the taxon IDs associated with the scientific name provided

Parameters

- **s_name** (*str*[813]) – the scientific name
- **rank** (*str*[814], *None*[815]) – return only a taxon_id of a specific rank
- **strict** (*book*) – if True and *rank* is not None, KeyError will be raised if multiple taxa have the same name and rank

Returns a reference to the list of IDs that have for *s_name*, if *rank* is None. If *rank* is not None and one taxon is found, its taxon_id is returned, or None if no taxon is found. If *strict* is True and *rank* is not None, the set of taxon_ids found is resturned.

Return type list[816]

Raises

- KeyError[817] – If multiple taxa are found, a *KeyError* exception is
- raised.

gen_name_map()
Changed in version 0.2.3: names are stored in the mapping as lowercase

Generate a name map, where to each scientific name in the taxonomy an id is associated.

get_lineage(*taxon_id*, *names=False*, *only_ranked=True*, *with_last=True*)
New in version 0.3.1.

Proxy for *get_lineage()*, with changed defaults

Parameters

- **taxon_id** (*int*[818]) – taxon_id to return the lineage
- **names** (*bool*[819]) – if the elements of the list are converted into the scientific names
- **only_ranked** (*bool*[820]) – only return the ranked taxa
- **with_last** (*bool*[821]) – include the taxon_id passed to the list

Returns the lineage of the passed taxon_id as a list of IDs or names

[811] https://docs.python.org/3/library/exceptions.html#KeyError
[812] https://docs.python.org/3/library/functions.html#int
[813] https://docs.python.org/3/library/stdtypes.html#str
[814] https://docs.python.org/3/library/stdtypes.html#str
[815] https://docs.python.org/3/library/constants.html#None
[816] https://docs.python.org/3/library/stdtypes.html#list
[817] https://docs.python.org/3/library/exceptions.html#KeyError
[818] https://docs.python.org/3/library/functions.html#int
[819] https://docs.python.org/3/library/functions.html#bool
[820] https://docs.python.org/3/library/functions.html#bool
[821] https://docs.python.org/3/library/functions.html#bool

Return type list[822]

get_lineage_string(*taxon_id*, *only_ranked=True*, *with_last=True*, *sep=';'*, *rank=None*)
New in version 0.3.3.

Generates a lineage string, with the possibility of getting another ranked taxon (via *Taxonomy.get_ranked_taxon()*) to another rank, such as *phylum*.

Parameters

- **taxon_id** (*int*[823]) – taxon_id to return the lineage
- **only_ranked** (*bool*[824]) – only return the ranked taxa
- **with_last** (*bool*[825]) – include the taxon_id passed to the list
- **sep** (*str*[826]) – separator used to join the lineage string
- **rank** (*int*[827] *or None*[828]) – if None the full lineage is returned, otherwise the lineage will be cut to the specified rank

Returns lineage string

Return type str[829]

get_name_map()
Returns a taxon_id->s_name dictionary

get_ranked_id(*taxon_id*, *rank=None*, *it=False*, *include_higher=True*)
New in version 0.3.4.

Gets the ranked taxon of another one. Useful when it's better to get a taxon_id instead of an instance of *TaxonTuple*. Internally, it relies on *Taxonomy.get_ranked_taxon()*.

Parameters

- **taxon_id** (*int*[830]) – taxon_id
- **rank** (*str*[831] *or None*[832]) – passed over
- **it** (*bool*[833]) – determines the return value. if True, a list is returned
- **include_higher** (*bool*[834]) – if True, any rank higher than the requested may be returned. If False and the rank cannot be returned, None is returned

Returns The type returned is based on the *it* paramenter. If *it* is True, the return value is a list with the *taxon_id* of the ranked taxon as the sole value. If False, the returned value is the *taxon_id*. *include_higher* determines if the return value should be *None* if the exact rank was not found and *include_higher* is False

Return type int[835] or list[836]

get_ranked_taxon(*taxon_id*, *rank=None*, *ranks=('superkingdom', 'kingdom', 'phylum', 'class', 'subclass', 'order', 'family', 'genus', 'species')*, *roots=False*)
Changed in version 0.1.13: added *roots* argument

[822] https://docs.python.org/3/library/stdtypes.html#list
[823] https://docs.python.org/3/library/functions.html#int
[824] https://docs.python.org/3/library/functions.html#bool
[825] https://docs.python.org/3/library/functions.html#bool
[826] https://docs.python.org/3/library/stdtypes.html#str
[827] https://docs.python.org/3/library/functions.html#int
[828] https://docs.python.org/3/library/constants.html#None
[829] https://docs.python.org/3/library/stdtypes.html#str
[830] https://docs.python.org/3/library/functions.html#int
[831] https://docs.python.org/3/library/stdtypes.html#str
[832] https://docs.python.org/3/library/constants.html#None
[833] https://docs.python.org/3/library/functions.html#bool
[834] https://docs.python.org/3/library/functions.html#bool
[835] https://docs.python.org/3/library/functions.html#int
[836] https://docs.python.org/3/library/stdtypes.html#list

Traverse the branch of which the *taxon* argument is the leaf backward, to get the specific rank to which the *taxon* belongs to.

> **Warning:** the *roots* options is kept for backward compatibility and should be be set to *False*

Parameters

- **taxon_id** – id of the taxon or instance of `UniprotTaxon`
- **rank** (*str*[837]) – string that specify the rank, if None, the first valid rank will be searched. (i.e. the first with a value different from '')
- **ranks** – tuple of all taxonomy ranks, default to the default module value
- **roots** (*bool*[838]) – if True, uses *TAXON_ROOTS* to solve the root taxa

Returns instance of *TaxonTuple* for the rank found.

is_ancestor(*leaf_id*, *anc_ids*)

Changed in version 0.1.13: now uses *is_ancestor()* and changed behavior

Checks if a taxon is the leaf of another one, or a list of taxa.

Parameters

- **leaf_id** (*int*[839]) – leaf taxon id
- **anc_ids** (*int*[840]) – ancestor taxon id(s)

Return bool True if the ancestor taxon is in the leaf taxon lineage

is_ranked_below(*taxon_id*, *rank*, *equal=True*)

New in version 0.4.0.

Tests if a taxon_id is below the requested rank.

Parameters

- **taxon_id** (*int*[841]) – taxo_id to test
- **rank** (*str*[842]) – rank requested
- **equal** (*bool*[843]) – determines if the taxon_id tested may be of the requested rank

Returns If the passed *taxon_id* is below the requested rank, it returns *True*. If *taxon_id* is of the rank requested and *equal* is True, the return value is *True*, if *equal* is False the return value is *False*. The return value is *False* otherwise.

Return type bool[844]

load_data(*file_handle*)

Changed in version 0.2.3: now can use read *msgpack* serialised files

Changed in version 0.1.13: now accepts file handles and compressed files (if file names)

Loads serialised data from file name "file_handle" and accepts compressed files.

if the *.msgpack* string is found in the file name, the *msgpack* package is used instead of pickle

Parameters **file_handle** (*str*[845], *file*) – file name to which save the instance data

[837] https://docs.python.org/3/library/stdtypes.html#str
[838] https://docs.python.org/3/library/functions.html#bool
[839] https://docs.python.org/3/library/functions.html#int
[840] https://docs.python.org/3/library/functions.html#int
[841] https://docs.python.org/3/library/functions.html#int
[842] https://docs.python.org/3/library/stdtypes.html#str
[843] https://docs.python.org/3/library/functions.html#bool
[844] https://docs.python.org/3/library/functions.html#bool
[845] https://docs.python.org/3/library/stdtypes.html#str

static parse_gtdb_lineage(*lineage*, *sep=';'*)
New in version 0.3.3.

Parse a GTDB lineage, one that defines the rank as a single letter, followed by __ for each taxon name. Taxa are separated by semicolon by default. Also the **domain** rank is renamed into **superkingdom** to allow mixing of taxonomies.

Returns dictionary with the parsed lineage, which can be passed to *Taxonomy.add_lineage()*

Return type dict[846]

read_from_gtdb_taxonomy(*file_handle*, *use_gtdb_name=True*, *sep='\t'*)
New in version 0.3.0.

Changed in version 0.3.1: replaced *domain* with *superkingdom* to support *get_lineage*

Reads a GTDB taxonomy file (tab separated genome_id/taxonomy) and populate the taxonomy instance. The method also return a dictionary of genome_id -> taxon_id.

Parameters

- **file_handle** (*file*) – file with the taxonomy
- **use_gtdb_name** (*bool*[847]) – if True, the names are kept as-is in the *s_name* attribute of *TaxonTuple* and the "cleaned" version in *c_name* (e.g. f__Ammonifexaceae -> Ammonifexaceae). If False, the values are switched
- **sep** (*str*[848]) – separator between the columns of the file

Returns dictionary of genome_id -> taxon_id, reflecting the created taxonomy

Return type dict[849]

Note: the taxon_id are generated, so there's no guarantee they will be the same in a successive execution

---

read_from_ncbi_dump(*nodes_file*, *names_file=None*, *merged_file=None*)
New in version 0.2.3.

Uses the *nodes.dmp* and optionally *names.dmp*, *merged.dmp* files from ftp://ftp.ncbi.nlm.nih.gov/pub/taxonomy/ to populate the taxonomy.

Parameters

- **nodes_file** (*str*[850], *file*) – file name or handle to the file
- **names_file** (*str*[851], *file*, *None*[852]) – file name or handle to the file, if None, names won't be added to the taxa
- **merged_file** (*str*[853], *file*, *None*[854]) – file name or handle to the file, if None, pointers to merged taxa won't be added

read_taxonomy(*f_handle*, *light=True*)
Changed in version 0.2.1: added *light* parameter

Deprecated since version 0.4.0: use *Taxonomy.read_from_ncbi_dump()*

[846] https://docs.python.org/3/library/stdtypes.html#dict
[847] https://docs.python.org/3/library/functions.html#bool
[848] https://docs.python.org/3/library/stdtypes.html#str
[849] https://docs.python.org/3/library/stdtypes.html#dict
[850] https://docs.python.org/3/library/stdtypes.html#str
[851] https://docs.python.org/3/library/stdtypes.html#str
[852] https://docs.python.org/3/library/constants.html#None
[853] https://docs.python.org/3/library/stdtypes.html#str
[854] https://docs.python.org/3/library/constants.html#None

Reads taxonomy from a file handle. The file needs to be a tab separated format return by a query on Uniprot. If *light* is True, lineage is not stored to decrease the memory usage. This is now the default.

New taxa will be added, duplicated taxa will be skipped.

Parameters **f_handle** (*handle*) – file handle of the taxonomy file.

save_data (*file_handle*)
Changed in version 0.2.3: now can use *msgpack* to serialise

Saves taxonomy data to a file handle or file name, can write compressed data if the file ends with ".gz", ".bz2"

if the *.msgpack* string is found in the file name, the *msgpack* package is used instead of pickle

Parameters **file_handle** (*str*[855], *file*) – file name to which save the instance data

class mgkit.taxon.**UniprotTaxonTuple**(*taxon_id*, *s_name*, *c_name*, *rank*, *lineage*, *parent_id*)
Bases: tuple[856]

c_name
Alias for field number 2

lineage
Alias for field number 4

parent_id
Alias for field number 5

rank
Alias for field number 3

s_name
Alias for field number 1

taxon_id
Alias for field number 0

mgkit.taxon.**UniprotTaxonomy**
alias of *mgkit.taxon.Taxonomy*

mgkit.taxon.**distance_taxa_ancestor**(*taxonomy*, *taxon_id*, *anc_id*)
New in version 0.1.16.

Function to calculate the distance between a taxon and the given ancestor

The distance is equal to the number of step in the taxonomy taken to arrive at the ancestor.

Parameters

- **taxonomy** – *Taxonomy* instance
- **taxon_id** (*int*[857]) – taxonomic identifier
- **anc_id** (*int*[858]) – taxonomic identifier of the ancestor

Raturns: int: distance between *taxon_id* and it ancestor *anc_id*

mgkit.taxon.**distance_two_taxa**(*taxonomy*, *taxon_id1*, *taxon_id2*)
New in version 0.1.16.

Calculate the distance between two taxa. The distance is equal to the sum steps it takes to traverse the taxonomy until their last common ancestor.

Parameters

[855] https://docs.python.org/3/library/stdtypes.html#str
[856] https://docs.python.org/3/library/stdtypes.html#tuple
[857] https://docs.python.org/3/library/functions.html#int
[858] https://docs.python.org/3/library/functions.html#int

- **taxonomy** – *Taxonomy* instance
- **taxon_id1** (*int*[859]) – taxonomic identifier of first taxon
- **taxon_id2** (*int*[860]) – taxonomic identifier of second taxon

Raturns: int: distance between *taxon_id1* and *taxon_id2*

mgkit.taxon.**get_ancestor_map**(*leaf_ids*, *anc_ids*, *taxonomy*)

This function returns a dictionary where every leaf taxon is associated with the right ancestors in anc_ids

ex. {clostridium: [bacteria, clostridia]}

mgkit.taxon.**get_lineage**(*taxonomy*, *taxon_id*, *names=False*, *only_ranked=False*, *with_last=False*)

New in version 0.2.1.

Changed in version 0.2.5: added *only_ranked*

Changed in version 0.3.0: added *with_last*

Returns the lineage of a taxon_id, as a list of taxon_id or taxa names

Parameters

- **taxonomy** – a *Taxonomy* instance
- **taxon_id** (*int*[861]) – taxon_id whose lineage to return
- **names** (*bool*[862]) – if True, the returned list contains the names of the taxa instead of the taxon_id
- **only_ranked** (*bool*[863]) – if True, only taxonomic levels whose rank is in data:*TAXON_RANKS* will be returned
- **with_last** (*bool*[864]) – if True, the passed taxon_id is included in the lineage

Returns lineage of the taxon_id, the elements are *int* if names is False, and *str* when *names* is True. If a taxon has no scientific name, the common name is used. If *only_ranked* is True, the returned list only contains ranked taxa (according to *TAXON_RANKS*).

Return type list[865]

mgkit.taxon.**is_ancestor**(*taxonomy*, *taxon_id*, *anc_id*)

Changed in version 0.1.16: if a *taxon_id* raises a KeyError, False is returned

Determine if the given taxon id (taxon_id) has anc_id as ancestor.

:param *Taxonomy* taxonomy: taxonomy used to test :param int taxon_id: leaf taxon to test :param int anc_id: ancestor taxon to test against

Return bool True if anc_id is an ancestor of taxon_id or their the same

mgkit.taxon.**last_common_ancestor**(*taxonomy*, *taxon_id1*, *taxon_id2*)

New in version 0.1.13.

Finds the last common ancestor of two taxon IDs. An alias to this function is in the same module, called *lowest_common_ancestor* for compatibility.

Parameters

- **taxonomy** – *Taxonomy* instance used to test

[859] https://docs.python.org/3/library/functions.html#int
[860] https://docs.python.org/3/library/functions.html#int
[861] https://docs.python.org/3/library/functions.html#int
[862] https://docs.python.org/3/library/functions.html#bool
[863] https://docs.python.org/3/library/functions.html#bool
[864] https://docs.python.org/3/library/functions.html#bool
[865] https://docs.python.org/3/library/stdtypes.html#list

- **taxon_id1** (*int*[866]) – first taxon ID
- **taxon_id2** (*int*[867]) – second taxon ID

Raturns: int: taxon ID of the lowest common ancestor

Raises *NoLcaFound* – if no common ancestor can be found

mgkit.taxon.**last_common_ancestor_multiple**(*taxonomy*, *taxon_ids*)
New in version 0.2.5.

Applies *last_common_ancestor()* to an iterable that yields *taxon_id* while removing any *None* values. If the list is of one element, that *taxon_id* is returned.

Parameters

- **taxonomy** – instance of *Taxonomy*
- **taxon_ids** (*iterable*) – an iterable that yields taxon_id

Returns the *taxon_id* that is the last common ancestor of all taxon_ids passed

Return type int[868]

Raises

- *NoLcaFound* – when no common ancestry is found or the number of
- *taxon_ids* is 0

mgkit.taxon.**lowest_common_ancestor**(*taxonomy*, *taxon_id1*, *taxon_id2*)
New in version 0.1.13.

Finds the last common ancestor of two taxon IDs. An alias to this function is in the same module, called *lowest_common_ancestor* for compatibility.

Parameters

- **taxonomy** – *Taxonomy* instance used to test
- **taxon_id1** (*int*[869]) – first taxon ID
- **taxon_id2** (*int*[870]) – second taxon ID

Raturns: int: taxon ID of the lowest common ancestor

Raises *NoLcaFound* – if no common ancestor can be found

mgkit.taxon.**parse_ncbi_taxonomy_merged_file**(*file_handle*)
New in version 0.2.3.

Parses the *merged.dmp* file where the merged taxon_id are stored. Available at ftp://ftp.ncbi.nlm.nih.gov/pub/taxonomy/

Parameters **file_handle** (*str*[871], *file*) – file name or handle to the file

Returns dictionary with merged_id -> taxon_id

Return type dict[872]

[866] https://docs.python.org/3/library/functions.html#int
[867] https://docs.python.org/3/library/functions.html#int
[868] https://docs.python.org/3/library/functions.html#int
[869] https://docs.python.org/3/library/functions.html#int
[870] https://docs.python.org/3/library/functions.html#int
[871] https://docs.python.org/3/library/stdtypes.html#str
[872] https://docs.python.org/3/library/stdtypes.html#dict

mgkit.taxon.**parse_ncbi_taxonomy_names_file**(*file_handle*, *name_classes=('scientific name', 'common name')*)

New in version 0.2.3.

Parses the *names.dmp* file where the names associated to a taxon_id are stored. Available at ftp://ftp.ncbi.nlm.nih.gov/pub/taxonomy/

Parameters

- **file_handle** (*str*[873], *file*) – file name or handle to the file
- **name_classes** (*tuple*[874]) – name classes to save, only the scientific and common name are stored

Returns dictionary with merged_id -> taxon_id

Return type dict[875]

mgkit.taxon.**parse_ncbi_taxonomy_nodes_file**(*file_handle*, *taxa_names=None*)

New in version 0.2.3.

Parses the *nodes.dmp* file where the nodes of the taxonomy are stored. Available at ftp://ftp.ncbi.nlm.nih.gov/pub/taxonomy/.

Parameters

- **file_handle** (*str*[876], *file*) – file name or handle to the file
- **taxa_names** (*dict*[877]) – dictionary with the taxa names (returned from *parse_ncbi_taxonomy_names_file()*)

Yields *TaxonTuple* – TaxonTuple instance

mgkit.taxon.**parse_uniprot_taxon**(*line*, *light=True*)

Changed in version 0.1.13: now accepts empty scientific names, for root taxa

Changed in version 0.2.1: added *light* parameter

Deprecated since version 0.4.0.

Parses a Uniprot taxonomy file (tab delimited) line and returns a UniprotTaxonTuple instance. If *light* is True, lineage is not stored to decrease the memory usage. This is now the default.

mgkit.taxon.**taxa_distance_matrix**(*taxonomy*, *taxon_ids*)

New in version 0.1.16.

Given a list of taxonomic identifiers, returns a distance matrix in a pairwise manner by using *distance_two_taxa()* on all possible two element combinations of *taxon_ids*.

Parameters

- **taxonomy** – *Taxonomy* instance
- **taxon_ids** (*iterable*) – list taxonomic identifiers

Returns matrix with the pairwise distances of all *taxon_ids*

Return type pandas.DataFrame

7.1.3 Module contents

Metagenomics Framework

mgkit.**DEBUG = False**

Debug switch for a few functions

[873] https://docs.python.org/3/library/stdtypes.html#str
[874] https://docs.python.org/3/library/stdtypes.html#tuple
[875] https://docs.python.org/3/library/stdtypes.html#dict
[876] https://docs.python.org/3/library/stdtypes.html#str
[877] https://docs.python.org/3/library/stdtypes.html#dict

exception mgkit.**DependencyError**(*package*)
Bases: Exception[878]

Raised if an optional requirement is needed

mgkit.**check_version**(*version*)

mgkit.**cite**(*file_handle=<_io.TextIOWrapper name='<stderr>' mode='w' encoding='UTF-8'>*)
Print citation to the specified stream

7.2 mgkit

[878] https://docs.python.org/3/library/exceptions.html#Exception

CHAPTER 8

Changes

8.1 0.4.1

Sanity checks for several mistakes, including never changed the Programming language version in the setup.py from 2.7. Tested installation under Python 3.6, with tox. Also removed the last bit of code using progressbar2.

8.2 0.4.0

This version was tested under Python 3.5, but the tests (with tox) were run also under Python 2.7. However, from the next release Python 2.7 will be removed gradually (as Python 2.7 won't be supported/patched anymore from next year).

8.2.1 Added

Added *–progress* option to several scripts

mgkit.counts.glm:

- *mgkit.counts.glm.optimise_alpha_scipy()*
- *mgkit.counts.glm.optimise_alpha_scipy_function()*

mgkit.graphs

- *mgkit.graphs.Reaction*
- *mgkit.graphs.merge_kgmls()*
- *mgkit.graphs.parse_kgml_reactions()*

mgkit.taxon:

- *mgkit.taxon.Taxonomy.is_ranked_below()*

8.2.2 Changed

Requires *pandas* version >=0.24 because now a pandas.SparseArray is used for *add-gff-info cov_samtools*. Before, when reading the depth files from **samtools** the array for each sequence was kept in memory, while now only the ones in the GFF file are used.

mgkit.align:

- *mgkit.align.SamtoolsDepth*: uses pandas.SparseArray now. It should use less memory, but needs pandas version > 0.24
- *mgkit.align.read_samtools_depth()*: now returns 3 array, instead of 2. Also added seq_ids to skip lines

mgkit.io.gff

- *mgkit.io.gff.from_gff*: added encoding parameter
- *mgkit.io.gff.parse_gff*: In some cases ASCII decoding is not enough, so it is parametrised now
- *mgkit.io.gff.split_gff_file*: added encoding parameter

mgkit.mappings.eggnog:

- *mgkit.mappings.eggnog.NOGInfo*: made file reading compatible with Python 3

mgkit.snps.funcs:

- *mgkit.snps.funcs.combine_sample_snps()*: added store_uids

8.2.3 Deprecated

- *mgkit.io.blast.add_blast_result_to_annotation()*
- *mgkit.taxon.Taxonomy.read_taxonomy()*: use Taxonomy.read_from_ncbi_dump()
- `mgkit.taxon.Taxonomy.parse_uniprot_taxon()`

8.2.4 Tests

Removed the last portions that used *nosetets* and better integrated pytest with setup.py. Now uses AppVeyor[879] for testing the build and running tests under Python 3.

In cases where the testing environment has no or limited internet connection, tests that require an internet connection can be skipped by setting the following environment variable before running the tests:

```
$ export MGKIT_TESTS_CONN_SKIP=T
```

8.3 0.3.4

General cleanup and testing release. Major changes:

- general moving to Python2 (2.7) and Python3 (3.5+) support, using the future package and when convenient checks for the version of python installed
- setup includes now all the optional dependencies, since this makes it easier to deal with conda environments
- move to pytest from nose, since it allows some functionality that interests me, along with the reorganisation of the test modules and skips of tests that cannot be executed (like mongodb)
- move from urlib to using *requests*, which also helps with python3 support

[879] https://ci.appveyor.com/project/setsuna80/mgkit

- more careful with some dependencies, like the lzma module and msgpack
- addition of more tests, to help the porting to python3, along with a tox configuration
- `matplotlib.pyplot` is still in the *mgkit.plots.unused*, but it is not imported when the parent package is, now. It is still needed in the *mgkit.plots.utils* functions, so the import has been moved inside the function. This should help with virtual environments and test suites
- renamed *mgkit.taxon.UniprotTaxonomy* to *mgkit.taxon.Taxonomy*, since it's really NCBI taxonomy and it's preferred to download the data from there. Same for *mgkit.taxon.UniprotTaxonTuple* to *mgkit.taxon.TaxonTuple*, with an alias for old name there, but will be removed in a later version
- *download_data* was removed. Taxonomy should be downloaded using *download-taxonomy.sh*, and the *mgkit.mappings* is in need of refactoring to remove old and now ununsed functionality
- added *mgkit.taxon.Taxonomy.get_ranked_id()*
- using a sphinx plugin to render the jupyter notebooks instead of old solution
- rerun most of the tutorial and updated commands for newest available software (samtools/bcftools) and preferred the SNP calling from bcftools

8.3.1 Scripts

This is a summary of notable changes, it is advised to check the changes in the command line interface for several scripts

- changed several scripts command line interface, to adapt to the use of *click*
- *taxon-utils lca* has one options only to specify the output format, also adding the option to output a format that can be used by *add-gff-info addtaxa*
- *taxon-utils filter* support the filtering of table files, when they are in a 2-columns format, such as those that are downloaded by *download-ncbi-taxa.sh*
- removed the *eggnog* and *taxonomy* commands from *add-gff-info*, the former since it's not that useful, the latter because it's possible to achieve the same results using taxon-utils with the new output option
- removed the *rand* command of *fastq-utils* since it was only for testing and the FastQ parser is the one from *mgkit.io.fastq*
- substantial changes where made to commands *values* and *sequence* of the *filter-gff* script
- *sampling-utils rand_seq* now can save the model used and reload it
- removed *download_data* and *download_profiles*, since they are not going to be used in the next tutorial and it is preferred now to use BLAST and then find the LCA with *taxon-utils*

8.3.2 Python3

At the time of writing all tests pass on Python 3.5, but more tests are needed, along with some new ones for the blast parser and the scripts. Some important changes:

- *mgkit.io.gff.Annotation* uses its *uid* to hash the instance. This allows the use in sets (mainly for filtering). However, hashing is not supported in *mgkit.io.gff.GenomicRange*.
- *mgkit.io.utils.open_file()* now *always* opens and writes files in binary mode. This is one of the suggestions to keep compatibility between 2.x and 3.x. So if used directly the output must be decoded from *ascii*, which is the format used in text files (at least bioinformatics). However, this is not required for the parsers, like *mgkit.io.gff.parse_gff()*, *mgkit.io.fasta.load_fasta()* along with others (and the correspective *write_* functions): they return unicode strings when parsing and decode into *ascii* when writing.

In general new projects will be worked on using Python 3.5 and the next releases will prioritise that (0.4.0 and later). If bugfixes are needed and Python 3 cannot be used, this version branch (0.3.x) will be used to fix bugs for users.

8.4 0.3.3

8.4.1 Added

- module *mgkit.counts.glm*, with functions used to help the fit of Generalised Linear Models (GLM)
- *mgkit.io.fastq.load_fastq_rename()*
- added *sync*, *sample_stream* and *rand_seq* commands to *sampling-utils* script
- *mgkit.utils.sequence.extrapolate_model()*
- *mgkit.utils.sequence.qualities_model_constant()*
- *mgkit.utils.sequence.qualities_model_decrease()*
- *mgkit.utils.sequence.random_qualities()*
- *mgkit.utils.sequence.random_sequences()*
- *mgkit.utils.sequence.random_sequences_codon()*
- `mgkit.taxon.UniprotTaxonomy.get_lineage_string()`
- `mgkit.taxon.UniprotTaxonomy.parse_gtdb_lineage()`
- *mgkit.net.uniprot.get_gene_info_iter()*

8.4.2 Changed

- *mgkit.io.fastq.write_fastq_sequence()*
- added *seq_id* as a special attribute to *mgkit.io.gff.Annotation.get_attr()*
- *mgkit.io.gff.from_prodigal_frag()* is tested and fixed
- added cache in *mgkit.utils.dictionary.HDFDict*
- *mgkit.utils.sequence.sequence_gc_content()* now returns 0.5 when denominator is 0
- *add-gff-info addtaxa -a* now accept *seq_id* as lookup, to use output from *taxon-utils lca* (after cutting output)

8.4.3 Deprecated

- *mgkit.io.fastq.convert_seqid_to_old()*

8.5 0.3.2

Removed deprecated code

8.6 0.3.1

This release adds several scripts and commands. Successive releases 0.3.x releases will be used to fix bugs and refine the APIs and CLI. Most importantly, since the publishing of the first paper using the framework, the releases will go torward the removal of as much deprecated code as possible. At the same time, a general review of the code to be able to run on Python3 (probably via the *six* package) will start. The general idea is to reach as a full removal of legacy code in 0.4.0, while full Python3 compatibility is the aim of 0.5.0, which also means dropping dependencies that are not compatible with Python3.

8.6.1 Added

- `mgkit.graphs.from_kgml()` to make a graph from a KGML file (allows for directionality)
- `mgkit.graphs.add_module_compounds()`: updates a graph with compounds information as needed
- `mgkit.kegg.parse_reaction()`: parses a reaction equation from Kegg
- added *–no-frame* option to *hmmer2gff - Convert HMMER output to GFF*, to use non translated protein sequences. Also changed the `mgkit.io.gff.from_hmmer()` function to enable this behaviour
- added options *–num-gt* and *–num-lt* to the *values* command of *filter-gff - Filter GFF annotations* to filter based on > and < inequality, in addition to >= and <=
- added *uid* as command in *fasta-utils - Fasta Utilities* to make unique fasta headers
- methods to make `mgkit.db.mongo.GFFDB` to behave like a dictionary (an annotation *uid* can be used as a key to retrieve it, instead of a query), this includes the possibility to iterate over it, but what is yielded are the values, not the keys (i.e. `mgkit.io.gff.Annotation` instances, not *uid*)
- added `mgkit.counts.func.from_gff()` to load count data stored inside a GFF, as is the case when the *counts* command of *add-gff-info - Add informations to GFF annotations* is used'
- added `mgkit.kegg.KeggClientRest.conv()` and `mgkit.kegg.KeggClientRest.find()` operations to `mgkit.kegg.KeggClientRest`
- `mgkit.kegg.KeggClientRest` now caches calls to several methods. The cache can be written to disk using `mgkit.kegg.KeggClientRest.write_cache()` or emptied via `mgkit.kegg.KeggClientRest.empty_cache()`
- added `mgkit.utils.dictionary.merge_dictionaries()` to merge multiple dictionaries where the keys contain different values
- added a Docker file to make a preconfigured mgkit/jupyter build
- added C functions (using Cython) for tetramer/kmer counting. The C functions are the default, with the pure python implementation having a _ appended to their names. This is because the Cython functions cannot have docstrings
- added `mgkit.io.gff.annotation_coverage_sorted()`
- added `mgkit.io.gff.Annotation.to_dict()`
- added `mgkit.plots.utils.legend_patches()` to create matplotlib patches, to be in legends
- added scripts download IDs to taxa tables from NCBI/Uniprot
- added `mgkit.io.utils.group_tuples_by_key()`
- added *cov* command to *get-gff-info - Extract informations to GFF annotations* and *filter-gff - Filter GFF annotations*
- added `mgkit.io.fasta.load_fasta_prodigal()`, to load the fasta file from prodigal for called genes (tested on aminoacids)
- added option to output a JSON file to the *lca* command in ref:*taxon-utils* and *cov* command in *get-gff-info - Extract informations to GFF annotations*

- added a bash script, *sort-gff.sh* to help sort a GFF
- added `mgkit.taxon.UniprotTaxonomy.get_lineage()` which simplifies the use of *mgkit.taxon.get_lineage()*
- added *mgkit.io.fastq.load_fastq()* as a simple parser for fastq files
- added a new script, *sampling-utils - Resampling Utilities*
- added *mgkit.utils.common.union_ranges()* and *mgkit.utils.common.complement_ranges()*
- added *to_hdf* command to *taxon-utils - Taxonomy Utilities* to create a HDF5 file to lookup taxa tables from NCBI/Uniprot
- added *–hdf-table* option to *addtaxa* command in *add-gff-info - Add informations to GFF annotations*
- `mgkit.taxon.UniprotTaxonomy.add_taxon()`, `mgkit.taxon.UniprotTaxonomy.add_lineage()` and `mgkit.taxon.UniprotTaxonomy.drop_taxon()`

8.6.2 Changed

- changed *domain* to *superkingdom* as for NCBI taxonomy in `mgkit.taxon.UniprotTaxonomy.read_from_gtdb_taxonomy()`
- updated scripts documentation to include installed but non advertised scripts (like translate_seq)
- `mgkit.kegg.KeggReaction` was reworked to only store the equation information
- some commands in *fastq-utils - Fastq Utilities* did not support standard in/out, also added the script usage to the script details
- translate_seq now supports standard in/out
- added *haplotypes* parameter to *mgkit.snps.funcs.combine_sample_snps()*
- an annotation from *mgkit.db.mongo.GFFDB* now doesn't include the lineage, because it conflicts with the string used in a GFF file
- an *mgkit.io.gff.Annotation.coverage()* now returns a *float* instead od a *int*
- moved code from package *mgkit.io* to *mgkit.io.utils*
- changed behaviour of *mgkit.utils.common.union_range()*
- removed `mgkit.utills.common.range_substract_()`
- added *progressbar2* as installation requirement
- changed how `mgkit.taxon.UniprotTaxonomy.find_by_name()`

8.6.3 Fixed

Besides smaller fixes:

- *mgkit.plots.abund.draw_circles()* behaviour when *sizescale* doesn't have the same shape as *order*
- parser is now correct for *taxon-utils - Taxonomy Utilities*, to include the Krona[880] options
- ondition when a blast output is empty, hence *lineno* is not initialised when a message is logged

[880] https://github.com/marbl/Krona/wiki

8.6.4 Deprecated

- translate_seq will be removed in version 0.4.0, instead use the similar command in *fasta-utils - Fasta Utilities*

8.7 0.3.0

A lot of bugs were fixed in this release, especially for reading NCBI taxonomy and using the *msgpack* format to save a UniprotTaxonomy instance. Also added a tutorial for profiling a microbial community using MGKit and BLAST (*Profile a Community with BLAST*)

8.7.1 Added

- `mgkit.align.read_samtools_depth()` to read the samtools depth format iteratively (returns a generator)
- `mgkit.align.SamtoolsDepth`, used to cache the samtools depth format, while requesting region coverage
- `mgkit.kegg.KeggModule.find_submodules()`, `mgkit.kegg.KeggModule.parse_entry2()`
- `mgkit.mappings.enzyme.get_mapping_level()`
- `mgkit.utils.dictionary.cache_dict_file()` to cache a large dictionary file (tab separated file with 2 columns), an example of its usage is in the documentation
- `mgkit.taxon.UniprotTaxonomy.read_from_gtdb_taxonomy()` to read a custom taxonomy from a tab separated file. The taxon_id are not guaranteed to be stable between runs
- added *cov_samtools* to *add-gff-info* script
- added `mgkit.workflow.fasta_utils` and correspondent script *fasta-utils*
- added options *-k* and *-kt* to *taxon_utils*, which outputs a file that can be used with Krona *ktImportText* (needs to use *-q* with this script)

8.7.2 Changed

- added *no_zero* parameter to `mgkit.io.blast.parse_accession_taxa_table()`
- changed behaviour of `mgkit.kegg.KeggModule` and some of its methods.
- added *with_last* parameter to `mgkit.taxon.get_lineage()`
- added *–split* option to *add-gff-info exp_syn* and *get-gff-info sequence* scripts, to emulate BLAST behaviour in parsing sequence headers
- added *-c* option to *add-gff-info addtaxa*

8.8 0.2.5

8.8.1 Changed

- added the *only_ranked* argument to `mgkit.taxon.get_lineage()`
- *add-gff-info addtaxa* (*add-gff-info - Add informations to GFF annotations*) doesn't preload the GFF file if a dictionary is used instead of the taxa table

- *blast2gff blastdb* ((*blast2gff - Convert BLAST output to GFF*) offers more options to control the format of the header in the DB used
- added the *sequence* command to *filter-gff* (*filter-gff - Filter GFF annotations*), to filter all annotations on a per-sequence base, based on mean bitscore or other comparisons

8.8.2 Added

- added `mgkit.counts.func.load_counts_from_gff()`
- added `mgkit.io.blast.parse_accession_taxa_table()`
- added `mgkit.plots.abund.draw_axis_internal_triangle()`
- added representation of `mgkit.taxon.UniprotTaxonomy`, it show the number of taxa in the instance
- added `mgkit.taxon.last_common_ancestor_multiple()`
- added *taxon_utils* (*taxon-utils - Taxonomy Utilities*) to filter GFF based on their taxonomy and find the last common ancestor for a reference sequence based on either GFF annotations or a list of taxon_ids (in a text file)

8.9 0.2.4

8.9.1 Changed

- `mgkit.utils.sequence.get_contigs_info()` now accepts a dictionary name->seq or a list of sequences, besides a file name (r536)
- *add-gff-info* **counts** command now removes trailing commas from the samples list
- the axes are turned off after the dendrogram is plo

8.9.2 Fixed

- the **snp_parser** script requirements were set wrong in *setup.py* (r540)
- uncommented lines to download sample data to build documentation (r533)
- *add-gff-info* **uniprot** command now writes the *lineage* attribute correctly (r538)

8.10 0.2.3

The installation dependencies are more flexible, with only *numpy* as being **required**. To install every needed packages, you can use:

```
$ pip install mgkit[full]
```

8.10.1 Added

- new option to pass the *query sequences* to **blast2gff**, this allows to add the correct frame of the annotation in the GFF
- added the attributes *evalue*, *subject_start* and *subject_end* to the output of *blast2gff*. The subject start and end position allow to understand on which frame of the *subject sequence* the match was found
- added the options to annotate the heatmap with the numbers. Also updated the relative example notebook

- Added the option to reads the taxonomy from NCBI dump files, using `mgkit.taxon.UniprotTaxonomy.read_from_ncbi_dump()`. This make it faster to get the taxonomy file
- added argument to return information from *mgkit.net.embl.datawarehouse_search()*, in the form of tab separated data. The argument *fields* can be used when *display* is set to **report**. An example on how to use it is in the function documentation
- added a bash script *download-taxonomy.sh* that download the taxonomy
- added script *venv-docs.sh* to build the documentation in HTML under a virtual environment. matplotlib on MacOS X raises a RuntimeError, because of a bug in virtualenv[881], the documentation can be first build with this, after the script *create-apidoc.sh* is create the API documentation. The rest of the documentation (e.g. the PDF) can be created with *make* as usual, afterwards
- added *mgkit.net.pfam*, with only one function at the moment, that returns the descriptions of the families.
- added *pfam* command to *add-gff-info*, using the mentioned function, it adds the description of the Pfam families in the GFF file
- added a new exception, used internally when an additional dependency is needed

8.10.2 Changed

- using the NCBI taxonomy dump has two side effects:
  - the scientific/common names are kept as is, not lower cased as was before
  - a *merged* file is provided for *taxon_id* that changed. While the old taxon_id is kept in the taxonomy, this point to the new taxon, to keep backward compatibility
- renamed the *add-gff-info gitaxa* command to *addtaxa*. It now accepts more data sources (dictionaries) and is more general
- changed *mgkit.net.embl.datawarehouse_search()* to automatically set the limit at 100,000 records
- the taxonomy can now be saved using msgpack[882], making it faster to read/write it. It's also more compact and better compression ratio
- the *mgkit.plots.heatmap.grouped_spine()* now accept the rotation of the labels as option
- added option to use another attribute for the *gene_id* in the *get-gff-info* script *gtf* command
- added a function to compare the version of MGKit used, throwing a warning, when it's different (*mgkit.check_version()*)
- removed test for old SNPs structures and added the same tests for the new one
- *mgkit.snps.classes.GeneSNP* now caches the number of synonymous and non-synonymous SNPs for better speed
- *mgkit.io.gff.GenomicRange.__contains__()* now also accepts a tuple (start, end) or another GenomicRange instance

8.10.3 Fixed

- a bug in the *gitaxa* (now *addtaxa*) command: when a taxon_id was not found in the table, the wrong *taxon_name* and *lineage* was inserted
- bug in *mgkit.snps.classes.GeneSNP* that prevented the correct addition of values
- fixed bug in *mgkit.snps.funcs.flat_sample_snps()* with the new class

[881] https://github.com/pypa/virtualenv/issues/54
[882] https://github.com/msgpack/msgpack-python

- *mgkit.io.gff.parse_gff()* now correctly handles comment lines and stops parsing if the fasta file at the end of a GFF is found

8.11 0.2.2

8.11.1 Added

- new commands for the **add-gff-info** script (*add-gff-info - Add informations to GFF annotations*):
  - *eggnog* to add information from eggNOG HMMs (at the moment the 4.5 Viral)
  - *counts* and *fpkms* to add count data (correctly exported to mongodb)
  - *gitaxa* to add taxonomy information directly from GI identifiers from NCBI
- added *blastdb* command to **blast2gff** script (*blast2gff - Convert BLAST output to GFF*)
- updated *MGKit GFF Specifications*
- added *gtf* command to **get-gff-info** script (*get-gff-info - Extract informations to GFF annotations*) to convert a GFF to GTF, that is accepted by featureCounts[883], in conjunction with the *counts* command of **add-gff-info**
- added method to *mgkit.snps.classes.RatioMixIn.calc_ratio_flag* to calculate special cases of pN/pS

8.11.2 Changed

- added argument in functions of the *mgkit.snps.conv_func* to bypass the default filters
- added *use_uid* argument to *mgkit.snps.funcs.combine_sample_snps()* to use the *uid* instead of the *gene_id* when calculating pN/pS
- added *flag_values* argument to *mgkit.snps.funcs.combine_sample_snps()* to use *mgkit.snps.classes.RatioMixIn.calc_ratio_flag* instead of *mgkit.snps.classes.RatioMixIn.calc_ratio*

8.11.3 Removed

- deprecated code from the **snps** package

8.12 0.2.1

8.12.1 Added

- added *mgkit.db.mongo*
- added *mgkit.db.dbm*
- added *mgkit.io.gff.Annotation.get_mappings()*
- added *mgkit.io.gff.Annotation.to_json()*
- added *mgkit.io.gff.Annotation.to_mongodb()*
- added *mgkit.io.gff.from_json()*
- added *mgkit.io.gff.from_mongodb()*

[883] http://bioinf.wehi.edu.au/featureCounts/

- added *mgkit.taxon.get_lineage()*
- added *mgkit.utils.sequence.get_contigs_info()*
- added *mongodb* and *dbm* commands to script *get-gff-info*
- added *kegg* command to *add-gff-info* script, caching results and *-d* option to *uniprot* command
- added *-ft* option to *blast2gff* script
- added *-ko* option to *download_profiles*
- added new HMMER tutorial
- added another notebook to the plot examples, for misc. tips
- added a script that downloads from figshare the tutorial data]
- added function to get an enzyme full name (*mgkit.mappings.enzyme.get_enzyme_full_name()*)
- added example notebook for using GFF annotations and the *mgkit.db.dbm*, *mgkit.db.mongo* modules

8.12.2 Changed

- *mgkit.io.blast.parse_uniprot_blast()*
- *mgkit.io.gff.Annotation*
- *mgkit.io.gff.GenomicRange*
- *mgkit.io.gff.from_hmmer()*
- mgkit.taxon.UniprotTaxonomy.read_taxonomy()
- *mgkit.taxon.parse_uniprot_taxon()*
- changed behaviour of *hmmer2gff* script
- changed tutorial notebook to specify the directory where the data is

8.12.3 Deprecated

- mgkit.filter.taxon.filter_taxonomy_by_lineage()
- mgkit.filter.taxon.filter_taxonomy_by_rank()

8.12.4 Removed

- removed old *filter_gff* script

8.13 0.2.0

- added creation of wheel distribution
- changes to ensure compatibility with alter pandas versions
- *mgkit.io.gff.Annotation.get_ec()* now returns a set, reflected changes in tests
- added a *–cite* option to scripts
- fixes to tutorial
- updated documentation for sphinx 1.3

- changes to diagrams
- added decoration to raise warnings for deprecated functions
- added possibility for `mgkit.counts.func.load_sample_counts()` info_dict to be a function instead of a dictionary
- consolidation of some eggNOG structures
- added more spine options in `mgkit.plots.heatmap.grouped_spine()`
- added a *length* property to `mgkit.io.gff.Annotation`
- changed *filter-gff* script to customise the filtering function, from the default one, also updated the relative documentation
- fixed a few plot functions

8.14 0.1.16

- changed default parameter for `mgkit.plots.boxplot.add_values_to_boxplot()`
- Added *include_only* filter option to the default snp filters `mgkit.consts.DEFAULT_SNP_FILTER`
- the default filter for SNPs now use an include only option, by default including only protozoa, archaea, fungi and bacteria in the matrix
- added *widths* parameter to def `mgkit.plots.boxplot.boxplot_dataframe()` function, added function `mgkit.plots.boxplot.add_significance_to_boxplot()` and updated example boxplot notebook for new function example
- *use_dist* and *dist_func* parameters to the `mgkit.plots.heatmap.dendrogram()` function
- added a few constants and functions to calculate the distance matrices of taxa: `mgkit.taxon.taxa_distance_matrix()`, `mgkit.taxon.distance_taxa_ancestor()` and `mgkit.taxon.distance_two_taxa()`
- `mgkit.kegg.KeggClientRest.link_ids()` now accept a dictionary as list of ids
- if the conversion of an Annotation attribute (first 8 columns) raises a ValueError in `mgkit.io.gff.from_gff()`, by default the parser keeps the string version (cases for phase, where is '.' instead of a number)
- treat cases where an attribute is set with no value in `mgkit.io.gff.from_gff()`
- added `mgkit.plots.colors.palette_float_to_hex()` to convert floating value palettes to string
- forces vertical alignment of tick labels in heatmaps
- added parameter to get a consensus sequence for an AA alignment, by adding the *nucl* parameter to `mgkit.utils.sequence.Alignment.get_consensus()`
- added `mgkit.utils.sequence.get_variant_sequence()` to get variants of a sequence, essentially changing the sequence according to the SNPs passed
- added method to get an aminoacid sequence from Annotation in `mgkit.io.gff.Annotation.get_aa_seq()` and added the possibility to pass a SNP to get the variant sequence of an Annotation in `mgkit.io.gff.Annotation.get_nuc_seq()`.
- added *exp_syn* command to *add-gff-info* script
- changed GTF file conversion
- changed behaviour of `mgkit.taxon.is_ancestor()`: if a *taxon_id* raises a KeyError, False is now returned. In other words, if the taxon_id is not found in the taxonomy, it's not an ancestor
- added `mgkit.io.gff.GenomicRange.__contains__()`. It tests if a position is inside the range

- added *mgkit.io.gff.GenomicRange.get_relative_pos()*. It returns a position relative to the GenomicRange start
- fixed documentation and bugs (Annotation.get_nuc_seq)
- added *mgkit.io.gff.Annotation.is_syn()*. It returns True if a SNP is synonymous and False if non-synonymous
- added *to_nuc* parameter to *mgkit.io.gff.from_nuc_blast()* function. It to_nuc is False, it is assumed that the hit was against an amino acidic DB, in which case the phase should always set to 0
- reworked internal of *snp_parser* script. It doesn't use SNPDat anymore
- updated tutorial
- added ipython notebook as an example to explore data from the tutorial
- cleaned deprecated code, fixed imports, added tests and documentation

8.15 0.1.15

- changed name of *mgkit.taxon.lowest_common_ancestor()* to *mgkit.taxon.last_common_ancestor()*, the old function name points to the new one
- added *mgkit.counts.func.map_counts_to_category()* to remap counts from one ID to another
- added *get-gff-info* script to extract information from GFF files
- script *download_data* can now download only taxonomy data
- added more script documentation
- added examples on gene prediction
- added function *mgkit.io.gff.from_hmmer()* to parse HMMER results and return *mgkit.io.gff.Annotation* instances
- added *mgkit.io.gff.Annotation.to_gtf()* to return a GTF line, *mgkit.io.gff.Annotation.add_gc_content()* and *mgkit.io.gff.Annotation.add_gc_ratio()* to calculate GC content and ratio respectively
- added *mgkit.io.gff.parse_gff_files()* to parse multiple GFF files
- added *uid_used* parameter to several functions in *mgkit.counts.func*
- added *mgkit.plots.abund* to plot abundance plots
- added example notebooks for plots
- HTSeq is now required only by the scripts that uses it, *snp_parser* and *fastq_utils*
- added function to convert numbers when reading from htseq count files
- changed behavior of *-b* option in *add-gff-info taxonomy* command
- added *mgkit.io.gff.get_annotation_map()*

8.16 0.1.14

- added ipthon notebooks to the documentation. As of this version the included ones (in *docs/source/examples*) are for two plot modules. Also added a bash script to convert them into rst files to be included with the documentation. The *.rst* are not versioned, and they must be rebuild, meaning that one of the requirements for building the docs is to have IPython[884] installed with the notebook extension

[884] http://ipython.org

- now importing some packages automatically import the subpackages as well
- refactored *mgkit.plots* into a package, with most of the original functions imported into it, for backward compatibility
- added *mgkit.graphs.build_weighted_graph()*
- added *box_vert* parameter in *mgkit.plots.boxplot.add_values_to_boxplot()*, the default will be changed in a later version (kept for compatibility with older scripts/notebooks)
- added an heatmap module to the plots package. Examples are in the notebook
- added *mgkit.align.covered_annotation_bp()* to find the number of bp covered by reads in annotations (as opposed to using the annotation length)
- added documentation to *mgkit.mappings.eggnog.NOGInfo* and an additional method
- added *mgkit.net.uniprot.get_uniprot_ec_mappings()* as it was used in a few scripts already
- added *mgkit.mappings.enzyme.change_mapping_level()* and other to deal with EC numbers. Also improved documentation with some examples
- added *mgkit.counts.func.load_sample_counts_to_genes()* and *mgkit.counts.func.load_sample_counts_to_taxon()*, for mapping counts to only genes or taxa. Also added *index* parameter in *mgkit.counts.func.map_counts()* to accomodate the changes
- added *mgkit.net.uniprot.get_ko_to_eggnog_mappings()* to get mappings of KO identifiers to eggNOG
- added *mgkit.io.gff.split_gff_file()* to split a gff into several ones, assuring that all annotations for a sequence is in the same file; useful to split massive GFF files before filtering
- added *mgkit.counts.func.load_deseq2_results()* to load DESeq2 results in *CSV* format
- added *mgkit.counts.scaling.scale_rpkm()* for scale with rpkm a count table
- added caching options to *mgkit.counts.func.load_sample_counts()* and others
- fixes and improvements to documentation

8.17 0.1.13

- added counts package, including functions to load HTSeq-counts results and scaling
- added *mgkit.filter.taxon.filter_by_ancestor()*, as a convenience function
- deprecated functions in *mgkit.io.blast* module, added more to parse blast outputs (some specific)
- *mgkit.io.fasta.load_fasta()* returns uppercase sequences, added a function (*mgkit.io.fasta.split_fasta_file()*) to split fasta files
- added more methods to *mgkit.io.gff.Annotation* to complete API from old annotations
- fixed *mgkit.io.gff.Annotation.dbq* property to return an **int** (bug in filtering with filter-gff)
- added function to extract the sequences covered by annotations, using the *mgkit.io.gff.Annotation.get_nuc_seq()* method
- added `mgkit.io.gff.correct_old_annotations()` to update old annotated GFF to new conventions
- added *mgkit.io.gff.group_annotations_by_ancestor()* and *mgkit.io.gff.group_annotations_sorted()*
- moved deprecated GFF classes/modules in `mgkit.io.gff_old`
- added *mgkit.io.uniprot* module to read/write Uniprot files

- added *mgkit.kegg.KeggClientRest.get_ids_names()* to remove old methods to get specific class names used to retrieve (they are deprecated at the moment)
- added *mgkit.kegg.KeggModule* to parse a Kegg module entry
- added *mgkit.net.embl.datawarehouse_search()* to search EMBL resources
- made *mgkit.net.uniprot.query_uniprot()* more flexible
- added/changed plot function in *mgkit.plots*
- added enum34 as a dependency for Python versions below 3.4
- changed classes to hold SNPs data: deprecated `mgkit.snps.classes.GeneSyn`, replaced by *mgkit.snps.classes.GeneSNP* which the enum module for *mgkit.snps.classes.SNPType*
- added *mgkit.taxon.NoLcaFound*
- fixed behaviour of `mgkit.taxon.UniprotTaxonomy.get_ranked_taxon()` for newer taxonomies
- change behaviour of `mgkit.taxon.UniprotTaxonomy.is_ancestor()` to use module *mgkit.taxon.is_ancestor()* and accept multiple taxon IDs to test
- `mgkit.taxon.UniprotTaxonomy.load_data()` now accept compressed data and file handles
- added *mgkit.taxon.lowest_common_ancestor()* to find the lowest common ancestor of two taxon IDs
- changed behaviour of *mgkit.taxon.parse_uniprot_taxon()*
- added functions to get GC content, ratio of a sequence and it composition to *mgkit.utils.sequence*
- added more options to **blast2gff** script
- added *coverage*, *taxonomy* and *unipfile* to **add-gff-info**
- refactored **snp_parser** to use new classes
- added possibility to use sorted GFF files as input for **filter-gff** to use less memory (the examples show how to use *sort* in Unix)

8.18 0.1.12

- added functions to elongate annotations, measure the coverage of them and diff GFF files in *mgkit.io.gff*
- added ranges_length and union_ranges to *mgkit.utils.common*
- added script filter-gff, filter_gff will be deprecated
- added script blast2gff to convert blast output to a GFF
- removed unneeded dependencies to build docs
- added script add-gff-info to add more annotations to GFF files
- added *mgkit.io.blast.parse_blast_tab()* to parse BLAST tabular format
- added *mgkit.io.blast.parse_uniprot_blast()* to return annotations from a BLAST tabular file
- added `mgkit.graph` module
- added classes *mgkit.io.gff.Annotation* and *mgkit.io.gff.GenomicRange* and deprecated old classes to handle GFF annotations (API not stable)
- added *mgkit.io.gff.DuplicateKeyError* raised in parsing GFF files
- added functions used to return annotations from several sources

- added option *gff_type* in `mgkit.io.gff.load_gff()`
- added *mgkit.net.embl.dbfetch()*
- added *mgkit.net.uniprot.get_gene_info()* and *mgkit.net.uniprot.query_uniprot()* *mgkit.net.uniprot.parse_uniprot_response()*
- added apply_func_to_values to *mgkit.utils.dictionary*
- added *mgkit.snps.conv_func.get_full_dataframe()*, *mgkit.snps.conv_func.get_gene_taxon_dataframe()*
- added more tests

8.19 0.1.11

- removed *rst2pdf* for generating a PDF for documentation. Latex is preferred
- corrections to documentation and example script
- removed need for joblib library in *translate_seq* script: used only if available (for using multiple processors)
- deprecated `mgkit.snps.funcs.combine_snps_in_dataframe()` and `mgkit.snps.funcs.combine_snps_in_dataframe()`: *mgkit.snps.funcs.combine_sample_snps()* should be used
- refactored some tests and added more
- added *docs_req.txt* to help build the documentation ont readthedocs.org
- renamed `mgkit.snps.classes.GeneSyn` gid and taxon attributes to gene_id and taxon_id. The old names are still available for use (via properties), but the will be taken out in later versions. Old pickle data should be loaded and saved again before in this release
- added a few convenience functions to ease the use of *combine_sample_snps()*
- added function *mgkit.snps.funcs.significance_test()* to test the distributions of genes share between two taxa.
- fixed an issue with deinterleaving sequence data from khmer
- added *mgkit.snps.funcs.flat_sample_snps()*
- Added method to *mgkit.kegg.KeggClientRest* to get names for all ids of a certain type (more generic than the various *get_*_names*)
- added first implementation of *mgkit.kegg.KeggModule* class to parse a Kegg module entry
- *mgkit.snps.conv_func.get_rank_dataframe()*, *mgkit.snps.conv_func.get_gene_map_dataframe()*

CHAPTER 9

Indices and tables

- genindex
- modindex
- search

Python Module Index

m

Index

Symbols

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

Y