Metagenomic framework Documentation

Release 0.3.3

Francesco Rubino

May 22, 2018

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://bitbucket.org/setsuna80/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] http://pythonhosted.org//mgkit/pipeline/tutorial.html
[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

2.1 Docker Instance (with Jupyter Notebook)

A preconfigured Docker instance (user: mgkit, no password) has been configured using the instructions in *Installing on Ubuntu Server 16.04*, including more packages for testing, available at Docker Hub (frubino/mgkit):

```
$ docker run -p 8881:8888 -v host-dir:/home/mgkit/notebooks/ -it frubino/mgkit
```

This command (assuming that Docker is already installed), will pull the instance and present with a bash terminal. IPython and Jupyter are installed and can be used. For a recap of the options:

- *-p 8888:8888* instruct to open the port 8888 on the host
- *-v* mount the directory *host-dir* into the virtual machine */home/mgkit/notebooks/* directory
- *-it* opens an interactive shell

The port to open is port 8881 on the host for using the Jupyter Notebook. The port can be change to what best fits the user. The same applies to the working directory.

After entering the virtual machine prompt, the command that is to be used to launch the notebook is:

```
$ jupyter notebook --ip=0.0.0.0
```

After which in a browser it will possible to access it at *localhost:8881*.

Note: The Dockerfile used to build the instance is available in the repository at: docs/source/extra/Dockerfile

2.2 Requirements

The library is written completely in Python[9] and has been tested with both version 2.6.x and 2.7.x. It has not been tested with Python 3.x, but the authors are trying to write code that is potentially runnable with the use of the *2to3* tool.

[9] http://www.python.org

Most UNIX systems provide a version of Python installed. Latest versions of MacOSX provides Python 2.7.x, while most Linux variants may have different versions installed, sometimes version 3.x, but they usually provide python 2.7.x packages (Archlinux[10] uses version 3.x by default, but a *python2* package is provided). You can check the Python version installed with:

```
$ python --version
```

The library requires these Python packages:

The installation dependencies are flexible, with only *numpy* (version 1.9.2 or later) as being **required**:

```
$ pip install mgkit
```

To install every needed packages, you can use:

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

Other options can be found in the *setup.py* file, in the *extra_requires* dictionary.

The optional dependencies includes:

- scipy
- pandas
- matplotlib
- pysam
- argparse (if Python 2.6 is installed, part of the standard library from 2.7)
- joblib: for script *translate_seq*, to use multiple processors
- HTSeq
- semidbm
- pymongo
- goatools[11] (required by `mgkit.mappings.go`), has package *fisher* as a dependency
- rpy2 >= 2.3.8 (required by `mgkit.utils.r_func`)

2.3 Installing on Ubuntu Server 16.04

You'll need to install the following packages with *apt-get*:

```
$ apt-get install -y velvet bowtie2 python-pip python \
  virtualenv python-dev zlib1g-dev libblas-dev \
  liblapack-dev gfortran libfreetype6-dev libpng-dev \
  fontconfig pkg-config
```

Create a virtual environment to ensure that the correct library versions are installed as explained in *Using virtualenv*.

2.4 Using pip

All dependencies are usually installed either through a package system provided by the running OS or through the pip[12] installer. If you're using a system that's shared with other people, you may not be able to install the

[10] https://www.archlinux.org/
[11] https://github.com/tanghaibao/goatools
[12] http://www.pip-installer.org/

dependencies system-wide, in which case the *–user* option in *pip* may solve the problem[24].

A system-wide installation with *pip* can be done with:

```
$ pip install path/to/library
```

while a user install is done with:

```
$ pip install --user path/to/library
```

all requirements will be downloaded/installed.

2.4.1 Using virtualenv

virtualenv[13] is a system that is used to isolate a Python installation, to make sure no conflicts arise with multiple packages. It's handy if you're developing or testing an application/library, as it provides a clean environment.

Assuming you've already installed *virtualenv*, a virtual environment can be created with:

```
$ virtualenv -p python2 mgkit-env
```

which creates a virtual environment in *mgkit-env*, with the interpreter used being the one linked to *python2*. Activating the environment requires using:

```
$ source mgkit-env/bin/activate
```

assuming you're in the same directory where you created the environment. The pip packager is installed by default with it, so we're going to use it to install the library if you have downloaded it already:

```
$ pip install path/to/library
```

or getting the last version from PyPI[14]:

```
$ pip install mgkit
```

You can also install a specific version:

```
$ pip install mgkit==0.2.0
```

2.4.2 Using the repository

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

2.5 Running Tests

The tests requires the *nosetests* package:

```
$ pip install nose
```

and the package *yanc* is used for coloring the output. If you don't want to install it you can edit the *setup.cfg* and *setup.py* files in the source distribution and delete the *with-yanc* before running the tests.

You can run the tests with:

[24] http://www.pip-installer.org/en/latest/user_guide.html#user-installs
[13] http://www.virtualenv.org/
[14] https://pypi.python.org/pypi
[15] https://bitbucket.org/setsuna80/mgkit

```
$ python setup.py nosetests
```

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

2.6 Building Documentation

Needs sphinx >=1.2.2

- sphinx_rtd_theme
- actdiag
- sphinxcontrib-actdiag
- blockdiag
- sphinxcontrib-blockdiag
- sphinxcontrib-napoleon (we'll be part of sphinx 1.3, needed until then)
- sphinx-argparse

Other libraries:

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

2.7 Troubleshooting

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).

2.7.1 HTSeq

Sometimes HTSeq or numpy fails to install in a clean environment; it's advised to install numpy first:

```
$ pip install numpy
```

and then reissue the library installation:

```
$ pip install path/to/library
```

2.7.2 MacOSX

The version of MacOSX is 10.9 that 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[16] or Macports[17], to install the compilers needed.

[16] http://brew.sh
[17] http://www.macports.org

If you want to use Xcode, you need to install the gfortran compiler, with the package provided here[18]. 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

Scipy

There are different solutions available if you have trouble installing the dependencies on MacOSX, one of which is hosted on this page[19], but installing from source is another option, provided that the Xcode and gfortran are installed.

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:

```
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.

2.7.3 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
```

[18] http://gcc.gnu.org/wiki/GFortranBinariesMacOS
[19] http://fonnesbeck.github.io/ScipySuperpack/
[20] http://matplotlib.org
[21] http://stackoverflow.com/questions/4092994/unable-to-install-matplotlib-on-mac-os-x
[22] http://www.scipy.org/scipylib/building/linux.html

```
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.

2.8 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.

[23] http://www.virtualenv.org/

CHAPTER 3

Metagenomic Pipeline

This section detailed information about example pipelines made using the framework

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.

For a pipeline using another approach, you can refer to the *HMMER Tutorial* section of the documentation. 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[28]
- Picard Tools[29][35]
- GATK[30][36]
- HTSeq[31]
- BLAST[32] or RAPSearch2[33]

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_data -x -p -m EMAIL
```

Where *EMAIL* should be replaced by your email address. The data will be saved in the directory *mg_data* to which we'll refer from now on.

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 informations 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
```

[27] http://bowtie-bio.sourceforge.net/bowtie2/
[28] http://samtools.sourceforge.net
[29] http://picard.sourceforge.net
[35] Picard Tools needs to be found in the directory picard-tools in the same directory as this tutorial.
[30] http://www.broadinstitute.org/gatk/
[36] 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://sourceforge.net/p/htseq/code/HEAD/tree/
[32] http://www.ncbi.nlm.nih.gov/books/NBK279690/
[33] http://omics.informatics.indiana.edu/mg/RAPSearch2/

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.fasta -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 | gzip > assembly.uniprot-filt.gff.gz
```

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

Taxonomic Refinement

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

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

```
$ get-gff-info sequence -f assembly.fasta 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 a supporting file to associate the results with the taxonomic information:

```
$ wget ftp://ftp.ncbi.nih.gov/pub/taxonomy/gi_taxid_nucl.dmp.gz
```

and run the script to add the taxonomic information to the GFF file, also using the LCA algorithm:

```
$ add-gff-info taxonomy -v -t gi_taxid_nucl.dmp.gz -b \
    assembly.uniprot.frag.tab -s 40 -d NCBI-NT -l -x \
    mg_data/taxonomy.pickle \
    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[34] and others, including the taxonomic information, a script is provided that downloads this information into a GFF file:

```
$ add-gff-info uniprot -v --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.fasta assembly.fasta
```

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

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

```
for file in *.fastq; do
    BASENAME=`basename $file .fastq`
    bowtie2 -N 1 -x assembly.fasta -U $file \
    --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 $file `basename $file .bam`-sort;
    rm $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.fasta > assembly.uniprot-update.gff

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

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 mapping and taxonomy information from Uniprot IDs to Kegg Orthologs, EC numbers and eggNOG.

3.1.8 SNP Calling

Before running samtools, which we'll use to do the SNP calling and GATK, to merge the vcf files, the reference *assembly.fa* must be indexed with Picard Tools (tested on version 1.30):

```
$ java -jar picard-tools/picard.jar CreateSequenceDictionary R=assembly.fasta␣
↪O=assembly.dict
```

SAMtools

For calling SNPs, we're going to use SAMtools, as it's the one having lower requirements for this tutorial. The output required by SNPdat and the later part of this tutorial is a vcf, so any software that can output can be used.

Running samtools to make the SNP calling requires a simple loop, as follows:

```
for file in *.bam; do
    samtools mpileup -Iuf assembly.fasta \
    $file | bcftools view -vcg - > `basename $file`.vcf;
done
```

After which, you need to merge all sample vcf files into one, so it can be analysed with the sample specific information, which is needed by the library. This can be done with various packages, but here we'l use GATK (tested on version 3.0-0-g6bad1c6):

```
$ export SAMPLES=$(for file in *.bam.vcf; do echo -V:`basename $file .bam.vcf`
↪$file ;done)
$ java -Xmx10g -jar GATK/GenomeAnalysisTK.jar \
  -R assembly.fasta -T CombineVariants -o assembly.vcf \
  -genotypeMergeOptions UNIQUIFY \
  $SAMPLES
$ unset SAMPLES
```

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 -v -g assembly.uniprot.gff -p assembly.vcf -a assembly.fasta $SAMPLES
$ unset SAMPLES
```

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
```

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 IPython Notebook

The IPython notebook with the data analysis is in the *Explore Data*. A converted python script is included in *Explore Data Python Script*

3.1.11 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.fasta
#remove velvet working directory
rm -R velvet_work

#To use the LCA option and other analysis we need a taxonomy file
download_data -x -p -m YOUR@EMAIL

#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.fasta -db uniprot_sprot.fasta -out \
#        assembly.uniprot.tab -outfmt 6

###RAPSearch
#Index
prerapsearch -d uniprot_sprot.fasta -n uniprot_sprot
#Run
rapsearch -q assembly.fasta -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.fasta 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
        wget ftp://ftp.ncbi.nih.gov/pub/taxonomy/gi_taxid_nucl.dmp.gz

        add-gff-info taxonomy -v -t gi_taxid_nucl.dmp.gz -b \
                assembly.uniprot.frag.tab -s 40 -d NCBI-NT -l -x \
                mg_data/taxonomy.pickle \
                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 -v --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.fasta assembly.fasta
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
        rm $file;
        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.fasta > assembly.uniprot-update.gff
#rename to continue the script
mv assembly.uniprot-update.gff assembly.uniprot.gff
unset SAMPLES

########
#SNP calling using samtools
for file in *.bam; do
        samtools mpileup -Iuf assembly.fasta \
        $file | bcftools view -vcg - > `basename $file`.vcf;
done
```

```
126
127 #Index fasta with Picard tools - GATK requires it
128 java -jar picard-tools/picard.jar CreateSequenceDictionary \
129         R=assembly.fasta O=assembly.dict
130
131 #merge vcf
132 export SAMPLES=$(for file in *.bam.vcf; do echo -V:`basename $file .bam.vcf` $file␣
    ↪;done)
133 java -Xmx10g -jar GATK/GenomeAnalysisTK.jar \
134           -R assembly.fasta -T CombineVariants -o assembly.vcf \
135           -genotypeMergeOptions UNIQUIFY \
136           $SAMPLES
137 unset SAMPLES
138
139 ########
140 #snp_parser
141 export SAMPLES=$(for file in *.bam; do echo -m `basename $file .bam`;done)
142 snp_parser -v -g assembly.uniprot.gff \
143         -p assembly.vcf \
144         -a assembly.fasta \
145          $SAMPLES
146 unset SAMPLES
147
148 ########
149 #Count reads
150 for file in *.bam; do
151         htseq-count -f bam -r pos -s no -t CDS -i uid -a 8 \
152         -m intersection-nonempty $file assembly.uniprot.gff \
153         > `basename $file .bam`-counts.txt
154 done
155
156 ########
157 #eggNOG mappings
158 wget http://eggnog.embl.de/version_3.0/data/downloads/COG.members.txt.gz
159 wget http://eggnog.embl.de/version_3.0/data/downloads/NOG.members.txt.gz
160 wget http://eggnog.embl.de/version_3.0/data/downloads/COG.funccat.txt.gz
161 wget http://eggnog.embl.de/version_3.0/data/downloads/NOG.funccat.txt.gz
162
163 ########
164 #EC names
165 wget ftp://ftp.expasy.org/databases/enzyme/enzclass.txt
```

3.1.12 Explore Data Python Script

3.2 HMMER Tutorial

This example pipeline explore three different aspects from the *Tutorial*):

1. normalisation of metagenomic data using khmer[37]
2. the use of another assembler, MEGAHIT[38]
3. Using Kegg to identify the ortholog genes from the nitrogen metabolism
4. making custom HMMER profiles
5. the use of samtools/bcftools for SNP calling

[37] http://khmer.readthedocs.org/
[38] https://github.com/voutcn/megahit

Hint: The normalisation assembly and profile building steps take a long time, with high relatively high memory requirements. Moreover, the profile building requires an active network connection. The complete assembly is available in this tutorial data, as well as the built HMM profile. These can be used if you are stuck on one of those steps.

3.2.1 Requirements

Warning: The requirements for assembly/normalisation are high for a desktop computer, with the khmer normalisation step using ~6GB of RAM to complete with a reasonable low false positive rate. The computer tested was running Mac OS X v10.11, with 16GB of RAM.

Table 3.1: Software Tested

Software	Version
wget	any
velvet	1.2.10
bowtie2	2.2.6/2.1.0
khmer	2.0
hmmer	3.1b2/3.1b1
clustalo	1.2.1
megahit	1.0.3
samtools	1.2.0
bcftools	1.0

Table 3.2: Python Packages

Package	Version
mgkit	0.2.1
HTSeq	0.6.1
pandas	0.17.1
pysam	0.8.4
scipy	0.16.1
semidbm	0.5.1
matplotlib	1.5
seaborn	0.6

On Mac OS X, some of the software requirements can be installed using Homebrew, with this command:

```
$ brew install wget homebrew/science/velvet homebrew/science/bowtie2 \
homebrew/science/samtools pyenv-virtualenv homebrew/science/hmmer \
homebrew/science/clustal-omega
```

Note: a lot of the shell code is possible in Bash, so you need to make sure the correct shell is loaded.

3.2.2 Metagenomic Data

The data that will be used is from the Analysis of metagenome in a full scale tannery wastewater treatment[39] project on ENA[40]. It has 5 samples, from waste water management:

[39] http://www.ebi.ac.uk/ena/data/view/PRJEB6461
[40] http://www.ebi.ac.uk/ena/

- I (Influent)
- B (Buffering)
- SA (Secondary aeration)
- PA (Primary aeration)
- SD (Sludge digestion)

As it involves waste water management, it's interesting to understand the diversity of the genes involved in the nitrogen metabolism.

The raw reads files can be downloaded with the following command:

```
$ wget ftp://ftp.sra.ebi.ac.uk/vol1/ERA315/ERA315794/fastq/B_R1.fastq.gz \
ftp://ftp.sra.ebi.ac.uk/vol1/ERA315/ERA315794/fastq/B_R2.fastq.gz \
ftp://ftp.sra.ebi.ac.uk/vol1/ERA315/ERA315794/fastq/I_R1.fastq.gz \
ftp://ftp.sra.ebi.ac.uk/vol1/ERA315/ERA315794/fastq/I_R2.fastq.gz \
ftp://ftp.sra.ebi.ac.uk/vol1/ERA315/ERA315794/fastq/PA_R1.fastq.gz \
ftp://ftp.sra.ebi.ac.uk/vol1/ERA315/ERA315794/fastq/PA_R2.fastq.gz \
ftp://ftp.sra.ebi.ac.uk/vol1/ERA315/ERA315794/fastq/SA_R1.fastq.gz \
ftp://ftp.sra.ebi.ac.uk/vol1/ERA315/ERA315794/fastq/SA_R2.fastq.gz \
ftp://ftp.sra.ebi.ac.uk/vol1/ERA315/ERA315794/fastq/SD_R1.fastq.gz \
ftp://ftp.sra.ebi.ac.uk/vol1/ERA315/ERA315794/fastq/SD_R2.fastq.gz
```

3.2.3 Digital Normalisation and QC

One aspect that makes the assembly of metagenomic particularly complex is the different coverage of the different organisms that are found in a sample. One solution is the use of a kmer counting such as **khmer**, as it allows to reduce the differences. This also reduces the memory requirements of the assembly.

The first step is to interleave the paired end reads into one file, which can be done using the *interleave-reads.py* script from the *khmer* package. As the quality, observer using FastQC[41] is poor in the last 20 bp, the following bash code uses Python and HTSeq to cut the last 20 bp from both reads files, using IO streams. This avoids the use of temporary files, but it's not mandatory.

```
$ interleave-reads.py --gzip -o all-interleaved.fq.gz \
<(
python - <<END
import HTSeq, sys, glob
files = glob.glob('*R1.fastq.gz')
for fname in files:
    for record in HTSeq.FastqReader(fname):
        record = record[:-20] # cut 20 bp
        # HTSeq adds '[part]' to the header, the next line cut it
        record.name = record.name[:-6]
        record.write_to_fastq_file(sys.stdout)
END
) \
<(
python - <<END
import HTSeq, sys, glob
files = glob.glob('*R2.fastq.gz')
for fname in files:
    for record in HTSeq.FastqReader(fname):
        record = record[:-20]
        record.name = record.name[:-6]
        record.write_to_fastq_file(sys.stdout)
END
)
```

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

Note: The interleave passage is especially important since some khmer scripts had problems parsing the new Casava header of Fastq files.

The file *all-interleaved.fq.gz* will be created, containing the trimmed sequences, interleaved.

The digital normalisation can be done using the *normalize-by-median.py*, part of the *khmer* package. In this dataset, with the parameters used, around 7% of data can be excluded from the assembly:

```
$ normalize-by-median.py -k 24 -p -o normalised.fq.gz \
  --gzip -M 6e9 all-interleaved.fq.gz
```

Producing a *normalised.fq.gz* file that can be used with an assembler.

3.2.4 Assembly

The assembly, as usual can be done with any assembler, as long as its output is a FASTA file. MEGAHIT can be used to assembler this data using the following command:

```
$ megahit --presets meta --verbose --min-contig-len 100 \
  --12 normalised.fq.gz -o megahit-out
```

One things that can create problems in software such as samtools, is the presence of spaces in the sequence headers. The following python code (executed in a BASH shell) can be used to give unique names to each sequence and keep in a file the names assigned by the assembler for any future reference:

```
$ python - <<END
from mgkit.io import fasta
from uuid import uuid4
import json

seq_dict = {}
with open('final-contigs.fa', 'w') as f:
    for name, seq in fasta.load_fasta('megahit-out/final.contigs.fa'):
        uid = str(uuid4())
        seq_dict[uid] = name
        fasta.write_fasta_sequence(f, uid, seq)

json.dump(seq_dict, open('seq-dict.json', 'w'))
END
```

This small script will read all sequences from the output of MEGAHIT, the *final.contigs.fa* file and replaces the original sequence headers with new ones, using the *uuid4* function. This function generate random unique identifiers without spaces. A dictionary with the original sequence headers is saved as a JSON file.

3.2.5 Download Data

It is necessary to download the taxonomy to download the genes from Kegg (for this tutorial). Moreover the taxonomy will be used in later steps and to download it, the MGKit script *download_data* can be used:

```
$ download_data -p -x -m EMAIL
```

This will save a *taxonomy.pickle* file in the subdirectory *mg_data*. More information at the script documentation (*download-data - Download Taxonomy from NCBI*)

3.2.6 Create Profiles

To download only a portion of Kegg, in this case the genes from the nitrogen metabolism, it's needed to use the Kegg REST API to retrieve the gene IDs. MGKit include a client class for this in the *mgkit.kegg* module, called *KeggClientRest*. Its method *link_ids* returns a data structure that contains the list of genes in the Nitrogen Metabolism (ID: ko00910).

The script can be executed on a bash shell with the following command:: $ export GENES='python - <<END from mgkit import kegg k = kegg.KeggClientRest() print " ".join(k.link_ids('ko', 'ko00910')['ko00910']) END'

The command will export an environment variable called *GENES* that can be passed to the *download_profiles* script:

```
$ download_profiles -o profiles -ko $GENES -r order -l bacteria \
-m EMAIL -t mg_data/taxonomy.pickle
```

The script will retrieve all the sequences in the *profiles* directory, for the genes in the *GENES* environment variable. The genes will be download as one for file for each order of bacteria AND gene. Each file will contain, all sequences for that ortholog, found in Uniprot for a particular order of bacteria. Orders with just one sequence will be skipped.

After the script finish its execution, the *GENES* variable can be unset:

```
$ unset GENES
```

More information about the script is in *download-profiles - Download Custom Profiles*.

Alignment and HMM profiles

The files created must be aligned using Clustal Omega or another similar software and for each alignment a HMM profile can be built using the *hmmbuild* from the HMMER package. An example, using a BASH loop:

```
$ for file in profiles/*.fa; do
        echo $file `basename $file .fa`
    clustalo -i $file -o profiles/`basename $file .fa`.afa ;
    hmmbuild profiles/`basename $file .afa`.hmm profiles/`basename $file .fa`.afa;
done
```

Each single profile can then concatenated using *cat*:

```
$ cat profiles/*.hmm > profiles.hmm
```

3.2.7 Gene Prediction

With the HMM profiles in one file, *hmmsearch* can be used to searc for similarity in the assembly. Before that, *hmmsearch* can only work on aminoacid sequences, so the assembly must be translated in the possible frames. The recommended way to do this is to use the *translate_seq* script included with MGKit:

```
$ translate_seq final-contigs.fa final-contigs.aa.fa
```

Warning: The problem with other software to translate the assembly is that the script that convert the result of *hmmsearch* into a GFF file needs the information about the frame and strand. *translate_seq* append a suffix to the sequence header to indicate it. As an example, for a sequence named *contig0001*, the following sequences headers will be found: *contig0001-f0*, *contig0001-f1*, *contig0001-f2*, *contig0001-r0*, *contig0001-r1*, *contig0001-r2*. The *f* stands for the forward (+ strand), *r* for the reverse (- strand), the number indicates the frame, from 1 to 2.

hmmsearch can then be launched using the following command:

```
$ hmmsearch -o /dev/null --domtbl hmmer_dom-table.txt profiles.hmm final-contigs.
↪aa.fa
```

The only file needed by MGKit is the domain table, stored in the file *hmmer_dom-table.txt*.

GFF Creation

The output of *hmmsearch* can then be supplied to the *hmmer2gff* script included in MGKit, which converts it to a GFF file:

```
$ hmmer2gff -d -o assembly.gff final-contigs.aa.fa hmmer_dom-table.txt
```

The command will create a *assembly.gff* file from all hits in the domain table. In this case the e-value filter was disabled (*-d* option), because the collection of files may be too small.

3.2.8 GFF Filtering

The GFF filtering works in the same way as explained in *Tutorial* and more informations can be found in the script manual (*filter-gff - Filter GFF annotations*). One thing to point out is that most scripts and commands in MGKit (and other software) allow the use of pipes, concatenating multiple commands in one line to avoid the use of temporary files. The following command can be run on a BAST shell:

```
$ cat assembly.gff | filter-gff values -b 40 | filter-gff \
overlap -s 100 | \ add-gff-info kegg -c EMAIL \
-v -d > assembly.filt.gff
```

The commands do the following, in sequence (between |):

1. output to the standard output the content of *assembly.gff*
2. only keeps annotations that have a bit score of at least 40
3. filters overlapping annotations (for at least 100 bp), keeping the one with the highest bit score
4. add the names of the genes getting them from Kegg

The result is then outputted into the *assembly.filt.gff* (after the >). This is not enforced, but can be used to speed up some commands, as nothing is written to disk, and avoid confusion when managing multiple temporary files.

3.2.9 GFF Additions

At this point, we have most of the information we need to continue with the analysis, but there is one mandatory and one optional step which can be done. The GFF after filtering can is not enough to continue with the diversity analysis, as we need coverage information about each predicted gene. We can also refine the taxonomic assignment, which is detailed in *Tutorial*.

Computing the gene coverage can be done before filtering, but the number of annotations would be too high, so it's preferred to add coverage information after filtering the GFF. Also, because we needs alignment files for each sample to compute the gene coverage, it is advised to makes the alignment files in parallel with the GFF filtering, to speed up the pipeline.

Alignments

To create alignments for each sample, the assembly file must first be indexed, with the folowing command:

```
$ bowtie2-build final-contigs.fa final-contigs
```

The following BASH loop creates the BAM file for each sample:

```
$ for file in *R1.fastq.gz; do
        BASENAME=`basename $file _R1.fastq.gz`
        echo $file $BASENAME "$BASENAME"_R2.fastq.gz
        bowtie2 -N 1 -x final-contigs --local --sensitive-local \
        -1 $file -2 "$BASENAME"_R2.fastq.gz \
        --rg-id $BASENAME --rg PL:Illumina --rg PU:Illumina-MiSeq \
        --rg SM:$BASENAME --no-unal \
        2> $BASENAME.log | samtools view -Sb - > $BASENAME.bam;
done
```

The loop uses the list of reads file from the first element of the pairs (R1.fastq.gz files) to automate the process, resulting in files that are in the form *SAMPLE.bam* (e.g. B.bam). A log file is also kept, using the same file name and *log* extension.

The alignments made need to be sorted, and the following BASH loop give an example of this:

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

Note: samtools 1.2 (at least) needs to specify the format and temp file prefix. Later version may not require it.

The result is BAM files with the sample names, as before.

Coverage and Expected SNPs

The alignments can now be used also to add coverage information to the GFF file, which is needed for another script in the pipeline. Because sample names are needed, as the per sample coverage information is store as *sample_cov*, adding a suffix *_cov* after the sample name, the following command infers the sample names from the BAM files:

```
$ export SAMPLES=$(for file in *.bam; do echo -a `basename $file .bam`,$file ;done)
```

The *SAMPLES* environment variable is used for the script *add-gff-info*, in particular its *coverage* command:

```
$ add-gff-info coverage $SAMPLES assembly.filt.gff | \
add-gff-info exp_syn -r final-contigs.fa > assembly.filt.cov.gff
```

To also add the information about the expected number of synonymous and non-synonymous changes for each annotation, the *exp_syn* command for the *add-gff-info* was used. The file is then saved as *assembly.filt.cov.gff*.

The *SAMPLES* variable can now be unset:

```
$ unset SAMPLES
```

3.2.10 SNP Calling

In this tutorial, it was decided to use samtools/bcftools to call SNPs, as GATK can require too much memory and time. The following command calls SNPs for all samples, writing a *assembly.vcf* file to disk:

```
$ samtools mpileup -t DP,SP,DPR,DV -ugf final-contigs.fa *.bam \
| bcftools call -vmO v > assembly.vcf
```

Note: samtools version 1.2 and bcftools version 1.0 were tested

Using the VCF file created, the *snp_parser* script included in MGKit can be used:

```
export SAMPLES=$(for file in *.bam; do echo -m `basename $file .bam`;done)

snp_parser -s -v -g assembly.filt.cov.gff -p assembly.vcf \
-a seqs/final-contigs.fa $SAMPLES

unset SAMPLES
```

The *SAMPLES* variable is dynamically created to help write the command line for *snp_parser* and after its execution can be unset. The command line is different compared to *Tutorial*, as *-s* was added to distinguish the type of sample information in the VCF, as outputted by *bcftools*, compared to one created using GATK (the default type for *snp_parser*).

3.2.11 IPython Notebook

The IPython notebook with the data analysis is in the *Explore Data*. A converted python script is included in *Explore Data Python Script*

3.2.12 Full Bash Script

```
#!/bin/bash

# Some tools require a contact email
export EMAIL=your@email

# Requirements
#
# software          - version tested
# mgkit           - 0.2.1
# wget                  - any
# velvet           - 1.2.10
# bowtie2           - 2.2.6 / 2.1.0
# khmer           - 2.0
# hmmer           - 3.1b2 / 3.1b1
# clustalo           - 1.2.1
# megahit          - 1.0.3

# python packages
# HTSeq>=0.6.1
# pandas>=0.17.1
# pysam>=0.8.4
# scipy>=0.16.1
# semidbm>=0.5.1
# matplotlib>=1.5
# seaborn>=0.6

# Requirements specific for Mac OS X (El Capitan, 10.11), uncomment these lines
# brew install wget homebrew/science/velvet homebrew/science/bowtie2 \
# homebrew/science/samtools pyenv-virtualenv homebrew/science/hmmer \
# homebrew/science/clustal-omega

# Memory ~6 GB for khmer normalisation
# ~3.5 GB for megahit
# ~0.5 GB for bowtie2

```

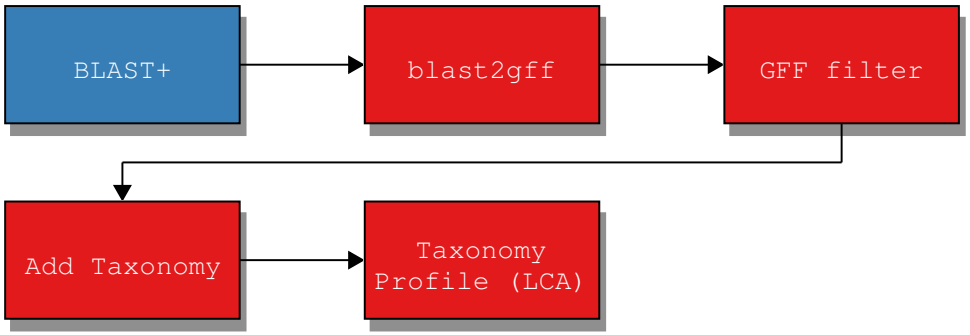
```
# Using data from the following project:
# http://www.ebi.ac.uk/ena/data/view/PRJEB6461
# Samples:
# I (Influent)
# B (Buffering)
# SA (Secondary aeration)
# PA (Primary aeration)
# SD (Sludge digestion)
mkdir seqs
cd seqs
#download data
wget ftp://ftp.sra.ebi.ac.uk/vol1/ERA315/ERA315794/fastq/B_R1.fastq.gz ftp://ftp.sra.ebi.ac.uk/vol1/ERA315/ERA315794/fastq/B_R2.fastq.gz ftp://ftp.sra.ebi.ac.uk/vol1/ERA315/ERA315794/fastq/I_R1.fastq.gz ftp://ftp.sra.ebi.ac.uk/vol1/ERA315/ERA315794/fastq/I_R2.fastq.gz ftp://ftp.sra.ebi.ac.uk/vol1/ERA315/ERA315794/fastq/PA_R1.fastq.gz ftp://ftp.sra.ebi.ac.uk/vol1/ERA315/ERA315794/fastq/PA_R2.fastq.gz ftp://ftp.sra.ebi.ac.uk/vol1/ERA315/ERA315794/fastq/SA_R1.fastq.gz ftp://ftp.sra.ebi.ac.uk/vol1/ERA315/ERA315794/fastq/SA_R2.fastq.gz ftp://ftp.sra.ebi.ac.uk/vol1/ERA315/ERA315794/fastq/SD_R1.fastq.gz ftp://ftp.sra.ebi.ac.uk/vol1/ERA315/ERA315794/fastq/SD_R2.fastq.gz

# Normalisation of the reads
# Khmer when using paired reads works best when are interleaved
# Moreover, it requires the older Casava type of header
# The sequences are trimmed by 20 bp and fed to the khmer
# script interleave-reads.py (the quality is not good in the last 20 bp)
interleave-reads.py --gzip -o all-interleaved.fq.gz \
<(
python - <<END
import HTSeq, sys, glob
files = glob.glob('*R1.fastq.gz')
for fname in files:
    for record in HTSeq.FastqReader(fname):
        record = record[:-20] # cut 20 bp
        # HTSeq adds '[part]' to the header, the next line cut it
        record.name = record.name[:-6]
        record.write_to_fastq_file(sys.stdout)
END
) \
<(
python - <<END
import HTSeq, sys, glob
files = glob.glob('*R2.fastq.gz')
for fname in files:
    for record in HTSeq.FastqReader(fname):
        record = record[:-20]
        record.name = record.name[:-6]
        record.write_to_fastq_file(sys.stdout)
END
)
# Normalisation takes out almost 7% of the reads
normalize-by-median.py -k 24 -p -o normalised.fq.gz --gzip -M 6e9 all-interleaved.fq.gz
# rm all-interleaved.fq.gz

# megahit manages to assemble the data
# add -t N to use more processors
megahit --presets meta --verbose --min-contig-len 100 --12 normalised.fq.gz -o megahit-out
# megahit used spaces in the sequence headers so it may create problems later
# one solution is to assign to each contig a new random sequence and then
# keep track of those in a json dictionary for later (if needed)
python - <<END
```

```
from mgkit.io import fasta
from uuid import uuid4
import json

seq_dict = {}
with open('final-contigs.fa', 'w') as f:
    for name, seq in fasta.load_fasta('megahit-out/final.contigs.fa'):
        uid = str(uuid4())
        seq_dict[uid] = name
        fasta.write_fasta_sequence(f, uid, seq)

json.dump(seq_dict, open('seq-dict.json', 'w'))
END
rm -R megahit-out
cd ..

#download offline data (only taxonomy)
download_data -p -x -m $EMAIL

# Get the nitrogen metabolism KOs
export GENES=`python - <<END
from mgkit import kegg
k = kegg.KeggClientRest()
print " ".join(k.link_ids('ko', 'ko00910')['ko00910'])
END`

download_profiles -o profiles -ko $GENES -r order -l bacteria -m $EMAIL -t mg_data/
↪taxonomy.pickle

unset GENES

#Profile alignments and build
for file in profiles/*.fa; do
        echo $file `basename $file .fa`
    clustalo -i $file -o profiles/`basename $file .fa`.afa ;
    hmmbuild profiles/`basename $file .afa`.hmm profiles/`basename $file .fa`.afa;
done
#Make one file for all profiles
cat profiles/*.hmm > profiles.hmm

#translation of the assembly in AA
translate_seq seqs/final-contigs.fa final-contigs.aa.fa

#HMMER command line
# add --cpu N to use more processors
hmmsearch -o /dev/null --domtbl hmmer_dom-table.txt profiles.hmm final-contigs.aa.
↪fa
#convert into annotations
hmmer2gff -d -o assembly.gff final-contigs.aa.fa hmmer_dom-table.txt
# Filter annotations and add information from Kegg
# most scripts working on GFF files allow to pipe their input/output
# First remove annotations with a bit score less than 40, then filter
# overlapping annoations and finally add gene descriptions and pathways for
# for each annotations. `cat` is not necessary, since the input/output can
# specified with a `-` (dash), which is standard
cat assembly.gff | filter-gff values -b 40 | filter-gff overlap -s 100 | \
add-gff-info kegg -c $EMAIL -v -d > assembly.filt.gff

# bowtie2
# index
bowtie2-build seqs/final-contigs.fa final-contigs
# alignments, read groups are added, using a sensitive approach for alignment
# the reads that are not aligned are not included in the BAM files (--no-unal
```

```
# option)
for file in seqs/*R1.fastq.gz; do
        BASENAME=`basename $file _R1.fastq.gz`
        echo $file $BASENAME seqs/"$BASENAME"_R2.fastq.gz
        bowtie2 -N 1 -x final-contigs --local --sensitive-local \
        -1 $file -2 seqs/"$BASENAME"_R2.fastq.gz \
        --rg-id $BASENAME --rg PL:Illumina --rg PU:Illumina-MiSeq \
        --rg SM:$BASENAME --no-unal \
        2> $BASENAME.log | samtools view -Sb - > $BASENAME.bam;
done

# samtools
# sorting (by position) the BAM files and indexing them
for file in *.bam; do
        # samtools 1.2 (at least) needs to specify the format and temp file prefix
        samtools sort -T tmp.$file -O bam -o `basename $file .bam`-sort.bam $file;
        mv `basename $file .bam`-sort.bam $file;
        samtools index $file;
done

#index for the assembly (required by GATK and samtools)
#.fai index
samtools faidx seqs/final-contigs.fa

#add coverage data
########
#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.filt.gff | add-gff-info \
        exp_syn -r seqs/final-contigs.fa > assembly.filt.cov.gff
unset SAMPLES

# samtools
# -u uncompressed
# -g binary BCF (it's piped to bcftools)
# -f reference fasta
# -t several important per sample information
# bcftools
# call - new command (instead of view)
# -v - vcf output
# -m - type of calling
# -O v - uncompressed vcf
samtools mpileup -t DP,SP,DPR,DV -ugf seqs/final-contigs.fa *.bam \
| bcftools call -vmO v > assembly.vcf

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

3.2.13 Explore Data Python Script

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 *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[42] 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

[42] 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[43] 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).

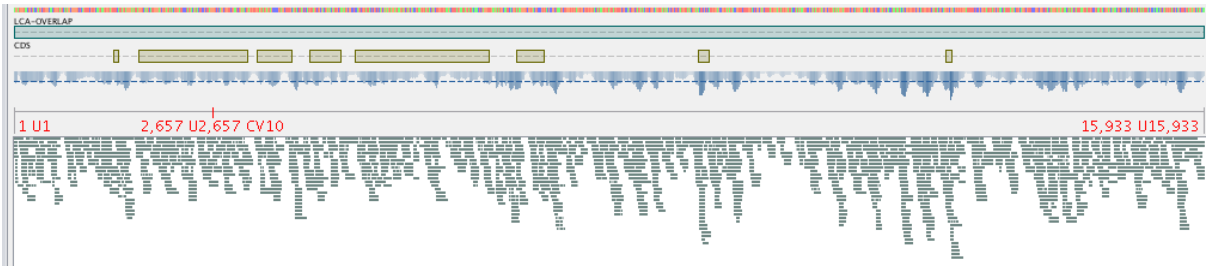
The produced files with **krona** extension can be the be used with the **ktImportText** (or **ImportText** if Krona Tools were not installed). The **-q** option of the script must be used:

```
$ 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[44]. 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 taxonomic group it belongs to.

There different ways to predict genes, with some relying on general function domains like the ones from PFam[45] or others. This type of collections is very useful in identifying proteins in an unknown sequence. The main

[43] https://github.com/marbl/Krona/wiki
[44] https://ics.hutton.ac.uk/tablet/
[45] http://pfam.xfam.org/

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[46] and Kegg Orthologs[47], 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[48]. 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

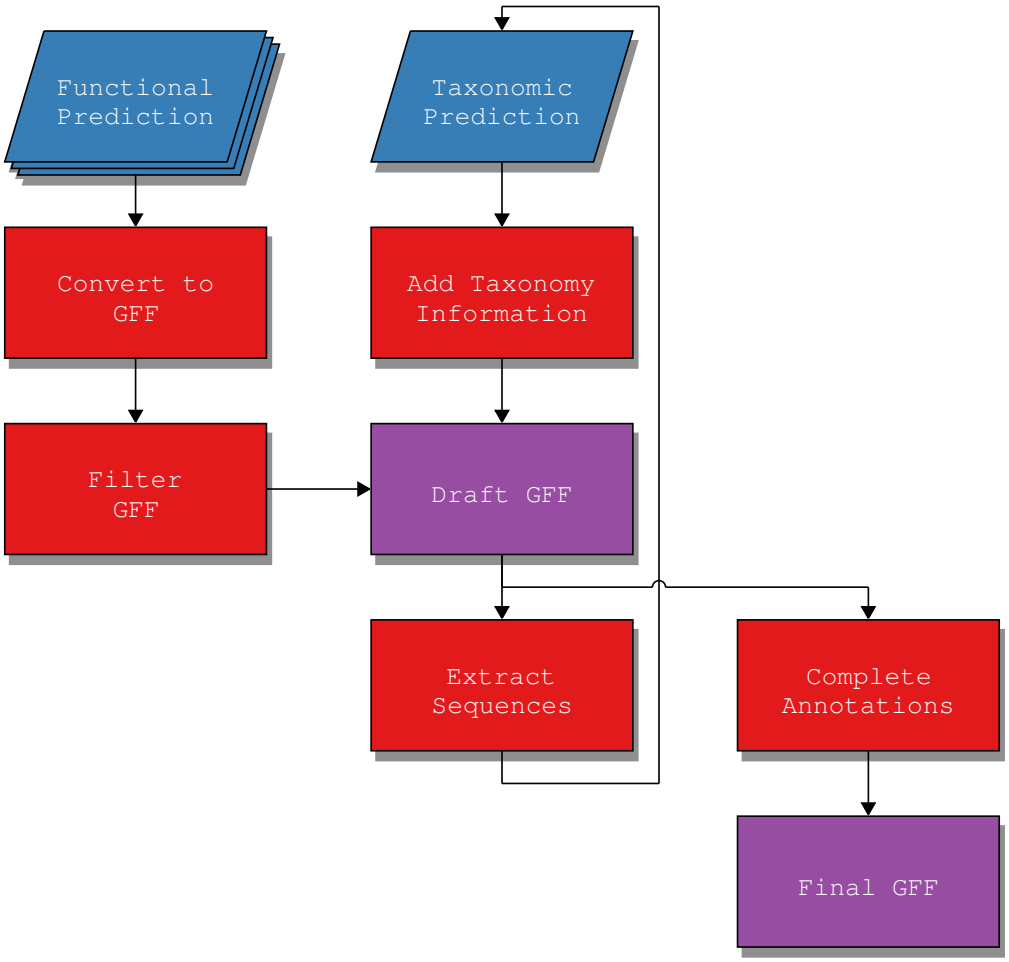
The prediction of genes requires both a collection and specific softwares to find homologous sequneces. There are two classes of software that can be used for gene prediction: one is profile based and the second uses similarity search.

An example of software using profile search is HMMER[49]. 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+[50], 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.

[46] http://eggnog.embl.de
[47] http://www.kegg.jp/kegg/ko.html
[48] http://www.uniprot.org
[49] http://hmmer.janelia.org/
[50] 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[51] 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[52] and RAPSearch2[53].

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.

[51] http://www.sequenceontology.org/gff3.shtml
[52] http://www.drive5.com/usearch/
[53] 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 *download-profiles - Download Custom 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[54] 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[55] gene overlaps.

Table 3.3: 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	

[54] http://www.clustal.org/omega/
[55] https://genome.ucsc.edu

Table 3.4: 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.

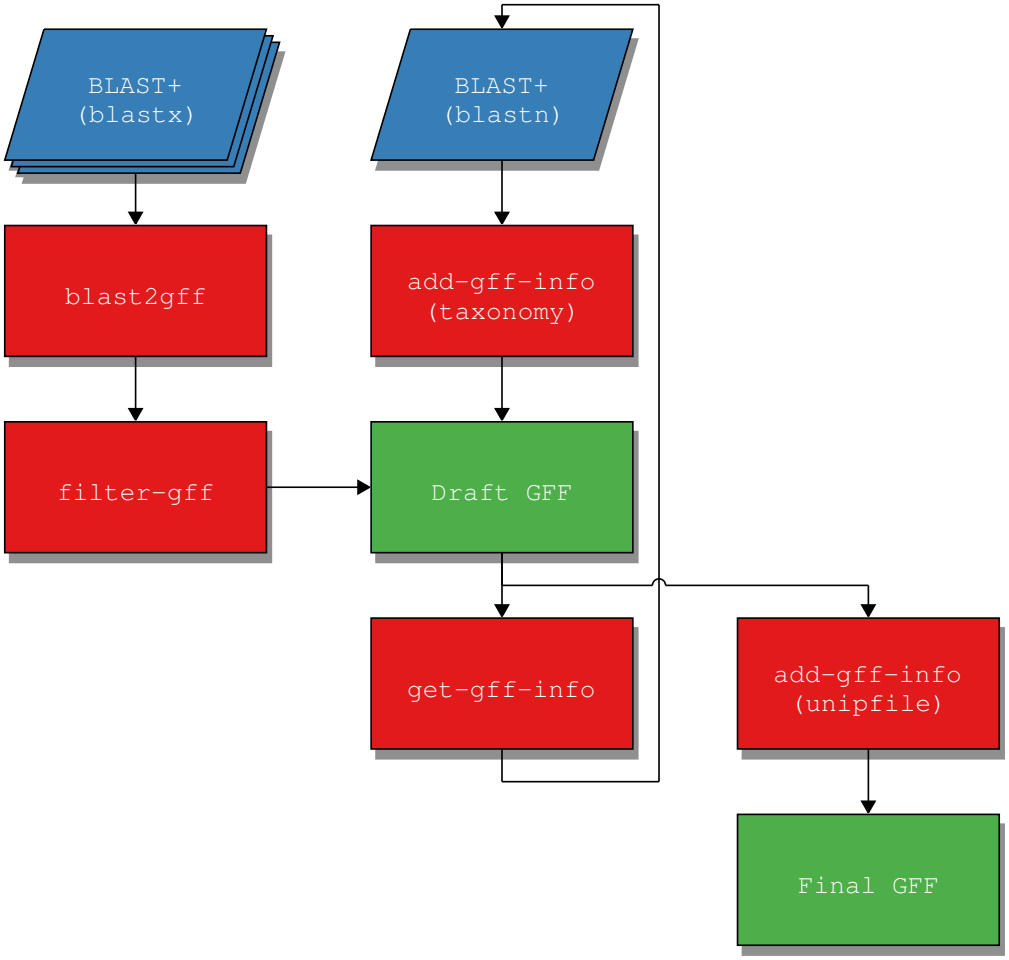
The process add mapping attributes to the GFF file, with eggNOG and Kegg Orthologs for example, while also completing the taxonomic assignment of annotations that were not assigned taxonomically. This can be done with an included script *add-gff-info - Add informations to GFF annotations* and the completed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

4.1.2 Options

4.2 filter-gff - Filter GFF annotations

4.2.1 Overview

4.2.2 Options

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

4.3.1 Overview

4.3.2 Options

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

4.4.1 Overview

4.4.2 Options

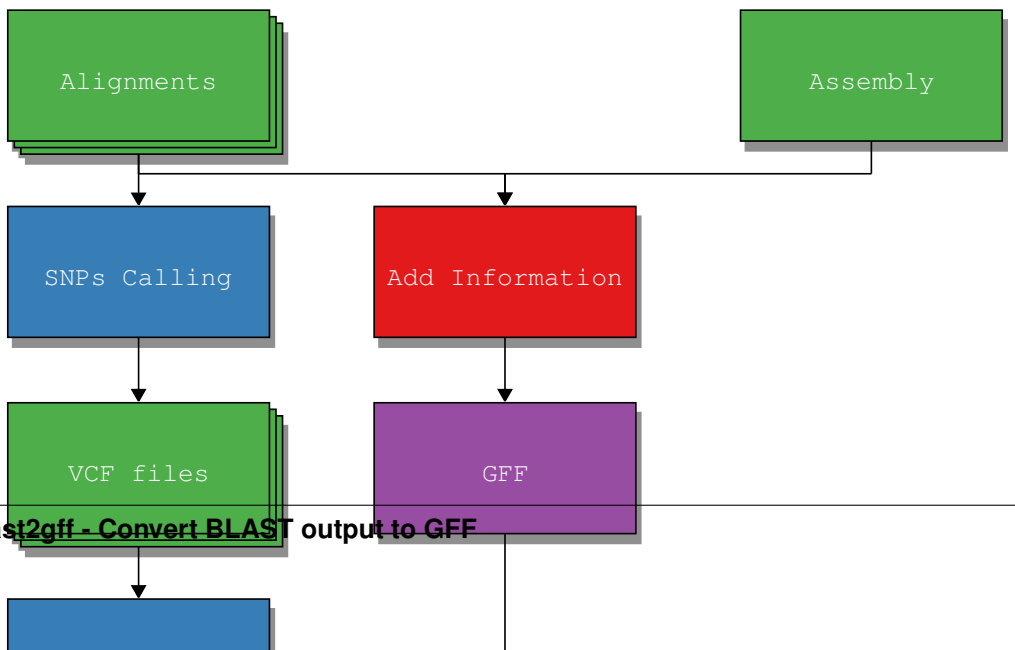
4.5 hmmer2gff - Convert HMMER output to GFF

4.5.1 Overview

4.5.2 Options

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[56] 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.[57]

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

4.6.2 Script Reference

4.6.3 Options

4.7 download-taxonomy.sh 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[58], 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[59]) that is not installed by default.

[56] http://code.google.com/p/snpdat/
[57] This step is done separately because it's both time consuming and can helps to paralellise later steps
[58] ftp://ftp.ncbi.nlm.nih.gov/pub/taxonomy/taxdump.tar.gz
[59] https://github.com/msgpack/msgpack-python

4.8 taxon-utils - Taxonomy Utilities

4.8.1 Overview

4.8.2 Options

4.9 fasta-utils - Fasta Utilities

4.9.1 Overview

4.9.2 Options

4.10 fastq-utils - Fastq Utilities

4.10.1 Overview

4.10.2 Options

4.11 json2gff - Convert JSON to GFF

4.11.1 Overview

4.11.2 Options

4.12 download-profiles - Download Custom Profiles

4.12.1 Overview

This script downloads sequence data for each gene of interest (ortholog) and all the specified taxa. The files that are downloaded with this script can then be used to create HMMER profiles, to search for similarity in a aminoacidic or nucleotidic sequence.

4.12.2 Limitations

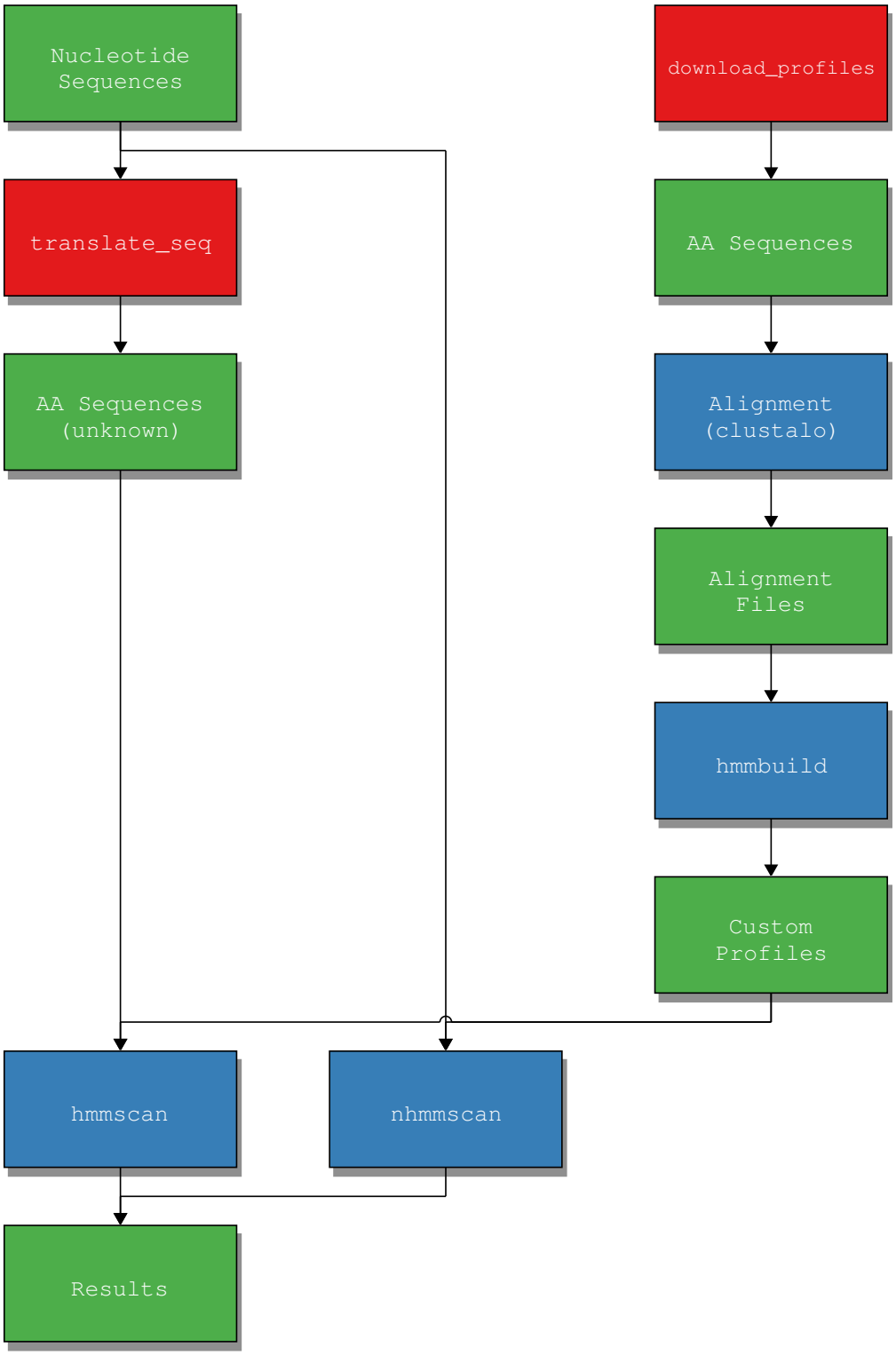
At the moment, the script uses Kegg Orthologs as the ortholog database.

> **Warning:** Some taxa may still black listed, because they are not relevant to the rumen microbiome. If you find such thing to occur to you, please contact me or open an issue on the repository.

4.12.3 Required Data

The script requires data from Kegg Orthologs and Uniprot to be downloaded, before it can be used. The script **download_data** (*download-data - Download Taxonomy from NCBI*) automates the process.

4.12.4 Workflow for Custom Profiles



The process of building the profiles to be used with HMMER is a step that involves several tasks (illustrated in the Workflow above):

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. This sctipt, *download_profiles* can be used. The aminoacidic sequences downloaded are then aligned using Clustal Omega (or other) and for each alignment a profile is built.

HMMER required the use of aminoacidic sequences, to be match against the profiles. The *translate_seq* script can be used to translate nucleotidic sequences into aminoacidic ones. However, the last version of HMMER should be able to match nucleotidic sequences, but it was not tested by us. The example Workflow above illustrate that.

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.

4.12.5 Usage

The default behaviour is to download all Kegg Orthologs for all taxa in the given taxonomy. Taxa can be filtered by both lineage (e.g. archaea, carnivora, etc.) and rank (e.g. genus, family, etc.). Another option is to specify the KO and taxa IDs to download.

Taxa Filters

The way a taxon is specified is through a few different rules:

- specific **taxon ids** in uniprot
- a specific **taxon rank** (e.g.: genus, phylum, etc.)
- optional lineage filter: the lineage filter make sure that the name specified is included in the lineage attribute in the taxonomy.

As an example, if the rank chosen is genus, and the lineage option is set to archaea, only the taxa whose rank is genus and that belong to the archaea subtree will be downloaded:

```
$ download_profiles -m EMAIL -r genus -l archaea mg_data/kegg.pickle \
-t mg_data/taxonomy.pickle
```

This allows to customise the level of specificity that we want in profiling and make the process of downloading faster. For metagenomic data, a good start is mixing different taxon ranks, using the order or genus for the genes and then specifying a lineage of interest.

Because each profile is indipendent from each other, it's useful to start the download with a certain rank and then run the profiling. During the profiling a new download can be started and so on.

Specific Genes and Taxa

It is possible to download only specific taxa and KO and can be done using the *-i* and *-ko* respectively. When *-ko* is used, loading Kegg Data with *-k* is not required and it is up to the user to ensure the correct genes or taxa.

An example to download only KO from 3 different taxa:

```
$ download_profiles -v -m EMAIL -ko K00016 -i 9611 9645 9682 \
-t mg_data/taxonomy.pickle
```

The same example using taxa filtering, instead (at the time of writing):

```
$ download_profiles -v -m EMAIL -ko K00016 -r genus -l carnivora \
-t mg_data/taxonomy.pickle
```

4.12.6 Changes

Changed in version 0.2.1: added *-ko* option, resolved issues caused by changes in library

4.12.7 Options

Download KO sequences from Uniprot

```
usage: download_profiles [-h] [-o OUTPUT_DIR] [-k KEGG_DATA] -m EMAIL
                         [-t TAXON_DATA] [-r TAXON_RANK] [-l LINEAGE]
                         [-i TAXON_ID [TAXON_ID ...]] [-ko KO_ID [KO_ID ...]]
                         [-R] [-a] [-v | --quiet] [--cite] [--manual]
                         [--version]
```

Named Arguments

-o, --output-dir	directory in which to store the downloaded files
	Default: "profile_files"
-k, --kegg-data	pickle file containing Kegg data
	Default: "data/kegg.pickle"
-m, --email	email address to use for Uniprot communications
-t, --taxon-data	pickle file containing taxonomy data
	Default: "data/taxonomy.pickle"
-r, --taxon-rank	taxon rank to download
-l, --lineage	lineage for filtering (e.g. archaea)
-i, --taxon-id	id(s) of taxa to download. If specified take precedence over lineage-rank
-ko, --ko-id	KO id(s)to download. If specified option -k is not needed
-R, --only-reviewed	Only download reviewed sequences
	Default: False
-a, --all-path	Download all KO from Kegg - exclude blacklist
	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

4.13 sampling-utils - Resampling Utilities

4.13.1 Overview

4.13.2 Options

4.14 download-data - Download Taxonomy from NCBI

A bash script called **download-taxonomy.sh** is installed with MGKit. The script downloads the required file (taxdump.tar.gz[60]) form the NCBI ftp taxonomy directory, using **wget**.

Note: If the file is found then it is not downladed. This is handy in case wget is not installed on the system by default (e.g. MacOS X)

The file is then decompressed using **tar** and a *files.txt* file created to be used by a simple script in the same directory. Both the directory and *files.txt* are deleted at the end, but not **taxdump.tar.gz**. A taxonomy file **taxonomy.pickle** is created in the same directory.

4.15 Download Taxa IDs from Uniprot

A script is included to download and prepare a tab separated list of all Taxa IDs associated with Uniprot IDs, so it can be used with *add-gff-info - Add informations to GFF annotations*. The script is called *download-uniprot-taxa.sh* and is installed with MGKit. By default both SwissProt and TrEMBL IDs are downloaded, but passing either *sp* or *SP* will download only SwissProt. The output file is called *uniprot-taxa*

4.16 Download Taxa IDs from NCBI

A script is included to download and prepare a tab separated list of all Taxa IDs associated with NCBI (GenBank) IDs, so it can be used with *add-gff-info - Add informations to GFF annotations*. The script is called *download-ncbi-taxa.sh* and is installed with MGKit. By default *nt* (nucleotide) IDs are downloaded, but passing either *prot* or *PROT* will download *nr* (protein) IDs. The output file is called *ncbi-nucl-taxa.gz* or *ncbi-prot-taxa.gz* depending of the downloaded data.

4.17 Download Required Data (Deprecated)

The scripts downloads the data that is used by the framework for some of its functions. It's mostly a shortcut to call the download_data function that is present in every module that is in the package mappings and in the kegg module.

The only option required, is the email contact for the person using the script; this is used to make sure that the API requirements in Uniprot are fullfilled and they can contact the person using the script is any problem arise.

Note: The default behavior is to download first the taxonomy data form Uniprot, Kegg and additional mapping data.

---

[60] ftp://ftp.ncbi.nlm.nih.gov/pub/taxonomy/taxdump.tar.gz

4.17.1 Taxonomy

It is downloaded from Uniprot and build a data structure that is used by several scripts and function in the package. The download can take some time.

Warning: if only the taxonomy is to be downloaded, both the *-x* and *-p* options must be passed to the script.

4.17.2 Kegg Onthologs

The Kegg data is the only "required" data at the moment, because it's used to download the sequence data (via the donwload_profiles script) for the profiling. It is the only data that can't be saved unless it's fully downloaded.

Kegg data is required by the mappers currently supported, and its download takes longer. The mappers handle timeouts and if exceptions are raised the data is saved and the download is resumed when the script is started again.

4.17.3 Other Mappings

The other mappings (from KO) are downloaded by default and this can be excluded by using the *-p* option. Mappings for Gene Onthology, eggNOG and CaZy are downloaded.

As the download of the mappings can take a lot of time, or break because of the number of requests to the web sites, checkpoints are saved often, so it can resumed at a later time.

Warning: the Gene Onthology module has specific requirements, so if they are not downloaded

4.17.4 Options

SNPs analysis, requires a vcf file and SNPDat results

```
usage: download_data [-h] [-o OUTPUT_DIR] [-k KEGG] [-c CAZY] [-g GO]
                     [-e EGGNOG] [-t TAXONOMY] [-p] [-x] -m EMAIL
                     [-v | --quiet] [--cite] [--manual] [--version]
```

Named Arguments

-o, --output-dir	Ouput directory
	Default: "mg_data"
-k, --kegg	Kegg data file name
	Default: "kegg.pickle"
-c, --cazy	CaZy data file name
	Default: "cazy.pickle"
-g, --go	Gene Onthology data file name
	Default: "go.pickle"
-e, --eggnog	eggNOG data file name
	Default: "eggnog.pickle"

-t, --taxonomy	Taxonomy data file name Default: "taxonomy.pickle"
-p, --no-mappings	Use to not download Mapping data Default: True
-x, --only-taxonomy	Use to only download Taxonomy data, no Kegg data Default: True
-m, --email	email address to use for Uniprot communications
-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

4.18 translate_seq - Translate Nucleotides to Aminoacids (Deprecated)

4.18.1 Overview

Deprecated in favour of a similar command in *fasta-utils - Fasta Utilities*

4.18.2 Options

CHAPTER 5

Example Notebooks

Example notebooks are included about using the library

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 6.1: 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()`[61]
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 6.2: 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*

[61] 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*[62]) – file handle or string with file name
- **samples** (*iterable*) – list of sample names, in the same order as *count_files*
- **cut_name** (*str*[63]) – 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

[62] https://docs.python.org/3/library/stdtypes.html#str
[63] 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*[64]) – 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*[65]) – function that accepts the sample name and returns tuple for a MultiIndex. Defaults to *None* so no transformation is performed
- **nozero** (*bool*[66]) – 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*[67]) – 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=<type 'int'>*)

Changed in version 0.1.15: added *conv_func* parameter

Loads an HTSeq-count result file

Parameters

- **file_handle** (*file or str*[68]) – 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*[69]) – 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

[64] https://docs.python.org/3/library/functions.html#iter
[65] https://docs.python.org/3/library/constants.html#None
[66] https://docs.python.org/3/library/functions.html#bool
[67] https://docs.python.org/3/library/stdtypes.html#str
[68] https://docs.python.org/3/library/stdtypes.html#str
[69] https://docs.python.org/3/library/stdtypes.html#dict

- **inc_anc** (*int*[70], *list*[71]) – ancestor taxa to include
- **rank** (*str*[72]) – rank to which map the counts
- **gene_map** (*dict*[73]) – dictionary with the gene mappings
- **ex_anc** (*int*[74], *list*[75]) – ancestor taxa to exclude
- **include_higher** (*bool*[76]) – if False, any rank different than the requested one is discarded
- **cached** (*bool*[77]) – if *True*, the function will use *mgkit.simple_cache.memoize* to cache some of the functions used
- **uid_used** (*None*[78], *dict*[79]) – 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*[80]) – 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

[70] https://docs.python.org/3/library/functions.html#int
[71] https://docs.python.org/3/library/stdtypes.html#list
[72] https://docs.python.org/3/library/stdtypes.html#str
[73] https://docs.python.org/3/library/stdtypes.html#dict
[74] https://docs.python.org/3/library/functions.html#int
[75] https://docs.python.org/3/library/stdtypes.html#list
[76] https://docs.python.org/3/library/functions.html#bool
[77] https://docs.python.org/3/library/functions.html#bool
[78] https://docs.python.org/3/library/constants.html#None
[79] https://docs.python.org/3/library/stdtypes.html#dict
[80] https://docs.python.org/3/library/functions.html#callable
[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

- **cached** (*bool*[87]) – if *True*, the function will use *mgkit.simple_cache.memoize* to cache some of the functions used
- **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*[90]) – 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*[91], *list*[92]) – ancestor taxa to include
- **rank** (*str*[93]) – rank to which map the counts
- **ex_anc** (*int*[94], *list*[95]) – ancestor taxa to exclude
- **include_higher** (*bool*[96]) – if False, any rank different than the requested one is discarded
- **cached** (*bool*[97]) – if *True*, the function will use *mgkit.simple_cache.memoize* to cache some of the functions used
- **uid_used** (*None*[98], *dict*[99]) – 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

[87] https://docs.python.org/3/library/functions.html#bool
[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#callable
[91] https://docs.python.org/3/library/functions.html#int
[92] https://docs.python.org/3/library/stdtypes.html#list
[93] https://docs.python.org/3/library/stdtypes.html#str
[94] https://docs.python.org/3/library/functions.html#int
[95] https://docs.python.org/3/library/stdtypes.html#list
[96] https://docs.python.org/3/library/functions.html#bool
[97] https://docs.python.org/3/library/functions.html#bool
[98] https://docs.python.org/3/library/constants.html#None
[99] https://docs.python.org/3/library/stdtypes.html#dict

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.

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*[100], *str*[101]) – 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*[102], *dict*[103]) – 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*[104]) – if False, counts for genes with no mappings in *gene_map* are discarded, if True, they a counted as *nomap_id*
- **nomap_id** (*str*[105]) – 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.UniprotTaxonomy.get_ranked_taxon()*

Parameters

- **taxonomy** – taxonomy instance
- **rank** (*str*[106]) – taxonomic rank requested

[100] https://docs.python.org/3/library/constants.html#None
[101] https://docs.python.org/3/library/stdtypes.html#str
[102] https://docs.python.org/3/library/constants.html#None
[103] https://docs.python.org/3/library/stdtypes.html#dict
[104] https://docs.python.org/3/library/functions.html#bool
[105] https://docs.python.org/3/library/stdtypes.html#str
[106] https://docs.python.org/3/library/stdtypes.html#str

- **taxon_id** (*int*[107]) – taxon_id to map
- **include_higher** (*bool*[108]) – if False, any rank different than the requested one is discarded

Returns if the mapping is successful, the ranked taxon_id is returned, otherwise *None* is returned

Return type (int[109], None[110])

mgkit.counts.glm module

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()*

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*

$$RPKM = \frac{10^9 \cdot C}{N \cdot L}$$

[107] https://docs.python.org/3/library/functions.html#int
[108] https://docs.python.org/3/library/functions.html#bool
[109] https://docs.python.org/3/library/functions.html#int
[110] https://docs.python.org/3/library/constants.html#None

Module contents

mgkit.db package

Submodules

mgkit.db.dbm module

mgkit.db.mongo module

Module contents

mgkit.filter package

Submodules

mgkit.filter.common module

Common consts/data for package filter

exception mgkit.filter.common.**FilterFails**
Bases: exceptions.Exception

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*[111], *float*[112]) – number of bases overlap that trigger the filtering
- **choose_func** (*None*[113], 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[114], Annotation)

[111] https://docs.python.org/3/library/functions.html#int
[112] https://docs.python.org/3/library/functions.html#float
[113] https://docs.python.org/3/library/constants.html#None
[114] https://docs.python.org/3/library/constants.html#None

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.

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*[115]) – function used to select the *losing* annotation; if *None*, it will be *choose_annotation()* with default values
- **sort_func** (`func`, *None*[116]) – by default the sorting key is the bitscore of the annotations
- **reverse** (*bool*[117]) – passed to *sorted*, by default is reversed

Returns a set with the annotations that pass the filtering

Return type set[118]

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*[119]) – key in the `mgkit.io.gff.Annotation.attr` dictionary
- **value** (*int*[120]) – the value to which we need to compare
- **greater** (*bool*[121]) – 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[122]

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

[115] https://docs.python.org/3/library/constants.html#None
[116] https://docs.python.org/3/library/constants.html#None
[117] https://docs.python.org/3/library/functions.html#bool
[118] https://docs.python.org/3/library/stdtypes.html#set
[119] https://docs.python.org/3/library/stdtypes.html#str
[120] https://docs.python.org/3/library/functions.html#int
[121] https://docs.python.org/3/library/functions.html#bool
[122] https://docs.python.org/3/library/functions.html#bool

Parameters

- **annotation** – mgkit.io.gff.Annotation instance
- **attr** (*str*[123]) – key in the mgkit.io.gff.Annotation.attr dictionary
- **value** (*int*[124]) – the value to which we need to compare
- **greater** (*bool*[125]) – if True the value must be greater than and if False lower than

Returns True if the test passes

Return type bool[126]

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*[127]) – key in the mgkit.io.gff.Annotation.attr dictionary
- **value** (*int*[128]) – the value to which we need to compare
- **equal** (*bool*[129]) – if True the value must be equal and if False equal value must be contained

Returns True if the test passes

Return type bool[130]

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*[131]) – 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[132]

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*[133]) – attribute of the annotation
- **value** (*int*[134]) – the value to which the attribute should be compared to

[123] https://docs.python.org/3/library/stdtypes.html#str
[124] https://docs.python.org/3/library/functions.html#int
[125] https://docs.python.org/3/library/functions.html#bool
[126] https://docs.python.org/3/library/functions.html#bool
[127] https://docs.python.org/3/library/stdtypes.html#str
[128] https://docs.python.org/3/library/functions.html#int
[129] https://docs.python.org/3/library/functions.html#bool
[130] https://docs.python.org/3/library/functions.html#bool
[131] https://docs.python.org/3/library/stdtypes.html#str
[132] https://docs.python.org/3/library/functions.html#bool
[133] https://docs.python.org/3/library/stdtypes.html#str
[134] https://docs.python.org/3/library/functions.html#int

- **greater** (*bool*[135]) – 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[136]

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*[137]) – the length to which the attribute should be compared to
- **greater** (*bool*[138]) – 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[139]

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.

[135] https://docs.python.org/3/library/functions.html#bool
[136] https://docs.python.org/3/library/functions.html#bool
[137] https://docs.python.org/3/library/functions.html#int
[138] https://docs.python.org/3/library/functions.html#bool
[139] https://docs.python.org/3/library/functions.html#bool

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:

```
func = lambda t_id, a_id: t_id == a_id
```

Parameters

- **taxon_id** (*int*[140]) – the taxon id to filter
- **filter_list** (*iterable*) – an iterable with taxon ids
- **exclude** (*bool*[141]) – if the filter is reversed (i.e. included if NOT found)
- **func** (*func or None*[142]) – 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[143]

Example

If using func and assuming that *taxonomy* is an instance of *UniprotTaxonomy* 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
```

[140] https://docs.python.org/3/library/functions.html#int
[141] https://docs.python.org/3/library/functions.html#bool
[142] https://docs.python.org/3/library/constants.html#None
[143] https://docs.python.org/3/library/functions.html#bool

Module contents

Package used to store filter functions (unless specific to a package)

mgkit.io package

Submodules

mgkit.io.blast module

mgkit.io.fasta module

Simple fasta parser and a few utility functions

mgkit.io.fasta.**load_fasta**(*f_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 **f_handle** (*str*[144], *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_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*[145], *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.

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*[146]) – fasta file with the input sequences
- **name_mask** (*str*[147]) – file name template for the splitted files, more informations are found in `mgkit.io.split_write()`
- **num_files** (*int*[148]) – 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

[144] https://docs.python.org/3/library/stdtypes.html#str
[145] https://docs.python.org/3/library/stdtypes.html#str
[146] https://docs.python.org/3/library/stdtypes.html#str
[147] https://docs.python.org/3/library/stdtypes.html#str
[148] https://docs.python.org/3/library/functions.html#int

- **file_handle** – file handle or string.
- **name** (*str*[149]) – header to write for the sequence
- **seq** (*str*[150]) – sequence to write
- **wrap** (*int*[151]) – 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.**check_fastq_type**(*qualities*)

Trys to guess the type of quality string used in a Fastq file

Parameters **qualities** (*str*[152]) – 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

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*[153]) – 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*[154]) – seq_id of the sequence (stripped of '@')
- **index_as_seq** (*bool*[155]) – 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

[149] https://docs.python.org/3/library/stdtypes.html#str
[150] https://docs.python.org/3/library/stdtypes.html#str
[151] https://docs.python.org/3/library/functions.html#int
[152] https://docs.python.org/3/library/stdtypes.html#str
[153] https://docs.python.org/3/library/stdtypes.html#str
[154] https://docs.python.org/3/library/stdtypes.html#str
[155] https://docs.python.org/3/library/functions.html#bool

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*[156], *file*) – fastq file to open, can be a file name or a file handle

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[157] – 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[158] – if the qualities are in a format different than sanger
- (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*[159]) – header to write for the sequence
- **seq** (*str*[160]) – sequence to write
- **qual** (*str*[161]) – quality string

mgkit.io.gff module

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.

[156] https://docs.python.org/3/library/stdtypes.html#str
[157] https://docs.python.org/3/library/exceptions.html#ValueError
[158] https://docs.python.org/3/library/exceptions.html#TypeError
[159] https://docs.python.org/3/library/stdtypes.html#str
[160] https://docs.python.org/3/library/stdtypes.html#str
[161] https://docs.python.org/3/library/stdtypes.html#str

`io.gff.from_glimmer3()` function.

Parameters **file_handle** (*str*[162], *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

mgkit.io.uniprot module

New in version 0.1.13.

Uniprot file formats

mgkit.io.uniprot.**parse_uniprot_mappings**(*file_handle*, *gene_ids=None*, *mappings=None*, *num_lines=10000000*)

Parses a Uniprot mapping file[163], returning a generator with the mappings.

Parameters

- **file_handle** (*str*[164], *file*) – file name or open file handle
- **gene_ids** (*None*[165], *set*[166]) – if not None, the returned mappings are for the gene IDs specified
- **mappings** (*None*[167], *set*[168]) – mappings to be returned
- **num_lines** (*None*[169], *int*[170]) – 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*)

Parses a Uniprot mapping file[171], returning a generator of dictionaries with the mappings requested.

Parameters

- **file_handle** (*str*[172], *file*) – file name or open file handle
- **gene_ids** (*None*[173], *set*[174]) – if not None, the returned mappings are for the gene IDs specified
- **mappings** (*None*[175], *set*[176]) – mappings to be returned

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

[162] https://docs.python.org/3/library/stdtypes.html#str
[163] ftp://ftp.uniprot.org/pub/databases/uniprot/current_release/knowledgebase/idmapping/idmapping.dat.gz
[164] https://docs.python.org/3/library/stdtypes.html#str
[165] https://docs.python.org/3/library/constants.html#None
[166] https://docs.python.org/3/library/stdtypes.html#set
[167] https://docs.python.org/3/library/constants.html#None
[168] https://docs.python.org/3/library/stdtypes.html#set
[169] https://docs.python.org/3/library/constants.html#None
[170] https://docs.python.org/3/library/functions.html#int
[171] ftp://ftp.uniprot.org/pub/databases/uniprot/current_release/knowledgebase/idmapping/idmapping.dat.gz
[172] https://docs.python.org/3/library/stdtypes.html#str
[173] https://docs.python.org/3/library/constants.html#None
[174] https://docs.python.org/3/library/stdtypes.html#set
[175] https://docs.python.org/3/library/constants.html#None
[176] https://docs.python.org/3/library/stdtypes.html#set

mgkit.io.utils module

Various utilities to help read and process files

exception mgkit.io.utils.**UnsupportedFormat**
Bases: exceptions.IOError

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*[177]) – 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.

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*[178]) – 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.

Opens a file using the extension as a guide to which module to use.

Parameters

- **file_name** (*str*[179]) – file name
- **mode** (*str*[180]) – 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.

[177] https://docs.python.org/3/library/stdtypes.html#str
[178] https://docs.python.org/3/library/functions.html#int
[179] https://docs.python.org/3/library/stdtypes.html#str
[180] https://docs.python.org/3/library/stdtypes.html#str

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*[181]) – 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*[182]) – the number of files to split the records

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

class mgkit.mappings.cazy.**Kegg2CazyMapper**(*fname=None*)
Bases: *mgkit.kegg.KeggMapperBase*

Class holding mappings KOs->CaZy

columns_string = 'database(cazy)'

map_kos_cazy(*kos*, *contact*, *skip=False*)
Map a list of KOs to CaZy ids

query_string = 'database:(type:ko {0}) AND database:(type:cazy)'

mgkit.mappings.cazy.**download_data**(*contact*, *kegg_data='kegg.pickle'*, *cazy_data='cazy.pickle'*)
Function to download CaZy data

mgkit.mappings.eggnog module

Module containing classes and functions to deal with eggNOG data

Todo:

- unify download of data from web

class mgkit.mappings.eggnog.**Kegg2NogMapper**(*fname=None*, *gen_ko_to_cat=True*)
Bases: *mgkit.kegg.KeggMapperBase*

Usage

to add a ko_id to the mapper use: * map_ko_to_eggnog(ko_id) which calls:

[181] https://docs.python.org/3/library/stdtypes.html#str
[182] https://docs.python.org/3/library/functions.html#int

- get_uniprot_from_ko
- get_eggnog_from_uniprot

return None if there was no problem or a dictionary with the exception thrown

to make mappings ko_id -> eggnog-categories: * get_cat_mapping_from_file (file are in the current directory) * gen_ko_to_cat

---

Todo:

- add method to load kos from file
- optimise querying (like Kegg2CazyMapper)

---

columns_string = 'database(eggnog)'

gen_ko_to_cat()

get_cat_ko_dict()

get_cat_mapping(*f_handle*)

get_cat_mapping_from_file(*files=('COG.funccat.txt', 'NOG.funccat.txt', 'KOG.funccat.txt')*)

Duplicate functionality of load_func_cat

Categories are single letter and their descriptioons are in EGGNOG_CAT, while the broader categories EGGNOG_CAT_MAP

get_ko_cat(*ko_id*)

get_ko_cat_dict()

get_ko_map()

load_data(*fname*)

map_ko_to_eggnog(*ko_id*, *contact*)

map_kos_eggnog(*kos*, *contact*)

query_string = 'database:(type:ko {0}) AND database:(type:eggnog)'

save_data(*fname*)

class mgkit.mappings.eggnog.**NOGInfo**

Bases: object[183]

New in version 0.1.14.

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

[183] 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 ID

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.**download_data**(*contact*, *kegg_data='kegg.pickle'*, *eggnog_data='eggnog.pickle'*, *base_url='http://eggnog.embl.de/version_3.0/data/downloads/'*)

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.eggnog.**print_ko_to_cat**(*data_file='eggnog.pickle'*, *descr=False*)

mgkit.mappings.enzyme module

New in version 0.1.14.

EC mappings

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*[184]) – dictionary generated by *mgkit.net.uniprot.get_gene_info()*
- **level** (*int*[185]) – 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

[184] https://docs.python.org/3/library/stdtypes.html#dict
[185] 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*[186]) – EC identifier
- **ec_names** (*dict*[187]) – a dictionary of names that can be produced using *parse_expasy_file()*
- **sep** (*str*[188]) – string used to join the names

Returns the enzyme classification name

Return type str[189]

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*[190]) – a string representing an EC number (e.g. 1.2.4.10)
- **level** (*int*[191]) – 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[192]

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.

[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
[190] https://docs.python.org/3/library/stdtypes.html#str
[191] https://docs.python.org/3/library/functions.html#int
[192] https://docs.python.org/3/library/stdtypes.html#str

Parameters

- **ec_map** (*dict*[193]) – dictionary genes to EC
- **level** (*int*[194]) – 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[195].

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

class mgkit.mappings.go.**Kegg2GOMapper**(*fname=None*)
Bases: *mgkit.kegg.KeggMapperBase*

Class holding mappings KOs->GO

columns_string = 'go-id'

load_go_names(*fname*)

load_goslim_mapping(*f_handle*)

map_kos_go(*kos*, *contact*, *skip=False*)
Map a list of KOs to GO ids

query_string = 'database:(type:ko {0})'

write_association_file(*f_handle*)

mgkit.mappings.go.**download_data**(*contact*, *kegg_data='kegg.pickle'*, *go_data='go_data.pickle'*)
Function to download 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[196]

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

[193] https://docs.python.org/3/library/stdtypes.html#dict
[194] https://docs.python.org/3/library/functions.html#int
[195] http://expasy.org
[196] http://en.wikipedia.org/wiki/Coefficient_of_variation

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*[197]) – dictionary of category->genes
- **root_taxon** (*str*[198]) – root taxon to group genes
- **name_dict** (*dict*[199]) – 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*[200]) – 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.UniprotTaxonomy*.

[197] https://docs.python.org/3/library/stdtypes.html#dict
[198] https://docs.python.org/3/library/stdtypes.html#str
[199] https://docs.python.org/3/library/stdtypes.html#dict
[200] https://docs.python.org/3/library/functions.html#int

```
>>> 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
- **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*[201]) – a dictionary in the form: gene_id->[cat1, cat2, .., catN]
- **normalise** (*bool*[202]) – 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*[203]) – dictionary with mappings for the attribute requested
- **attr** (*str*[204]) – 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

exception mgkit.net.embl.**EntryNotFound**

Bases: `exceptions.Exception`

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.

[201] https://docs.python.org/3/library/stdtypes.html#dict
[202] https://docs.python.org/3/library/functions.html#bool
[203] https://docs.python.org/3/library/stdtypes.html#dict
[204] https://docs.python.org/3/library/stdtypes.html#str

exception mgkit.net.embl.**NoEntryFound**
Bases: exceptions.Exception

Raised if no sequences where found by *get_sequences_by_ids()*, the check is based on the NONE_FOUND variable.

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[205] with more details about the databases domains at this page[206]

Parameters

- **query** (*str*[207]) – query for the search enging
- **domain** (*str*[208]) – database domain to search
- **result** (*str*[209]) – domain result requested
- **display** (*str*[210]) – display option (format to retrieve the entries)
- **offset** (*int*[211]) – the offset of the search results, defaults to the first
- **length** (*int*[212]) – number of results to retrieve at the specified offset and the limit is automatically set a 100,000 records for query
- **contact** (*str*[213]) – email of the user
- **download** (*str*[214]) – type of response. Gzip responses are automatically decompressed
- **url** (*str*[215]) – base URL for the resource
- **fields** (*None*[216], *iterable*) – must be an iterable of fields to be returned if display is set to *report*

Returns the raw request

Return type str[217]

Examples

Querying EMBL for all sequences of type rRNA of the *Clostridium* genus. Only from the EMBL release database in fasta format:

[205] http://www.ebi.ac.uk/ena/about/browser#data_warehouse
[206] http://www.ebi.ac.uk/ena/data/warehouse/usage
[207] https://docs.python.org/3/library/stdtypes.html#str
[208] https://docs.python.org/3/library/stdtypes.html#str
[209] https://docs.python.org/3/library/stdtypes.html#str
[210] https://docs.python.org/3/library/stdtypes.html#str
[211] https://docs.python.org/3/library/functions.html#int
[212] https://docs.python.org/3/library/functions.html#int
[213] https://docs.python.org/3/library/stdtypes.html#str
[214] https://docs.python.org/3/library/stdtypes.html#str
[215] https://docs.python.org/3/library/stdtypes.html#str
[216] https://docs.python.org/3/library/constants.html#None
[217] https://docs.python.org/3/library/stdtypes.html#str

```
>>> 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
```

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[218]

Parameters

- **embl_ids** (*str*[219], *iterable*) – list or single sequence id to retrieve
- **db** (*str*[220]) – database from which retrieve the sequence data
- **contact** (*str*[221]) – email contact to use as per EMBL guidlines
- **out_format** (*str*[222]) – output format, depends on database
- **num_req** (*int*[223]) – 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[224]

mgkit.net.embl.**get_sequences_by_ids**(*embl_ids*, *contact=None*, *out_format='fasta'*, *num_req=10*, *embl_db='embl_cds'*, *compress=True*, *strict=False*)

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.

[218] http://www.ebi.ac.uk/Tools/dbfetch/syntax.jsp
[219] https://docs.python.org/3/library/stdtypes.html#str
[220] https://docs.python.org/3/library/stdtypes.html#str
[221] https://docs.python.org/3/library/stdtypes.html#str
[222] https://docs.python.org/3/library/stdtypes.html#str
[223] https://docs.python.org/3/library/functions.html#int
[224] https://docs.python.org/3/library/stdtypes.html#list

- 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

- **embl_ids** (*iterable, str*[225]) – list of ids to download
- **contact** (*str*[226]) – email address to be passed in the query
- **format** (*str*[227]) – format of the entry
- **num_req** (*int*[228]) – number of entries to download with each request
- **embl_db** (*str*[229]) – db to which the ids refer to
- **compress** (*bool*[230]) – if True, the function tries to obtain a compressed response and decompress it on the fly
- **strict** (*bool*[231]) – if True, a check on the number of entries retrieved is performed

Returns the entries requested

Return type str[232]

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

[225] https://docs.python.org/3/library/stdtypes.html#str
[226] https://docs.python.org/3/library/stdtypes.html#str
[227] https://docs.python.org/3/library/stdtypes.html#str
[228] https://docs.python.org/3/library/functions.html#int
[229] https://docs.python.org/3/library/stdtypes.html#str
[230] https://docs.python.org/3/library/functions.html#bool
[231] https://docs.python.org/3/library/functions.html#bool
[232] https://docs.python.org/3/library/stdtypes.html#str

Parameters **key** (*str*[233]) – 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[234]

mgkit.net.uniprot module

Contains function and constants for Uniprot access

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*[235]) – gene id(s) to get informations for
- **columns** (*list*[236]) – list of columns
- **max_req** (*int*[237]) – number of maximum *gene_ids* per request
- **contact** (*str*[238]) – 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[239]

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.

[233] https://docs.python.org/3/library/stdtypes.html#str
[234] https://docs.python.org/3/library/stdtypes.html#dict
[235] https://docs.python.org/3/library/stdtypes.html#str
[236] https://docs.python.org/3/library/stdtypes.html#list
[237] https://docs.python.org/3/library/functions.html#int
[238] https://docs.python.org/3/library/stdtypes.html#str
[239] https://docs.python.org/3/library/stdtypes.html#dict

Parameters

- **ko_ids** (*iterable*) – an iterable of KO IDs
- **contact** (*str*[240]) – email address to be passed in the query (requested Uniprot API)

Returns The format of the resulting dictionary is ko_id -> {eggnog_id1, ..}

Return type dict[241]

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*[242]) – string that identify the DB for elements in entry_ids
- **db_to** (*str*[243]) – string that identify the DB to which map entry_ids
- **out_format** (*str*[244]) – format of the mapping; 'list' and 'tab' are processed
- **contact** (*str*[245]) – 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*[246]) – KO id of the sequences to download
- **taxonomy** (*int*[247]) – id of the taxon
- **contact** (*str*[248]) – email address to be passed in the query (requested by Uniprot API)
- **reviewed** (*bool*[249]) – 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*[250]) – id used in the query

[240] https://docs.python.org/3/library/stdtypes.html#str
[241] https://docs.python.org/3/library/stdtypes.html#dict
[242] https://docs.python.org/3/library/stdtypes.html#str
[243] https://docs.python.org/3/library/stdtypes.html#str
[244] https://docs.python.org/3/library/stdtypes.html#str
[245] https://docs.python.org/3/library/stdtypes.html#str
[246] https://docs.python.org/3/library/stdtypes.html#str
[247] https://docs.python.org/3/library/functions.html#int
[248] https://docs.python.org/3/library/stdtypes.html#str
[249] https://docs.python.org/3/library/functions.html#bool
[250] https://docs.python.org/3/library/stdtypes.html#str

- **query** (*str*[251]) – query passed to the Uniprot API, ko_id is replaced using `str.format()`
- **column** (*str*[252]) – column used in the results table used to map the ids
- **contact** (*str*[253]) – 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*[254]) – string response from Uniprot
- **simple** (*bool*[255]) – 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[256]

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[257]

Parameters

- **query** (*str*[258]) – query to submit, as put in the input box
- **columns** (*None*[259], *iterable*) – list of columns to return
- **format** (*str*[260]) – response format
- **limit** (*int*[261], *None*[262]) – number of entries to return or *None* to request all entries
- **contact** (*str*[263]) – email address to be passed in the query (requested Uniprot API)
- **baseurl** (*str*[264]) – base url for the REST API, can be either `UNIPROT_GET` or `UNIPROT_TAXONOMY`

Returns raw response from the query

Return type str[265]

[251] https://docs.python.org/3/library/stdtypes.html#str
[252] https://docs.python.org/3/library/stdtypes.html#str
[253] https://docs.python.org/3/library/stdtypes.html#str
[254] https://docs.python.org/3/library/stdtypes.html#str
[255] https://docs.python.org/3/library/functions.html#bool
[256] https://docs.python.org/3/library/stdtypes.html#dict
[257] http://www.uniprot.org/faq/28
[258] https://docs.python.org/3/library/stdtypes.html#str
[259] https://docs.python.org/3/library/constants.html#None
[260] https://docs.python.org/3/library/stdtypes.html#str
[261] https://docs.python.org/3/library/functions.html#int
[262] https://docs.python.org/3/library/constants.html#None
[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/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*, *compress=True*, *agent=None*)
Utility function that compresses the request and add the user agent header if supplied.

Parameters

- **url** (*str*[266]) – url to request
- **data** (*str*[267]) – data to add to the request
- **compress** (*bool*[268]) – if the response should be compressed
- **agent** (*str*[269]) – 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*, *compress=True*, *agent=None*)
Utility function that compresses the request and add the user agent header if supplied.

Wrapper of *url_open()* which reads the full response

Parameters

- **url** (*str*[270]) – url to request
- **data** (*str*[271]) – data to add to the request
- **compress** (*bool*[272]) – if the response should be compressed
- **agent** (*str*[273]) – if supplied, the 'User-Agent' header we'll be added to the request

Returns the response data

Module contents

Package with functions/classes used in accessing network resources

[266] https://docs.python.org/3/library/stdtypes.html#str
[267] https://docs.python.org/3/library/stdtypes.html#str
[268] https://docs.python.org/3/library/functions.html#bool
[269] https://docs.python.org/3/library/stdtypes.html#str
[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#bool
[273] https://docs.python.org/3/library/stdtypes.html#str

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*[274]) – 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*[275], *float*[276], *tuple*[277]) – color used to draw the triangle
- **linewidth** (*float*[278]) – line width

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

[274] https://docs.python.org/3/library/functions.html#float
[275] https://docs.python.org/3/library/stdtypes.html#str
[276] https://docs.python.org/3/library/functions.html#float
[277] https://docs.python.org/3/library/stdtypes.html#tuple
[278] https://docs.python.org/3/library/functions.html#float

- **csize** (*int*[279]) – the base size of the circles
- **alpha** (*float*[280]) – transparency of the circles, between 0 and 1 included
- **sizescale** (*None*[281], *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*[282], *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*[283]) – width of the circle line
- **edgecolor** (*str*[284]) – 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*[285], *iterable*) – either *None* for solid lines or matplotlib line markers. These are in sync between the internal lines and the axes.
- **linewidth** (*float*[286]) – line width for the axes, the internal lines are equal to *0.75 * linewidth*
- **fontsize** (*float*[287]) – 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*[288]) – contains the three coordinates to project

Returns the projected point in a planar-simplex

Return type tuple[289]

mgkit.plots.boxplot module

New in version 0.1.14.

[279] https://docs.python.org/3/library/functions.html#int
[280] https://docs.python.org/3/library/functions.html#float
[281] https://docs.python.org/3/library/constants.html#None
[282] https://docs.python.org/3/library/constants.html#None
[283] https://docs.python.org/3/library/functions.html#float
[284] https://docs.python.org/3/library/stdtypes.html#str
[285] https://docs.python.org/3/library/constants.html#None
[286] https://docs.python.org/3/library/functions.html#float
[287] https://docs.python.org/3/library/functions.html#float
[288] https://docs.python.org/3/library/stdtypes.html#tuple
[289] https://docs.python.org/3/library/stdtypes.html#tuple

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[290]

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*[291]) – colors used for the values
- **alpha** (*float*[292]) – alpha value for the colour
- **s** (*int*[293]) – size of the marker drawn
- **marker** (*str*[294]) – one of the accepted matplotlib markers
- **linewidth** (*float*[295]) – width of the line used to draw the marker
- **box_vert** (*bool*[296]) – specify if the original boxplot is vertical or not

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*[297]) – the 2 values are the coordinates for the top line, and the the lowest bound for the whisker
- **box_vert** (*bool*[298]) – if the boxplot is vertical

[290] http://matplotlib.org/api/markers_api.html
[291] https://docs.python.org/3/library/stdtypes.html#dict
[292] https://docs.python.org/3/library/functions.html#float
[293] https://docs.python.org/3/library/functions.html#int
[294] https://docs.python.org/3/library/stdtypes.html#str
[295] https://docs.python.org/3/library/functions.html#float
[296] https://docs.python.org/3/library/functions.html#bool
[297] https://docs.python.org/3/library/stdtypes.html#tuple
[298] https://docs.python.org/3/library/functions.html#bool

- **fontsize** (*float*[299]) – 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*[300]) – a map that converts the items in plot_order to a label used on the plot X axes
- **fonts** (*dict*[301]) – dictionary with properties for x axis labels, DEFAULT_BOXPLOT_FONTCONF is used by default
- **fill_box** (*bool*[302]) – if True each box is filled with the same colour of its outline
- **colours** (*dict*[303]) – 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*[304]) – 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

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

[299] https://docs.python.org/3/library/functions.html#float
[300] https://docs.python.org/3/library/stdtypes.html#dict
[301] https://docs.python.org/3/library/stdtypes.html#dict
[302] https://docs.python.org/3/library/functions.html#bool
[303] https://docs.python.org/3/library/stdtypes.html#dict
[304] https://docs.python.org/3/library/stdtypes.html#dict

- **ax** – an axis instance
- **label_map** (*dict*[305]) – a map that converts the items in plot_order to a label used on the plot X ax
- **fonts** (*dict*[306]) – dictionary with properties for x axis labels, `DEFAULT_BOXPLOT_FONTCONF` is used by default
- **fill_box** (*bool*[307]) – if True each box is filled with the same colour of its outline
- **colours** (*dict*[308]) – 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*[309]) – dictionary of colours for each boxplot, a set of colours can be obtained using func:*map_taxon_to_colours*
- **box_vert** (*bool*[310]) – if False the boxplots are drawn horizontally
- **widths** (*float*[311]) – 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

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

[305] https://docs.python.org/3/library/stdtypes.html#dict
[306] https://docs.python.org/3/library/stdtypes.html#dict
[307] https://docs.python.org/3/library/functions.html#bool
[308] https://docs.python.org/3/library/stdtypes.html#dict
[309] https://docs.python.org/3/library/stdtypes.html#dict
[310] https://docs.python.org/3/library/functions.html#bool
[311] https://docs.python.org/3/library/functions.html#float

- **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*[312], *matplotlib.colors.ListedColormap*) – color map to use
- **xticks** (*None*[313], *dict*[314]) – dictionary with additional options to pass to *set_xticklabels*
- **yticks** (*None*[315], *dict*[316]) – dictionary with additional options to pass to *set_yticklabels*
- **fontsize** (*int*[317]) – font size to use for the labels
- **meshopts** (*None*[318], *dict*[319]) – additional options to pass to `matplotlib.pyplot.pcolormesh()`
- **annot** (*bool*[320]) – if True the values of the matrix will be added
- **annot_args** (*None*[321], *dict*[322]) – 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.

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*[323]) – to specify the axis, either *x* or *y*
- **spine** (*str*[324]) – position of the spine. if *which* is **x** accepted values are *top* and *bottom*, if which is **y** *left* and *right* are accepted

[312] https://docs.python.org/3/library/constants.html#None
[313] https://docs.python.org/3/library/constants.html#None
[314] https://docs.python.org/3/library/stdtypes.html#dict
[315] https://docs.python.org/3/library/constants.html#None
[316] https://docs.python.org/3/library/stdtypes.html#dict
[317] https://docs.python.org/3/library/functions.html#int
[318] https://docs.python.org/3/library/constants.html#None
[319] https://docs.python.org/3/library/stdtypes.html#dict
[320] https://docs.python.org/3/library/functions.html#bool
[321] https://docs.python.org/3/library/constants.html#None
[322] https://docs.python.org/3/library/stdtypes.html#dict
[323] https://docs.python.org/3/library/stdtypes.html#str
[324] https://docs.python.org/3/library/stdtypes.html#str

- **spine_opts** (*dict*[325]) – additional options to pass to the spine class
- **start** (*int*[326]) – 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*[327]) – clustering method used, internally `scipy.cluster.hierarchy.linkage()` is used.
- **orientation** (*str*[328]) – direction for the plot. *top*, *bottom*, *left* and *right* are accepted; *top* will draw the leaves at the bottom.
- **use_dist** (*bool*[329]) – 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*[330]) – passed to *mgkit.plots.utils.get_grid_figure()*
- **cmap** (*None*[331], *matplotlib.colors.ListedColormap*) – color map to use
- **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.**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

[325] https://docs.python.org/3/library/stdtypes.html#dict
[326] https://docs.python.org/3/library/functions.html#int
[327] https://docs.python.org/3/library/stdtypes.html#str
[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#tuple
[331] https://docs.python.org/3/library/constants.html#None

- **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.UniprotTaxonomy.get_taxon_root()` to determine the root of a taxon.

Parameters

- **taxa** (*iterable*) – iterable of taxon ids
- **taxonomy** (UniprotTaxonomy) – 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**(**args*, ***kwargs*)

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.UniprotTaxonomy.get_taxon_root()` to determine the root of a taxon.

Parameters

- **taxa** (*iterable*) – iterable of taxon ids
- **taxonomy** (UniprotTaxonomy) – 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*[332]) – colour of the bars for each taxon
- **log_scale** (*bool*[333]) – if True the y axis is log scaled
- **fig_aspect** (*tuple*[334]) – tuple with figure size

[332] https://docs.python.org/3/library/stdtypes.html#dict
[333] https://docs.python.org/3/library/functions.html#bool
[334] https://docs.python.org/3/library/stdtypes.html#tuple

- **xlabels_size** (*int*[335]) – size of the taxon labels
- **index** – optional `pandas.Index` used to reindex the series
- **file_name** (*str*[336]) – 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*[337]) – `numpy.array` with shape n, 2
- **labels** (*array*[338]) – labels to categorise samples
- **colours** (*dict*[339]) – dictionary whose keys are the labels and the values are valid `matplotlib` colours
- **pointsize** (*int*[340]) – point size
- **title** (*str*[341]) – plot title
- **xlabel** (*str*[342]) – label for x axis
- **ylabel** (*str*[343]) – 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*[344]) – `numpy.array` with shape n, 3
- **labels** (*array*[345]) – labels to categorise samples
- **colours** (*dict*[346]) – dictionary whose keys are the labels and the values are valid `matplotlib` colours
- **pointsize** (*int*[347]) – point size
- **title** (*str*[348]) – plot title
- **xlabel** (*str*[349]) – label for x axis
- **ylabel** (*str*[350]) – label for y axis
- **zlabel** (*str*[351]) – label for z axis

[335] https://docs.python.org/3/library/functions.html#int
[336] https://docs.python.org/3/library/stdtypes.html#str
[337] https://docs.python.org/3/library/array.html#module-array
[338] https://docs.python.org/3/library/array.html#module-array
[339] https://docs.python.org/3/library/stdtypes.html#dict
[340] https://docs.python.org/3/library/functions.html#int
[341] https://docs.python.org/3/library/stdtypes.html#str
[342] https://docs.python.org/3/library/stdtypes.html#str
[343] https://docs.python.org/3/library/stdtypes.html#str
[344] https://docs.python.org/3/library/array.html#module-array
[345] https://docs.python.org/3/library/array.html#module-array
[346] https://docs.python.org/3/library/stdtypes.html#dict
[347] https://docs.python.org/3/library/functions.html#int
[348] https://docs.python.org/3/library/stdtypes.html#str
[349] https://docs.python.org/3/library/stdtypes.html#str
[350] https://docs.python.org/3/library/stdtypes.html#str
[351] https://docs.python.org/3/library/stdtypes.html#str

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*[352]) – dictionary that contains the data
- **xlabel** (*str*[353]) – label for x axis
- **ylabel** (*str*[354]) – label for y axis
- **title** (*str*[355]) – graph title
- **colours** – colours used in for the different datasets; defaults to TAXON_COLOURS
- **file_name** (*str*[356]) – 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*[357]) – 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

Parameters

- **rows** (*int*[358]) – number of rows
- **columns** (*int*[359]) – number of columns
- **dpi** (*int*[360]) – dpi used for the figure
- **figsize** (*tuple*[361]) – size of the figure in inches

Returns the figure and axes objects

Return type tuple[362]

[352] https://docs.python.org/3/library/stdtypes.html#dict
[353] https://docs.python.org/3/library/stdtypes.html#str
[354] https://docs.python.org/3/library/stdtypes.html#str
[355] https://docs.python.org/3/library/stdtypes.html#str
[356] https://docs.python.org/3/library/stdtypes.html#str
[357] https://docs.python.org/3/library/functions.html#float
[358] https://docs.python.org/3/library/functions.html#int
[359] https://docs.python.org/3/library/functions.html#int
[360] https://docs.python.org/3/library/functions.html#int
[361] https://docs.python.org/3/library/stdtypes.html#tuple
[362] https://docs.python.org/3/library/stdtypes.html#tuple

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*[363]) – dpi used for the figure
- **figsize** (*tuple*[364]) – size of the figure in inches
- **aspect** (*str*[365], *float*[366]) – aspect ratio to be passed to figure.add_subplot

Returns the figure and axes objects

Return type tuple[367]

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[368]

Module contents

New in version 0.1.14.

mgkit.snps package

Submodules

mgkit.snps.classes module

Manage SNP data.

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

str – unique id for the isoform (to be referenced in a GFF file)

[363] https://docs.python.org/3/library/functions.html#int
[364] https://docs.python.org/3/library/stdtypes.html#tuple
[365] https://docs.python.org/3/library/stdtypes.html#str
[366] https://docs.python.org/3/library/functions.html#float
[367] https://docs.python.org/3/library/stdtypes.html#tuple
[368] https://docs.python.org/3/library/stdtypes.html#list

gene_id
str – gene id

taxon_id
int – gene taxon

exp_syn
int – expected synonymous changes

exp_nonsyn
int – expected non-synonymous changes

coverage
int – gene coverage

snps
list – 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*.

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.

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*[369]) – SNP position, relative to the gene start
- **change** (*str*[370]) – nucleotidic change
- **snp_type** (*enum*[371]) – 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*[372]) – json representation, as returned by *GeneSNP.to_json()*

gene_id = None

nonsyn
Returns the expected non-synonymous changes

snps = None

[369] https://docs.python.org/3/library/functions.html#int
[370] https://docs.python.org/3/library/stdtypes.html#str
[371] https://docs.python.org/3/library/enum.html#module-enum
[372] https://docs.python.org/3/library/stdtypes.html#str

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[373]

uid = None

class mgkit.snps.classes.**RatioMixIn**
Bases: object[374]

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.

$$\frac{pN}{pS} = \frac{{}^{oN}/_{eN}}{{}^{oS}/_{eS}} \tag{7.1}$$

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

- **flag_value** (*bool*[375]) – 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*[376]) – 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

[373] https://docs.python.org/3/library/stdtypes.html#str
[374] https://docs.python.org/3/library/functions.html#object
[375] https://docs.python.org/3/library/functions.html#bool
[376] https://docs.python.org/3/library/functions.html#bool

All other cases return a NaN value

Return type float[377]

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[378]

New in version 0.1.13.

Enum that defines SNP types. Supported at the moment:

- unknown = 0
- 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()*.

[377] https://docs.python.org/3/library/functions.html#float
[378] https://docs.python.org/3/library/enum.html#enum.Enum

Parameters

- **snp_data** (*dict*[379]) – dictionary sample->GeneSyn of SNPs data
- **taxonomy** – Uniprot Taxonomy
- **min_num** (*int*[380]) – minimum number of samples in which a valid pN/pS is found
- **index_type** (*str*[381], *None*[382]) – 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*[383]) – dictionary sample->GeneSyn of SNPs data
- **taxonomy** – Uniprot Taxonomy
- **min_num** (*int*[384]) – minimum number of samples in which a valid pN/pS is found
- **gene_map** (*dict*[385]) – dictionary of mapping for the gene_ids in in SNPs data
- **index_type** (*str*[386], *None*[387]) – 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

[379] https://docs.python.org/3/library/stdtypes.html#dict
[380] https://docs.python.org/3/library/functions.html#int
[381] https://docs.python.org/3/library/stdtypes.html#str
[382] https://docs.python.org/3/library/constants.html#None
[383] https://docs.python.org/3/library/stdtypes.html#dict
[384] https://docs.python.org/3/library/functions.html#int
[385] https://docs.python.org/3/library/stdtypes.html#dict
[386] https://docs.python.org/3/library/stdtypes.html#str
[387] https://docs.python.org/3/library/constants.html#None

- **snp_data** (*dict*[388]) – dictionary sample->GeneSyn of SNPs data
- **taxonomy** – Uniprot Taxonomy
- **min_num** (*int*[389]) – minimum number of samples in which a valid pN/pS is found
- **gene_map** (*dict*[390]) – dictionary of mapping for the gene_ids in in SNPs data
- **index_type** (*str*[391], *None*[392]) – 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*[393]) – dictionary sample->GeneSyn of SNPs data
- **taxonomy** – Uniprot Taxonomy
- **min_num** (*int*[394]) – minimum number of samples in which a valid pN/pS is found
- **rank** (*str*[395]) – taxon rank to map. Valid ranks are found in `mgkit.taxon.TAXON_RANKS`
- **index_type** (*str*[396], *None*[397]) – 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*[398]) – minimum coverage allowed (included)

[388] https://docs.python.org/3/library/stdtypes.html#dict
[389] https://docs.python.org/3/library/functions.html#int
[390] https://docs.python.org/3/library/stdtypes.html#dict
[391] https://docs.python.org/3/library/stdtypes.html#str
[392] https://docs.python.org/3/library/constants.html#None
[393] https://docs.python.org/3/library/stdtypes.html#dict
[394] https://docs.python.org/3/library/functions.html#int
[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/constants.html#None
[398] https://docs.python.org/3/library/functions.html#int

Returns True if the gene has enough coverage

Return type bool[399]

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*[400]) – 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[401]

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 *UniprotTaxonomy*)
- **`filter_list`** (*iterable*) – list of taxon IDs to include/exclude
- **`exclude`** (*bool*[402]) – 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[403]

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.

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

[399] https://docs.python.org/3/library/functions.html#bool
[400] https://docs.python.org/3/library/functions.html#bool
[401] https://docs.python.org/3/library/functions.html#bool
[402] https://docs.python.org/3/library/functions.html#bool
[403] https://docs.python.org/3/library/functions.html#bool

- **dataframe** – pandas.Series instance with a MultiIndex (gene-taxon)
- **taxonomy** – taxon.UniprotTaxonomy instance with the full taxonomy
- **taxon_rank** (*str*[404]) – 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*)

Changed in version 0.2.2: added *use_uid* argument

Changed in version 0.3.1: added *haplotypes*

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*[405]) – dictionary with the *GeneSNP* instances
- **min_num** (*int*[406]) – 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*
- **index_type** (*str*[407], *None*[408]) – 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*[409]) – if True, uses the *GeneSNP.uid* instead of *GeneSNP.gene_id*
- **flag_values** (*bool*[410]) – if True, mgkit.snps.classes.GeneSNP.calc_ratio_flag() will be used, instead of mgkit.snps.classes.GeneSNP.calc_ratio()
- **haplotypes** (*bool*[411]) – if *flag_values* is False, and *haplotypes* is True, the 0/0 case will be returned as 0 instead of NaN

Returns pandas.DataFrame with the pN/pS values for the input SNPs, with the columns being the samples.

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.

[404] https://docs.python.org/3/library/stdtypes.html#str
[405] https://docs.python.org/3/library/stdtypes.html#dict
[406] https://docs.python.org/3/library/functions.html#int
[407] https://docs.python.org/3/library/stdtypes.html#str
[408] https://docs.python.org/3/library/constants.html#None
[409] https://docs.python.org/3/library/functions.html#bool
[410] https://docs.python.org/3/library/functions.html#bool
[411] https://docs.python.org/3/library/functions.html#bool

Parameters

- **snps_data** (*dict*[412]) – dictionary with the instances of `classes.GeneSNP`
- **min_cov** (*int*[413]) – 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[414]

`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*[415]) – 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.

Parameters

- **ratios** (*dict*[416]) – dictionary instance id->iterable
- **aggr_func** (*function*) – any function returning a value that can be used as a key in sorting
- **reverse** (*bool*[417]) – 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()`.

[412] https://docs.python.org/3/library/stdtypes.html#dict
[413] https://docs.python.org/3/library/functions.html#int
[414] https://docs.python.org/3/library/stdtypes.html#dict
[415] https://docs.python.org/3/library/stdtypes.html#dict
[416] https://docs.python.org/3/library/stdtypes.html#dict
[417] https://docs.python.org/3/library/functions.html#bool

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*[418]) – 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

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*[419]) – 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.UniprotTaxonomy.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*[420]) – taxon ID to be mapped
- **rank** (*str*[421]) – taxon rank used (mgkit.taxon.TAXON_RANKS)
- **include_higher** (*bool*[422]) – determines if a rank higher than the one requested is to be returned

Returns iterator with the values or empty

[418] https://docs.python.org/3/library/stdtypes.html#dict
[419] https://docs.python.org/3/library/functions.html#int
[420] https://docs.python.org/3/library/functions.html#int
[421] https://docs.python.org/3/library/stdtypes.html#str
[422] https://docs.python.org/3/library/functions.html#bool

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*[423]) – a1 leftmost number
- **a1e** (*int*[424]) – a1 rightmost number
- **a2s** (*int*[425]) – a2 leftmost number
- **a2e** (*int*[426]) – 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*[427]) – number to test
- **start** (*int*[428]) – leftmost number
- **end** (*int*[429]) – 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

[423] https://docs.python.org/3/library/functions.html#int
[424] https://docs.python.org/3/library/functions.html#int
[425] https://docs.python.org/3/library/functions.html#int
[426] https://docs.python.org/3/library/functions.html#int
[427] https://docs.python.org/3/library/functions.html#int
[428] https://docs.python.org/3/library/functions.html#int
[429] https://docs.python.org/3/library/functions.html#int

- **intervals** (*intervals*) – iterable where each element is a closed range (tuple)
- **end** (*int*[430]) – 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[431]

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.

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*[432]) – start position for the first range
- **end1** (*int*[433]) – end position for the first range
- **start2** (*int*[434]) – start position for the second range
- **end2** (*int*[435]) – 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[436], tuple[437])

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.

Parameters **ranges** (*iterable*) – each element is a tuple like *(1, 10)*

Returns sum of all ranges lengths

Return type int[438]

[430] https://docs.python.org/3/library/functions.html#int
[431] https://docs.python.org/3/library/stdtypes.html#list
[432] https://docs.python.org/3/library/functions.html#int
[433] https://docs.python.org/3/library/functions.html#int
[434] https://docs.python.org/3/library/functions.html#int
[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#tuple
[438] https://docs.python.org/3/library/functions.html#int

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[439] or None[440])

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[441]

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=<type 'int'>*, *cache=True*)

Bases: object[442]

Changed in version 0.3.3: added *cache* in __init__

[439] https://docs.python.org/3/library/stdtypes.html#tuple
[440] https://docs.python.org/3/library/constants.html#None
[441] https://docs.python.org/3/library/stdtypes.html#list
[442] https://docs.python.org/3/library/functions.html#object

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*[443]) – dictionary whose values are iterables
- **func** (func) – function to apply to the dictionary values

Returns dictionary with transformed values

Return type dict[444]

class mgkit.utils.dictionary.**cache_dict_file**(*iterator*, *skip_lines=0*)

Bases: object[445]

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

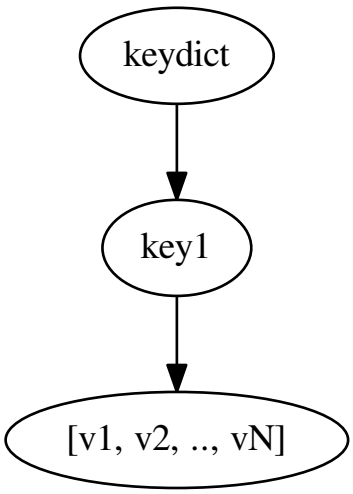
```
>>> from mgkit.io.blast import parse_accession_taxa_table
>>> i = parse_accession_taxa_table('nucl_gb.accession2taxid.gz', key=0)
>>> d = cache_dict_file(i)
>>> d['AH001684']
4400
```

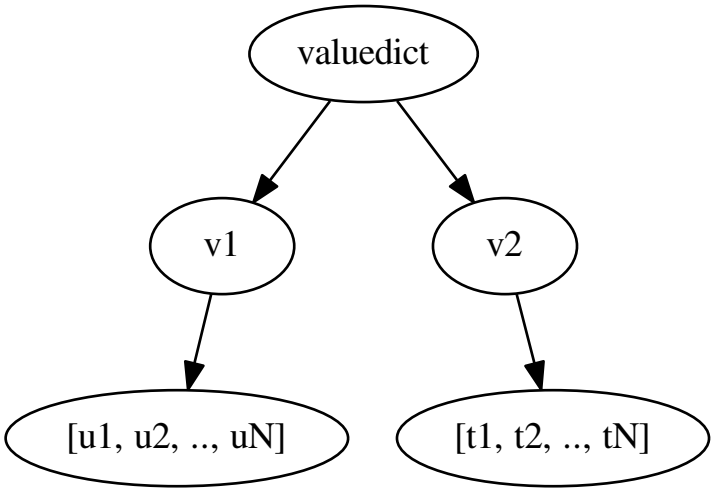
next()

mgkit.utils.dictionary.**combine_dict**(*keydict*, *valuedict*)

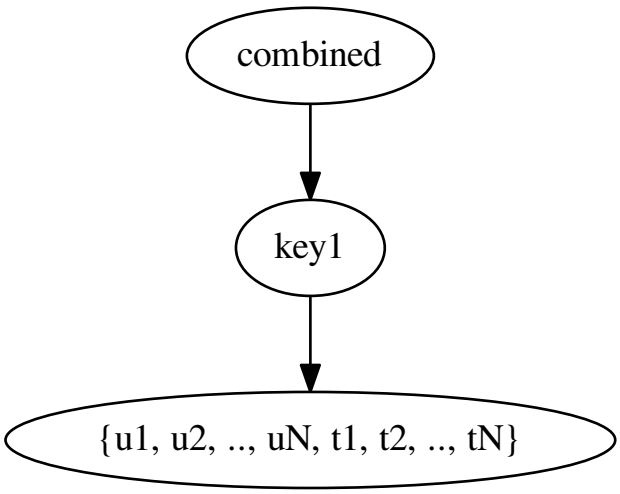
Combine two dictionaries when the values of keydict are iterables. The combined dictionary has the same keys as keydict and the its values are sets containing all the values associated to keydict values in valuedict.

[443] https://docs.python.org/3/library/stdtypes.html#dict
[444] https://docs.python.org/3/library/stdtypes.html#dict
[445] https://docs.python.org/3/library/functions.html#object





Resulting dictionary will be



Parameters

- **keydict** (*dict*[446]) – dictionary whose keys are the same as the returned dictionary
- **valuedict** (*dict*[447]) – 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 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*[448]) – dictionary whose keys are the same as the returned dictionary
- **valuedict** (*dict*[449]) – 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*[450]) – dictionary key->list
- **min_num** (*int*[451]) – minimum number of elements in the value iterable

Return dict filtered dictionary

[446] https://docs.python.org/3/library/stdtypes.html#dict
[447] https://docs.python.org/3/library/stdtypes.html#dict
[448] https://docs.python.org/3/library/stdtypes.html#dict
[449] https://docs.python.org/3/library/stdtypes.html#dict
[450] https://docs.python.org/3/library/stdtypes.html#dict
[451] https://docs.python.org/3/library/functions.html#int

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.

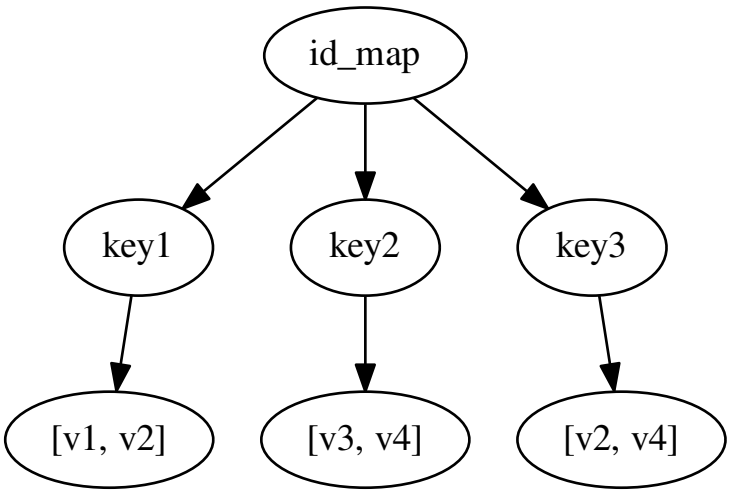
Parameters

- **s_id** (*dict*[452]) – element to look for in the dictionary's values
- **d** (*object*[453]) – dictionary to search in

Return list list of keys in which d was found

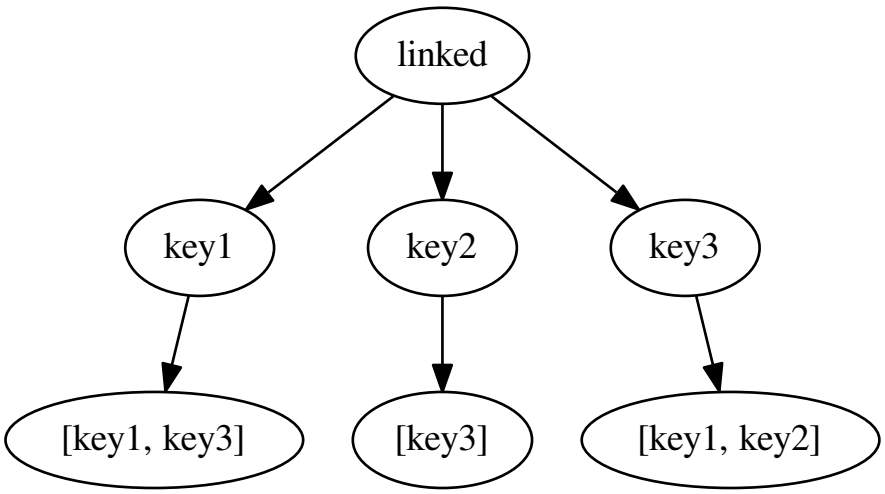
mgkit.utils.dictionary.**link_ids**(*id_map*, *black_list=None*)

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.



Becomes:

[452] https://docs.python.org/3/library/stdtypes.html#dict
[453] https://docs.python.org/3/library/functions.html#object



Parameters

- **id_map** (*dict*[454]) – 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

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*[455]) – 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*[456]) – dictionary to be splitted
- **threshold** (*number*) – must be comparable to threshold
- **aggr_func** (*func*) – function used to aggregate the dictionary values

[454] https://docs.python.org/3/library/stdtypes.html#dict
[455] https://docs.python.org/3/library/stdtypes.html#dict
[456] https://docs.python.org/3/library/stdtypes.html#dict

- **key_filter** (*iterable*) – if specified, only these key will be in the resulting dictionary

Returns two dictionaries

mgkit.utils.sequence module

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

Module contents

Package that contains utility functions/classes

mgkit.workflow package

Submodules

mgkit.workflow.add_coverage module

mgkit.workflow.add_gff_info module

mgkit.workflow.assembly module

mgkit.workflow.blast2gff module

mgkit.workflow.download_data module

The scripts downloads the data that is used by the framework for some of its functions. It's mostly a shortcut to call the download_data function that is present in every module that is in the package mappings and in the kegg module.

The only option required, is the email contact for the person using the script; this is used to make sure that the API requirements in Uniprot are fullfilled and they can contact the person using the script is any problem arise.

Note: The default behavior is to download first the taxonomy data form Uniprot, Kegg and additional mapping data.

Taxonomy

It is downloaded from Uniprot and build a data structure that is used by several scripts and function in the package. The download can take some time.

Warning: if only the taxonomy is to be downloaded, both the *-x* and *-p* options must be passed to the script.

Kegg Onthologs

The Kegg data is the only "required" data at the moment, because it's used to download the sequence data (via the donwload_profiles script) for the profiling. It is the only data that can't be saved unless it's fully downloaded.

Kegg data is required by the mappers currently supported, and its download takes longer. The mappers handle timeouts and if exceptions are raised the data is saved and the download is resumed when the script is started again.

Other Mappings

The other mappings (from KO) are downloaded by default and this can be excluded by using the *-p* option. Mappings for Gene Onthology, eggNOG and CaZy are downloaded.

As the download of the mappings can take a lot of time, or break because of the number of requests to the web sites, checkpoints are saved often, so it can resumed at a later time.

Warning: the Gene Onthology module has specific requirements, so if they are not downloaded

`mgkit.workflow.download_data.`**`main`**`()`
Main function

`mgkit.workflow.download_data.`**`set_parser`**`()`
Sets command line arguments parser

mgkit.workflow.download_profiles module

Overview

This script downloads sequence data for each gene of interest (ortholog) and all the specified taxa. The files that are downloaded with this script can then be used to create HMMER profiles, to search for similarity in a aminoacidic or nucleotidic sequence.

Limitations

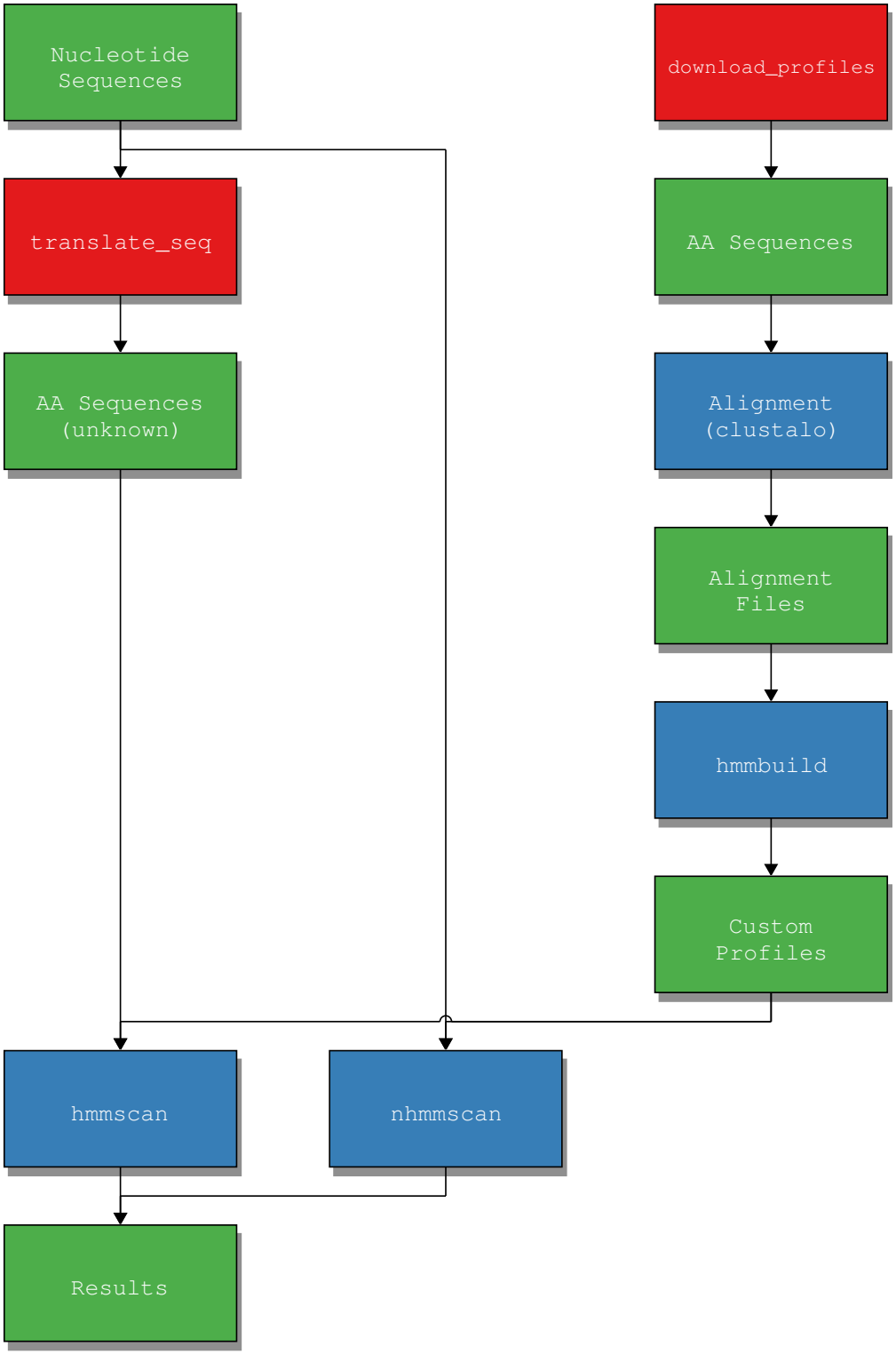
At the moment, the script uses Kegg Orthologs as the ortholog database.

Warning: Some taxa may still black listed, because they are not relevant to the rumen microbiome. If you find such thing to occur to you, please contact me or open an issue on the repository.

Required Data

The script requires data from Kegg Orthologs and Uniprot to be downloaded, before it can be used. The script **download_data** (*download-data - Download Taxonomy from NCBI*) automates the process.

Workflow for Custom Profiles



The process of building the profiles to be used with HMMER is a step that involves several tasks (illustrated in the Workflow above):

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. This sctipt, *download_profiles* can be used. The aminoacidic sequences downloaded are then aligned using Clustal Omega (or other) and for each alignment a profile is built.

HMMER required the use of aminoacidic sequences, to be match against the profiles. The *translate_seq* script can be used to translate nucleotidic sequences into aminoacidic ones. However, the last version of HMMER should be able to match nucleotidic sequences, but it was not tested by us. The example Workflow above illustrate that.

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.

Usage

The default behaviour is to download all Kegg Orthologs for all taxa in the given taxonomy. Taxa can be filtered by both lineage (e.g. archaea, carnivora, etc.) and rank (e.g. genus, family, etc.). Another option is to specify the KO and taxa IDs to download.

Taxa Filters

The way a taxon is specified is through a few different rules:

- specific **taxon ids** in uniprot
- a specific **taxon rank** (e.g.: genus, phylum, etc.)
- optional lineage filter: the lineage filter make sure that the name specified is included in the lineage attribute in the taxonomy.

As an example, if the rank chosen is genus, and the lineage option is set to archaea, only the taxa whose rank is genus and that belong to the archaea subtree will be downloaded:

```
$ download_profiles -m EMAIL -r genus -l archaea mg_data/kegg.pickle \
-t mg_data/taxonomy.pickle
```

This allows to customise the level of specificity that we want in profiling and make the process of downloading faster. For metagenomic data, a good start is mixing different taxon ranks, using the order or genus for the genes and then specifying a lineage of interest.

Because each profile is indipendent from each other, it's useful to start the download with a certain rank and then run the profiling. During the profiling a new download can be started and so on.

Specific Genes and Taxa

It is possible to download only specific taxa and KO and can be done using the *-i* and *-ko* respectively. When *-ko* is used, loading Kegg Data with *-k* is not required and it is up to the user to ensure the correct genes or taxa.

An example to download only KO from 3 different taxa:

```
$ download_profiles -v -m EMAIL -ko K00016 -i 9611 9645 9682 \
-t mg_data/taxonomy.pickle
```

The same example using taxa filtering, instead (at the time of writing):

```
$ download_profiles -v -m EMAIL -ko K00016 -r genus -l carnivora \
-t mg_data/taxonomy.pickle
```

Changes

Changed in version 0.2.1: added *-ko* option, resolved issues caused by changes in library

mgkit.workflow.download_profiles.**add_profiles_to_length**(*seqs*, *length_data*)
Adds the average profile length to the dictionary

mgkit.workflow.download_profiles.**choose_ko_ids**(*kegg_data*, *options*)
Returns the list of mapping ID->Name according to the options passed

mgkit.workflow.download_profiles.**choose_taxa**(*taxonomy*, *options*)
Returns the list of ids to look for in Uniprot

mgkit.workflow.download_profiles.**download_ko_sequences**(*ko_id*, *taxon_ids*, *reviewed*, *contact*)
Downloads the sequences associated to all taxon IDs provided

mgkit.workflow.download_profiles.**filter_found_taxa**(*taxon_ids_found*, *taxon_ids*, *taxonomy*)
Filter the taxa found in Uniprot, making sure that they at a lower level of those requested

mgkit.workflow.download_profiles.**filter_taxonomy_by_lineage**(*taxonomy*, *taxon_ids*, *lineage*)

mgkit.workflow.download_profiles.**filter_taxonomy_by_rank**(*taxonomy*, *taxon_ids*, *rank*)

mgkit.workflow.download_profiles.**load_data**(*taxon_data*, *length_data_name*)
Loads data for script

mgkit.workflow.download_profiles.**main**()
Main function

mgkit.workflow.download_profiles.**map_ko_to_uniprot**(*ko_id*, *taxon_ids*, *reviewed*, *contact*)
Returns the taxon IDs found in Uniprot for a specific id

mgkit.workflow.download_profiles.**set_parser**()
argument parser configuration

mgkit.workflow.download_profiles.**write_ko_sequences**(*seqs*, *taxonomy*, *output_dir*)
Writes fasta sequences to disc

mgkit.workflow.extract_gff_info module

mgkit.workflow.fasta_utils module

mgkit.workflow.fastq_utils module

mgkit.workflow.filter_gff module

mgkit.workflow.hmmer2gff module

mgkit.workflow.json2gff module

mgkit.workflow.nuc2aa module

mgkit.workflow.sampling_utils module

mgkit.workflow.snp_parser module

mgkit.workflow.taxon_utils module

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[457]

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[458]

New in version 0.2.6.

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.**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

[457] https://docs.python.org/3/library/argparse.html#argparse.Action
[458] https://docs.python.org/3/library/argparse.html#argparse.Action

7.1.2 Submodules

mgkit.align module

Module dealing with BAM/SAM files

class mgkit.align.**SamtoolsDepth**(*file_handle*, *num_seqs=10000*)
Bases: object[459]

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

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*[460]) – sequence for which to return mean coverage
- **start** (*int*[461]) – start of the region
- **end** (*int*[462]) – end of the region

Returns mean coverage of the requested region

Return type float[463]

mgkit.align.**add_coverage_info**(*annotations*, *bam_files*, *samples*, *attr_suff='_cov'*)
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

[459] https://docs.python.org/3/library/functions.html#object
[460] https://docs.python.org/3/library/stdtypes.html#str
[461] https://docs.python.org/3/library/functions.html#int
[462] https://docs.python.org/3/library/functions.html#int
[463] https://docs.python.org/3/library/functions.html#float

- **annotations** (*iterable*) – an iterable that returns annotations
- **min_cov** (*int*[464]) – minumum coverage for a base to counted
- **progress** (*bool*[465]) – 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[466]

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*[467]) – sequence id
- **feat_from** (*int*[468]) – start position of feature
- **feat_to** (*int*[469]) – end position of feature

Return int coverage array for the annotation

mgkit.align.**read_samtools_depth**(*file_handle*, *num_seqs=10000*)

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*[470]) – number of sequence that fires a log message

Yields *tuple* – the first element is the sequence identifier and the second one is the *numpy* array with the positions

mgkit.consts module

Module containing constants for the filter package

[464] https://docs.python.org/3/library/functions.html#int
[465] https://docs.python.org/3/library/functions.html#bool
[466] https://docs.python.org/3/library/stdtypes.html#dict
[467] https://docs.python.org/3/library/stdtypes.html#str
[468] https://docs.python.org/3/library/functions.html#int
[469] https://docs.python.org/3/library/functions.html#int
[470] https://docs.python.org/3/library/functions.html#int

mgkit.graphs module

New in version 0.1.12.

Graph module

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*[471]) – 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*)
New in version 0.1.12.

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
- default is not None: the node attribute will be set to *default*

Parameters

- **graph** – the graph to annotate
- **attr** (*str*[472]) – the attribute to annotate
- **id_map** (*dict*[473]) – the dictionary with the values for each node
- **default** – the value used in case an *id* is not found in *id_map*

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*[474]) – dictionary with the links
- **name** (*str*[475]) – name of the graph
- **edge_type** (*str*[476]) – an optional name for the *edge_type* attribute set for each edge
- **weight** (*float*[477]) – the weight assigned to each edge in the graph

Returns an instance of `networkx.Graph`

Return type graph

[471] https://docs.python.org/3/library/stdtypes.html#dict
[472] https://docs.python.org/3/library/stdtypes.html#str
[473] https://docs.python.org/3/library/stdtypes.html#dict
[474] https://docs.python.org/3/library/stdtypes.html#dict
[475] https://docs.python.org/3/library/stdtypes.html#str
[476] https://docs.python.org/3/library/stdtypes.html#str
[477] https://docs.python.org/3/library/functions.html#float

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*[478]) – dictionary with the links
- **name** (*str*[479]) – name of the graph
- **edge_type** (*str*[480]) – an optional name for the *edge_type* attribute set for each edge
- **weight** (*float*[481]) – 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

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*[482]) – KGML file as a string, or anything that can be passed to ElementTree

[478] https://docs.python.org/3/library/stdtypes.html#dict
[479] https://docs.python.org/3/library/stdtypes.html#str
[480] https://docs.python.org/3/library/stdtypes.html#str
[481] https://docs.python.org/3/library/functions.html#float
[482] https://docs.python.org/3/library/stdtypes.html#str

- **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*[483]) – 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

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.**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*[484]

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[485]

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

[483] https://docs.python.org/3/library/stdtypes.html#set
[484] https://docs.python.org/3/library/functions.html#object
[485] 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 = <_sre.SRE_Pattern object>

cpd_re = <_sre.SRE_Pattern object at 0x42982f0>

empty_cache(*methods=None*)
New in version 0.3.1.

Empties the cache completely or for a specific method(s)

Parameters **methods** (*iterable, str*[486]) – 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',
```

[486] https://docs.python.org/3/library/stdtypes.html#str

```
 '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*[487]) – kegg id of the resource to get
- **option** (*str*[488]) – 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 = <_sre.SRE_Pattern object>

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*[489]) – the target db
- **ids** – can be either a single id as a string or a list of ids
- **strip** (*bool*[490]) – if the prefix (e.g. ko:K00601) should be stripped

[487] https://docs.python.org/3/library/stdtypes.html#str
[488] https://docs.python.org/3/library/stdtypes.html#str
[489] https://docs.python.org/3/library/stdtypes.html#str
[490] https://docs.python.org/3/library/functions.html#bool

- **max_len** (*int*[491]) – 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*[492]) – 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 = <_sre.SRE_Pattern object>

rn_name_re = <_sre.SRE_Pattern object>

write_cache(*file_handle*)

New in version 0.3.1.

Write the cache to file

class mgkit.kegg.**KeggCompound**(*cp_id=None*, *description=''*)

Bases: object[493]

Kegg compound

__eq__(*other*)

```
>>> KeggCompound('test') == KeggCompound('test')
True
>>> KeggCompound('test') == 1
False
```

__ne__(*other*)

```
>>> KeggCompound('test') != KeggCompound('test1')
True
>>> KeggCompound('test') != 1
True
```

class mgkit.kegg.**KeggData**(*fname=None*, *gen_maps=True*)

Bases: object[494]

gen_ko_map()

gen_maps()

get_cp_names()

[491] https://docs.python.org/3/library/functions.html#int
[492] https://docs.python.org/3/library/stdtypes.html#str
[493] https://docs.python.org/3/library/functions.html#object
[494] https://docs.python.org/3/library/functions.html#object

get_ko_cp_links(*path_filter=None*, *description=False*)

get_ko_cp_links_alt(*direction='out'*, *description=False*)

get_ko_names()

get_ko_pathway_map(*black_list=None*)

get_ko_pathways(*ko_id*)

get_ko_rn_links(*path_filter=None*, *description=False*)

get_pathway_ko_map(*black_list=None*)

get_rn_cp_links(*path_filter=None*, *description=False*)

get_rn_names()

load_data(*fname*)

maps = None

pathways = None

save_data(*fname*)

class mgkit.kegg.**KeggMapperBase**(*fname=None*)

Bases: object[495]

Base object for Kegg mapping classes

get_id_map()

Returns a mapping->KOs dictionary (a reverse mapping to get_ko_map)

get_id_names()

Returns a copy of the mapping names

get_ko_map()

Returns a copy of the KO->mapping dictionary

static 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*[496]) – id used in the query
- **query** (*str*[497]) – query passed to the Uniprot API, ko_id is replaced using str.format()
- **column** (*str*[498]) – column used in the results table used to map the ids
- **contact** (*str*[499]) – email address to be passed in the query (requested Uniprot API)

Note: each mapping in the column is separated by a ;

load_data(*fname*)

Loads mapping data to disk

save_data(*fname*)

Saves mapping data to disk

[495] https://docs.python.org/3/library/functions.html#object
[496] https://docs.python.org/3/library/stdtypes.html#str
[497] https://docs.python.org/3/library/stdtypes.html#str
[498] https://docs.python.org/3/library/stdtypes.html#str
[499] https://docs.python.org/3/library/stdtypes.html#str

class mgkit.kegg.**KeggModule**(*entry=None*, *old=False*)
Bases: object[500]

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

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*[501], *iterable*) – if None the returned edges are for the whole module, if an iterable (converted to a set[502]), only edges for those reactions are returned

Yields *tuple* – the elements are the compounds and reactions in the module

class mgkit.kegg.**KeggOrtholog**(*ko_id=None*, *description=''*, *reactions=None*)
Bases: object[503]

Kegg Ortholog gene

__eq__(*other*)

[500] https://docs.python.org/3/library/functions.html#object
[501] https://docs.python.org/3/library/constants.html#None
[502] https://docs.python.org/3/library/stdtypes.html#set
[503] https://docs.python.org/3/library/functions.html#object

```
>>> KeggOrtholog('test') == KeggOrtholog('test')
True
>>> KeggOrtholog('test') == 1
False
```

__ne__(*other*)

```
>>> KeggOrtholog('test') != KeggOrtholog('test1')
True
>>> KeggOrtholog('test') != 1
True
```

class mgkit.kegg.**KeggPathway**(*path_id=None*, *description=None*, *genes=None*)
Bases: object[504]

Kegg Pathway

__eq__(*other*)

```
>>> KeggPathway('test') == KeggPathway('test')
True
>>> KeggPathway('test') == 1
False
```

__ne__(*other*)

```
>>> KeggPathway('test') != KeggPathway('test1')
True
>>> KeggPathway('test') != 1
True
```

class mgkit.kegg.**KeggReaction**(*entry*)
Bases: object[505]

Changed in version 0.3.1: reworked, only stores the equation

Kegg Reaction, used for parsing the equation line

left_cp = None

right_cp = None

rn_id = None

mgkit.kegg.**download_data**(*fname='kegg.pickle'*, *contact=None*)

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*[506]) – reaction string

Returns left and right components as *sets*

Return type tuple[507]

Raises ValueError[508] – if the

[504] https://docs.python.org/3/library/functions.html#object
[505] https://docs.python.org/3/library/functions.html#object
[506] https://docs.python.org/3/library/stdtypes.html#str
[507] https://docs.python.org/3/library/stdtypes.html#tuple
[508] https://docs.python.org/3/library/exceptions.html#ValueError

mgkit.logger module

Module configuring log information

mgkit.logger.**config_log**(*level=10*, *output=<open file '<stderr>', mode 'w'>*)
Minimal configuration of :mod‘logging‘ module, default to debug level and the output is printed to standard error

Parameters

- **level** (*int*[509]) – 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*[510]) – logging level
- **output** (*file*) – file to which write the log

mgkit.simple_cache module

class mgkit.simple_cache.**memoize**(*func*)
Bases: dict[511]

a cache found on the PythonDecoratorLibrary[512]

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: exceptions.Exception

New in version 0.1.13.

Raised if no lowest common ancestor can be found in the taxonomy

class mgkit.taxon.**UniprotTaxonTuple**(*taxon_id*, *s_name*, *c_name*, *rank*, *lineage*, *parent_id*)
Bases: tuple[513]

__getnewargs__()
Return self as a plain tuple. Used by copy and pickle.

__getstate__()
Exclude the OrderedDict from pickling

__repr__()
Return a nicely formatted representation string

_asdict()
Return a new OrderedDict which maps field names to their values

[509] https://docs.python.org/3/library/functions.html#int
[510] https://docs.python.org/3/library/functions.html#int
[511] https://docs.python.org/3/library/stdtypes.html#dict
[512] https://wiki.python.org/moin/PythonDecoratorLibrary#Alternate_memoize_as_dict_subclass
[513] https://docs.python.org/3/library/stdtypes.html#tuple

_replace(*_self*, ***kwds*)
Return a new UniprotTaxonTuple object replacing specified fields with new values

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

class mgkit.taxon.**UniprotTaxonomy**(*fname=None*)
Bases: object[514]

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

__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[515]) – the lineage as a keyword arguments

Returns the taxon_id of the last element in the lineage

Return type int[516]

[514] https://docs.python.org/3/library/functions.html#object
[515] https://docs.python.org/3/library/stdtypes.html#dict
[516] https://docs.python.org/3/library/functions.html#int

Raises

- `ValueError`[517] – 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*[518]) – scientific name of the taxon
- **common_name** (*str*[519]) – common name
- **rank** (*str*[520]) – rank, defaults to 'no rank'
- **parent_id** (*int*[521]) – 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[522]

Raises

- `KeyError`[523] – 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*[524]) – 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*[525]) – the scientific name
- **rank** (*str*[526], *None*[527]) – 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

[517] https://docs.python.org/3/library/exceptions.html#ValueError
[518] https://docs.python.org/3/library/stdtypes.html#str
[519] https://docs.python.org/3/library/stdtypes.html#str
[520] https://docs.python.org/3/library/stdtypes.html#str
[521] https://docs.python.org/3/library/functions.html#int
[522] https://docs.python.org/3/library/functions.html#int
[523] https://docs.python.org/3/library/exceptions.html#KeyError
[524] https://docs.python.org/3/library/functions.html#int
[525] https://docs.python.org/3/library/stdtypes.html#str
[526] https://docs.python.org/3/library/stdtypes.html#str
[527] https://docs.python.org/3/library/constants.html#None

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[528]

Raises

- `KeyError`[529] – 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*[530]) – taxon_id to return the lineage
- **names** (*bool*[531]) – if the elements of the list are converted into the scientific names
- **only_ranked** (*bool*[532]) – only return the ranked taxa
- **with_last** (*bool*[533]) – include the taxon_id passed to the list

Returns the lineage of the passed taxon_id as a list of IDs or names

Return type list[534]

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 *UniprotTaxonomy.get_ranked_taxon()*) to another rank, such as *phylum*.

Parameters

- **taxon_id** (*int*[535]) – taxon_id to return the lineage
- **only_ranked** (*bool*[536]) – only return the ranked taxa
- **with_last** (*bool*[537]) – include the taxon_id passed to the list
- **sep** (*str*[538]) – separator used to join the lineage string
- **rank** (*int*[539] *or None*[540]) – if None the full lineage is returned, otherwise the lineage will be cut to the specified rank

Returns lineage string

Return type str[541]

[528] https://docs.python.org/3/library/stdtypes.html#list
[529] https://docs.python.org/3/library/exceptions.html#KeyError
[530] https://docs.python.org/3/library/functions.html#int
[531] https://docs.python.org/3/library/functions.html#bool
[532] https://docs.python.org/3/library/functions.html#bool
[533] https://docs.python.org/3/library/functions.html#bool
[534] https://docs.python.org/3/library/stdtypes.html#list
[535] https://docs.python.org/3/library/functions.html#int
[536] https://docs.python.org/3/library/functions.html#bool
[537] https://docs.python.org/3/library/functions.html#bool
[538] https://docs.python.org/3/library/stdtypes.html#str
[539] https://docs.python.org/3/library/functions.html#int
[540] https://docs.python.org/3/library/constants.html#None
[541] https://docs.python.org/3/library/stdtypes.html#str

get_name_map()
Returns a taxon_id->s_name dictionary

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

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*[542]) – 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*[543]) – if True, uses `TAXON_ROOTS` to solve the root taxa

Returns instance of `UniprotTaxon` 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*[544]) – leaf taxon id
- **anc_ids** (*int*[545]) – ancestor taxon id(s)

Return bool True if the ancestor taxon is in the leaf taxon lineage

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*[546], *file*) – file name to which save the instance data

Raises

- `DependencyError` – if the file name contains *.msgpack* and the
- package is not installed

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 *UniprotTaxonomy.add_lineage()*

[542] https://docs.python.org/3/library/stdtypes.html#str
[543] https://docs.python.org/3/library/functions.html#bool
[544] https://docs.python.org/3/library/functions.html#int
[545] https://docs.python.org/3/library/functions.html#int
[546] https://docs.python.org/3/library/stdtypes.html#str

Return type dict[547]

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*[548]) – if True, the names are kept as-is in the *s_name* attribute of *UniprotTaxonTuple* and the "cleaned" version in *c_name* (e.g. f__Ammonifexaceae -> Ammonifexaceae). If False, the values are switched
- **sep** (*str*[549]) – separator between the columns of the file

Returns dictionary of genome_id -> taxon_id, reflecting the created taxonomy

Return type dict[550]

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*[551], *file*) – file name or handle to the file
- **names_file** (*str*[552], *file*, *None*[553]) – file name or handle to the file, if None, names won't be added to the taxa
- **merged_file** (*str*[554], *file*, *None*[555]) – 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

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

[547] https://docs.python.org/3/library/stdtypes.html#dict
[548] https://docs.python.org/3/library/functions.html#bool
[549] https://docs.python.org/3/library/stdtypes.html#str
[550] https://docs.python.org/3/library/stdtypes.html#dict
[551] https://docs.python.org/3/library/stdtypes.html#str
[552] https://docs.python.org/3/library/stdtypes.html#str
[553] https://docs.python.org/3/library/constants.html#None
[554] https://docs.python.org/3/library/stdtypes.html#str
[555] https://docs.python.org/3/library/constants.html#None

Parameters **file_handle** (*str*[556], *file*) – file name to which save the instance data

Raises

- DependencyError – if the file name contains *.msgpack* and the
- package is not installed

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** – *UniprotTaxonomy* instance
- **taxon_id** (*int*[557]) – taxonomic identifier
- **anc_id** (*int*[558]) – 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

- **taxonomy** – *UniprotTaxonomy* instance
- **taxon_id1** (*int*[559]) – taxonomic identifier of first taxon
- **taxon_id2** (*int*[560]) – 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 *UniprotTaxonomy* instance
- **taxon_id** (*int*[561]) – taxon_id whose lineage to return
- **names** (*bool*[562]) – if True, the returned list contains the names of the taxa instead of the taxon_id

[556] https://docs.python.org/3/library/stdtypes.html#str
[557] https://docs.python.org/3/library/functions.html#int
[558] https://docs.python.org/3/library/functions.html#int
[559] https://docs.python.org/3/library/functions.html#int
[560] https://docs.python.org/3/library/functions.html#int
[561] https://docs.python.org/3/library/functions.html#int
[562] https://docs.python.org/3/library/functions.html#bool

- **only_ranked** (*bool*[563]) – if True, only taxonomic levels whose rank is in data:*TAXON_RANKS* will be returned
- **with_last** (*bool*[564]) – 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[565]

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 *UniprotTaxonomy* 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** – *UniprotTaxonomy* instance used to test
- **taxon_id1** (*int*[566]) – first taxon ID
- **taxon_id2** (*int*[567]) – 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 *UniprotTaxonomy*
- **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[568]

Raises

- *NoLcaFound* – when no common ancestry is found or the number of
- *taxon_ids* is 0

[563] https://docs.python.org/3/library/functions.html#bool
[564] https://docs.python.org/3/library/functions.html#bool
[565] https://docs.python.org/3/library/stdtypes.html#list
[566] https://docs.python.org/3/library/functions.html#int
[567] https://docs.python.org/3/library/functions.html#int
[568] https://docs.python.org/3/library/functions.html#int

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** – *UniprotTaxonomy* instance used to test
- **taxon_id1** (*int*[569]) – first taxon ID
- **taxon_id2** (*int*[570]) – 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*[571], *file*) – file name or handle to the file

Returns dictionary with merged_id -> taxon_id

Return type dict[572]

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*[573], *file*) – file name or handle to the file
- **name_classes** (*tuple*[574]) – name classes to save, only the scientific and common name are stored

Returns dictionary with merged_id -> taxon_id

Return type dict[575]

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*[576], *file*) – file name or handle to the file
- **taxa_names** (*dict*[577]) – dictionary with the taxa names (returned from *parse_ncbi_taxonomy_names_file()*)

[569] https://docs.python.org/3/library/functions.html#int
[570] https://docs.python.org/3/library/functions.html#int
[571] https://docs.python.org/3/library/stdtypes.html#str
[572] https://docs.python.org/3/library/stdtypes.html#dict
[573] https://docs.python.org/3/library/stdtypes.html#str
[574] https://docs.python.org/3/library/stdtypes.html#tuple
[575] https://docs.python.org/3/library/stdtypes.html#dict
[576] https://docs.python.org/3/library/stdtypes.html#str
[577] https://docs.python.org/3/library/stdtypes.html#dict

Yields *UniprotTaxonTuple* – UniprotTaxonTuple 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

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** – *UniprotTaxonomy* 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

exception mgkit.**DependencyError**(*package*)
Bases: exceptions.Exception

Raised if an optional requirement is needed

mgkit.**check_version**(*version*)

mgkit.**cite**(*file_handle=<open file '<stderr>', mode 'w'>*)
Print citation to the specified stream

7.2 mgkit.filter.gff_old module

7.3 mgkit.utils.r_func module

7.4 mgkit

CHAPTER 8

Changes

8.1 0.3.3

8.1.1 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()*

8.1.2 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)

8.1.3 Deprecated

- *mgkit.io.fastq.convert_seqid_to_old()*

8.2 0.3.2

Removed deprecated code

8.3 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.

8.3.1 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()*

8.3.2 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 - Translate Nucleotides to Aminoacids (Deprecated)*)
- *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 - Translate Nucleotides to Aminoacids (Deprecated)* 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()*

8.3.3 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[578] options
- ondition when a blast output is empty, hence *lineno* is not initialised when a message is logged

8.3.4 Deprecated

- *translate_seq - Translate Nucleotides to Aminoacids (Deprecated)* will be removed in version 0.4.0, instead use the similar command in *fasta-utils - Fasta Utilities*

8.4 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*)

8.4.1 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)

8.4.2 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*

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

8.5 0.2.5

8.5.1 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

8.5.2 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)

8.6 0.2.4

8.6.1 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

8.6.2 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)

8.7 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]
```

8.7.1 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[579], 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

8.7.2 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[580], 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

[579] https://github.com/pypa/virtualenv/issues/54
[580] https://github.com/msgpack/msgpack-python

8.7.3 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
- `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

8.8 0.2.2

8.8.1 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[581], 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

8.8.2 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*

8.8.3 Removed

- deprecated code from the **snps** package

8.9 0.2.1

8.9.1 Added

- added `mgkit.db.mongo`
- added `mgkit.db.dbm`

[581] http://bioinf.wehi.edu.au/featureCounts/

- 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()`
- 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

8.9.2 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

8.9.3 Deprecated

- `mgkit.filter.taxon.filter_taxonomy_by_lineage()`
- `mgkit.filter.taxon.filter_taxonomy_by_rank()`

8.9.4 Removed

- removed old *filter_gff* script

8.10 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

8.11 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

8.12 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()`

8.13 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[582] installed with the notebook extension
- 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

8.14 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

[582] http://ipython.org

- 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)

8.15 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

8.16 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

K

L

M

N

O

P

Q

R

S

T

U

W