## Contents:

1  Build information  
2  PyPI version and Licensing  
3  conda version  
4  Description  
5  Reporting problems and requesting improvements  
   5.1  Citing pyani  
   5.2  About pyani  
   5.3  QuickStart Guide  
   5.4  Requirements  
   5.5  Installation Guide  
   5.6  Basic Use  
   5.7  Examples  
   5.8  pyani subcommands  
   5.9  Testing  
   5.10  Contributing to pyani  
   5.11  Licensing  
   5.12  pyani package  
   5.13  Indices and tables  

Python Module Index  
Index
CHAPTER 1

Build information
CHAPTER 2

PyPI version and Licensing
If you’re feeling impatient, please head over to the *QuickStart Guide*
**Description**

*pyani* is a program and Python package that provides support for calculating average nucleotide identity (ANI) and related measures for whole genome comparisons, and for rendering relevant graphical and tabular summary output. Where available, it natively takes advantage of multicore systems, and can integrate with SGE or OGE-compatible job schedulers to manage the computationally-heavy sequence comparisons.

Installing the *pyani* Python package also installs the program *pyani*, which enables command-line based analysis of genomes. Results are stored in a private SQLite3 database local to the machine, which permits addition of genomes to a previous analysis without having to recalculate all previously-performed comparisons, facilitating incremental analysis and visualisation, and enabling incremental maintenance of results for a taxonomic group as new genome sequences become available.

If you use *pyani* in your work, we would be grateful if you could please cite us as indicated on the *Citations* page.
CHAPTER 5

Reporting problems and requesting improvements

If you encounter bugs or errors, or would like to suggest ways in which pyani can be improved, please raise a new issue at the pyani GitHub issues page.

If you’d like to fix a bug or make an improvement yourself, contributions are welcomed, and guidelines on how to do this can be found at the Contributing to pyani documentation page.

5.1 Citing pyani

We would be grateful if you could please cite the following manuscript in your work if you have found pyani useful:


@Article{C5AY02550H,
  author ="Pritchard, Leighton and Glover, Rachel H. and Humphris, Sonia and Elphinstone, John G. and Toth, Ian K.",
  title ="Genomics and taxonomy in diagnostics for food security: soft-rotting enterobacterial plant pathogens",
  journal ="Anal. Methods",
  year ="2016",
  volume ="8",
  issue ="1",
  pages ="12-24",
  publisher ="The Royal Society of Chemistry",
  doi ="10.1039/C5AY02550H",
  url ="http://dx.doi.org/10.1039/C5AY02550H",
  abstract ="Soft rot Enterobacteriaceae (SRE) are bacterial plant pathogens that cause blackleg, wilt and soft rot diseases on a broad range of important crop and ornamental plants worldwide. These organisms (spanning the genera Erwinia, Pectobacterium, Dickeya, and Pantoea) cause significant economic and yield losses in the field, and in storage. They are transmissible through surface water by trade and other movement of plant material and soil, and in some cases are"
subject to international legislative and quarantine restrictions. Effective detection and diagnosis in support of food security legislation and epidemiology is dependent on the ability to classify pathogenic isolates precisely. Diagnostics and classification are made more difficult by the influence of horizontal gene transfer and sometimes inaccurate nomenclatural and taxonomic assignments that persist in strain collections and online sequence databases. Here we briefly discuss the relationship between taxonomy, genotype and phenotype in the SRE, and their implications for diagnostic testing and legislation. We present novel whole-genome classifications of the SRE, illustrating inconsistencies between the established taxonomies and evidence from completely sequenced isolates. We conclude with a perspective on the future impact of widespread whole-genome sequencing and classification methods on detection and identification of bacterial plant pathogens in support of legislative and policy efforts in food security.”

Tip: When citing pyani in your text, please consider quoting the precise release version (if you are using a specific release), or the specific commit hash (if you are using the ‘bleeding-edge’ development version), e.g.

"...using PYANI (v0.3.0)...

or

"...using PYANI (commit bb1cb5c)...

This will enable precise reproduction of this part of your work by others.

5.1.1 Publications citing pyani

If you are using pyani, you are in good company. These authors and manuscripts have employed pyani to help with their whole-genome classification:

(If I’ve missed out your paper, please do put in a pull request on GitHub or get in touch and I’ll be happy to add it!)

2021

- Abdullah et al. (2021) “Comparative analysis of whole genome sequences of Leptospira spp. from RefSeq database provide interspecific divergence and repertoire of virulence factors” bioRxiv doi:10.1101/2021.01.12.426470
- Biffignandi et al (2021) “Genome of Superficieibacter maynardsmithii, a novel, antibiotic susceptible representative of Enterobacteriaceae” G3 doi:10.1093/g3journal/jkab019
- Boeuf et al. (2021) “Meta-pangenomics Reveals Depth-dependent Shifts in Metabolic Potential for the Ubiquitous Marine Bacterial SAR324 Lineage” Research Squared doi:10.21203/rs.3.rs-225427/v1

• Huang et al. (2021) “Phenotypic properties and genotyping analysis of Bacillus cereus group isolates from dairy and potato products” LWT doi:10.1016/j.lwt.2021.110853


• Lee et al. (2021) “Bifidobacterium bifidum strains synergize with immune checkpoint inhibitors to reduce tumour burden in mice” Nat. Microbiol. doi:10.1038/s41564-020-00831-6

• Liu et al. (2021) “Corynebacterium anserum sp. nov., isolated from the faeces of greater white-fronted geese (Anser albifrons) at Poyang Lake, PR China” Int. J. Syst. Evol. Microbiol. doi:10.1099/ijsem.0.004637


• Luo et al. (2021) “Isolation and characterization of new phage vB_CtuP_A24 and application to control Cronobacter spp. in infant milk formula and lettuce” Food Res. Int. doi:10.1016/j.foodres.2021.110109


• Ma et al. (2021) “Identification of Pectobacterium versatile causing blackleg of potato in New York State” Plant Disease doi:10.1094/PDIS-09-20-2089-RE


• Singh et al. (2021) “Genome-based reclassification of Amycolatopsis eurytherma as a later heterotypic synonym of Amycolatopsis thermoflava” Int. J. Syst. Evol. Microbiol. doi:10.1099/ijsem.0.004642
• Vargas-Peralta et al. (2021) “Identification of Pectobacterium species isolated from the soft rot of tetecho (Neobuxbaumia tetetzo), a columnar cactus, and associated metagenomics” bioRxiv doi:10.1101/2021.02.01.429127

• Vincent et al. (2021) “AsaGEI2d: a new variant of a genomic island identified in a group of Aeromonas salmonicida subsp. salmonicida isolated from France, which bears the pAsa7 plasmid” FEMS Micro. Lett. doi:10.1093/femsle/fnab021

• Young et al. (2021) “Defining the Rhizobium leguminosarum Species Complex” genes doi:10.3390/genes12010111

• Zvi-Kedem et al. (2021) “The worm affair: Genetic diversity in two species of symbionts that co-occur in tubeworms from the Mediterranean Sea” bioRxiv doi:10.1101/2021.01.27.428081

2020


• Albert et al. (2020) “Comparative Pangenomics of the Mammalian Gut Commensal Bifidobacterium longum” Microorganisms doi:10.3390/microorganisms8100007


• Christman et al. (2020) “Methanogens Within a High Salinity Oil Reservoir From the Gulf of Mexico” Front. Microbiol. doi:10.3389/fmicb.2020.570714


• Dangel et al. (2020) “Corynebacterium silvaticum sp. nov., a unique group of NTTB corynebacteria in wild boar and roe deer” Int. J. Syst. Evol. Microb. doi:10.1099/ijsem.0.004195

• de Andrade Alves et al. “First isolation and whole-genome sequencing of a Shewanella algae strain from a swine farm in Brazil” BMC Microbiol. doi:10.1186/s12866-020-02040-x


• Duar et al. (2020) “Comparative Genome Analysis of Bifidobacterium longum subsp. infantis Strains Reveals Variation in Human Milk Oligosaccharide Utilization Genes among Commercial Probiotics” Nutrients doi:10.3390/nu12113247


• Feng et al. (2020) “Phylogenetic and genomic analysis reveals high genomic openness and genetic diversity of Clostridium perfringens” Microb. Geno. doi:10.1099/mgen.0.000441

• Ferrerira et al. “Genome-based reclassification of Azospirillum brasilense Sp245 as the type strain of Azospirillum balaniorum sp. nov” Int. J. Syst. Evol. Micro. doi:10.1099/ijsem.0.004517

• Fishbein et al. (2020) “Randomized Controlled Trial of Oral Vancomycin Treatment in Clostridioides difficile-Colonized Patients” mSphere doi:10.1128/mSphere.01296-20


• Freitas et al. (2020) “Yeast communities associated with cacti in Brazil and the description of Kluyveromyces starmeri sp. nov. based on phylogenomic analyses” Yeast doi:10.1002/jea.3528


• Gardon et al. (2020) “A drift-barrier model drives the genomic landscape of a structured bacterial population” Molecular Ecol. doi:10.1111/mec.15628

• Girard et al. (2020) “Reliable Identification of Environmental Pseudomonas Isolates Using the rpoD Gene” Microorganisms doi:10.3390/microorganisms8081166


• Hulin et al. (2020) “Cherry picking by pseudomonads: after a century of research on canker, genomics provides insights into the evolution of pathogenicity towards stone fruits” Plant Pathology doi:10.1111/ppa.13189

• Ibarra Caballero et al. (2020) “Genome comparison and transcriptome analysis of the invasive brown root rot pathogen, Phellinus noxius, from different geographic regions reveals potential enzymes associated with degradation of different wood substrates” Fungal Biology doi:10.1016/j.funbio.2019.12.007


• Kim et al. (2020) “Genome analysis of Lactobacillus plantarum subsp. plantarum KCCP11226 reveals a well-conserved C30 carotenoid biosynthetic pathway” 3 Biotech. doi:10.1007/s13205-020-2149-y


Li et al. (2020) “Comparative Genomics Reveals Broad Genetic Diversity, Extensive Recombination and Nascent Ecological Adaptation in Micrococcus luteus” Research Square doi:10.21203/rs.3.rs-62334/v1

Li et al. (2020) “Isolation and Characterization of Bacillus cereus Phage vB_BceP-DLc1 Reveals the Largest Member of the Phi29-Like Phages” Microorganisms doi:10.3390/microorganisms8111750


Mu et al. (2020) “Bradymonabacteria, a novel bacterial predator with versatile survival strategies in saline environments” Microbiome doi:10.21203/rs.2.20535/v1

• Müller et al. (2020) “Aliarcobacter butzleri from Water Poultry: Insights into Antimicrobial Resistance, Virulence and Heavy Metal Resistance” Genes doi:10.3390(genesis11091104


• Mullins et al. (2020) “Reclassification of the biocontrol agents Bacillus subtilis BY-2 and Tu-100 as Bacillus velezensis and insights into the genomic and specialized metabolite diversity of the species” Microbiol. doi:10.1099/mic.0.000986


• Norris et al. (2020) “Acidithiobacillus ferrianus sp. nov.: an ancestral extremely acidophilic and facultatively anaerobic chemolithoautotroph” Extremophiles doi:0.1007/s00792-020-01157-1

• Oshkin et al. (2020) “Pan-Genome-Based Analysis as a Framework for Demarcating Two Closely Related Methanotroph Genera Methylocystis and Methylosinus” Microorganisms doi:10.3390/microorganisms8050768

• Pandey et al. (2020) “Evidence of homologous recombination as a driver of diversity in Brachyspira pilosicoli” Micro. Genom. doi:10.1099/mgen.0.000470

• Paulsen et al. (2020) “Production of the antimicrobial compound tetrabromopyrrol and the Pseudomonas quinolone system precursor, 2-heptyl-4-quinolone, by a novel marine species Pseudoalteromonas galatheae sp. nov.” Sci. Rep. doi:10.1038/s41598-020-78439-3


• Pasanen et al. (2020) “Pectobacterium parvum sp. nov., having a Salmonella SPI-1-like Type III secretion system and low virulence” Int. J. Syst. Evol. Microb. doi:10.1099/ijsem.0.004057

• Peral-Aranega et al. (2020) “Bacteria Belonging to Pseudomonas typographi sp. nov. from the Bark Beetle Ips typographus Have Genomic Potential to Aid in the Host Ecology” Insect doi:10.3390/insects11090593

• Pierry et al. (2020) “High-quality draft genome sequence resources of eight Xylella fastidiosa strains isolated from citrus, coffee, plum and hibiscus in South America” Phytopathology doi:10.1094/PHYTO-05-20-0162-A


• Portier et al. (2020) “Updated taxonomy of Pectobacterium genus in the CIRM-CFBP bacterial collection: when newly described species reveal “old” endemic population” Preprints doi:10.20944/preprints202008.0608.v1

• Rackaityte (2020) “Viable bacterial colonization is highly limited in the human intestine in utero” Nature Medicine doi:10.1038/s41591-020-0761-3


• Shen et al. (2020) “Helicobacter monodelphidis sp. nov. and Helicobacter didelphidarum sp. nov., isolated from grey short-tailed opossums (Monodelphis domestica) with endemic cloacal prolapses” Int. J. Syst. Evol. Micro. doi:10.1099/ijsem.0.004424


• Thapa et al. (2020) “Genome-wide analyses of Liberibacter species provides insights into evolution, phylogenetic relationships, and virulence factors” Mol. Plant Path. doi:10.1111mpp.12925


• Tsukimi et al. (2020) “Draft Genome Sequences of Bifidobacterium animalis Consecutively Isolated from Healthy Japanese Individuals” J. Genomics doi:10.7150/jgen.38516

• Vijayan et al. (2020) “Bacteria known to induce settlement of larvae of Hydrodes elegans are rare in natural inductive biofilm” Aquatic Microb. Ecol. doi:10.3354/ame01925


• Wang et al. (2020) “Comparative Genomics Analysis of Lactobacillus ruminis from Different Niches” Genes doi:10.3390/genes11010070


• Weiser et al. (2020) “A Novel Inducible Prophage from Burkholderia vietnamiensis G4 is Widely Distributed across the Species and Has Lytic Activity against Pathogenic Burkholderia” Viruses doi:10.3390/v12060601


• Wist et al. (2020) “Phenotypic and Genotypic Traits of Vancomycin-Resistant Enterococci from Healthy Food-Producing Animals” Microorganisms doi:10.3390/microorganisms8020261


5.1. Citing pyani

• Zhang et al. (2020) “Deinococcus detaillensis sp. nov., isolated from humus soil in Antarctica” Arch. Microbiol. doi:10.1007/s00203-020-01920-0


2019


• Acevedo et al. (2019) “Bacillus claus ar sp. nov. is a new Bacillus cereus group species isolated from soil” BioRxiv doi:10.1101/508077


• Boukerb et al. (2019) “Campylobacter armoricus sp. nov., a novel member of the Campylobacter lari group isolated from surface water and stools from humans with enteric infection” Int. J. Syst. Evol. Micro. doi:10.1099/ijsem.0.003836


• Doud et al. (2019) “Function-driven single-cell genomics uncovers cellulose-degrading bacteria from the rare biosphere” ISME J. doi:10.1038/s41396-019-0557-y


• Feng et al. (2019) “Complete genome sequence of Hahella sp. KA22, a prodigiosin-producing algicidal bacterium” Marine Genomics doi:10.1016/j.margen.2019.04.003


• Ghosh et al. (2019) “Reanalysis of Lactobacillus paracasei Lbs2 Strain and Large-Scale Comparative Genomics Places Many Strains into Their Correct Taxonomic Position” Microorganisms doi:10.3390/microorganisms7110487


• Kaminsky et al. (2019) “Genomic Analysis of γ-Hexachlorocyclohexane-Degrading Sphingopyxis lindanitol-eras WS5A3p Strain in the Context of the Pangenome of Sphingopyxis” Genes doi:0.3390/genes10090688

• Khan et al. (2019) “Genomic and physiological analyses reveal that extremely thermophilic Caldicellulosiruptor changbaiensis deploys unique cellulose attachment mechanisms” BioRxiv doi:10.1101/622977


5.1. Citing pyani
• Kiu et al. (2019) “Phylogenomic analysis of gastroenteritis-associated Clostridium perfringens in England and Wales over a 7-year period indicates distribution of clonal toxigenic strains in multiple outbreaks and extensive involvement of enterotoxin-encoding (CPE) plasmids” Micro. Genom. doi:10.1099/mgen.0.000297


• Kochetkova et al. (2019) “Tepidiforma bonchosmolovskayae gen. nov., sp. nov., a moderately thermophilic Chloroflexi bacterium from a Chukotka hot spring (Arctic, Russia), representing a novel class, Tepidiformia, which includes the previously uncultivated lineage OLB14” Int. J. Syst. Evol. Microbiol. doi:10.1099/ijsem.0.003902


• McIntyre et al. (2019) “Elevation of Pectobacterium carotovorum subsp. odoriferum to species level as Pectobacterium odoriferum sp. nov., proposal of Pectobacterium brasiliense sp. nov. and Pectobacterium actinidiae sp. nov., emended description of Pectobacterium carotovorum and description of Pectobacterium versatile sp. nov., isolated from streams and symptoms on diverse plants” Int. J Syst. Evol. Biol doi:10.1099/ijsem.0.003611


• Stevens et al. (2019) “Whole-genome-based phylogeny of Bacillus cytotoxicus reveals different clades within the species and provides clues on ecology and evolution” Sci. Rep. doi:10.1038/s41598-018-36254-x


• Thorell et al. (2019) “Isolates from colonic spirochaetosis in humans show high genomic divergence and carry potential pathogenic features but are not detected by 16S amplicon sequencing using standard primers for the human microbiota” BioRxiv doi:10.1101/544502


• Vincent et al. (2019) “A Mesophilic Aeromona salmonicida Strain Isolated from an Unsuspected Host, the Micratory Bird Pied Avocet” Microorganisms doi:10.3390/microorganisms7120592


5.1. Citing pyani


• Zakham et al. (2019) “Molecular diagnosis and enrichment culture identified a septic pseudoarthrosis due to an infection with Erysipelatoclostridium ramosum” Int. J. Inf. Dis. doi:10.1016/j.ijid.2019.02.001


2018


• Ferretti et al. (2018) “Mother-to-Infant Microbial Transmission from Different Body Sites Shapes the Developing Infant Gut Microbiome” Cell Host Microbe doi:10.1016/j.chom.2018.06.005


• Jung et al. (2018) “Complete genome sequence of Bifidobacterium choerinum FMB-1, a resistant starch-degrading bacterium” J. Biotech. doi:10.1016/j.jbiotec.2018.03.009
• Samad et al. (2017) “Comparative genome analysis of the vineyard weed endophyte Pseudomonas viridiflava CDRTc14 showing selective herbicidal activity” Sci. Rep. doi:10.1038/s41598-017-16495-y
• Sant’Anna et al. (2018) “Genome-based reclassification of Paenibacillus dauci as a later heterotypic synonym of Paenibacillus shenyangensis” Int. J. Syst. Evol. Micro. doi:10.1099/ijsem.0.003127
• Vincent & Charette (2018) “Completion of genome of Aeromonas salmonicida subsp. salmonica 01-B526 reveals how sequencing technologies can influence sequence quality and result interpretations” New Microb. New Inf. doi:10.1016/j.nmni.2018.05.007
• Wilhelm (2018) “Following the terrestrial tracks of Caulobacter - redefining the ecology of a reputed aquatic oligotroph” ISME J doi:10.1038/s41396-018-0257-z


2017

• Anderson et al. (2017) “Genomic variation in microbial populations inhabiting the marine subseafloor at deep-sea hydrothermal vents” Nat. Comm. doi:10.1038/s41467-017-01228-6

• Ding et al. (2017) “Loss of the ssrA genome island led to partial debromination in the PBDE respiring Dehalococcoides mccartyi strain GY50” Env. Micro. doi:10.1111/1462-2920.13817


• Tohno et al. (2017) “Lactobacillus silagincola sp. nov. and Lactobacillus pentosiphilus sp. nov., isolated from silage” Int. J Syst. Evol. Biol doi:10.1099/ijsem.0.002196


2016

- Burstein et al. (2016) “New CRISPR–Cas systems from uncultivated microbes” Nature doi:10.1038/nature21059

5.2 About pyani

pyani is a Python package and standalone program for calculation of whole-genome similarity measures. It is designed to be used with draft or complete prokaryote (bacterial and archaeal) genomes, and implements the following methods:

- ANIb (average nucleotide identity using BLAST+)
- ANIblastall (average nucleotide identity using legacy BLAST)
- ANIm (average nucleotide identity using MUMmer)
- TETRA (4-mer sequence profiles)

5.2.1 Funding and Support

The development of pyani has been made possible by generous funding from a number of sources, including:

- The Scottish Government
  - 2011-2016 workpackages
  - 2016-2019 workpackage RD2.1.4

As is the case with much bioinformatics software, ongoing development does not receive continuous direct funding support. If you use pyani, please do cite this tool in your publications and outputs - this helps us justify future funding applications to maintain and improve the software.

You can find more information about how to cite pyani on the Citing pyani page.
5.3 QuickStart Guide

5.3.1 Installation

To use pyani you will need to install it on a local machine (your laptop, desktop, server or cluster). Installation is easiest using one of the two more popular Python package managers:

1. conda

pyani is available through the bioconda channel of Anaconda:

```
conda install -c bioconda pyani
```

2. PyPI

pyani is available via the PyPI package manager for Python:

```
pip install pyani
```

**Tip:** pyani can also be installed directly from source, or run as a Docker image. More detailed, and alternative, installation instructions can be found on the Installation Guide page.

5.3.2 pyani Walkthrough

The general procedure for any pyani analysis is:

1. Collect genomes for analysis (and index them)
2. Create a database to hold genome data and analysis results
3. Perform ANI/TETRA/etc. analysis
4. Report and visualise analysis results
5. Generate species hypotheses (classify the input genomes) using the analysis results

To see options available for the pyani program, use the -h (help) option:

```
pyani -h
```

An example ANIm (ANI with MUMmer) analysis using pyani is provided as a walkthrough below. The sequence of commands used is:

```
pyani download --email my.email@my.domain -t 203804 C_blochmannia
pyani createdb
pyani anim C_blochmannia C_blochmannia_ANIm
   --name "C. blochmannia run 1"
   --labels C_blochmannia/labels.txt --classes C_blochmannia/classes.txt
pyani report --runs C_blochmannia_ANIm/ --formats html,excel,stdout
pyani report --run_results 1 --formats html,excel,stdout C_blochmannia_ANIm/
pyani report --run_matrices 1 --formats html,excel,stdout C_blochmannia_ANIm/
pyani plot --formats png,pdf --method seaborn C_blochmannia_ANIm 1
```
Tip: If you have the pyani source code, you can run the walkthrough commands by executing make walkthrough at the command-line, in the repository root. You can clean up the walkthrough output with make clean_walkthrough.

1. Collect genomes

It is possible to use genomes you have already placed into a local directory with pyani, but for this walkthrough a new set of genomes will be obtained from GenBank, using the pyani download command.

Tip: To read more about using local files with pyani, please see the Indexing Genomes documentation. To read more about downloading genomes from NCBI, please see the Downloading Genomes from NCBI documentation.

Attention: To use their online resources programmatically, NCBI require that you provide your email address for contact purposes if jobs go wrong, and for their own usage statistics. This should be specified with the --email <EMAIL ADDRESS> argument of pyani download.

Using the pyani download subcommand, we download all available genomes for Candidatus Blochmannia from NCBI. The taxon ID for this grouping is 203804, and this ID is passed as the -t argument. The final (compulsory) argument is the path to the directory into which the genome data will be downloaded.

pyani download --email my.email@my.domain -t 203804 C_blochmannia

This creates a new directory (C_blochmannia) with the following contents:

```
$ tree C_blochmannia
C_blochmannia
    GCF_000011745.1_ASM1174v1_genomic.fna
    GCF_000011745.1_ASM1174v1_genomic.fna.gz
    GCF_000011745.1_ASM1174v1_genomic.md5
    ...
    GCF_000973545.1_ASM97354v1_hashes.txt
    classes.txt
    labels.txt
```

Each downloaded genome is represented by four files: the genome sequence (FASTA: *.fna, compressed: *.fna.gz), an NCBI hashes file (*.hashes.txt) and an MD5 hash of the genome sequence file (*.md5).

Two additional files are created, summarising all genomes in the subdirectory:

- classes.txt: defines a class to which each input genome belongs. This is used for determining membership of groups for each genome, and annotating graphical output.
- labels.txt: provides text which will be used to label each input genome in the graphical output from pyani

2. Create database

pyani uses a local SQLite3 database to store genome data and analysis results. Existing databases can be re-used. For this walkthrough we create a new, empty database by executing the command:

```
pyani createdb
```
Tip: This creates the new database in a default location (pyani/pyanidb), but the name and location of this database can be controlled with the pyani createdb command (see the Creating a Local pyani Database documentation). The path to the database can be specified in each of the subsequent commands, to enable maintenance and sharing of multiple analysis runs.

3. Conduct ANIm analysis

We run ANIm on the downloaded genomes by specifying first the directory containing the genome data (here, C_blochmannia) then the path to a directory which will contain the analysis results (C_blochmannia_ANIm for this walkthrough).

We also provide a name for the analysis (--name, for later human-readable reference), with optional files defining labels for each genome to be used when plotting output (--labels) and a set of classes to which each genome belongs (--classes) for downstream analysis:

```bash
pyani anim C_blochmannia C_blochmannia_ANIm \
   --name "C. blochmannia run 1" \ 
   --labels C_blochmannia/labels.txt --classes C_blochmannia/classes.txt
```

This command runs ANIm analysis on the genomes in the specified C_blochmannia directory. As we did not specify a database, the analysis results will be stored in the default database we created earlier (pyani/pyanidb), where they will be identified by the name C. blochmannia run 1. The comparison result files will be written to the C_blochmannia_ANIm directory.

4. Reporting Analyses and Analysis Results

We can list all the runs contained in the (default) database, using the command:

```bash
pyani report --runs C_blochmannia_ANIm/ --formats html,excel,stdout
```

This will report the relevant information to new files in the C_blochmannia_ANIm directory.

```bash
$ tree -L 1 C_blochmannia_ANIm/
C_blochmannia_ANIm/
   nucmer_output
   runs.html
   runs.tab
   runs.xlsx
```

Tip: By default the pyani report command will create a tab-separated text file with the .tab suffix, but by using the --formats option, we have also created an HTML file, and an Excel file with the same data. The stdout option also prints the output table to the terminal window.

By inspecting the runs.tab file (or any of the other runs.* files) we see that our walkthrough analysis has run ID 1. So we can use this ID to get tables of specific information for that run, such as:

- the genomes that were analysed in all runs
  ```bash
  pyani report --runs_genomes --formats html,excel,stdout C_blochmannia_ANIm/
  ```

- the complete set of pairwise comparison results for a single run (listed by comparison)
comparison results as matrices (percentage identity and coverage, number of aligned bases and “similarity errors”, and a Hadamard matrix of identity multiplied by coverage).

pyani report --run_matrices 1 --formats html,excel,stdout C_blochmannia_ANIm/

Attention: The --run_results and --run_matrices options take a single run ID or a comma-separated list of IDs (such as 1,3,4,5,9) as an argument, and will produce output for each specified run ID.

Graphical output

Graphical output is obtained by executing the pyani plot subcommand, specifying the output directory and run ID. Optionally, output file formats and the graphics drawing method can be specified.

pyani plot --formats png,pdf --method seaborn C_blochmannia_ANIm 1

Supported output methods are:

- seaborn
- mpl (matplotlib)
- plotly

and each generates five plots corresponding to the matrices that pyani report produces:

- percentage identity of aligned regions
- percentage coverage of each genome by aligned regions
- number of aligned bases on each genome
- number of “similarity errors” on each genome
- a Hadamard matrix of percentage identity multiplied by percentage coverage for each comparison

Several graphics output formats are available, including .png, .pdf and .svg.

5.4 Requirements

The pyani package requires several other programs, packages and tools to run and develop. Many of these are automatically installed alongside pyani, but some packages and tools must be installed separately.

This page describes requirements for pyani and how/why they are used.

Tip: For more information about installation of specific packages, please see the Installation Guide page.

5.4.1 Python3

pyani is written in Python, and the modern version of Python is Python3. The legacy version of Python will not be maintained past 2020. pyani is written to use many features of Python3 and will not run on Python2.
Fig. 1: Percentage identity matrix for *Candidatus Blochmannia* ANIm analysis

Each cell represents a pairwise comparison between the named genomes on rows and columns, and the number in the cell is the pairwise identity of aligned regions. The dendrograms are single-linkage clustering trees generated from the matrix of pairwise identity results. The default colour scheme colours cells with identity > 0.95 as red, and those with < 0.95 as blue. This division corresponds to a widely-used convention for bacterial species boundaries.
Fig. 2: Percentage coverage matrix for Candidatus Blochmannia ANIm analysis
Each cell represents a pairwise comparison between the named genomes on rows and columns, and the number in the cell is pairwise coverage of each genome by aligned regions in the comparison. The dendrograms are single-linkage clustering trees generated from the matrix of pairwise coverage results. The default colour scheme colours cells with identity > 0.50 as red, and those with < 0.50 as blue. This division corresponds to a strict majority of each genome in the comparison being alignable (a plausible minimum requirement for two sequences being considered “the same thing”).

5.4. Requirements
5.4.2 NCBI-BLAST+

To carry out ANIb (average nucleotide identity using BLAST) analysis, genome sequences are compared using the BLAST+ tool, provided by NCBI. The BLAST+ tool is the current, maintained version, and is completely rewritten with respect to the legacy BLAST package (see below).

• NCBI-BLAST+

5.4.3 MUMmer v3.23

To carry out ANIm (average nucleotide identity using MUMmer) analysis, genome sequences are compared using the nucmer tool, part of the MUMmer package. Currently, pyani uses an older version of MUMmer for this analysis, pinned at version 3.23. pyani has not yet been tested with MUMmer 4.x.

• MUMmer3
• MUMmer4

5.4.4 Legacy NCBI-BLAST

An alternative implementation of ANIb (average nucleotide identity using BLAST), included for compatibility checks with other ANI calculation software is provided in pyani through the legacy script average_nucleotide_identity.py. The use of the legacy aniblastall analysis is not recommended, and NCBI do not recommend use of the legacy NCBI-BLAST tool. However, the legacy software can still be downloaded and installed, for the curious and those who wish to test legacy compatibility.

• Legacy NCBI-BLAST (not supported)

5.4.5 SQLite3

The output generated by pyani analyses is stored in a local database, provided by SQLite3, for rapid querying and recovery. This allows for persistent storage of results without the need to keep the original alignment files, and for incremental addition of new analyses. SQLite is installed with Python.

• SQLite

5.4.6 Open Grid Scheduler

When running on a cluster, pyani currently schedules jobs using the Sun Grid Engine/Open Grid Engine/Open Grid Scheduler syntax. Your cluster will require a compatible scheduler for pyani to distribute jobs appropriately.

• Open Grid Scheduler

5.4.7 Python Packages

pyani relies on functionality provided by a number of additional Python packages, and we gratefully acknowledge their contribution:

• Biopython: for working with biological data formats
• Matplotlib: for graphical output
• **NetworkX**: for graph calculations and representation
• **Numpy**: for matrix calculations
• **OpenPyXL**: for Microsoft Excel output compatibility
• **Pandas**: for dataframe operations
• **Pillow**: for graphics manipulation and rendering
• **SciPy**: for scientific computing operations
• **Seaborn**: for graphical output
• **SQLAlchemy**: (pinned at v1.2.18 for compatibility reasons) for interaction with SQLite3
• **tqdm**: provides progress bars for user interaction

### Development

We rely on a number of additional packages to aid *pyani* development, and if you set up a development environment as recommended in *Contributing to pyani*, then the following Python packages will be installed or expected to be present:

• **bandit**: to check for security issues in the codebase
• **black**: to enforce consistent, opinionated code formatting
• **codecov**: to generate code coverage output for the codecov.io service
• **coverage**: to generate code coverage output for local inspection
• **doc8**: to check docstring formatting syntax
• **flake8**: for code linting
• **jinja2**: for output/docfile templating
• **pre-commit**: for checking code style and quality prior to *git* commit
• **pylint**: for code linting
• **pytest**: to manage and run automated testing
• **pytest-cov**: to integrate *pytest* with codecov and coverage
• **pytest-ordering**: to ensure *pytest* test ordering
• **sphinx**: to generate documentation
• **sphinx-rtd-theme**: to provide local ReadTheDocs style formatting

### 5.5 Installation Guide

We support four ways to install and run *pyani* on your system:

1. Installation with **Anaconda** (i.e. the *conda* package manager) **[Recommended]**
2. Installation **via** *pip* (i.e. from PyPI)
3. Installation from source (i.e. download from GitHub)
4. Installation of a **Docker** image
Note: If you wish to contribute to development of pyani, you will require developer tools and packages that are not included in these installation instructions. To set up a local environment suitable for developing pyani, please refer to the Contributing to pyani page.

5.5.1 1. Installation with Anaconda

Anaconda <https://www.anaconda.com/> is a Python language distribution that includes the conda package manager. Several channels, themed collections of packages, are available through the conda package manager. The latest release of pyani should always be available from the bioconda channel.

We recommend installation of pyani using the Anaconda3 or Miniconda3 distributions, and the conda package manager. This provides a straightforward way of managing third-party tool (e.g. MUMmer and NCBI-BLAST) installation, and allows for installation of pyani in a virtual environment, protected from other installations on the system.

1.1 Install Anaconda or Miniconda

If not already available on your system, install Anaconda3 or Miniconda3 on your system, following the instructions for your system on their respective websites:

• Anaconda3 download
• Miniconda3 download

After installation, you can check whether conda is available by issuing the following command in your terminal:

```
$ conda -V
conda 4.7.10
```

1.2 (optional) Create a new conda environment

Creation of a new conda environment is not necessary for installation of pyani, but it can be sometimes useful to specify the available version of Python, or to keep the versions of third party tools and Python packages separate from the system-level install. This may be particularly useful if, for example, the default system-level Python is Python2, as Python3 is required for pyani.

The following commands will create and activate a new conda environment called pyani, using Python 3.7:

```
conda create --name pyani --yes python=3.7  # --yes accepts all suggestions
conda activate pyani
```

If successful, you should see the prefix (pyani) before your terminal prompt. To exit the current conda environment, issue the following command:

```
conda deactivate
```

• Managing conda environments

1.3 Add required conda channels

pyani and many of the packages it requires are provided in the conda channel called bioconda. Channels are locations where conda will look for packages, and they are typically grouped by topic area.
To install `pyani`, you will need to make available the `defaults`, `conda-forge` and `bioconda` channels, with the following commands:

```
conda config --add channels defaults
conda config --add channels bioconda
conda config --add channels conda-forge
```

- Managing conda channels

### 1.4 Install pyani

The bioconda distribution of `pyani` will install all necessary packages and software required to run the software, including NCBI-BLAST+ and MUMmer, with the following command:

```
conda install --yes pyani
```

When installation is complete, you can check for the availability of the `pyani` programs with the following commands:

```
$ pyani --version
pyani 0.2.9
$ average_nucleotide_identity.py --version
average_nucleotide_identity.py: pyani 0.2.9
$ genbank_get_genomes_by_taxon.py --version
genbank_get_genomes_by_taxon.py: pyani 0.2.9
```

**Attention:** If you wish to use the ANIblastall legacy ANIb method, then the legacy NCBI-BLAST tools need to be installed. These are not available through `conda` and must be installed manually for your system, as described below.

### 5.5.2 2. Installation with pip

PyPI is the Python Packaging Index, a repository of software for the Python language. Packages from PyPI can be installed using the `pip` package installer, which should come preinstalled with your system’s Python3. The latest release of `pyani` should always be available from PyPI.

### 2.1 (optional) Create and activate a new virtual environment

As with the `conda` installation route above, it can be useful to separate the version of `Python` you use for `pyani`, and any installed packages and tools, from the system-level `Python` installation. There are multiple tools available to do this, and for convenience we list some below:

- Anaconda distribution
- pipenv
- pyenv
- virtualenv
2.2 Install third-party tools

Two third-party software tools are needed to perform ANIm and ANIb analysis:

- **MUMmer3** for ANIm
- **NCBI-BLAST+** for ANIb

These tools are not part of the PyPI distribution of pyani, and should be installed according to the instructions on their respective websites.

**Tip:** If you are using a conda environment for pyani, you can install both tools with a single command: `conda install --yes blast mummer`.

**Attention:** If you wish to use the ANIblastall legacy ANIb method, then the legacy NCBI-BLAST tools need to be installed. These are not available through conda and must be installed manually for your system, as described below.

2.3 Install pyani

The pyani programs, and their Python dependencies, can be installed with the command:

```
pip install pyani
```

When installation is complete, you can check for the availability of the pyani programs with the following commands:

```
$ pyani --version
pyani 0.2.9
$ average_nucleotide_identity.py --version
average_nucleotide_identity.py: pyani 0.2.9
$ genbank_get_genomes_by_taxon.py --version
genbank_get_genomes_by_taxon.py: pyani 0.2.9
```

5.5.3 3. Installation from source

The source code for the current pyani release can always be found on GitHub:

- Current pyani release source

3.1 (optional) Create and activate a new virtual environment

As with the conda installation route above, it can be useful to separate the version of Python you use for pyani, and any installed packages and tools, from the system-level Python installation. There are multiple tools available to do this, and for convenience we list some below:

- Anaconda distribution
- pipenv
- pyenv
- virtualenv
3.2 Install third-party tools

Two third-party software tools are needed to perform ANIm and ANIb analysis:

- MUMmer3 for ANIm
- NCBI-BLAST+ for ANIb

These tools are not part of the PyPI distribution of pyani, and should be installed according to the instructions on their respective websites.

**Tip:** If you are using a conda environment for pyani, you can install both tools with a single command: conda install --yes blast mummer.

**Attention:** If you wish to use the ANIblastall legacy ANIb method, then the legacy NCBI-BLAST tools need to be installed. These are not available through conda and must be installed manually for your system, as described below.

3.3 Download and extract the pyani source code

Click on one of the two links on the pyani releases page, or use a download tool such as curl or wget to download the link to a convenient location. For example:

```bash
wget https://github.com/widdowquinn/pyani/archive/v0.2.9.tar.gz
```

Then extract the source code archive. This will create a new directory called pyani-<CURRENT_VERSION> in your current location:

```bash
$ tar -zxf v0.2.9.tar.gz
$ ls
[...]
pyani-0.2.9/
[...]
```

3.4 Install pyani

Change directory in the terminal to the source code directory, e.g.:

```bash
$ cd pyani-0.2.9
$ ls

CHANGES.md LICENSE
→ config.yml requirements.txt
CITATIONS MANIFEST.in setup.cfg
→
CONTRIBUTORS.md Makefile setup.py
→pyani/
→requirements-dev.txt test-requirements.txt
Dockerfile-average_nucleotide_identity README-docker.md
→requirements-pip.txt tests/
Dockerfile-genbank_get_genomes_by_taxon

pyani can be installed using Python3’s setup tools, using the command:

```bash
python3 setup.py install
```
python setup.py install

This will download and install the Python packages that pyani needs to run. When installation is complete, you can check for the availability of the pyani programs with the following commands:

```
$ pyani --version
pyani 0.2.9
$ average_nucleotide_identity.py --version
average_nucleotide_identity.py: pyani 0.2.9
$ genbank_get_genomes_by_taxon.py --version
genbank_get_genomes_by_taxon.py: pyani 0.2.9
```

5.5.4 4. Installation with Docker

Docker is a platform for running applications that uses containerisation to share virtual machines that come pre-installed with all required tools and dependencies. The latest pyani programs should always be available as separate Docker containers from DockerHub:

- average_nucleotide_identity.py
- genbank_get_genomes_by_taxon.py

In order to use the containerised versions of pyani, you must have Docker installed and working on your system. To do so, please follow the instructions at the Docker website:

- Docker website

4.1 Running pyani with Docker

To pull (if necessary) and run the pyani programs in a Docker image on your local system, use the following commands (with the Docker daemon running):

```
docker run -v ${PWD}:/host_dir leightonpritchard/average_nucleotide_identity:v<REQUIRED_VERSION>
docker run -v ${PWD}:/host_dir leightonpritchard/genbank_get_genomes_by_taxon:v<REQUIRED_VERSION>
docker run -v ${PWD}:/host_dir leightonpritchard/pyani:v<REQUIRED_VERSION>
```

Tip: If no tag is specified, then Docker will attempt to use the :latest tag, which may not exist.

Note: The -v ${PWD}:/host_dir links the Docker image to the current working directory, and enables the pyani programs to see files below the current execution point in your filesystem. This is necessary for the analysis to proceed.

5.6 Basic Use

5.6.1 Downloading Genomes from NCBI

This page describes some typical use cases for downloading genomes from NCBI using the pyani download subcommand. This command downloads all assembled genomes found beneath a given taxon identifier, at the NCBI
GenBank database. You will need to know the NCBI taxon ID for each taxon you wish to download.

**Attention:** To use their online resources programmatically, NCBI require that you provide your email address for contact purposes if jobs go wrong, and for their own usage statistics. This should be specified with the `--email <EMAIL_ADDRESS>` argument of `pyani download`.

For more information about the `pyani download` subcommand, please see the `pyani download` page, or issue the command `pyani download -h` to see the inline help.

### Download all genomes in a single taxon

The basic form of the command is:

```
pyani download --email my.email@my.domain -t <TAXON_ID> <OUTPUT_DIRECTORY>
```

This instructs `pyani` to use the `download` subcommand to obtain all available genome assemblies below the taxon ID `<TAXON_ID>`, passed with the `-t` argument, and place the downloaded files - along with label and class information files created by `pyani` in the subdirectory `<OUTPUT_DIRECTORY>`.

For example, if we wished to download all available assemblies for the bacterium *Pseudomonas flexibilis* we would identify the taxon ID to be 706570, and use this as the argument to `-t`, placing the output in a convenient subdirectory (e.g. genomes), with the command:

```
$ pyani download --email my.email@my.domain -t 706570 genomes
```

This displays each assembly as a download is attempted, and places all output in the named subdirectory:

```
$ tree genomes/
genomes/
    GCA_001312105.1_ASM131210v1_genomic.fna
    GCA_001312105.1_ASM131210v1_genomic.fna.gz
    GCA_001312105.1_ASM131210v1_genomic.md5
    GCA_001312105.1_ASM131210v1_hashes.txt
    GCF_000802425.1_ASM80242v1_genomic.fna
    GCF_000802425.1_ASM80242v1_genomic.fna.gz
    GCF_000802425.1_ASM80242v1_genomic.md5
    GCF_000802425.1_ASM80242v1_hashes.txt
```

(continues on next page)
Each genome is downloaded in compressed format (.fna.gz files) and expanded in-place to give the FASTA file (.fna) files. The MD5 hash of each FASTA file is also calculated (.md5). This will be used by pyani to uniquely identify that assembly throughout the analysis process.

```
$ head genomes/GCA_001312105.1_ASM131210v1_genomic.md5
e55cd3d913a198ac60af8d509c02ab4 genomes/GCA_001312105.1_ASM131210v1_genomic.fna
```

pyani also creates two files:

- **classes.txt**: each genome is assigned a class which is used to annotate genomes in the graphical output. pyani attempts to infer genus and species as the default class
- **labels.txt**: each genome is assigned a text label, which is used to label genomes in the graphical output. pyani attempts to infer genus, species, and strain ID as the default label

```
$ head genomes/classes.txt
1ac4941c569f32f044eba0a8540d4704 GCF_900155995.1_IMG-taxon_2681812811_annotated_genomic_Pseudomonas flexibilis
86643417980701f70b2569a5b3a2320 GCF_900101515.1_IMG-taxon_2596583557_annotated_genomic_Pseudomonas flexibilis
```

These files are used to associate labels and classes to the genome files in the pyani database, specific to the analysis run.

Both **classes.txt** and **labels.txt** can be edited to suit the user’s classification and labelling scheme.
Download all genomes from multiple taxa

To download genomes from more than one taxon, you can provide a comma-separated list of taxon IDs to the `pyani download` subcommand, e.g.:

```
pyani download --email my.email@my.domain -t <TAXON_ID1>,<TAXON_ID2>,... <OUTPUT_DIRECTORY>
```

The following command can be used to download assemblies from three different *Pseudomonas* taxa (*P. flexibilis*: 706570, *P. mosselli*: 78327, and *P. fulva*: 47880):

```
$ pyani download --email my.email@my.domain -t 706570,78327,47880 multi_taxa
GCF_900155995.1_IMG-taxon_2681812811_annotated_assembly_genomic.fna.gz: 2097152it ➜ [00:00, 3081776.59it/s]
GCF_900155995.1_IMG-taxon_2681812811_annotated_assembly_hashes.txt: 1048576it [00:00, 63489526.95it/s]
GCF_900101515.1_IMG-taxon_2596583557_annotated_assembly_genomic.fna.gz: 2097152it ➜ [00:00, 3194885.99it/s]
GCF_900101515.1_IMG-taxon_2596583557_annotated_assembly_hashes.txt: 1048576it [00:00, 77838775.82it/s]
```

**Dry-run test (identify, but do not download, files)**

If you only want to see which genomes will be downloaded from NCBI with a given `pyani download` subcommand, but not download them, then you can use the `--dry-run` option. For example:

```
$ pyani download --email my.email@my.domain -t 706570,78327,47880 multi_taxa --dry-run
WARNING: Dry run only: will not overwrite or download
WARNING: (dry-run) skipping download of GCF_900155995.1
WARNING: (dry-run) skipping download of GCF_900101515.1
WARNING: (dry-run) skipping download of GCA_001312105.1
WARNING: (dry-run) skipping download of GCF_000806415.1
(....)
```

**Download genomes for compilation of a custom Kraken database**

Kraken is a bioinformatics tool that assigns taxonomic identities to short DNA sequences, such as Illumina or Nanopore reads. Several wide-ranging Kraken databases are available for download, and around the community, but it can sometimes be useful to construct a custom local Kraken database specific for your organism or taxon of interest (e.g. for filtering out contaminating or suspect reads).

The `pyani download` command can prepare downloaded genome files for immediate use with the Kraken database-building tools, by specifying the `--kraken` option:

```
$ pyani download --email my.email@my.domain -t 706570,78327,47880 genomes_kraken --kraken
GCF_900155995.1_IMG-taxon_2681812811_annotated_assembly_genomic.fna.gz: 2097152it ➜ [00:00, 3085741.03it/s]
GCF_900155995.1_IMG-taxon_2681812811_annotated_assembly_hashes.txt: 1048576it [00:00, 140958511.30it/s]
GCF_900101515.1_IMG-taxon_2596583557_annotated_assembly_genomic.fna.gz: 2097152it ➜ [00:01, 1902023.68it/s]
GCF_900101515.1_IMG-taxon_2596583557_annotated_assembly_hashes.txt: 1048576it [00:00, 31428985.47it/s]
```

(continues on next page)
Using this option does affect downstream performance or use of `pyani` only in that the two different input files for the same genome will have distinct hashes:

```
$ head multi_taxa/GCA_001312105.1_ASM131210v1_genomic.fna
>BBCY01000001.1 Pseudomonas tuomuerensis JCM 14085 DNA, contig: JCM14085.contig00001,
ACCAGCATCTGGCGGATCAGGTCGCGGGCCTTCTCGGCCGATTGGCGGATGCGCCCGAGGTAGCGGCCGAGCGGCGCGTCGCCGCGCTCGCCCGCCAGCTCCTCGGCCATCTGCGTGTAGCCGAGCATGCTGGTCAGCAGGTTGTTGAAGTCGTGGGCAATGCCGCCGGTCAGGTGGCCGATGGCTTCCATGCGCTGCGCCTGGCGCAGCTGCTGTTCCAGCGCCGCCCGCTCCACCTCG

$ head genomes_kraken/GCA_001312105.1_ASM131210v1_genomic.fna
>BBCY01000001.1|kraken:taxid|706570 BBCY01000001.1 Pseudomonas tuomuerensis JCM 14085 DNA, contig: JCM14085.contig00001, whole genome shotgun sequence
ACCAGCATCTGGCGGATCAGGTCGCGGGCCTTCTCGGCCGATTGGCGGATGCGCCCGAGGTAGCGGCCGAGCGGCGCGTCGCCGCGCTCGCCCGCCAGCTCCTCGGCCATCTGCGTGTAGCCGAGCATGCTGGTCAGCAGGTTGTTGAAGTCGTGGGCAATGCCGCCGGTCAGGTGGCCGATGGCTTCCATGCGCTGCGCCTGGCGCAGCTGCTGTTCCAGCGCCGCCCGCTCCACCTCG
```

and so will not be considered to be the “same sequence” when repeating comparisons.

### 5.6.2 Indexing Genomes

Indexing genomes is a necessary step in `pyani` to prepare input genomes for analysis.

**Attention:** If you use the `pyani download` subcommand (see *Downloading Genomes from NCBI*) to obtain genomes for analysis, then indexing is carried out automatically. However, if you collect a local set of genomes (e.g. from your own sequencing project), then you will need to index the genomes with the `pyani index` subcommand.

For more information about the `pyani index` subcommand, please see the `pyani index` page, or issue the command `pyani index -h` to see the inline help.

**What does indexing do?**

In the context of `pyani`, *indexing* refers to generating an index code that is unique to each input genome FASTA file in the input directory. The index code is the MD5 hash for the FASTA file.

This MD5 index code is used to identify each specific input genome sequence (and associated metadata) so that duplicate comparisons can be readily identified, and previous results reused from the `pyani` database, if they are available.

Indexing also generates two files (see *Downloading Genomes from NCBI*):

- **classes.txt**: each genome is assigned a *class* which is used to annotate genomes in the graphical output. `pyani` attempts to infer genus and species as the default class.
• *labels.txt*: each genome is assigned a text label, which is used to label genomes in the graphical output. *pyani* attempts to infer genus, species, and strain ID as the default label.

These files are used to associate labels and classes to the genome files in the *pyani* database, specific to the analysis run. Both *classes.txt* and *labels.txt* can be edited to suit the user's classification and labelling scheme.

### Index a directory of FASTA files

The basic form of the command is:

```
pyani index <GENOME_DIRECTORY>
```

This instructs *pyani* to search `<GENOME_DIRECTORY>` for files with a standard FASTA suffix (`.fna`, `.fasta`, `.fa`, `.fas`, `.fsa`). For each file found, it calculates the MD5 hash and writes it to an accompanying file with extension `.md5`. The hash is then associated with a genome label and a genome class, written to the two files *labels.txt* and *classes.txt* (see above).

For example, if we have a directory called `unindexed` that contains some FASTA format genome sequence files:

```
$ tree unindexed
unindexed
    GCA_001312105.1_ASM131210v1_genomic.fna
    GCF_000834555.1_ASM83455v1_genomic.fna
    GCF_005796105.1_ASM579610v1_genomic.fna

We could run the `pyani index` command:

```
$ pyani index unindexed/
$ tree unindexed
unindexed
    GCA_001312105.1_ASM131210v1_genomic.fna
    GCA_001312105.1_ASM131210v1_genomic.md5
    GCF_000834555.1_ASM83455v1_genomic.fna
    GCF_000834555.1_ASM83455v1_genomic.md5
    GCF_005796105.1_ASM579610v1_genomic.fna
    GCF_005796105.1_ASM579610v1_genomic.md5
    classes.txt
    labels.txt
```

This creates an `.md5` file for each genome, and corresponding *classes.txt* and *labels.txt* files:

```
$ head unindexed/GCA_001312105.1_ASM131210v1_genomic.fna
>BBCY01000001.1 Pseudomonas tuomuerensis JCM 14085 DNA, contig: JCM14085.contig00001, whole genome shotgun sequence
ACCCAGCATCTGGCGGATCAGGTCGCGGGCCTTCTCGGCCGATTGGCGGATGCGCCCGAGGTAGCGGCCGAGCGGCGCGTCGCCGCCCGCTCGCCCGCCAGCTCCTCGGCGCCATTTGCCGAGTGACCGGCCAGCGCGCGCGAGCGGCGCGTCGCCGCCCGGCCAGCTCCTCGGCCGTGATGCCGACATGTGGTGCAGCAGGGTTGAAGTGCCTGTTGGCCAA
$ head unindexed/GCA_001312105.1_ASM131210v1_genomic.md5
e55cd3d913a198ac60afd8d509c02ab4 unindexed/GCA_001312105.1_ASM131210v1_genomic.fna
$ head unindexed/classes.txt
527f35b3eb9b5643d9f GCF_000834555.1_ASM83455v1_genomic
   -> Pseudomonas fulva strain MEJ086 contig_1, whole genome shotgun sequence
b00c5bf6368b083b68b25e7e28a40 GCF_005796105.1_ASM579610v1_genomic
   -> Pseudomonas mosselii strain SC006 Scaffold1, whole genome shotgun sequence
e55cd3d913a198ac60afd8d509c02ab4 GCA_001312105.1_ASM131210v1_genomic
   -> Pseudomonas tuomuerensis JCM 14085 DNA, contig: JCM14085.contig00001, whole genome shotgun sequence
$ head unindexed/labels.txt
```

(continues on next page)
527f35b3eb9dd371d8d5043dd9f  GCF_000834555.1_ASM83455v1_genomic → Pseudomonas fulva strain MEJ086 contig_1, whole genome shotgun sequence

b00c5b1f636b8083b68b128e7ee828a40  GCF_005796105.1_ASM579610v1_genomic → Pseudomonas mosselii strain SC006 Scaffold1, whole genome shotgun sequence

e55cd3d913a198ac60a0f8d509c02ab4  GCA_001312105.1_ASM131210v1_genomic → Pseudomonas tuomuerensis JCM 14085 DNA, contig: JCM14085.contig00001, whole genome → shotgun sequence

**Tip:** The class and label information produced by `pyani` index is different to that generated with `pyani download`. Genus, species and strain identifiers can reliably be obtained from NCBI metadata when downloading genomes, but with user-provided sequences the information may not be encoded in the sequence description line in a standard manner.

As a result, when using `pyani` index it is often useful to edit the `classes.txt` and `labels.txt` directly, or generate these files in some other way.

### 5.6.3 Creating a Local `pyani` Database

`pyani` stores genome information and analysis results in a persistent local SQLite3 database. This allows for reuse of previous comparisons and reanalysis of datasets without having to rerun the analysis. It also means that the genome comparison results don’t have to be stored in full on disk, saving space.

**Note:** To conduct a `pyani` analysis, there needs to be an existing database in-place.

To create a new, empty database you can use the `pyani createdb` command.

For more information about the `pyani createdb` subcommand, please see the `pyani createdb` page, or issue the command `pyani download -h` to see the inline help.

**Create a new empty `pyani` database**

The basic form of the command is:

```
pyani createdb
```

This instructs `pyani` to create a new, empty database for analysis at the default location.

**Note:** The default location for the `pyani` database is in a hidden directory: `.pyani/pyanidb`. All other `pyani` subcommands will look in this location for the database, unless told otherwise using the `--dbpath` option.

For example:

```
$ ls .pyani
ls: .pyani: No such file or directory
$ pyani createdb
$ ls .pyani
pyanidb
```
Create an empty `pyani` database at a specific location

**Tip:** If you use `pyani` for a number of distinct taxa, it can be convenient to create a new database for each project, to avoid performance issues as the database grows in size, filled by data that does not contribute to the analysis.

The following command can be used to specify the location of the newly-created `pyani` database:

```
pyani createdb --dbpath <PATH_TO_DATABASE>
```

where `<PATH_TO_DATABASE>` is the intended location of the database. For instance, to create a new database specific for an analysis we’ll call `multitaxa`, we could use the command:

```
$ ls .pyani  
pyanidb  
$ pyani createdb --dbpath .pyani/multitaxadb
$ ls .pyani  
multitaxadb pyanidb
```

The new database can then be specified in other `pyani` subcommands, using the `--dbpath` option.

### 5.6.4 Running ANIm analysis

`pyani` implements average nucleotide identity analysis using MUMmer3 (ANIm) as defined in Richter & Rosselló-Móra (2009) (doi:10.1073/pnas.0906412106). To run ANIm on a set of input genomes, use the `pyani anim` subcommand.

In brief, the analysis proceeds as follows for a set of input prokaryotic genomes:

1. MUMmer3 is used to perform pairwise comparisons between each possible pair of input genomes, to identify homologous (alignable) regions.
2. For each comparison, the alignment output is parsed, and the following values are calculated:
   - total number of aligned bases on each genome
   - fraction of each genome that is aligned (the coverage)
   - the proportion of all aligned regions that is identical in each genome (the ANI)
   - the number of unaligned or non-identical bases (the similarity errors)
   - the product of coverage and ANI

The output values are recorded in the `pyani` database.

**Note:** A single MUMmer comparison is performed between each pair of genomes. Input genomes are sorted into alphabetical order by filename, and the query sequence is the genome that occurs earliest in the list; the subject sequence is the genome that occurs latest in the list.

**Tip:** The MUMmer comparisons are embarrassingly parallel, and can be distributed across cores on an Open Grid Scheduler-compatible cluster, using the `--scheduler SGE` option.
Attention: pyani anim requires that a working copy of MUMmer3 is available. Please see Installation Guide for information about installing this package.

For more information about the pyani anim subcommand, please see the pyani anim page, or issue the command pyani anim -h to see the inline help.

**Perform ANIm analysis**

The basic form of the command is:

```bash
pyani anim <INPUT_DIRECTORY> <OUTPUT_DIRECTORY>
```

This instructs pyani to perform ANIm on the genome FASTA files in `<INPUT_DIRECTORY>`, and write any output files to `<OUTPUT_DIRECTORY>`. For example, the following command performs ANIm on genomes in the directory genomes and writes output to a new directory genomes_ANIm:

```bash
pyani anim genomes genomes_ANIm
```

Note: While running, pyani anim will show progress bars unless these are disabled with the option --disable_tqdm.

This command will write the intermediate nucmer/MUMmer output to the directory genomes_ANIm, in a subdirectory called nucmer_output, where the results can be inspected if required.

```bash
$ ls genomes_ANIm/
nucmer_output
```

Attention: To view the output ANIm results, you will need to use the pyani report or pyani plot subcommands. Please see pyani report and pyani plot for more details.

**Perform ANIm analysis with Open Grid Scheduler**

The MUMmer comparison step of ANIm is embarrassingly parallel, and nucmer jobs can be distributed across cores in a cluster using the Open Grid Scheduler. To enable this during the analysis, use the --scheduler SGE option:

```bash
pyani anim --scheduler SGE genomes genomes_ANIm
```

Note: Jobs are submitted as array jobs to keep the scheduler queue short.

Note: If --scheduler SGE is not specified, all MUMmer jobs are run locally with Python's multiprocessing module.

**Controlling parameters of Open Grid Scheduler**

It is possible to control the following features of Open Grid Scheduler via the pyani anim subcommand:
• The array job size (by default, comparison jobs are batched in arrays of 10,000)
• The prefix string for the job, as reported in the scheduler queue
• Arguments to the qsub job submission command

These allow for useful control of job execution. For example, the command:

```
pyani anim --scheduler SGE --SGEgroupsize 5000 genomes genomes_ANIm
```

will batch MUMmer jobs in groups of 500 for the scheduler. The command:

```
pyani anim --scheduler SGE --jobprefix My_Ace_Job genomes genomes_ANIm
```

will prepend the string My_Ace_Job to your job in the scheduler queue. And the command:

```
pyani anim --scheduler SGE --SGEargs "-m e -M my.name@my.domain" 5000 genomes genomes_ANIm
```

will email my.name@my.domain when the jobs finish.

References


5.7 Examples

5.7.1 Using non-NCBI genomes

It is usual to want to include or work only with genomes that have been generated locally, or that were not downloaded from NCBI using `pyani download`. To use these genomes with the `pyani` analysis subcommands, the genomes must be indexed.

To index a set of genomes, use the `pyani index` subcommand on the input directory. To index the directory `mygenomes`, for example:

```
pyani index mygenomes
```

This will create a .md5 file (containing the hash) for each genome, as well as class and label files listing all the input genomes.

All genomes in the `mygenomes` directory will now be available for use in `pyani`.

Class and labels files

Using the `pyani download` command will create two files, by default `classes.txt` and `labels.txt` containing identifiers for each input genome that are used in later analysis and visualisation. These files are also created when `pyani index` is used as above.

The location of the labels and classes files may be changed using the `--labels` and `--classes` arguments, for example:

1 `indexing` here refers to constructing a hash of the genome: a short representation of the entire genome’s contents that can be used to identify it uniquely.

5.7. Examples
5.8 pyani subcommands

pyani has a subcommand structure, where the command pyani is followed immediately by a subcommand to let the software know which action you wish it to perform, in the format pyani <subcommand>. For example, to create a database you would use the createdb subcommand:

```
pyani createdb
```

This document links out to detailed instructions for each of the pyani subcommands.

5.8.1 pyani download

The download subcommand controls download of genome files from the NCBI Assembly database for input to pyani.

```
                --email EMAIL [--api_key API_KEYPATH]
                [--retries RETRIES] [--batchsize BATCHSIZE]
                [--timeout TIMEOUT] [-f] [--noclobber]
                [--labels LABELFNAME] [--classes CLASSFNAME]
                [--kraken] [--dry-run]
                outdir
```

**Positional arguments**

`outdir` The `outdir` argument should be the path to a directory into which genome files will be downloaded. If the directory exists, a warning will be given and the download will not proceed, to avoid overwriting existing data. To force writing into an existing directory, use the `-f` option.

**Flagged arguments**

```
--api_key PATH_TO_API_KEY The program will attempt to use an NCBI API key (see here) located at PATH_TO_API_KEY. Default: ~/.ncbi/api_key

--batchsize BATCHSIZE The download process will attempt to download assemblies in multiples of BATCHSIZE. Default: 10000

--classes CLASSFNAME Write a set of labels (one per downloaded genome) to the file CLASSFNAME in outdir. Default: classes.txt

--disable_tqdm Disable the tqdm progress bar while the download process runs. This is useful when testing to avoid aesthetic problems with test output.

--dry-run Perform all actions of the download process except for downloading files.

--email EMAIL COMPULSORY. Provide the email address EMAIL to NCBI so that they can track problems.

-f, --force Force use of the outdir directory when downloaded genomes, even if it already exists.

-h, --help Display usage information for pyani download.
```
--kraken Add taxonomy information to the FASTA file headers of downloaded genomes. This allows the genomes to be readily used to construct databases for the Kraken software package.

-1 LOGFILE, --logfile LOGFILE Provide the location LOGFILE to which a logfile of the download process will be written.

--labels LABELFNAME Write a set of labels (one per downloaded genome) to the file LABELFNAME in outdir. Default: labels.txt

--noclobber Do not overwrite individual files in the outdir directory, when used with -f.

--retries RETRIES The download process will attempt to download each batch of assemblies a maximum of RETRIES times. Default: 20

-t TAXON, --taxon TAXON COMPULSORY All genomes below taxon ID TAXON of a node in the NCBI Taxonomy database will be downloaded to the location specified by outdir.

--timeout TIMEOUT The download process will wait a maximum of TIMEOUT seconds before abandoning a URL connection attempt. Default: 10

-v, --verbose Provide verbose output to STDOUT

5.8.2 pyani index

The index subcommand will index the genome files it finds the passed directory indir, generating label and class files, and files that contain an MD5 hash of the nucleotide sequence of each genome.


Positional arguments

indir The indir argument should be the path to a directory containing genome sequence data as FASTA files (one per genome assembly).

Flagged arguments

--classes CLASSFNAME Write a set of labels (one per genome sequence file) to the file CLASSFNAME in indir. Default: classes.txt

--disable_tqdm Disable the tqdm progress bar while the download process runs. This is useful when testing to avoid aesthetic problems with test output.

-h, --help Display usage information for pyani index.

-1 LOGFILE, --logfile LOGFILE Provide the location LOGFILE to which a logfile of the download process will be written.

--labels LABELFNAME Write a set of labels (one per genome sequence file) to the file LABELFNAME in indir. Default: labels.txt

-v, --verbose Provide verbose output to STDOUT

5.8. pyani subcommands 49
5.8.3 pyani createdb

The `createdb` subcommand creates a new, empty database for `pyani` to use in subsequent analysis runs.

```
                      [--dbpath DBPATH] [-f]

Flagged arguments

--disable_tqdm  Disable the `tqdm` progress bar while the download process runs. This is useful when testing to avoid aesthetic problems with test output.

--dbpath DBPATH  Path to the location where the database will be created. Default: `.pyani/pyanidb`

-h, --help       Display usage information for `pyani` index.

-l LOGFILE, --logfile LOGFILE  Provide the location `LOGFILE` to which a logfile of the download process will be written.

-v, --verbose    Provide verbose output to `STDOUT`
```

5.8.4 pyani anim

The `anim` subcommand will carry out ANIm analysis using genome files contained in the `indir` directory, writing result files to the `outdir` directory, and recording data about each comparison and run in a local SQLite3 database.

```
                    [--scheduler {multiprocessing,SGE}] [--workers WORKERS]
                    [--SGEgroupsize SGEGROUPSIZE] [--SGEargs SGEARGS]
                    [--jobprefix JOBPREFIX] [--name NAME] [--classes CLASSES]
                    [--labels LABELS] [--recovery] [--dbpath DBPATH]
                    [--nucmer_exe NUCMER_EXE] [--filter_exe FILTER_EXE]
                    [--maxmatch] [--nofilter]
                    indir outdir

Positional arguments

indir  Path to the directory containing indexed genome files to be used for the analysis.

outdir  Path to a directory where comparison output files will be written.

Flagged arguments

--classes CLASSFNAME  Use the set of classes (one per genome sequence file) found in the file `CLASSFNAME` in `indir`. Default: `classes.txt`

--dbpath DBPATH  Path to the location of the local `pyani` database to be used. Default: `.pyani/pyanidb`

--disable_tqdm  Disable the `tqdm` progress bar while the download process runs. This is useful when testing to avoid aesthetic problems with test output.

--filter_exe FILTER_EXE  Path to the MUMmer `delta-filter` executable. Default: `delta-filter`

-h, --help       Display usage information for `pyani` index.
--jobprefix JOBPREFIX Use the string JOBPREFIX as a prefix for SGE job submission names. Default: PYANI

--labels LABELFNAME Use the set of labels (one per genome sequence file) found in the file LABELFNAME in indir. Default: labels.txt

--name NAME Use the string NAME to identify this ANIm run in the pyani database.

--nucmer_exe NUCMER Exe Path to the MUMmer nucmer executable. Default: nucmer

-l LOGFILE, --logfile LOGFILE Provide the location LOGFILE to which a logfile of the download process will be written.

--maxmatch Use the MUMmer --maxmatch option to include all nucmer matches.

--nofilter Do not use delta-filter to restrict nucmer output to 1:1 matches.

--recovery Use existing NUCmer comparison output if available, e.g. if recovering from a failed job submission. Using this option will not generate a new comparison if the old output files exist.

--scheduler {multiprocessing, SGE} Specify the job scheduler to be used when parallelising genome comparisons: one of multiprocessing (use many cores on the current machine) or SGE (use an SGE or OGE job scheduler). Default: multiprocessing.

--SGEargs SGEARGS Pass additional arguments SGEARGS to qsub when running the SGE-distributed jobs.

--SGEgroupsize SGEGROUPSIZE Create SGE arrays containing SGEGROUPSIZE comparison jobs. Default: 10000

-v, --verbose Provide verbose output to STDOUT

--workers WORKERS Spawn WORKERS worker processes with the --scheduler multiprocessing option. Default: 0 (use all cores)

5.8.5 pyani anib

5.8.6 pyani report

5.8.7 pyani plot

5.8.8 pyani classify

5.8.9 pyani listdeps

The listdeps subcommand writes an account of the local platform, the installed Python version, and installed dependencies (Python packages and third-party software tools) to STDOUT.

usage: pyani.py listdeps [-h] [-l LOGFILE] [-v]

Flagged arguments

--disable_tqdm Disable the tqdm progress bar while the download process runs. Does nothing.

-h, --help Display usage information for pyani listdeps.

-l LOGFILE, --logfile LOGFILE Provide the location LOGFILE to which a logfile of the subcommand output will be written.

-v, --verbose Provide verbose output to STDOUT. This actually provides no additional information.

5.8. pyani subcommands

51
5.9 Testing

We are currently writing tests formatted for the pytest package, for testing pyani.

Warning: Some tests are still targeted at nosetests, which is in maintenance mode and, although we are still in transition, our plan is to change the test framework completely to use pytest.

5.9.1 Test directory structure

All tests, supporting input and target data, are to be found in the tests/ subdirectory of the pyani repository.

- Input data for tests is located under the tests/test_input directory, in subdirectories named for the general operation that is being tested.
- Target data (known correct values) for tests is located under the tests/test_targets directory, in subdirectories named for the general operation being tested.
- Test output is written to the tests/test_output directory, in subdirectories named for the general operation being tested.

Tip: If you wish to write new tests for pyani, we ask that your test data and operations conform to this structure.

The tests/test_failing_data directory contains data that is known to cause problems for pyani in that at least two of the input genomes have no appreciable sequence identity.

5.9.2 Running tests

To run tests with nosetests, change directory to the root of the pyani repository, and invoke a nosetests command.

Run all tests locally

To run all tests locally on your machine, issue the following command from the repository root:

```
nosetests -v
```

This will cause nose to run all tests under the tests/ subdirectory.

Run individual tests

Tests are grouped in files with filenames that match test_*.py. We aim to write tests as classes that subclass unittest.TestCase, as described in the nosetests documentation. An example of this style can be found in the tests/test_anim.py test file.

This style allows us to run tests at several levels of granularity, specifying all tests (see above), all tests within a module (e.g. test_anim.py), or all tests within a class in that test file.

For example, to run all ANIm-related tests, we can issue:

```
nosetests -v tests/test_anim.py
```
To run all tests of `nucmer` command line generation, we can specify a single class within that file using the command:

```
nosetests -v tests/test_anim.py:TestNUCmerCmdline
```

And to test only “multiple command generation” we can issue the following:

```
nosetests -v tests/test_anim.py:TestNUCmerCmdline.test_multi_cmd_generation
```

## 5.10 Contributing to `pyani`

### 5.10.1 Reporting bugs and errors

If you find a bug, or an error in the code or documentation, please report this by raising an issue at the [GitHub issues page](https://github.com/pyani/pyani/issues) for `pyani`:

- [GitHub issues page](https://github.com/pyani/pyani/issues)

### 5.10.2 Contributing code or documentation

We gratefully accept code and other contributions. A list of contributors can be found via the [Github contributors link](https://github.com/pyani/pyani/graphs/contributors).

You are welcome to help develop `pyani`, fix a bug, improve documentation, or contribute in any other way. To make everyone’s lives easier in this process, we ask that you please follow the guidelines for developers below:

**Pre-commit checks and style guide**

So far as is possible, we aim to follow the coding conventions as described in PEP8 and PEP257, but we have adopted black code styling, which does vary from the PEPs in places.

We use the `flake8` tool for style checks, and this can be installed as follows (with two useful plugins):

```
pip install flake8 flake8-docstrings flake8-blind-except
```

`flake8` can then be run directly on the codebase with

```
flake8 bin/
flake8 pyani/
```

We use the `black` tool for code style checking, which can be installed with:

```
pip install black
```

The `flake8` and `black` styles can be enforced as pre-commit hooks using the `pre-commit` package (included in `requirements.txt`).

The `black` and `flake8` hooks are defined in `.pre-commit-config.yaml`; custom settings for `flake8` are held in `.flake8` (all files are under version control).

To enable pre-commit checks in the codebase on your local machine (once `pre-commit` has been installed), execute the following command in the root directory of this repository:

```
pre-commit install
```
Checking changes to the documentation

Much of the repository documentation is written in Markdown files, but the main documentation (which you are reading) is prepared for ReadTheDocs, which uses reStructuredText and Sphinx. The Sphinx configuration is described in `docs/conf.py` (under version control).

So long as Sphinx is installed on your machine, you can check your documentation changes locally, building inplace by changing to the `docs/` directory and issuing:

```
make html
```

This will place a compiled version of the documentation under `_build/html`, which you can inspect before committing to the repository.

Tip: To build the documentation in ReadTheDocs style, you will need to install the corresponding theme with `pip install sphinx_rtd_theme` or `conda install sphinx_rtd_theme`.

For now, docstrings in the source code are not required to be in any controlled syntax, such as reStructuredText, but this may change.

Making changes and pull requests

1. Fork the `pyani` repository under your account at GitHub.
2. Clone your fork to your development machine.
3. Create a new branch in your forked repository with an informative name like `fix_issue_107`, using `git checkout -b fix_issue_107`.
4. Make the changes you need and commit them to your local branch.
5. Run the repository tests (see the `Testing` documentation for more details).
6. If the tests all pass, push the changes to your fork, and submit a pull request against the original repository.
7. Indicate one of the `pyani` developers as an assignee to review your pull request when you submit your pull request.

The assigned developer will then review your pull request, and merge it or continue the conversation, as appropriate.

5.10.3 Suggestions for improvement

If you would like to make a suggestion for how we could improve `pyani`, we welcome contributions at the GitHub issues page.

5.11 Licensing

Unless otherwise indicated, all code is subject to the following agreement:

(c) The James Hutton Institute 2014-2019  (c) The University of Strathclyde 2019  Author: Leighton Pritchard
Contact: leighton.pritchard@strath.ac.uk
5.11.1 The MIT License

Copyright (c) 2014-2019 The James Hutton Institute Copyright (c) 2019 The University of Strathclyde

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

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

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

5.12 pyani package

Module with main code for pyani application/package.

```python
exception pyani.PyaniException
    Bases: Exception

    General exception for pyani.
```

5.12.1 Subpackages

pyani.scripts package

Modules to support use of pyani as a script.

```python
exception pyani.scripts.PyaniScriptException (msg='Error in pyani.py script')
    Bases: Exception

    General exception for pyani.py script.
```

```python
def pyani.scripts.make_outdir (outdir: pathlib.Path, force: bool, noclobber: bool) → None
    Create output directory (allows for force and noclobber).
```

Parameters

- **outdir** – Path, path to output directory
- **force** – bool, True if an existing directory will be reused
- **noclobber** – bool, True if existing files are not overwritten

The intended outcomes are: outdir doesn’t exist: create outdir outdir exists: raise exception outdir exists, --force only: remove the directory tree outdir exists, --force --noclobber: continue with existing directory tree
So long as the outdir is created with this function, we need only check for args.noclobber elsewhere to see how to proceed when a file exists.

Subpackages

pyani.scripts.parsers package

Module providing command-line parser definitions.

```
pyani.scripts.parsers.parse_cmdline(  
    argv: Optional[List[T]] = None) -> argparse.Namespace
```

Parse command-line arguments for script.

**Parameters**

- `argv` – Namespace, command-line arguments

The script offers a single main parser, with subcommands for the actions:

- **download**  download all available NCBI assemblies below the passed taxonomy ID
- **index**  index genome sequence files in a subdirectory, for analysis
- **createdb**  generate SQLite database for data and analysis results
- **anim**  conduct ANIm analysis
- **anib**  conduct ANIb analysis
- **aniblastall**  conduct ANIblastall analysis
- **report**  generate output describing analyses, genomes, and results
- **plot**  generate graphical output describing results
- **classify**  produce graph-based classification of genomes on the basis of ANI analysis

Submodules

pyani.scripts.parsers.anib_parser module

Provides parser for anib subcommand.

```
pyani.scripts.parsers.anib_parser.build(  
    subps: argparse._SubParsersAction,  
    parents: Optional[List[argparse.ArgumentParser]] = None) -> None
```

Return a command-line parser for the anib subcommand.

**Parameters**

- `subps` – collection of subparsers in main parser
- `parents` – parsers from which arguments are inherited

The terminology may be confusing, but in practice the main parser collects command-line arguments that are then available to this parser, which inherits options from the parsers in `parents` in addition to those defined below.

pyani.scripts.parsers.aniblastall_parser module

Provides parser for aniblastall subcommand.
Return a command-line parser for the aniblastall subcommand.

Parameters

- `subps` – collection of subparsers in main parser
- `parents` – parsers from which arguments are inherited

**pyani.scripts.parsers.anim_parser module**

Provides parser for anim subcommand.

Return a command-line parser for the anim subcommand.

Parameters

- `subps` – collection of subparsers in main parser
- `parents` – parsers from which arguments are inherited

**pyani.scripts.parsers.classify_parser module**

Provides parser for classify subcommand.

Return a command-line parser for the classify subcommand.

Parameters

- `subps` – collection of subparsers in main parser
- `parents` – parsers from which arguments are inherited

The classify subcommand takes specific arguments:

- `--cov_min` (minimum coverage threshold for an edge)
- `--id_min` (minimum identity threshold for an edge)
- `--resolution` (number of identity thresholds to test)

**pyani.scripts.parsers.common_parser module**

Provides parser for arguments common to all subcommands.

Return the common argument parser for all script subcommands.

Parameters

- `subps` – collection of subparsers in main parser
• **parents** – parsers from which arguments are inherited

Common arguments are:

- `-l`, –logfile (specify logfile output path)  
- `-v`, –verbose (produce verbose output on STDOUT)

---

### pyani.scripts.parsers.createdb_parser module

Provides parser for createdb subcommand.

```python
pyani.scripts.parsers.createdb_parser.build(subps: argparse._SubParsersAction, parents: Optional[List[argparse.ArgumentParser]] = None) → None
```

Return a command-line parser for the createdb subcommand.

**Parameters**

- **subps** – collection of subparsers in main parser
- **parents** – parsers from which arguments are inherited

### pyani.scripts.parsers.download_parser module

Provides parser for download subcommand.

```python
pyani.scripts.parsers.download_parser.build(subps: argparse._SubParsersAction, parents: Optional[List[argparse.ArgumentParser]] = None) → None
```

Return a command-line parser for the download subcommand.

**Parameters**

- **subps** – ArgumentParser.subparser
- **parents** – additional Parser objects

The download subcommand takes specific arguments:

- `-t`, –taxon (NCBI taxonomy IDs - comma-separated list, or one ID)  
- `-email` (email for providing to Entrez services)  
- `--api_key` (path to file containing personal API key for Entrez)  
- `--retries` (number of Entrez retry attempts to make)  
- `--batchsize` (number of Entrez records to download in a batch)  
- `--timeout` (how long to wait for Entrez query timeout)  
- `-f`, –force (allow existing directory overwrite)  
- `--noclobber` (don’t replace existing files)  
- `--labels` (path to write labels file)  
- `--classes` (path to write classes file)

### pyani.scripts.parsers.index_parser module

Provides parser for index subcommand.

```python
pyani.scripts.parsers.index_parser.build(subps: argparse._SubParsersAction, parents: Optional[List[argparse.ArgumentParser]] = None) → None
```

Return a command-line parser for the index subcommand.

**Parameters**

- **subps** – collection of subparsers in main parser
- **parents** – parsers from which arguments are inherited

The index subcommand takes a single positional argument:
• indir (directory containing input genome sequence files)

**pyani.scripts.parsers.plot_parser module**

Provides parser for plot subcommand.

```python
pyani.scripts.parsers.plot_parser.build(subps: argparse._SubParsersAction, parents: Optional[List[argparse.ArgumentParser]] = None) → None
```

Return a command-line parser for the plot subcommand.

**Parameters**

- **subps** – collection of subparsers in main parser
- **parents** – parsers from which arguments are inherited

The plot subcommand takes specific arguments:

- **--method** (graphics method to use)

**pyani.scripts.parsers.report_parser module**

Provides parser for report subcommand.

```python
pyani.scripts.parsers.report_parser.build(subps: argparse._SubParsersAction, parents: Optional[List[argparse.ArgumentParser]] = None) → None
```

Return a command-line parser for the report subcommand.

**Parameters**

- **subps** – collection of subparsers in main parser
- **parents** – parsers from which arguments are inherited

**pyani.scripts.parsers.run_common_parser module**

Provides parser for arguments common to analysis run subcommands.

```python
pyani.scripts.parsers.run_common_parser.build() → argparse.ArgumentParser
```

Return the common argument parser for analysis run subcommands.

**Parameters**

- **subps** – collection of subparsers in main parser
- **parents** – parsers from which arguments are inherited

Common arguments are:

- **--name** (human-readable name for the run)
- **--labels** (genome labels for this run)
- **--classes** (genome classes for this run)
pyani Documentation, Release 0.2.9

pyani.scripts.parsers.scheduling_parser module

Provides parser for arguments common to job scheduling.

```python
pyani.scripts.parsers.scheduling_parser.build() → argparse.ArgumentParser
```

Return the common argument parser for job scheduling.

**Parameters**

- `subps` – collection of subparsers in main parser
- `parents` – parsers from which arguments are inherited

Common arguments are:

pyani.scripts.subcommands package

Module providing subcommands for pyani scripts.

Submodules

pyani.scripts.subcommands.subcmd_anib module

Provides the anib subcommand for pyani.

```python
```

Return path to fragmented sequence file and JSON of fragment lengths.

**Parameters**

- `inpath` – Path to genome file
- `outdir` – Path to directory to hold fragmented files
- `fragsize` – size of genome fragments

Returns a tuple of `(path, json)` where `path` is the path to the fragment file and `json` is a JSON-ified dictionary of fragment lengths, keyed by fragment sequence ID.

```python
```

Return list of ComparisonJobs.

**Parameters**

- `comparisons` – list of (Genome, Genome) tuples for which comparisons are needed
- `existingfiles` – list of pre-existing BLASTN+ outputs
- `fragfiles` –
- `fraglens` –
- `args` – Namespace, command-line arguments
Perform ANIb on all genome files in an input directory.

**Parameters**

- `args` – Namespace, command-line arguments


All FASTA format files (selected by suffix) in the input directory are fragmented into (by default 1020nt) consecutive sections, and a BLAST+ database constructed from the whole genome input. The BLAST+ blastn tool is then used to query each set of fragments against each BLAST+ database, in turn.

For each query, the BLAST+.tab output is parsed to obtain alignment length, identity and similarity error count. Alignments below a threshold are not included in the calculation (this introduces systematic bias with respect to ANIm). The results are processed to calculate the ANI percentages, coverage, and similarity error.

The calculated values are stored in the local SQLite3 database.

**pyani.scripts.subcommands.subcmd_aniblastall module**

Provides the aniblastall subcommand for pyani.

Perform ANIblastall on all genome files in an input directory.

**Parameters**

- `args` –
- `logger` –

**pyani.scripts.subcommands.subcmd_anim module**

Provides the anim subcommand for pyani.

**class**

**pyani.scripts.subcommands.subcmd_anim.ComparisonJob**

-Bases: tuple

Pairwise comparison job for the SQLAlchemy implementation.

- `filtercmd` – Alias for field number 2
- `job` – Alias for field number 5
- `nucmercmd` – Alias for field number 3
- `outfile` – Alias for field number 4
- `query` – Alias for field number 0
- `subject` – Alias for field number 1
class pyani.scripts.subcommands.subcmd_anim.ComparisonResult
    Bases: tuple
    Convenience struct for a single nucmer comparison result.
    aln_length
        Alias for field number 2
    pid
        Alias for field number 4
    qcov
        Alias for field number 7
    qid
        Alias for field number 0
    qlen
        Alias for field number 5
    scov
        Alias for field number 8
    sid
        Alias for field number 1
    sim_errs
        Alias for field number 3
    slen
        Alias for field number 6

class pyani.scripts.subcommands.subcmd_anim.ProgData
    Bases: tuple
    Convenience struct for comparison program data/info.
    program
        Alias for field number 0
    version
        Alias for field number 1

class pyani.scripts.subcommands.subcmd_anim.ProgParams
    Bases: tuple
    Convenience struct for comparison parameters.
    Use default of zero for fragsize or else db queries will not work as SQLite/Python nulls do not match up well
    fragsize
        Alias for field number 0
    maxmatch
        Alias for field number 1

class pyani.scripts.subcommands.subcmd_anim.RunData
    Bases: tuple
    Convenience struct describing an analysis run.
    cmdline
        Alias for field number 3
**date**
Alias for field number 2

**method**
Alias for field number 0

**name**
Alias for field number 1

```python
class generate_joblist:
    def __init__(self, comparisons: List[Tuple], existingfiles: List[pathlib.Path], args: argparse.Namespace):
        self.comparisons = comparisons
        self.existingfiles = existingfiles
        self.args = args

    def generate(self) -> List[ComparisonJob]:
        # Generate joblist
```

Return list of ComparisonJobs.

**Parameters**

- **comparisons** – list of (Genome, Genome) tuples
- **existingfiles** – list of pre-existing nucmer output files
- **args** – Namespace of command-line arguments for the run

```python
class run_anim_jobs:
    def __init__(self, joblist: List[ComparisonJob], args: argparse.Namespace):
        self.joblist = joblist
        self.args = args

    def run(self) -> None:
        # Run ANIm nucmer jobs
```

Pass ANIm nucmer jobs to the scheduler.

**Parameters**

- **joblist** – list of ComparisonJob namedtuples
- **args** – command-line arguments for the run

```python
class subcmdAnim:
    def __init__(self, args: argparse.Namespace):
        self.args = args

    def perform(self) -> None:
        # Perform ANIm on all genome files in an input directory
```

Perform ANIm on all genome files in an input directory.

**Parameters**

- **args** – Namespace, command-line arguments


All FASTA format files (selected by suffix) in the input directory are compared against each other, pairwise, using NUCmer (whose path must be provided).

For each pairwise comparison, the NUCmer .delta file output is parsed to obtain an alignment length and similarity error count for every unique region alignment between the two organisms, as represented by sequences in the FASTA files. These are processed to calculated aligned sequence lengths, average nucleotide identity (ANI) percentages, coverage (aligned percentage of whole genome - forward direction), and similarity error count for each pairwise comparison.

The calculated values are deposited in the SQLite3 database being used for the analysis.

For each pairwise comparison the NUCmer output is stored in the output directory for long enough to extract summary information, but for each run the output is gzip compressed. Once all runs are complete, the outputs for each comparison are concatenated into a single gzip archive.
Update the Comparison table with the completed result set.

**Parameters**

- `joblist` – list of ComparisonJob namedtuples
- `run` – Run ORM object for the current ANIm run
- `session` – active pyanidb session via ORM
- `nucmer_version` – version of nucmer used for the comparison
- `args` – command-line arguments for this run

The Comparison table stores individual comparison results, one per row.

**pyani.scripts.subcommands.subcmd_classify module**

Provides the classify subcommand for pyani.

```python
class pyani.scripts.subcommands.subcmd_classify.SubgraphData
    Bases: tuple

    Subgraph clique/classification output.

    cliqueinfo
        Alias for field number 2

    graph
        Alias for field number 1

    interval
        Alias for field number 0
```

Generate classifications for an analysis.

**Parameters**

- `args` – Namespace, command-line arguments

```python
pyani.scripts.subcommands.subcmd_classify.subcmd_classify(args: arg-parse.Namespace) → int
```

Return graphs trimmed from lowest to highest attribute value.

**Parameters**
• `ingraph` – `nx.Graph` of genomes as nodes, having edges weighted by the property named in attribute
• `args` – Namespace, parsed command-line arguments
• `attribute` – str, name of the property by which the graph edges should be trimmed

A generator which, starting from the initial graph, yields in sequence a series of graphs from which the edge(s) with the lowest threshold value attribute were removed. The generator returns a tuple of:

(threshold, graph, analyse_cliques(graph))

This will be slow with moderate-large graphs

**pyani.scripts.subcommands.subcmd_createdb module**

Provides the createdb subcommand for pyani.

```python
pyani.scripts.subcommands.subcmd_createdb.subcmd_createdb(args: argparse.Namespace) → int
Create an empty pyani database.
```

**Parameters**

• `args` – Namespace, command-line arguments
• `logger` – logging object

**pyani.scripts.subcommands.subcmd_download module**

Provides the download subcommand for pyani.

```python
class pyani.scripts.subcommands.subcmd_download.Skipped
Bases: tuple
Convenience struct for holding information about skipped genomes.

accession
Alias for field number 1

dltype
Alias for field number 5

organism
Alias for field number 2

strain
Alias for field number 3

taxon_id
Alias for field number 0

url
Alias for field number 4
```

```python
pyani.scripts.subcommands.subcmd_download.configure_entrez(args: argparse.Namespace) → Optional[str]
Configure Entrez email, return API key.
```

**Parameters** `args` – Namespace, command-line arguments
Returns None if no API key found

```
pyani.scripts.subcommands.subcmd_download.dl_info_to_str(esummary, uid_class) → str
```

Return descriptive string for passed download data.

Parameters

- `esummary`
- `uid_class`

```
pyani.scripts.subcommands.subcmd_download.download_data(args: argparse.Namespace, api_key: Optional[str], asm_dict: Dict[str, List[T]]) → Tuple[List[T], List[T], List[T]]
```

Download the accessions indicated in the passed dictionary.

Parameters

- `args` – Namespace of command-line arguments
- `api_key` – str, API key for NCBI downloads
- `asm_dict` – dictionary of assembly UIDs to download, keyed by taxID

Returns lists of information about downloaded genome classes and labels, and a list of skipped downloads (as Skipped objects).

```
pyani.scripts.subcommands.subcmd_download.download_genome(args: argparse.Namespace, filestem: str, tid: str, uid: str, uid_class)
```

Download single genome data to output directory.

Parameters

- `args` – Namespace, command-line arguments
- `filestem` – str, output filestem
- `tid` – str, taxonID
- `uid` – str, assembly UID
- `uid_class`

```
pyani.scripts.subcommands.subcmd_download.extract_genomes(args: argparse.Namespace, dlstatus: pyani.download.DLStatus, esummary)
```

Extract genome files in passed dlstatus.

Parameters

- `args` – Namespace of command-line arguments
- `dlstatus`
- `esummary`
Return dictionary of assembly UIDs to download, keyed by taxID.

**Parameters**

- **args** – Namespace of command-line arguments

Hash genome files in passed dlstatus.

**Parameters**

- **args** – Namespace of command-line arguments
- **dlstatus** –
- **filestem** – str, filestem for output
- **uid_class** –

Returns NCBI API key if present, None otherwise.

**Parameters**

- **args** – Namespace of command-line arguments

Download assembled genomes in subtree of passed NCBI taxon ID.

**Parameters**

- **args** – Namespace, command-line arguments

Provides the index subcommand for pyani.

**Parameters**

- **args** – Namespace, received command-line arguments
- **logger** – logging object

Identify the genome files in the input directory, and generate a single MD5 for each so that <genome>.fna produces <genome>.md5

Genome files (FASTA) are identified from the file extension.

Provides the plot subcommand for pyani.
pyani.scripts.subcommands.subcmd_plot.subcmd_plot(args: argparse.Namespace) → int

Produce graphical output for an analysis.

Parameters **args** – Namespace of command-line arguments

This is graphical output for representing the ANI analysis results, and takes the form of a heatmap, or heatmap with dendrogram.


Write distribution plots for each matrix type.

Parameters
- **run_id** – int, run_id for this run
- **matdata** – MatrixData object for this distribution plot
- **args** – Namespace for command-line arguments
- **outfmts** – list of output formats for files

pyani.scripts.subcommands.subcmd_plot.write_heatmap(run_id: int, matdata: pyani.pyani_tools.MatrixData, result_labels: Dict[KT, VT], result_classes: Dict[KT, VT], outfmts: List[str], args: argparse.Namespace) → None

Write a single heatmap for a pyani run.

Parameters
- **run_id** – int, run_id for this run
- **matdata** – MatrixData object for this heatmap
- **result_labels** – dict of result labels
- **result_classes** – dict of result classes
- **args** – Namespace for command-line arguments
- **outfmts** – list of output formats for files

pyani.scripts.subcommands.subcmd_plot.write_run_heatmaps(run_id: int, session, outfmts: List[str], args: argparse.Namespace) → None

Write all heatmaps for a specified run to file.

Parameters
- **run_id** – int, run identifier in database session
- **session** – Session, active SQLite session
- **outfmts** – list of output format types
- **args** – Namespace, command line arguments
**pyani.scripts.subcommands.subcmd_report module**

Provides the report subcommand for pyani.

```python
class pyani.scripts.subcommands.subcmd_report.ReportParams
    Bases: tuple
    Report query/header data.
    headers
        Alias for field number 2
    name
        Alias for field number 0
    statement
        Alias for field number 1
```

```python
```

Return processed list of output formats for writing reports.

**Parameters**

- **args** – Namespace of command-line arguments

```python
```

Write tabular report of pyani runs from database.

**Parameters**

- **args** – Namespace of command-line arguments
- **session** – SQLAlchemy database session
- **formats** – list of output formats
- **params** – ReportParams namedtuple

```python
pyani.scripts.subcommands.subcmd_report.subcmd_report(args: argparse.Namespace) -> int
```

Present report on ANI results and/or database contents.

**Parameters**

- **args** – Namespace, command-line arguments

The report subcommand takes any of several long options that do one of two things:

1. perform a single action.
2. set a parameter/format

These will typically take an output path to a file or directory into which the report will be written (whatever form it takes). By default, text output is written in plain text format, but for some outputs this can be modified by an ‘excel’ or ‘html’ format specifier, which writes outputs in that format, where possible.

**Submodules**

**pyani.scripts.average_nucleotide_identity module**

Script that calculates ANI measures for a directory of genomes.
This script calculates Average Nucleotide Identity (ANI) according to one of a number of alternative methods described in, e.g.


ANI is proposed to be the appropriate in silico substitute for DNA-DNA hybridisation (DDH), and so useful for delineating species boundaries. A typical percentage threshold for species boundary in the literature is 95% ANI (e.g. Richter et al. 2009).

All ANI methods follow the basic algorithm:

- Align the genome of organism 1 against that of organism 2, and identify the matching regions
- Calculate the percentage nucleotide identity of the matching regions, as an average for all matching regions

Methods differ on: (1) what alignment algorithm is used, and the choice of parameters (this affects the aligned region boundaries); (2) what the input is for alignment (typically either fragments of fixed size, or the most complete assembly available); (3) whether a reciprocal comparison is necessary or desirable.

ANIm: uses MUMmer (NUCmer) to align the input sequences. ANIb: uses BLASTN to align 1000nt fragments of the input sequences

TETRA: calculates tetranucleotide frequencies of each input sequence

This script takes as main input a directory containing a set of correctly-formatted FASTA multiple sequence files. All sequences for a single organism should be contained in only one sequence file. The names of these files are used for identification, so it would be advisable to name them sensibly.

Output is written to a named directory. The output files differ depending on the chosen ANI method.

**ANIm:** MUMmer/NUCmer .delta files, describing the sequence alignment; tab-separated format plain text tables describing total alignment lengths, and total alignment percentage identity

**ANIb:** FASTA sequences describing 1000nt fragments of each input sequence; BLAST nucleotide databases - one for each set of fragments; and BLASTN output files (tab-separated tabular format plain text) - one for each pairwise comparison of input sequences. There are potentially a lot of intermediate files.

**TETRA:** Tab-separated text file describing the Z-scores for each tetranucleotide in each input sequence.

In addition, all methods produce a table of output percentage identity (ANIm and ANIb) or correlation (TETRA), between each sequence.

If graphical output is chosen, the output directory will also contain PDF files representing the similarity between sequences as a heatmap with row and column dendrograms.

**DEPENDENCIES**

- Biopython (http://www.biopython.org)
- BLAST+ executable in the $PATH, or available on the command line (ANIb) (ftp://ftp.ncbi.nlm.nih.gov/blast/executables/blast+/LATEST/)
- MUMmer executables in the $PATH, or available on the command line (ANIm) (http://mummer.sourceforge.net/)
For graphical output

- R with shared libraries installed on the system, for graphical output ([http://cran.r-project.org/](http://cran.r-project.org/))
- Rpy2 ([http://rpy.sourceforge.net/rpy2.html](http://rpy.sourceforge.net/rpy2.html))

```python
pyani.scripts.average_nucleotide_identity.calculate_anim(args: argparse.Namespace, infiles: List[pathlib.Path], org_lengths: Dict[KT, VT]) -> pyani.pyani_tools.ANIResults
```

Returns ANIm result dataframes for files in input directory.

**Parameters**

- `args` – Namespace, command-line arguments
- `logger` – logging object
- `infiles` – list of paths to each input file
- `org_lengths` – dict, input sequence lengths, keyed by sequence


All FASTA format files (selected by suffix) in the input directory are compared against each other, pairwise, using NUCmer (which must be in the path). NUCmer output is stored in the output directory.

The NUCmer .delta file output is parsed to obtain an alignment length and similarity error count for every unique region alignment between the two organisms, as represented by the sequences in the FASTA files.

These are processed to give matrices of aligned sequence lengths, average nucleotide identity (ANI) percentages, coverage (aligned percentage of whole genome), and similarity error count for each pairwise comparison.

```python
pyani.scripts.average_nucleotide_identity.calculate_tetra(infiles: List[pathlib.Path]) -> pandas.core.frame.DataFrame
```

Calculate TETRA for files in input directory.

**Parameters**

- `logger` – logging object
- `infiles` – list, paths to each input file

Calculates TETRA correlation scores, as described in:


and


```python
pyani.scripts.average_nucleotide_identity.compress_delete_outdir(outdir: pathlib.Path, logger: logging.Logger) -> None
```

Compress the contents of the passed directory to .tar.gz and delete.
pyani.scripts.average_nucleotide_identity.draw(args: argparse.Namespace, filestems: List[str], gformat: str) → None

Draw ANIb/ANIm/TETRA results.

**Parameters**

- **args** – Namespace, command-line arguments
- **logger** – logging object
- **filestems** –
  - filestems for output files
- **gformat** –
  - the format for output graphics

pyani.scripts.average_nucleotide_identity.get_method(args: argparse.Namespace) → Tuple

Return function and config for the chosen method.

**Parameters**

- **args** – Namespace of command-line arguments
- **logger** – logging object

The dictionary defines pairs of method function and configurations, keyed by method name.

pyani.scripts.average_nucleotide_identity.last_exception() → str

Return last exception as a string, or use in logging.

pyani.scripts.average_nucleotide_identity.make_outdirs(args: argparse.Namespace) →

Make the output directory, if required.

**Parameters**

- **args** – Namespace of command-line options
- **logger** – logging object

If the output directory already exists and args.force is not set True, stop with an error.

**If args.force is set...** If args.noclobber is not set True, delete the output directory tree If args.noclobber is set True, use the existing output directory, and keep any existing output


Return tuple of fragment files, and fragment sizes.

**Parameters**

- **args** – Namespace of command-line arguments
- **logger** – logging object
- **infiles** – iterable of sequence files to fragment
- **blastdir** – path of directory to place BLASTN databases of fragments

Splits input FASTA sequence files into the fragments (a requirement for ANIb methods), and writes BLAST databases of these fragments, and fragment lengths of sequences, to local files.

```python
pyani.scripts.average_nucleotide_identity.parse_cmdline(argv: Optional[List[T]] = None) → argparse.Namespace
```
Parse command-line arguments for script.

**Parameters**
- **argv** – list of arguments from command-line

```python
pyani.scripts.average_nucleotide_identity.process_arguments(args: Optional[argparse.Namespace]) → argparse.Namespace
```
Process command-line arguments.

**Parameters**
- **args** – Namespace of command-line arguments

Either returns parsed arguments or - if only the script name is used, shows the version and exits.

```python
```
Run BLAST commands for ANIb methods.

**Parameters**
- **args** – Namespace of command-line options
- **logger** – logging object
- **infiles** – iterable of sequence files to compare
- **blastdir** – path of directory to fragment BLASTN databases

Runs BLAST database creation and comparisons, returning the cumulative return values of the BLAST tool subprocesses, and the fragment sizes for each input file.

```python
pyani.scripts.average_nucleotide_identity.run_main(argsin: Optional[argparse.Namespace] = None) → int
```
Run main process for `average_nucleotide_identity.py` script.

**Parameters**
- **argsin** – Namespace, command-line arguments
- **logger** – logging object

```python
```
Return a random subsample of the passed input files.

**Parameters**
• **args** – Namespace, command-line arguments
• **logger** – logging object
• **infiles** – list of input files for analysis

```python
pyani.scripts.average_nucleotide_identity.test_class_label_paths(args: argparse.Namespace,
logger: logging.Logger) → None
```

Raise error and exit if label and class files exist.

**Parameters**

- **args** – Namespace of command-line arguments
- **logger** – logging object

Exits if class and label files are not found

```python
pyani.scripts.average_nucleotide_identity.test_scheduler(args: argparse.Namespace,
logger: logging.Logger) → None
```

Test if the specified scheduler can be used.

**Parameters**

- **args** – Namespace of command-line arguments
- **logger** – logging object

Exits if the scheduler is invalid

```python
pyani.scripts.average_nucleotide_identity.unified_anib(args: argparse.Namespace,
infiles: List[pathlib.Path],
org_lengths: Dict[str, int]) →
pyani.pyani_tools.ANIResults
```

Calculate ANIb for files in input directory.

**Parameters**

- **args** – Namespace of command-line options
- **logger** – logging object
- **infiles** – iterable of paths to each input file
- **org_lengths** – dict of input sequence lengths keyed by sequence name

Calculates ANI by the ANIb method, as described in Goris et al. (2007) Int J Syst Evol Micr 57: 81-91. doi:10.1099/ijs.0.64483-0. There are some minor differences depending on whether BLAST+ or legacy BLAST (BLASTALL) methods are used.

All FASTA format files (selected by suffix) in the input directory are used to construct BLAST databases, placed in the output directory. Each file’s contents are also split into sequence fragments of length options.fragsize, and the multiple FASTA file that results written to the output directory. These are BLASTNed, pairwise, against the databases.

The BLAST output is interrogated for all fragment matches that cover at least 70% of the query sequence, with at least 30% nucleotide identity over the full length of the query sequence. This is an odd choice and doesn’t correspond to the twilight zone limit as implied by Goris et al. We persist with their definition, however.
Only these qualifying matches contribute to the total aligned length, and total aligned sequence identity used to
calculate ANI.

The results are processed to give matrices of aligned sequence length (aln_lengths.tab), similarity error counts
(sim_errors.tab), ANIs (perc_ids.tab), and minimum aligned percentage (perc_aln.tab) of each genome, for each
pairwise comparison. These are written to the output directory in plain text tab-separated format.

```python
pyani.scripts.average_nucleotide_identity.write(args: argparse.Namespace, results: pd.DataFrame) -> None
```

Write ANIb/ANIm/TETRA results to output directory.

**Parameters**

- `args` – Namespace, command-line arguments
- `logger` – logging object
- `results` – Results object from analysis

Each dataframe is written to an Excel-format file (if args.write_excel is True), and plain text tab-separated file
in the output directory. The order of result output must be reflected in the order of filestems.

**pyani.scripts.delta_filter_wrapper module**

Wrapper for MUMmer 3.23 delta-filter script.

It is required in order to catch STDOUT ahead of the SGE job runner so that pyani can run on SGE/OGE scheduling
systems.

The wrapper does not modify the output of delta-filter, and is called in exactly the same way, passing arguments
through directly. The only departure from this is that the first argument is the path to delta-filter, and the final argument
denotes the output filtered delta file path, so that redirection (which no longer works with SGE/OGE) is not necessary.

For example, the delta-filter command

```
delta-filter [options] <delta file> > <filtered delta file>
```

becomes

```
delta_filter_wrapper.py delta-filter [options] <delta file> <filtered delta file>
```

This wrapper is not very robust, but will be improved in later versions of pyani.

```python
pyani.scripts.delta_filter_wrapper.run_main() -> int
```

Run main process for delta_filter_wrapper.py.

**pyani.scripts.genbank_get_genomes_by_taxon module**

Script to download from NCBI all genomes in a specified taxon subtree.

This script takes an NCBI taxonomy identifier (or string, though this is not always reliable for taxonomy tree sub-
graphs... ) and downloads all genomes it can find from NCBI in the corresponding taxon subgraph that has the passed
argument as root.

```python
exception pyani.scripts.genbank_get_genomes_by_taxon.NCBIDownloadException
    bases: Exception
```

General exception for failed NCBI download.
pyani.scripts.genbank_get_genomes_by_taxon.entrez_batch_webhistory(args, record, expected, batchsize, *fnargs, **fnkwargs)

Recover Entrez data from a prior NCBI webhistory search.

Parameters
- **args** – Namespace, command-line arguments
- **record** – Entrez webhistory record
- **expected** – int, number of expected search returns
- **batchsize** – int, number of search returns to retrieve in each batch
- ***fnargs** – tuple, arguments to Efetch
- ****fnkwargs** – dict, keyword arguments to Efetch

Recovery is performed in in batches of defined size, using Efetch. Returns all results as a list.

pyani.scripts.genbank_get_genomes_by_taxon.entrez_retry(args, func, *fnargs, **fnkwargs)

Retry the passed function a defined number of times.

Parameters
- **args** – Namespace, command-line arguments
- **func** – func, Entrez function to attempt
- ***fnargs** – tuple, arguments to the Entrez function
- ****fnkwargs** – dict, keyword arguments to the Entrez function

pyani.scripts.genbank_get_genomes_by_taxon.extract_archive(archivepath)

Return path to extracted gzip file.

Parameters **archivepath** – Path, path to gzipped file with “.tar.gz” suffix

pyani.scripts.genbank_get_genomes_by_taxon.extract_filestem(data)

Extract filestem from Entrez eSummary data.

Parameters **data** – Entrez eSummary

Function expects esummary[‘DocumentSummarySet’][‘DocumentSummary’][0]

Some illegal characters may occur in AssemblyName - for these, a more robust regex replace/escape may be required. Sadly, NCBI don’t just use standard percent escapes, but instead replace certain characters with underscores: white space, slash, comma, hash, brackets.

pyani.scripts.genbank_get_genomes_by_taxon.get_asm_uids(args, taxon_uid)

Return a set of NCBI UIDs associated with the passed taxon.

Parameters
- **args** – Namespace, command-line arguments
- **taxon_uid** – str, NCBI taxon ID

This query at NCBI returns all assemblies for the taxon subtree rooted at the passed taxon_uid.
Return the NCBI AssemblyAccession and AssemblyName for an assembly.

**Parameters**

- **args** – Namespace, command-line arguments
- **asm_uid** – NCBI assembly UID
- **fmt** – str, format to retrieve assembly information

Returns organism data for class/label files also, as well as accession, so we can track whether downloads fail because only the most recent version is available.

AssemblyAccession and AssemblyName are data fields in the eSummary record, and correspond to downloadable files for each assembly at ftp://ftp.ncbi.nlm.nih.gov/genomes/all/GC[[]AF[]]/nnn/nnn/nnn/<AA>_<AN> where <AA> is AssemblyAccession, and <AN> is AssemblyName, and the choice of GCA vs GCF, and the three values of nnn are taken from <AA>

Return last exception as a string, or use in logging.

Report to logger if alternative assemblies were downloaded.

Make the output directory, if required.

Parse command-line arguments.

Parse command-line arguments.

Download assembly sequence to a local directory.

**Parameters**

- **args** – Namespace, command-line arguments
- **filestem** – str, filestem for output file
• **ftpstem** – str, URI stem for NCBI FTP site
• **fmt** – str, format for output file

The filestem corresponds to `<AA>_<AN>`, where `<AA>` and `<AN>` are AssemblyAccession and AssemblyName: data fields in the eSummary record. These correspond to downloadable files for each assembly at ftp://ftp.ncbi.nlm.nih.gov.genomes/all/GC{[}A{]}F{[}]_/nnn/nnn/nnn/<AA>_<AN>/ where `<AA>` is AssemblyAccession, and `<AN>` is AssemblyName. The choice of GCA vs GCF, and the values of nnn, are derived from `<AA>`.

The files in this directory all have the stem `<AA>_<AN>_<suffix>`, where suffixes are: assembly_report.txt assembly_stats.txt feature_table.txt.gz genomic.fna.gz genomic.gbff.gz genomic.gff.gz protein.faa.gz protein.gpff.gz rm_out.gz rm.run wgsmaster.gbff.gz

This function downloads the genomic_fna.gz file, and extracts it in the output directory name specified when the script is called.

```python
pyani.scripts.genbank_get_genomes_by_taxon.run_main(args=None)
```
Run main process for average_nucleotide_identity.py script.

**Parameters**
- **args** – Namespace, command-line arguments

```python
pyani.scripts.genbank_get_genomes_by_taxon.set_ncbi_email(args: argparse.Namespace) → None
```
Set contact email for NCBI.

**Parameters**
- **args** – Namespace, command-line arguments

```python
pyani.scripts.genbank_get_genomes_by_taxon.write_contigs(args, asm_uid, contig_uids, batchsize=10000)
```
Write assembly contigs to a single FASTA file.

**Parameters**
- **args** – Namespace, command-line arguments
- **asm_uid** – str, NCBI assembly UID
- **contig_uids** –
- **batchsize** – int

FASTA records are returned, as GenBank and even GenBankWithParts format records don’t reliably give correct sequence in all cases.

The script returns two strings for each assembly, a ‘class’ and a ‘label’ string - this is for use with, e.g. pyani.

**pyani.scripts.logger module**

**pyani.scripts.pyani_script module**

Implements the pyani script for classifying prokaryotic genomes.

```python
pyani.scripts.pyani_script.add_log_headers()
```
Add headers to log output.

```python
pyani.scripts.pyani_script.run_main(argv: Optional[List[str]] = None) → int
```
Run main process for pyani.py script.

**Parameters**
- **argv** –
pyani.scripts.tools module

5.12.2 Submodules

pyani.anib module

Code to implement the ANIb average nucleotide identity method.

Calculates ANI by the ANIb method, as described in Goris et al. (2007) Int J Syst Evol Micr 57: 81-91. doi:10.1099/ijs.0.64483-0.

From Goris et al.

"...The genomic sequence from one of the genomes in a pair (the query) was cut into consecutive 1020 nt fragments. The 1020 nt cut-off was used to correspond with the fragmentation of the genomic DNA to approximately 1 kb fragments during the DDH experiments. [...] The 1020 nt fragments were then used to search against the whole genomic sequence of the other genome in the pair (the reference) by using the BLASTN algorithm; the best BLASTN match was saved for further analysis. The BLAST algorithm was run using the following settings: X=150 (where X is the drop-off value for gapped alignment), q=-1 (where q is the penalty for nucleotide mismatch) and F=F (where F is the filter for repeated sequences); the rest of the parameters were used at the default settings. These settings give better sensitivity than the default settings when more distantly related genomes are being compared, as the latter target sequences that are more similar to each other. [...] The ANI between the query genome and the reference genome was calculated as the mean identity of all BLASTN matches that showed more than 30% overall sequence identity (recalculated to an identity along the entire sequence) over an alignable region of at least 70% of their length. This cut-off is above the ‘twilight zone’ of similarity searches in which an inference of homology is error prone because of low levels of Reverse searching, i.e. in which the reference genome is used as the query, was also performed to provide reciprocal values."

All input FASTA format files are used to construct BLAST databases. Each file’s contents are also split into sequence fragments of length options.fragsize, and the multiple FASTA file that results written to the output directory. These are BLASTNed, pairwise, against the databases.

BLAST output is interrogated for all fragment matches that cover at least 70% of the query sequence, with at least 30% nucleotide identity over the full length of the query sequence. This is an odd choice and doesn’t correspond to the twilight zone limit as implied by Goris et al. We persist with their definition, however. Only these qualifying matches contribute to the total aligned length, and total aligned sequence identity used to calculate ANI.

pyani.anib.build_db_jobs(infiles: List[pathlib.Path], blastcmds: pyani.pyani_tools.BLASTcmds) → Dict[KT, VT]

Return dictionary of db-building commands, keyed by dbname.

Parameters

• infiles –
• blastcmds –


Return single blastall command.

Parameters

• fname1 –
• fname2 –
• outdir –
• blastall_exe – str, path to BLASTALL executable

Return a single blastn command.

Parameters
- **fname1** –
- **fname2** –
- **outdir** –
- **blastn_exe** – str, path to blastn executable


Return formatdb command and path to output file.

Parameters
- **filename** – Path, input filename
- **outdir** – Path, path to output directory
- **blastdb_exe** – Path, path to the formatdb executable


Return makeblastdb command and path to output file.

Parameters
- **filename** – Path, input filename
- **outdir** – Path, directory for output
- **blastdb_exe** – Path, path to the makeblastdb executable


Chop sequences of the passed files into fragments, return filenames.

Parameters
- **infiles** – collection of paths to each input sequence file
- **outdirname** – Path, path to output directory
- **fragsize** – Int, the size of sequence fragments

Takes every sequence from every file in infiles, and splits them into consecutive fragments of length fragsize, (with any trailing sequences being included, even if shorter than fragsize), writing the resulting set of sequences to a file with the same name in the specified output directory.

All fragments are named consecutively and uniquely (within a file) as fragNNNNNN. Sequence description fields are retained.

Returns a tuple (filenames, fragment_lengths) where filenames is a list of paths to the fragment sequence files, and fragment_lengths is a dictionary of sequence fragment lengths, keyed by the sequence files, with values being a dictionary of fragment lengths, keyed by fragment IDs.

Return list of makeblastdb command-lines for ANIb/ANIblastall.

Parameters

- **filenames** – a list of paths to input FASTA files
- **outdir** – path to output directory
- **blastdb_exe** – path to the makeblastdb executable
- **mode** – str, ANIb analysis type (ANIb or ANIblastall)


Return a list of blastn command-lines for ANIm.

Parameters

- **filenames** – a list of paths to fragmented input FASTA files
- **outdir** – path to output directory
- **blastn_exe** – path to BLASTN executable
- **mode** – str, analysis type (ANIb or ANIblastall)

Assumes that the fragment sequence input filenames have the form ACCESSION-fragments.ext, where the corresponding BLAST database filenames have the form ACCESSION.ext. This is the convention followed by the fragment_FASTA_files() function above.

pyani.anib.get_fraglength_dict(fastafiles: List[pathlib.Path]) → Dict[KT, VT]

Return dictionary of sequence fragment lengths, keyed by query name.

Parameters **fastafiles** – list of paths to FASTA input whole sequence files

Loops over input files and, for each, produces a dictionary with fragment lengths, keyed by sequence ID. These are returned as a dictionary with the keys being query IDs derived from filenames.

pyani.anib.get_fragment_lengths(fastafile: pathlib.Path) → Dict[KT, VT]

Return dictionary of sequence fragment lengths, keyed by fragment ID.

Parameters **fastafile** –

Biopython’s SeqIO module is used to parse all sequences in the FASTA file.

NOTE: ambiguity symbols are not discounted.

pyani.anib.get_version(blast_exe: pathlib.Path = PosixPath('blastn')) → str

Return BLAST+ blastn version as a string.

Parameters **blast_exe** – path to blastn executable

We expect blastn to return a string as, for example

```
$ blastn -version
blastn: 2.9.0+
Package: blast 2.9.0, build Jun 10 2019 09:40:53
```

This is concatenated with the OS name.
pyani Documentation, Release 0.2.9


Return BLASTcmds object for construction of BLAST commands.

Parameters

- **mode** – str, the kind of ANIb analysis (ANIb or ANIblastall)
- **outdir** –
- **format_exe** –
- **blast_exe** –
- **prefix** –


Return job dependency graph, based on the passed input sequence files.

Parameters

- **infiles** – list of paths to input FASTA files
- **fragfiles** – list of paths to fragmented input FASTA files
- **blastcmds** –

By default, will run ANIb - it is possible to make a mess of passing the wrong executable for the mode you’re using.

All items in the returned graph list are BLAST executable jobs that must be run after the corresponding database creation. The Job objects corresponding to the database creation are contained as dependencies. How those jobs are scheduled depends on the scheduler (see run_multiprocessing.py, run_sge.py)

pyani.anib.parse_blast_tab(filename: pathlib.Path, fraglengths: Dict[KT, VT], mode: str = 'ANIb') → Tuple[int, int, int]

Return (alignment length, similarity errors, mean_pid) tuple.

Parameters

- **filename** – Path, path to .blast_tab file
- **fraglengths** – Optional[Dict], dictionary of fragment lengths for each genome.
- **mode** – str, analysis type (ANIb or ANIblastall)

Calculate the alignment length and total number of similarity errors (as we would with ANIm), as well as the Goris et al.-defined mean identity of all valid BLAST matches for the passed BLASTALL alignment .blast_tab file.

“ANI between the query genome and the reference genome was calculated as the mean identity of all BLASTN matches that showed more than 30% overall sequence identity (recalculated to an identity along the entire sequence) over an alignable region of at least 70% of their length.”


Return tuple of ANIb results for .blast_tab files in the output dir.

Parameters

- **blast_dir** – Path, path to the directory containing .blast_tab file
- **org_lengths** – Dict, the base count for each input sequence
• **fraglengths** – dictionary of query sequence fragment lengths, only needed for BLASTALL output

• **mode** – str, analysis type (ANIb or ANIblastall)

• **logger** – a logger for messages

Returns the following pandas dataframes in an ANIResults object; query sequences are rows, subject sequences are columns:

• alignment_lengths - non-symmetrical: total length of alignment

• percentage_identity - non-symmetrical: ANIb (Goris) percentage identity

• alignment_coverage - non-symmetrical: coverage of query

• similarity_errors - non-symmetrical: count of similarity errors

May throw a ZeroDivisionError if one or more BLAST runs failed, or a very distant sequence was included in the analysis.

**pyani.anim module**

Code to implement the ANIm average nucleotide identity method.


All input FASTA format files are compared against each other, pairwise, using NUCmer (binary location must be provided). NUCmer output will be stored in a specified output directory.

The NUCmer .delta file output is parsed to obtain an alignment length and similarity error count for every unique region alignment. These are processed to give matrices of aligned sequence lengths, similarity error counts, average nucleotide identity (ANI) percentages, and minimum aligned percentage (of whole genome) for each pairwise comparison.


Return a tuple of corresponding NUCmer and delta-filter commands.

**Parameters**

• **fname1** – path to query FASTA file

• **fname2** – path to subject FASTA file

• **outdir** – path to output directory

• **nucmer_exe** –

• **filter_exe** –

• **maxmatch** – Boolean flag indicating whether to use NUCmer’s -maxmatch option. If not, the -mum option is used instead

The split into a tuple was made necessary by changes to SGE/OGE. The delta-filter command must now be run as a dependency of the NUCmer command, and be wrapped in a Python script to capture STDOUT.

**NOTE:** This command-line writes output data to a subdirectory of the passed outdir, called “nucmer_output”. 

---

*5.12. pyani package*

Return list of NUCmer command-lines for ANIm.

Parameters

• filenames – a list of paths to input FASTA files
• outdir – path to output directory
• nucmer_exe – location of the nucmer binary
• maxmatch – Boolean flag indicating to use NUCmer’s -maxmatch option

The first element returned is a list of NUCmer commands, and the second a corresponding list of delta_filter_wrapper.py commands. The NUCmer commands should each be run before the corresponding delta-filter command.

TODO: This return value needs to be reworked as a collection.

Loop over all FASTA files generating NUCmer command lines for each pairwise comparison.


Return list of Jobs describing NUCmer command-lines for ANIm.

Parameters

• filenames – Iterable, Paths to input FASTA files
• outdir – str, path to output directory
• nucmer_exe – str, location of the nucmer binary
• filter_exe –
• maxmatch – Boolean flag indicating to use NUCmer’s -maxmatch option
• jobprefix –

Loop over all FASTA files, generating Jobs describing NUCmer command lines for each pairwise comparison.

pyani.anim.get_fasta_files(dirname: pathlib.Path = PosixPath('')) → Iterable[T_co]

Return iterable of FASTA files in the passed directory.

Parameters
dirname – str, path to input directory

pyani.anim.get_version(nucmer_exe: pathlib.Path = PosixPath('nucmer')) → str

Return NUCmer package version as a string.

Parameters nucmer_exe – path to NUCmer executable

We expect NUCmer to return a string on STDERR as

```
$ nucmer
NUCmer (NUCleotide MUMmer) version 3.1
```

we concatenate this with the OS name.

pyani.anim.parse_delta(filename: pathlib.Path) → Tuple[int, int]

Return (alignment length, similarity errors) tuple from passed .delta.
**Parameters**

- **filename** – Path, path to the input .delta file

Extracts the aligned length and number of similarity errors for each aligned uniquely-matched region, and returns the cumulative total for each as a tuple.

Similarity errors are defined in the .delta file spec (see below) as non-positive match scores. For NUCmer output, this is identical to the number of errors (non-identities and indels).

Delta file format has seven numbers in the lines of interest: see [http://mummer.sourceforge.net/manual/](http://mummer.sourceforge.net/manual/) for specification

- start on query
- end on query
- start on target
- end on target
- error count (non-identical, plus indels)
- **similarity errors (non-positive match scores)** [NOTE: with PROmer this is equal to error count]
- stop codons (always zero for nucmer)

To calculate alignment length, we take the length of the aligned region of the reference (no gaps), and process the delta information. This takes the form of one value per line, following the header sequence. Positive values indicate an insertion in the reference; negative values a deletion in the reference (i.e. an insertion in the query). The total length of the alignment is then:

```
reference_length + insertions - deletions
```

For example:

```
A = ABCDACBDAC$ B = BCCDACDCAC$ Delta = (1, -3, 4, 0) A = ABCDACBDAC$ B = .BCCDAC.DCAC$
```

A is the reference and has length 11. There are two insertions (positive delta), and one deletion (negative delta). Alignment length is then $11 + 1 = 12$.

**pyani.anim.process_deltadir**

```
pyani.anim.process_deltadir(delta_dir: pathlib.Path, org_lengths: Dict[KT, VT], logger: Optional[logging.Logger] = None) -> pyani.pyani_tools.ANIResults
```

Return tuple of ANIm results for .deltas in passed directory.

**Parameters**

- **delta_dir** – Path, path to the directory containing .delta files
- **org_lengths** – dictionary of total sequence lengths, keyed by sequence

Returns the following pandas dataframes in an ANIResults object; query sequences are rows, subject sequences are columns:

- alignment_lengths - symmetrical: total length of alignment
- percentage_identity - symmetrical: percentage identity of alignment
- alignment_coverage - non-symmetrical: coverage of query and subject
- similarity_errors - symmetrical: count of similarity errors

May throw a ZeroDivisionError if one or more NUCmer runs failed, or a very distant sequence was included in the analysis.
pyani Documentation, Release 0.2.9

pyani.blast module

Code for handling BLAST output files.

```python
pyani.blast.parse_blasttab(fhandle: TextIO) → List[List[str]]
```

Return the passed BLAST tab output file as a list of lists.

**Parameters**

- `fhandle` – TextIO, filehandle containing BLAST output file

This is used when testing for conserved BLAST output, as the exact format of the BLAST result can depend on the software version. For instance, the locally-installed version may be BLASTN+ 2.6.0, which reports match identity to 3sf, and the version in CI may be BLASTN+ 2.2.28, which reports to 2sf.

Returning a list of lines, parsed into the appropriate data type, allows for direct comparison of line content independent of formatting.

pyani.download module

Module providing functions useful for downloading genomes from NCBI.

```python
class pyani.download.ASMIDs
```

Matching Assembly ID information for a query taxID.

- `asm_ids` – Alias for field number 2
- `query` – Alias for field number 0
- `result_count` – Alias for field number 1

```python
class pyani.download.Classification
```

Taxonomic classification for an isolate.

- `genus` – Alias for field number 1
- `organism` – Alias for field number 0
- `species` – Alias for field number 2
- `strain` – Alias for field number 3

```python
class pyani.download.DLFileData
```

Convenience struct for file download data.

- `filestem` – Alias for field number 0
- `ftpstem` – Alias for field number 1
**suffix**

Alias for field number 2

**class** pyani.download.DLStatus (**url**: str, **hashurl**: str, **outfname**: pathlib.Path, **outfhash**: pathlib.Path, **skipped**: bool, **error**: Optional[str] = None)

```
Bases: object
Download status data.
```

**exception** pyani.download.FileExistsException (**msg**: str = 'Specified file exists')

```
Bases: Exception
A specified file exists.
```

**class** pyani.download.Hashstatus

```
Bases: tuple
Status report on file hash comparison.
```

**filehash**

Alias for field number 2

**localhash**

Alias for field number 1

**passed**

Alias for field number 0

**exception** pyani.download.NCBIDownloadException (**msg**: str = 'Error downloading file from NCBI')

```
Bases: Exception
General exception for failed NCBI download.
```

**pyani.download.check_hash** (**fname**: pathlib.Path, **hashfile**: pathlib.Path) → pyani.download.Hashstatus

Check MD5 of passed file against downloaded NCBI hash file.

**Parameters**

- **fname** – Path, path to local hash file
- **hashfile** – Path, path to NCBI hash file

**pyani.download.compile_url** (**filestem**: str, **suffix**: str, **ftpstem**: str) → Tuple[str, str]

Compile download URLs given a passed filestem.

**Parameters**

- **filestem** –
- **suffix** –
- **ftpstem** –

The filestem corresponds to `<AA>_<AN>`, where `<AA>` and `<AN>` are AssemblyAccession and AssemblyName: data fields in the eSummary record. These correspond to downloadable files for each assembly at ftp://ftp.ncbi.nlm.nih.gov/ncbi/ftp.db/ftp.db/<AA>/ where `<AA>` is AssemblyAccession, and `<AN>` is AssemblyName. The choice of GCA vs GCF, and the values of nnn, are derived from `<AA>`

The files in this directory all have the stem `<AA>_<AN>_<suffix>`, where suffixes are: assembly_report.txt, assembly_stats.txt, feature_table.txt, genomic.fna.gz, genomic.gbff.gz, genomic.gff.gz, protein.faa.gz, protein.gpff.gz, rm_out.gz, rm.run, wgsmaster.gbff.gz
Construct paths to output files for genome and hash.

**Parameters**

- **filestem** – str, output filename stem
- **suffix** – str, output filename suffix
- **outdir** – Path, path to output directory

Return MD5 hash of the passed file contents.

**Parameters**

- **fname** – Path, path to file for hashing

We can ignore the Bandit B303 error as we’re not using the hash for cryptographic purposes.

Return class and label text from UID classification.

**Parameters**

- **classification** – Classification named tuple (org, genus, species, strain)
- **filestem** – str, filestem of input genome file
- **genomehash** – str, MD5 hash of genome data

The ‘class’ data is the organism as provided in the passed Classification named tuple; the ‘label’ data is genus, species and strain information from the same tuple. The label is intended to be human-readable, the class data to be a genuine class identifier.

Returns a tuple of two strings: (label, class).

The two strings are tab-separated strings: `<HASH>t<FILE>t<CLASS/LABEL>`. The hash is used to help uniquely identify the genome in the database (label/class is unique by a combination of hash and run ID).

Download genome and accompanying MD5 hash from NCBI.

**Parameters**

- **args** – Namespace for command-line arguments
- **outdir** – Path to output directory for downloads
- **timeout** – int: timeout for download attempt
- **dlfiledata** – namedtuple of info for file to download
- **dltype** – reference database to use: RefSeq or GenBank
- **disable_tqdm** – disable progress bar

This function tries the (assumed to be passed) RefSeq FTP URL first and, if that fails, then attempts to download the corresponding GenBank data.

We attempt to gracefully skip genomes with download errors.

Download remote URL to a local directory.
Parameters

- **url** – URL of remote file for download
- **outfname** – Path, path to write output
- **timeout** –
- **disable_tqdm** – Boolean, show tqdm progress bar?

This function downloads the contents of the passed URL to the passed filename, in buffered chunks.

```python
pyani.download.entrez_batch(func)
```

Decorator to compile batches from the wrapped function into a single set of results.

The `entrez_batch` decorator should go outside the `entrez_retry` decorator.

```python
pyani.download.entrez_batched_webhistory(*args, expected=None, batchsize=None, **kwargs)
```

```python
pyani.download.entrez_esearch(*args, retries=1, **kwargs)
```

```python
pyani.download.entrez_esummary(*args, retries=1, **kwargs)
```

```python
pyani.download.entrez_retry(func)
```

Decorator to retry the wrapped function up to `retries` times.

```python
```

Extract contents of `fname` to `ename` using gunzip.

**Parameters**

- **fname** – str, path to input compressed file
- **ename** – str, path to output uncompressed file

Returns status of subprocess call.

```python
pyani.download.extract_filestem(esummary) → str
```

Extract filestem from Entrez eSummary data.

**Parameters**

- **esummary** –

Function expects `esummary`['DocumentSummarySet'][0]['DocumentSummary'][0]

Some illegal characters may occur in `AssemblyName` - for these, a more robust regex replace/escape may be required. Sadly, NCBI don’t just use standard percent escapes, but instead replace certain characters with underscores: white space, slash, comma, hash, brackets.

```python
pyani.download.extract_hash(hashfile: pathlib.Path, name: str) → str
```

Return MD5 hash from file of name:MD5 hashes.

**Parameters**

- **hashfile** – Path, path to file containing name:MD5 pairs
- **name** – str, name associated with hash

```python
pyani.download.get_asm_uids(taxon_uid: str, retries: int) → pyani.download.ASMIDs
```

Return set of NCBI UIDs associated with the passed taxon UID.

**Parameters**

- **taxon_uid** – str, NCBI taxID for taxon to download
- **retries** – int, number of download retry attempts

This query at NCBI returns all assemblies for the taxon subtree rooted at the passed taxon_uid.
pyani Documentation, Release 0.2.9

pyani.download.get_ncbi_classification(esummary) → pyani.download.Classification
Return organism, genus, species, strain info from eSummary data.

Parameters
esummary –

pyani.download.get_ncbi_esummary(asm_uid, retries, api_key=None) → Tuple
Obtain full eSummary info for the passed assembly UID.

Parameters

• asm_uid –
• retries –
• api_key –

pyani.download.last_exception() → str
Return last exception as a string.

pyani.download.make_asm_dict(taxon_ids: List[str], retries: int) → Dict[KT, VT]
Return a dict of assembly UIDs, keyed by passed taxon IDs.

Parameters

• taxon_ids –
• retries –

Takes the passed list of taxon IDs and calls get_asm_uids to generate a dictionary linking each taxon ID to a list of assembly IDs at NCBI.

Download genome contigs and MD5 hash data from NCBI.

Parameters

• filestem –
• suffix –
• ftpstem –
• outdir –
• timeout –
• disable_tqdm – Boolean, show tqdm progress bar?

pyani.download.set_ncbi_email(email: str) → None
Set contact email for NCBI.

Parameters
eemail – str, email address to give to Entrez at NCBI

pyani.download.split_taxa(taxa: str) → List[str]
Return list of taxon ids from the passed comma-separated list.

Parameters
taxa – str, comma-separated list of valid NCBI taxonomy IDs

The function checks the passed taxon argument against a regular expression that permits comma-separated numerical symbols only.
**pyani.nucmer module**

Code for handling NUCmer output files.

```python
class pyani.nucmer.DeltaAlignment (refstart: int, refend: int, qrystart: int, qryend: int, errs: int, simerrs: int, stops: int)
Bases: object

Represents a single alignment region and scores for a pairwise comparison.
```

```python
Bases: object

Represents a comparison between two sequences in a .delta file.
```

```python
add_alignment (aln: pyani.nucmer.DeltaAlignment) → None
```

Parameters:

- `aln` – DeltaAlignment object

```python
class pyani.nucmer.DeltaData (name: str, handle: TextIO = None)
Bases: object

Class to hold MUMmer/nucmer output “delta” data.

This is required because the ordering within files differs depending on MUMmer build, for the same version (as evidenced by differences between OSX and Linux builds), and a means of testing for equality of outputs is necessary.

The output file structure and format is described at http://mummer.sourceforge.net/manual/#nucmeroutput

Each file is represented as:

- `header`: first line of the .delta file, naming the two input comparison files; stored as a tuple (path1, path2), returned as the combined string; the individual files are stored as self._query and self._subject
- `program`: name of the MUMmer program that produced the output
- `query`: path to the query sequence file
- `subject`: path to the subject sequence file

```python
class pyani.nucmer.DeltaData (name: str, handle: TextIO = None)
Bases: object

Class to hold MUMmer/nucmer output “delta” data.

This is required because the ordering within files differs depending on MUMmer build, for the same version (as evidenced by differences between OSX and Linux builds), and a means of testing for equality of outputs is necessary.

The output file structure and format is described at http://mummer.sourceforge.net/manual/#nucmeroutput

Each file is represented as:

- `header`: first line of the .delta file, naming the two input comparison files; stored as a tuple (path1, path2), returned as the combined string; the individual files are stored as self._query and self._subject
- `program`: name of the MUMmer program that produced the output
- `query`: path to the query sequence file
- `subject`: path to the subject sequence file

```python
class pyani.nucmer.DeltaData (name: str, handle: TextIO = None)
Bases: object

Class to hold MUMmer/nucmer output “delta” data.

This is required because the ordering within files differs depending on MUMmer build, for the same version (as evidenced by differences between OSX and Linux builds), and a means of testing for equality of outputs is necessary.

The output file structure and format is described at http://mummer.sourceforge.net/manual/#nucmeroutput

Each file is represented as:

- `header`: first line of the .delta file, naming the two input comparison files; stored as a tuple (path1, path2), returned as the combined string; the individual files are stored as self._query and self._subject
- `program`: name of the MUMmer program that produced the output
- `query`: path to the query sequence file
- `subject`: path to the subject sequence file

```python
class pyani.nucmer.DeltaData (name: str, handle: TextIO = None)
Bases: object

Class to hold MUMmer/nucmer output “delta” data.

This is required because the ordering within files differs depending on MUMmer build, for the same version (as evidenced by differences between OSX and Linux builds), and a means of testing for equality of outputs is necessary.

The output file structure and format is described at http://mummer.sourceforge.net/manual/#nucmeroutput

Each file is represented as:

- `header`: first line of the .delta file, naming the two input comparison files; stored as a tuple (path1, path2), returned as the combined string; the individual files are stored as self._query and self._subject
- `program`: name of the MUMmer program that produced the output
- `query`: path to the query sequence file
- `subject`: path to the subject sequence file

```python
class pyani.nucmer.DeltaData (name: str, handle: TextIO = None)
Bases: object

Class to hold MUMmer/nucmer output “delta” data.

This is required because the ordering within files differs depending on MUMmer build, for the same version (as evidenced by differences between OSX and Linux builds), and a means of testing for equality of outputs is necessary.

The output file structure and format is described at http://mummer.sourceforge.net/manual/#nucmeroutput

Each file is represented as:

- `header`: first line of the .delta file, naming the two input comparison files; stored as a tuple (path1, path2), returned as the combined string; the individual files are stored as self._query and self._subject
- `program`: name of the MUMmer program that produced the output
- `query`: path to the query sequence file
- `subject`: path to the subject sequence file

```python
class pyani.nucmer.DeltaData (name: str, handle: TextIO = None)
Bases: object

Class to hold MUMmer/nucmer output “delta” data.

This is required because the ordering within files differs depending on MUMmer build, for the same version (as evidenced by differences between OSX and Linux builds), and a means of testing for equality of outputs is necessary.

The output file structure and format is described at http://mummer.sourceforge.net/manual/#nucmeroutput

Each file is represented as:

- `header`: first line of the .delta file, naming the two input comparison files; stored as a tuple (path1, path2), returned as the combined string; the individual files are stored as self._query and self._subject
- `program`: name of the MUMmer program that produced the output
- `query`: path to the query sequence file
- `subject`: path to the subject sequence file

```python
class pyani.nucmer.DeltaData (name: str, handle: TextIO = None)
Bases: object

Class to hold MUMmer/nucmer output “delta” data.

This is required because the ordering within files differs depending on MUMmer build, for the same version (as evidenced by differences between OSX and Linux builds), and a means of testing for equality of outputs is necessary.

The output file structure and format is described at http://mummer.sourceforge.net/manual/#nucmeroutput

Each file is represented as:

- `header`: first line of the .delta file, naming the two input comparison files; stored as a tuple (path1, path2), returned as the combined string; the individual files are stored as self._query and self._subject
- `program`: name of the MUMmer program that produced the output
- `query`: path to the query sequence file
- `subject`: path to the subject sequence file

```python
class pyani.nucmer.DeltaData (name: str, handle: TextIO = None)
Bases: object

Class to hold MUMmer/nucmer output “delta” data.

This is required because the ordering within files differs depending on MUMmer build, for the same version (as evidenced by differences between OSX and Linux builds), and a means of testing for equality of outputs is necessary.

The output file structure and format is described at http://mummer.sourceforge.net/manual/#nucmeroutput

Each file is represented as:

- `header`: first line of the .delta file, naming the two input comparison files; stored as a tuple (path1, path2), returned as the combined string; the individual files are stored as self._query and self._subject
- `program`: name of the MUMmer program that produced the output
- `query`: path to the query sequence file
- `subject`: path to the subject sequence file
class `pyani.nucmer.DeltaIterator` *(handle: TextIO)*

Bases: object

Iterator for MUMmer .delta files.

Returns a stream of DeltaMetadata, DeltaComparison and DeltaAlignment objects when iterated over a filehandle.

The .delta file structure and format is described at [http://mummer.sourceforge.net/manual/#nucmeroutput](http://mummer.sourceforge.net/manual/#nucmeroutput)

class `pyani.nucmer.DeltaMetadata`

Bases: object

Represents the metadata header for a MUMmer .delta file.

**pyani.pyani_classify module**

Module providing functions to generate clusters/species hypotheses.

class `pyani.pyani_classify.CliquesInfo`

Bases: tuple

Summary of clique structure.

`all_k_complete`

Alias for field number 2

`n_nodes`

Alias for field number 0

`n_subgraphs`

Alias for field number 1

`pyani.pyani_classify.all_components_k_complete(graph: networkx.classes.graph.Graph)`

Return True if all components in passed graph are k-complete. → bool

Parameters `graph` – NetworkX Graph object

`pyani.pyani_classify.analyse_cliques(graph: networkx.classes.graph.Graph)` → `pyani.pyani_classify.CliquesInfo`

Return Cliquesinfo NamedTuple describing clique data for a graph.

Parameters `graph` – NetworkX Graph object

`pyani.pyani_classify.build_graph_from_results(results, label_dict: Dict[int, str], cov_min: float = 0, id_min: float = 0)` → `networkx.classes.graph.Graph`

Return undirected graph representing the passed ANIResults object.

The passed ANIResults object is converted to an undirected graph where nodes on the graph represent genomes, and edges represent pairwise comparisons having the minimum coverage and identity indicated.

Parameters

- `results` – Run object from pyani_orm
- `label_dict` – dictionary of genome labels for result matrices the dict is keyed by the index/column values for the results matrices
- `cov_min` – minimum coverage for an edge
• \texttt{id\_min} –
  \hspace{1cm} \text{minimum identity for an edge}

\texttt{pyani.pyani\_classify.k\_complete\_component\_status}(\texttt{graph: networkx.classes.graph.Graph}) 
\hspace{1cm} \rightarrow 
\texttt{List[bool]}

Return list of Booleans of whether connected components of the graph are k-complete.

\textbf{Parameters} \texttt{graph} – NetworkX Graph object

For each component in the passed graph, a list of Booleans is calculated, representing whether each node has property P: the degree of the node is equal to the number of nodes in that component, minus 1.

The \texttt{all()} gives a Boolean indicating whether all nodes in that component have property P.

\texttt{pyani.pyani\_classify.remove\_low\_weight\_edges}(\texttt{graph: networkx.classes.graph.Graph}, \texttt{threshold: float}, \texttt{attribute: str = ‘identity’}) 
\hspace{1cm} \rightarrow 
\texttt{Tuple[net\texttt{workx.classes.graph.Graph}, List[T]]}

Return graph and edgelist where edges having weight < threshold are removed.

\textbf{Parameters}

• \texttt{graph} – NetworkX Graph
• \texttt{threshold} – float, minimum edge weight
• \texttt{attribute} – String, attribute to use as weight

\textbf{pyani.pyani\_config module}

Configuration settings for the pyani package.

\texttt{pyani.pyani\_config.get\_colormap}(\texttt{dataframe: pandas.core.frame.DataFrame}, \texttt{matname: str}) 
\hspace{1cm} \rightarrow 
\texttt{Tuple[str, Any, Any]}

Return colormap parameters for a dataframe.

\textbf{Parameters}

• \texttt{dataframe} –
• \texttt{matname} –

The colormap is dependent on the type of analysis that was done.

\texttt{pyani.pyani\_config.params\_mpl}(\texttt{dfm: pandas.core.frame.DataFrame}) 
\hspace{1cm} \rightarrow 
\texttt{Dict[str, Tuple[str, Any, Any]]}

Return dict of matplotlib parameters, dependent on dataframe.

\textbf{Parameters} \texttt{dfm} –

DEPRECATED FROM v0.3 onwards

\textbf{pyani.pyani\_files module}

Code to handle files for average nucleotide identity calculations.

\textbf{exception pyani.pyani\_files.PyaniFilesException}

\hspace{1cm} \textbf{Bases: pyani.PyaniException}

Exception raised by pyani when file interaction goes bad.

Return a list of existing output files at dirpath.

Parameters

• dirpath – Path, path to existing output directory
• program – str, name of program to use for comparisons
• args – Namespace, command-line arguments for the run


Return a list of (FASTA file, hash file) tuples in passed directory.

Parameters dirname – Path, path to input directory

Raises an IOError if the corresponding hash for a FASTA file does not exist

pyani.pyani_files.get_fasta_files (dirname: pathlib.Path = PosixPath(.) → List[pathlib.Path]

Return a list of FASTA files in the passed directory.

Parameters dirname – Path, path to input directory

pyani.pyani_files.get_fasta_paths (dirname: pathlib.Path = PosixPath(.) → List[pathlib.Path]

Parameters

• dirname – Path, path to directory containing input FASTA files
• extlist – List, file suffixes for FASTA files

Returns the full path to each file.


Return files in passed directory, filtered by extension.

Parameters

• dirname – Path, path to input directory
• *ext – optional iterable of arguments describing permitted file extensions

pyani.pyani_files.get_sequence_lengths (fastafilenames: Iterable[pathlib.Path]) → Dict[str, int]

Return dictionary of sequence lengths, keyed by organism.

Parameters fastafilenames – Iterable[Path], paths to input FASTA files

Biopython’s SeqIO module is used to parse all sequences in the FASTA file corresponding to each organism, and the total base count in each is obtained.

NOTE: ambiguity symbols are not discounted.

pyani.pyani_files.load_classes_labels (path: pathlib.Path) → Dict[str, str]

Return a dictionary of genome classes or labels keyed by hash.

Parameters path – Path, path to classes or labels file

The expected format of the classes and labels files is: <HASH><FILESTEM><CLASS><LABEL>, where <HASH> is the MD5 hash of the genome data (this is not checked); <FILESTEM> is the path to the genome file (this is intended to be a record for humans to audit, it’s not needed for the database interaction; and <CLASS><LABEL> is the class or label associated with that genome.
pyani.pyani_files.read_fasta_description (filename: pathlib.Path) → str
Return the first description string from a FASTA file.

Parameters filename – Path, path to FASTA file

pyani.pyani_files.read_hash_string (filename: pathlib.Path) → Tuple[str, str]
Return the hash and file strings from the passed hash file.

Parameters filename – Path, path to file containing hash string

pyani.pyani_graphics module
Code to implement graphics output for ANI analyses.
class pyani.pyani_graphics.Params (params: Tuple, labels: Optional[Dict[KT, VT]] = None, classes: Optional[Dict[KT, VT]] = None)
Bases: object
Convenience class to hold heatmap rendering parameters.

vdiff
Return difference between max and min values for presentation.

pyani.pyani_jobs module
Code to manage jobs for pyani.
In order to be a little more consistent behind the scenes for schedulers, and to allow for a fairly hacky approach to scheduling on SGE, a job dependency graph is used.
Commands to be run are stored in Jobs. A Job’s dependency is stored so that the Job will not be executed until its dependency is executed.
When used in ANI analysis, the way jobs are used depends on the scheduler.
With multiprocessing, we place all root jobs in a single pool; then all first-level dependencies will go in a second (dependent) pool that is not run until the first is completed, and so on. It’s not very efficient, but should work equivalently to the original code that handled asynchronous pools directly.
With SGE, the dependencies can be managed independently, and effectively interleaved by the scheduler with no need for pools.
This code is essentially a frozen and cut-down version of pysge (https://github.com/widdowquinn/pysge)
class pyani.pyani_jobs.Job (name: str, command: str, queue: Optional[str] = None)
Bases: object
Individual job to be run, with list of dependencies.

add_dependency (job) → None
Add passed job to the dependency list for this Job.

Parameters job – Job to be added to the Job’s dependency list
This Job should not execute until all dependent jobs are completed.

remove_dependency (job) → None
Remove passed job from this Job’s dependency list.

Parameters job – Job to be removed from the Job’s dependency list

wait (interval: float = 0.01) → None
Wait until the job finishes, and poll SGE on its status.
Parameters interval – float, number of seconds to wait before polling SGE

Bases: object

Class that stores a group of jobs, permitting parameter sweeps.

add_dependency(job) -> None
Add the passed job to the dependency list for this JobGroup.

Parameters job – Job, job to be added to the JobGroup’s dependency list

This JobGroup should not execute until all dependent jobs are completed

generate_script() -> None
Create the SGE script that will run the jobs in the JobGroup.

remove_dependency(job) -> None
Remove passed job from this JobGroup’s dependency list.

Parameters job – Job, job to be removed from the JobGroup’s dependency list

wait(interval: float = 0.01) -> None
Wait for a defined period, then poll SGE for job status.

Parameters interval – int, seconds to wait before polling SGE

pyani.pyani ORM module

Module providing useful functions for manipulating pyani’s SQLite3 db.
This SQLAlchemy-based ORM replaces the previous SQL-based module

class pyani.pyani ORM.BlastDB(**kwargs)
Bases: sqlalchemy.ext.declarative.api.Base

Describes relationship between genome, run, source BLAST database and query fragments.

Each genome and run combination can be assigned a single BLAST database for the comparisons

• fragpath path to fragmented genome (query in ANIb)
• dbpath path to source genome database (subject in ANIb)
• fragsizes JSONified dict of fragment sizes
• dbcmd command used to generate database

blastdb_id
dbcmd
dbpath
fragpath
fragsizes
genome
genome_id
run
run_id
class pyani.pyani_orm.Comparison(**kwargs)
    Bases: sqlalchemy.ext.declarative.api.Base

    Describes a single pairwise comparison between two genomes.
    aln_length
    comparison_id
    cov_query
    cov_subject
    fragsize
    identity
    maxmatch
    program
    query
    query_id
    runs
    sim_errs
    subject
    subject_id
    version

class pyani.pyani_orm.Genome(**kwargs)
    Bases: sqlalchemy.ext.declarative.api.Base

    Describes an input genome for a pyani run.
    • genome_id primary key
    • genome_hash MD5 hash of input genome file (in path)
    • path path to FASTA genome file
    • length length of genome (total bases)
    • description genome description
    blastdbs
description
    genome_hash
    genome_id
    labels
    length
    path
    query_comparisons
    runs
    subject_comparisons
class pyani.pyani_orm.Label(**kwargs)
    Bases: sqlalchemy.ext.declarative.api.Base
    
    Describes relationship between genome, run and genome label.
    Each genome and run combination can be assigned a single label
    
    class_label
    genome
    genome_id
    label
    label_id
    run
    run_id

class pyani.pyani_orm.LabelTuple
    Bases: tuple
    
    Label and Class for each file.
    
    class_label
        Alias for field number 1
    label
        Alias for field number 0

exception pyani.pyani_orm.PyaniORMException
    Bases: pyani.PyaniException
    
    Exception raised when ORM or database interaction fails.

class pyani.pyani_orm.Run(**kwargs)
    Bases: sqlalchemy.ext.declarative.api.Base
    
    Describes a single pyani run.
    
    blastdbs
    cmdline
    comparisons
    date
    df_alnlength
    df_coverage
    df_hadamard
    df_identity
    df_simerrors
    genomes
    labels
    method
    name
    run_id
status

pyani.pyani.orm.add_run(session, method, cmdline, date, status, name)

Create a new Run and add it to the session.

Parameters

- **session** – live SQLAlchemy session of pyani database
- **method** – string describing analysis run type
- **cmdline** – string describing pyani command-line for run
- **date** – datetime object describing analysis start time
- **status** – string describing status of analysis
- **name** – string - name given to the analysis run

Creates a new Run object with the passed parameters, and returns it.


Add genomes for a run to the database.

Parameters

- **session** – live SQLAlchemy session of pyani database
- **run** – Run object describing the parent pyani run
- **indir** – path to the directory containing genomes
- **classpath** – path to the file containing class information for each genome
- **labelpath** – path to the file containing class information for each genome

This function expects a single directory (indir) containing all FASTA files for a run, and optional paths to plain text files that contain information on class and label strings for each genome.

If the genome already exists in the database, then a Genome object is recovered from the database. Otherwise, a new Genome object is created. All Genome objects will be associated with the passed Run object.

The session changes are committed once all genomes and labels are added to the database without error, as a single transaction.

pyani.pyani.orm.create_db(dbpath: pathlib.Path) → None

Create an empty pyani SQLite3 database at the passed path.

Parameters **dbpath** – path to pyani database

pyani.pyani.orm.filter_existing_comparisons(session, run, comparisons, program, version, fragsize: Optional[int] = None, maxmatch: Optional[bool] = None) → List[T]

Filter list of (Genome, Genome) comparisons for those not in the session db.

Parameters

- **session** – live SQLAlchemy session of pyani database
- **run** – Run object describing parent pyani run
- **comparisons** – list of (Genome, Genome) query vs subject comparisons
- **program** – program used for comparison
- **version** – version of program for comparison
- **fragsize** – fragment size for BLAST databases
• **maxmatch** – maxmatch used with nucmer comparison

When passed a list of (Genome, Genome) comparisons as comparisons, check whether the comparison exists in
the database and, if so, associate it with the passed run. If not, then add the (Genome, Genome) pair to a list for
returning as the comparisons that still need to be run.

```python
pyani.pyani_orm.get_comparison_dict(session: Any) → Dict[Tuple, Any]
```

Return a dictionary of comparisons in the session database.

**Parameters**

- `session` – live SQLAlchemy session of pyani database

**Returns**

Returns Comparison objects, keyed by (_query_id, _subject_id, _program, _version, _fragsize, _maxmatch)
tuple

```python
pyani.pyani_orm.get_matrix_classes_for_run(session: Any, run_id: int) → Dict[str, List[T]]
```

Return dictionary of genome classes, keyed by row/column ID.

**Parameters**

- `session` – live SQLAlchemy session
- `run_id` – the Run.run_id value for matrices

The class labels should be valid for identity, coverage and other complete matrix results accessed via the .df_* attributes of a run

Labels are returned keyed by the string of the genome ID, for compatibility with matplotlib.

```python
pyani.pyani_orm.get_matrix_labels_for_run(session: Any, run_id: int) → Dict[KT, VT]
```

Return dictionary of genome labels, keyed by row/column ID.

**Parameters**

- `session` – live SQLAlchemy session
- `run_id` – the Run.run_id value for matrices

The labels should be valid for identity, coverage and other complete matrix results accessed via the .df_* attributes of a run.

Labels are returned keyed by the string of the genome ID, for compatibility with matplotlib.

```python
pyani.pyani_orm.get_session(dbpath: pathlib.Path) → Any
```

Connect to an existing pyani SQLite3 database and return a session.

**Parameters**

- `dbpath` – path to pyani database

```python
pyani.pyani_orm.update_comparison_matrices(session, run) → None
```

Update the Run table with summary matrices for the analysis.

**Parameters**

- `session` – active pyanidb session via ORM
- `run` – Run ORM object for the current ANIm run

**pyani.pyani_report module**

Module providing functions for presenting analysis/db output.

```python
pyani.pyani_report.colour_coverage(series: pandas.core.series.Series, threshold: float = 0.95, colour: str = '#FF2222') → List[str]
```

Highlight percent coverage over a threshold.

**Parameters**
• **series** –
• **threshold** – float, threshold for cell highlighting
• **colour** – str, hex colour for highlighted cells

**pyani.pyani_report.colour_identity** *(series: pandas.core.series.Series, threshold: float = 0.95, colour: str = ‘#FF2222’) → List[str]*

Highlight percentage identities over a threshold.

**Parameters**

- **series** –
- **threshold** – float, threshold for cell highlighting
- **colour** – str, hex colour for highlighted cells

**pyani.pyani_report.colour_numeric** *(val: float, threshold: float = 0.95, colour: str = ‘#FF2222’) → str*

Highlight numeric values over a threshold.

**Parameters**

- **val** –
- **threshold** – float, threshold for cell highlighting
- **colour** – str, hex colour for highlighted cell

**pyani.pyani_report.colour_rows** *(series: pandas.core.series.Series, even_colour: str = ‘#DDECF5’, odd_colour: str = ‘#6CB6E4’) → List[str]*

Return alternating colours for rows in a dataframe.

**Parameters**

- **series** – pd.Series
- **even_colour** – str, hex colour for even rows
- **odd_colour** – str, hex colour for odd rows

**pyani.pyani_report.header_font** () → Dict[str, Any]

Return header HTML font style.

**pyani.pyani_report.hover_highlight** *(hover_colour: str = ‘#FFFF99’) → Dict[str, Any]*

Return HTML style to colour dataframe row when hovering.

**Parameters**

- **hover_colour** – str, hex colour for hover highlight

**pyani.pyani_report.table_padding** () → Dict[str, Any]

Return HTML for table cell padding.


Write database result table to output file in named format.

**Parameters**

- **dfm** – pd.Dataframe
- **path** – Path to output file
- **formats** – tuple of str, output file formats
- **index** – Boolean
**pyani Documentation, Release 0.2.9**

- `show_index` – Boolean
- `colour_num` – use colours for values in HTML output

    colours are used for identity/coverage tables

```python
```

Add CSS styling to a dataframe and write as HTML.

**Parameters**

- `path` – path to write output file
- `dfm` – dataframe to be written out
- `index` – column to be set as index (if necessary)

```python
pyani.pyani_report.write_to_stdout(stem: str, dfm: pandas.core.frame.DataFrame, show_index: bool = False, line_width: float = None) → None
```

Write dataframe in tab-separated form to STDOUT.

**Parameters**

- `stem` – str
- `dfm` – pd.DataFrame
- `show_index` – Boolean, include index in output table
- `line_width` –

**pyani.pyani_tools module**

Code to support pyani.

```python
class pyani.pyani_tools.ANIResults(labels: List[str], mode: str)
Bases: object
```

Holds ANI dataframe results.

```python
add_coverage(qname: str, sname: str, qcover: float, scover: Optional[float] = None) → None
```

Add percentage coverage values to self.alignment_coverage.

**Parameters**

- `qname` –
- `sname` –
- `value` –
- `sym` –

```python
add_pid(qname: str, sname: str, value: float, sym: bool = True) → None
```

Add a percentage identity value to self.percentage_identity.

**Parameters**

- `qname` –
- `sname` –
- `value` –
• sym

add_sim_errors(qname: str, sname: str, value: float, sym: bool = True) → None
Add a similarity error value to self.similarity_errors.

Parameters
• qname –
• sname –
• value –
• sym –

add_tot_length(qname: str, sname: str, value: float, sym: bool = True) → None
Add a total length value to self.alignment_lengths.

Parameters
• qname –
• sname –
• value –
• sym –

data
Return list of (dataframe, filestem) tuples.

hadamard
Return Hadamard matrix (identity * coverage).

class pyani.pyani_tools.BLASTcmds(funcs: pyani.pyani_tools.BLASTfunctions, exes: pyani.pyani_tools.BLASTexes, prefix: str, outdir: pathlib.Path)
Bases: object
Class for construction of BLASTN and database formatting commands.

build_blast_cmd(fname: pathlib.Path, dbname: pathlib.Path)
Return BLASTN command.

Parameters
• fname – Path to query file
• dbname – Path to database

build_db_cmd(fname: pathlib.Path) → str
Return database format/build command.

Parameters fname –

get_db_name(fname: pathlib.Path) → str
Return database filename.

Parameters fname –

class pyani.pyani_tools.BLASTexes
Bases: tuple
Convenience structure to hold BLAST executables.

blast_exe
Alias for field number 1
format_exe
   Alias for field number 0

class pyani.pyani_tools.BLASTfunctions
   Bases: tuple
   Convenience structure to hold BLAST functions.

blastn_func
   Alias for field number 1

db_func
   Alias for field number 0

class pyani.pyani_tools.Dependencies
   Bases: tuple
   Convenience struct for third-party dependency presence.

blast
   Alias for field number 0

legacy_blast
   Alias for field number 1

mummer
   Alias for field number 2

class pyani.pyani_tools.MatrixData
   Bases: tuple
   Convenience struct for matrix data returned by ORM.

data
   Alias for field number 1

graphic_args
   Alias for field number 2

name
   Alias for field number 0

pyani.pyani_tools.get_genome_length(filename: pathlib.Path) \to\ int
   Return total length of all sequences in a FASTA file.

   Parameters
   
   • filename – path to FASTA file

pyani.pyani_tools.get_labels(filename: pathlib.Path, logger: Optional[logging.Logger] = None) \to\ Dict[KT, VT]
   Return dictionary of alternative sequence labels, or None.

   Parameters
   
   • filename – path to file containing tab-separated table of labels
   • logger – logging object

   Input files should be formatted as <hash>t<key>t<label>, one pair per line.

pyani.pyani_tools.has_dependencies() \to\ pyani.pyani_tools.Dependencies
   Return NamedTuple indicating if 3rd dependencies are available.

pyani.pyani_tools.label_results_matrix(matrix: pandas.core.frame.DataFrame, labels: Dict[KT, VT]) \to\ pandas.core.frame.DataFrame
   Return results matrix dataframe with labels.
Parameters

- **matrix** – results dataframe deriving from Run object
- **labels** – dictionary of genome labels labels must be keyed by index/col values from matrix

Applies the labels from the dictionary to the dataframe in matrix, and returns the result.

```python
pyani.pyani_tools.termcolor(logstr: str, color: Optional[str] = None, bold: Optional[bool] = False) -> str
```

Return the passed logstr, wrapped in terminal colouring.

**pyani.run_multiprocessing module**

Code to run a set of command-line jobs using multiprocessing.

For parallelisation on multi-core desktop/laptop systems, etc. we use Python’s multiprocessing module to distribute command-line jobs.

```python
pyani.run_multiprocessing.multiprocessing_run(cmdlines: List[T], workers: Optional[int] = None) -> int
```

Distributes passed command-line jobs using multiprocessing.

Parameters

- **cmdlines** – iterable, command line strings
- **workers** – int, number of workers to use for multiprocessing

Returns the sum of exit codes from each job that was run. If all goes well, this should be 0. Anything else and the calling function should act accordingly.

```python
```

Create list of jobsets at different depths of dependency tree.

Parameters

- **job** –
- **cmdsets** –
- **depth** –

This is a recursive function (is there something quicker in the itertools module?) that descends each ‘root’ job in turn, populating each

```python
pyani.run_multiprocessing.run_dependency_graph(jobgraph, workers: Optional[int] = None, logger: Optional[logging.Logger] = None) -> int
```

Create and run pools of jobs based on the passed jobgraph.

Parameters

- **jobgraph** – list of jobs, which may have dependencies.
- **workers** – int, number of workers to use with multiprocessing
- **logger** – a logger module logger (optional)

The strategy here is to loop over each job in the list of jobs (jobgraph), and create/populate a series of Sets of commands, to be run in reverse order with multiprocessing_run as asynchronous pools.
**pyani Documentation, Release 0.2.9**

**pyani.run_sge module**

Code to run a set of command-line jobs using SGE/Grid Engine.

For parallelisation on multi-node system, we use some custom code to submit jobs.

```python
pyani.run_sge.build_and_submit_jobs(root_dir: pathlib.Path, jobs: Iterable[T_co], sgeargs: Optional[str] = None) → None
```

Submit passed iterable of Job objects to SGE.

**Parameters**

- `root_dir` – root directory for SGE and job output
- `jobs` – list of Job objects, describing each job to be submitted
- `sgeargs` – str, additional arguments to `qsub`

This places SGE’s output in the passed root directory

```python
pyani.run_sge.build_directories(root_dir: pathlib.Path) → None
```

Construct the subdirectories output, stderr, stdout, and jobs.

**Parameters**

- `root_dir` – path of root directory in which to place output

Subdirectories are created in the passed root directory. These subdirectories have the following roles:

- `jobs` Stores the scripts for each job
- `stderr` Stores the stderr output from SGE
- `stdout` Stores the stdout output from SGE
- `output` Stores output (if the scripts place the output here)

- `root_dir` Path to the top-level directory for creation of subdirectories

```python
pyani.run_sge.build_job_scripts(root_dir: pathlib.Path, jobs: List[T]) → None
```

Construct script for each passed Job in the jobs iterable.

**Parameters**

- `root_dir` – Path to output directory
- `jobs` –

```python
pyani.run_sge.build_joblist(jobgraph) → List[T]
```

Return a list of jobs, from a passed jobgraph.

**Parameters**

- `jobgraph` –

```python
pyani.run_sge.compile_jobgroups_from_joblist(joblist: List[T], jgprefix: str, sgegroupsize: int) → List[T]
```

Return list of jobgroups, rather than list of jobs.

**Parameters**

- `joblist` –
- `jgprefix` – str, prefix for SGE jobgroup
- `sgegroupsize` – int, number of jobs in each SGE jobgroup

```python
pyani.run_sge.extract_submittable_jobs(waiting: List[T]) → List[T]
```

Obtain list of jobs that are able to be submitted from pending list.

**Parameters**

- `waiting` – list of Job objects

```python
```

Create set of jobs reflecting dependency tree.

**Parameters**
The set contains jobs at different depths of the dependency tree, retaining dependencies as strings, not Jobs.

pyani.run_sge.run_dependency_graph(jobgraph, jgprefix: str = 'ANIm_SGE_JG', sgegroupsize: int = 10000, sgeargs: Optional[str] = None) -> None

Create and runs SGE scripts for jobs based on passed jobgraph.

Parameters

- **jobgraph** – list of jobs, which may have dependencies.
- **verbose** – flag for multiprocessing verbosity
- **jgprefix** – a prefix for the submitted jobs, in the scheduler
- **sgegroupsize** – the maximum size for an array job submission
- **sgeargs** – additional arguments to qsub

The strategy here is to loop over each job in the dependency graph and, because we expect a single main delta-filter (wrapped) job, with a single nucmer dependency for each analysis, we can split the dependency graph into two lists of corresponding jobs, and run the corresponding nucmer jobs before the delta-filter jobs.

pyani.run_sge.split_seq(iterable: Iterable[T_co], size: int) -> Generator[T_co, T_contra, V_co]

Split a passed iterable into chunks of a given size.

Parameters

- **iterable** – iterable
- **size** – int, number of items to return in each chunk

pyani.run_sge.submit_jobs(root_dir: pathlib.Path, jobs: Iterable[T_co], sgeargs: Optional[str] = None) -> None

Submit passed jobs to SGE server with passed directory as root.

Parameters

- **root_dir** – path to output directory
- **jobs** – list of Job objects
- **sgeargs** – str, additional arguments for qsub

pyani.run_sge.submit_safe_jobs(root_dir: pathlib.Path, jobs: Iterable[T_co], sgeargs: Optional[str] = None) -> None

Submit passed list of jobs to SGE server with dir as root for output.

Parameters

- **root_dir** – path to output directory
- **jobs** – iterable of Job objects
- **sgeargs** – str, additional arguments for qsub

pyani.tetra module

Code to implement the TETRA average nucleotide identity method.

Provides functions for calculation of TETRA as described in:

and


`pyani.tetra.calculate_correlations(tetra_z: Dict[str, Dict[str, float]]) → pandas.core.frame.DataFrame`

Return dataframe of Pearson correlation coefficients.

**Parameters**

- `tetra_z` – dict, Z-scores, keyed by sequence ID

Calculates Pearson correlation coefficient from Z scores for each tetranucleotide. This is done longhand here, which is fast enough, but for robustness we might want to do something else… (TODO).

Note that we report a correlation by this method, rather than a percentage identity.

`pyani.tetra.calculate_tetra_zscore(filename: pathlib.Path) → Dict[str, float]`

Return TETRA Z-score for the sequence in the passed file.

**Parameters**

- `filename` – path to sequence file

Calculates mono-, di-, tri- and tetranucleotide frequencies for each sequence, on each strand, and follows Teeling et al. (2004) in calculating a corresponding Z-score for each observed tetranucleotide frequency, dependent on the mono-, di- and tri- nucleotide frequencies for that input sequence.

`pyani.tetra.calculate_tetra_zscores(infilenames: Iterable[T_co]) → Dict[str, Dict[str, float]]`

Return dictionary of TETRA Z-scores for each input file.

**Parameters**

- `infilenames` – iterable of paths to input sequence files

`pyani.tetra.tetra_clean(instr: str) → bool`

Return True if string contains only unambiguous IUPAC nucleotide symbols.

**Parameters**

- `instr` – str, nucleotide sequence

We are assuming that a low frequency of IUPAC ambiguity symbols doesn’t affect our calculation.

5.13 Indices and tables

- genindex
- modindex
- search
Python Module Index

p
pyani, 55
pyani.anib, 79
pyani.anim, 83
pyani.blast, 86
pyani.download, 86
pyani.nucmer, 91
pyani.pyani_classify, 92
pyani.pyani_config, 93
pyani.pyani_files, 93
pyani.pyani_graphics, 95
pyani.pyani_jobs, 95
pyani.pyani_orm, 96
pyani.pyani_report, 100
pyani.pyani_tools, 102
pyani.run_multiprocessing, 105
pyani.run_sge, 106
pyani.scripts, 55
pyani.scripts.average_nucleotide_identity, 69
pyani.scripts.delta_filter_wrapper, 75
pyani.scripts.genbank_get_genomes_by_taxon, 75
pyani.scripts.parsers, 56
pyani.scripts.parsers.anib_parser, 56
pyani.scripts.parsers.aniblastall_parser, 56
pyani.scripts.parsers.anim_parser, 57
pyani.scripts.parsers.classify_parser, 57
pyani.scripts.parsers.common_parser, 57
pyani.scripts.parsers.createdb_parser, 58
pyani.scripts.parsers.download_parser, 58
pyani.scripts.parsers.index_parser, 58
pyani.scripts.parsers.plot_parser, 59
pyani.scripts.parsers.report_parser, 59
pyani.scripts.parsers.run_common_parser, 59
pyani.scripts.parsers.scheduling_parser, 60
pyani.scripts.pyani_script, 78
pyani.scripts.subcommands, 60
pyani.scripts.subcommands.subcmd_anib, 60
pyani.scripts.subcommands.subcmd_aniblastall, 61
pyani.scripts.subcommands.subcmd_anim, 61
pyani.scripts.subcommands.subcmd_classify, 64
pyani.scripts.subcommands.subcmd_createdb, 65
pyani.scripts.subcommands.subcmd_download, 65
pyani.scripts.subcommands.subcmd_index, 67
pyani.scripts.subcommands.subcmd_plot, 67
pyani.scripts.subcommands.subcmd_report, 69
pyani.tetra, 107
Index

A

accession (pyani.scripts.subcommands.subcmd_download attribute), 65
add_alignment() (pyani.nucmer.DeltaComparison method), 91
add_coverage() (pyani.pyani_tools.ANIResults method), 102
add_dependency() (pyani.pyani_jobs.Job method), 95
add_dependency() (pyani.pyani_jobs.JobGroup method), 96
add_log_headers() (in module pyani.scripts.pyani_script), 78
add_pid() (pyani.pyani_tools.ANIResults method), 102
add_run() (in module pyani.pyani_orm), 99
add_run_genomes() (in module pyani.pyani_orm), 99
add_sim_errors() (pyani.pyani_tools.ANIResults method), 103
add_tot_length() (pyani.pyani_tools.ANIResults method), 92
all_componentsk_complete() (in module pyani.pyani_classify), 92
all_k_complete (pyani.pyani_classify.Cliquesinfo attribute), 92
aln_length (pyani.pyani_orm.Comparison attribute), 97
aln_length (pyani.scripts.subcommands.subcmd_anim.ComparisonResult attribute), 62
analyse_cliques() (in module pyani.pyani_classify), 92
ANIResults (class in pyani.pyani_tools), 103
Blastcmds (class in pyani.pyani_tools), 103
blastDB (class in pyani.pyani_orm), 96
blastdb_id (pyani.pyani_orm.BlastDB attribute), 96
blastdbs (pyani.pyani_orm.Genome attribute), 97
blastdbs (pyani.pyani_orm.Run attribute), 98
BLASTexes (class in pyani.pyani_tools), 103
BLASTfunctions (class in pyani.pyani_tools), 104
blastn_func (pyani.pyani_tools.BLASTfunctions attribute), 104
build() (in module pyani.scripts.parsers.anib_parser), 56
build() (in module pyani.scripts.parsers.aniblastall_parser), 56
build() (in module pyani.scripts.parsers.anim_parser), 57
build() (in module pyani.scripts.parsers.classify_parser), 57
build() (in module pyani.scripts.parsers.common_parser), 57
build() (in module pyani.scripts.parsers.createdb_parser), 58
build() (in module pyani.scripts.parsers.download_parser), 58
build() (in module pyani.scripts.parsers.index_parser), 58
build() (in module pyani.scripts.parsers.plot_parser), 59
build() (in module pyani.scripts.parsers.report_parser), 59
build() (in module pyani.scripts.parsers.run_common_parser), 59
build() (in module pyani.scripts.parsers.scheduling_parser), 60
build_and_submit_jobs() (in module pyani.run_sge), 106
build_blast_cmd()
C

calculate_animal() (pyani.pyani_tools.BLASTcmds method), 103

calculate_correlations() (in module pyani.tetra), 108

calculate_tetra() (in module pyani.scripts.average_nucleotide_identity), 71

calculate_tetra_zscore() (in module pyani.tetra), 108

calculate_tetra_zscores() (in module pyani.tetra), 108

class_label (pyani.pyani_orm.Label attribute), 98

class_label (pyani.pyani_orm.LabelTuple attribute), 98

Cassification (class in pyani.download), 86

clique_info (pyani.scripts.subcommands.subcmd_animal.Run attribute), 64

Clique_info (class in pyani.pyani_classify), 92

cmdline (pyani.pyani_orm.Run attribute), 98

cmdline (pyani.scripts.subcommands.subcmd_animal.Run attribute), 62

collect_existing_output() (in module pyani.pyani_files), 93

colour_coverage() (in module pyani.pyani_report), 100

colour_identity() (in module pyani.pyani_report), 101

colour_numeric() (in module pyani.pyani_report), 101

colour_rows() (in module pyani.pyani_report), 101

Comparison (class in pyani.pyani_orm), 96

comparison_id (pyani.pyani_orm.Comparison attribute), 97

ComparisonJob (class in pyani.scripts.subcommands.subcmd_animal), 61

ComparisonResult (class in pyani.scripts.subcommands.subcmd_animal), 61
df_simerrors (pyani.pyani_orm.Run attribute), 98
dl_info_to_str() (in module pyani.scripts.subcommands.subcmd_download), 66
DLFileData (class in pyani.download), 86
DLStatus (class in pyani.download), 87
dltype (pyani.scripts.subcommands.subcmd_download.Skipped attribute), 65
download_data() (in module pyani.scripts.subcommands.subcmd_download), 66
download_genome() (in module pyani.scripts.subcommands.subcmd_download), 66
download_genome_and_hash() (in module pyani.download), 88
download_url() (in module pyani.download), 88
draw() (in module pyani.scripts.average_nucleotide_identity), 71

E
etrez_batch() (in module pyani.download), 89
entrez_batch_webhistory() (in module pyani.scripts.genbank_get_genomes_by_taxon), 75
entrez_batched_webhistory() (in module pyani.download), 89
entrez_esearch() (in module pyani.download), 89
entrez_esummary() (in module pyani.download), 89
entrez_retry() (in module pyani.download), 89
entrez_retry() (in module pyani.scripts.genbank_get_genomes_by_taxon), 76
extract_archive() (in module pyani.scripts.genbank_get_genomes_by_taxon), 76
extract_contigs() (in module pyani.download), 89
extract_filestem() (in module pyani.download), 89
extract_filestem() (in module pyani.scripts.genbank_get_genomes_by_taxon), 76
extract_genomes() (in module pyani.scripts.subcommands.subcmd_download), 66
extract_hash() (in module pyani.download), 89
extract_submittable_jobs() (in module pyani.run_sge), 106

F
FileExistsException, 87
filehash (pyani.download.Hashstatus attribute), 87
filestem (pyani.download.DLFileData attribute), 86
filter_existing_comparisons() (in module pyani.pyani_orm), 99
filtercmd (pyani.scripts.subcommands.subcmd_anim.ComparisonJob attribute), 61
format_exe (pyani.pyani_tools.BLASTexes attribute), 103
fragment.fasta_file() (in module pyani.scripts.subcommands.subcmd_anib), 60
fragment.fasta_files() (in module pyani.anib), 80
fragpath (pyani.pyani_orm.BlastDB attribute), 96
fragsize (pyani.pyani_orm.ProgParams attribute), 62
fragsizes (pyani.pyani_orm.BlastDB attribute), 96
generate_blastdb_commands() (in module pyani.anib), 80
generate_blastn_commands() (in module pyani.anib), 81
generate_joblist() (in module pyani.scripts.subcommands.subcmd_anib), 60
generate_joblist() (in module pyani.scripts.subcommands.subcmd_anim), 63
generate_nucmer_commands() (in module pyani.anib), 83
generate_nucmer_jobs() (in module pyani.anib), 84
generate_script() (pyani.pyani_jobs.JobGroup method), 96

G
Genome (class in pyani.pyani_orm), 97
genome (pyani.pyani_orm.Label attribute), 98
genome_hash (pyani.pyani_orm.Genome attribute), 97
genome_id (pyani.pyani_orm.BlastDB attribute), 96
genome_id (pyani.pyani_orm.Genome attribute), 97
genome_id (pyani.pyani_orm.Label attribute), 97
genomes (pyani.pyani_orm.Run attribute), 98
genus (pyani.download.Classification attribute), 86
get_asm_uids() (in module pyani.download), 89
get_asm_uids() (in module pyani.scripts.genbank_get_genomes_by_taxon), 76
get_colormap() (in module pyani.pyani_config), 93
get_comparison_dict() (in module pyani.pyani_orm), 100
get_db_name()  (in module pyani.pyani_tools.BLASTcmds method), 103
get_fasta_and_hash_paths()  (in module pyani.pyani_files), 94
get_fasta_files()  (in module pyani.anim), 84
get_fasta_files()  (in module pyani.pyani_files), 94
get_fasta_paths()  (in module pyani.pyani_files), 94
get_fraglength_dict()  (in module pyani.anib), 81
get_fragment_lengths()  (in module pyani.anib), 81
get_genome_length()  (in module pyani.pyani_tools), 104
get_input_files()  (in module pyani.pyani_files), 94
get_labels()  (in module pyani.pyani_tools), 104
get_matrix_classes_for_run()  (in module pyani.pyani_orm), 100
get_matrix_labels_for_run()  (in module pyani.pyani_orm), 100
get_method()  (in module pyani.scripts.average_nucleotide_identity), 72
get_ncbi_asm()  (in module pyani.scripts.genbank_get_genomes_by_taxon), 76
get_ncbi_classification()  (in module pyani.download), 89
get_ncbi_esummary()  (in module pyani.download), 90
get_sequence_lengths()  (in module pyani.pyani_files), 94
get_session()  (in module pyani.pyani_orm), 100
get_tax_asm_dict()  (in module pyani.scripts.subcommands.subcmd_download), 66
get_version()  (in module pyani.pyani_tools), 103
get_version()  (in module pyani.anim), 84
graph  (in module pyani.scripts.subcommands.subcmd_classify.SubgraphData attribute), 64
graphic_args  (in module pyani.pyani_tools.MatrixData attribute), 104
header_font()  (in module pyani.pyani_report), 101
headers  (in module pyani.scripts.subcommands.subcmd_report.ReportParams attribute), 69
hover_highlight()  (in module pyani.pyani_report), 101
identity  (in module pyani.pyani_orm.Comparison attribute), 97
interval  (in module pyani.scripts.subcommands.subcmd_classify.SubgraphData attribute), 64
Job  (class in module pyani.pyani_jobs), 95
job  (in module pyani.scripts.subcommands.subcmd_anim.ComparisonJob attribute), 61
JobGroup  (class in module pyani.pyani_jobs), 96
k_complete_component_status()  (in module pyani.pyani_classify), 93
Label  (class in module pyani.pyani_orm), 97
label  (in module pyani.pyani_orm.Label attribute), 98
label_id  (in module pyani.pyani_orm.Label attribute), 98
label_results_matrix()  (in module pyani.pyani_tools), 104
labels  (in module pyani.pyani_orm.Genome attribute), 97
labels  (in module pyani.pyani_orm.Run attribute), 98
LabelTuple  (class in module pyani.pyani_orm), 98
last_exception()  (in module pyani.pyani_download), 90
last_exception()  (in module pyani.scripts.average_nucleotide_identity), 72
last_exception()  (in module pyani.scripts.genbank_get_genomes_by_taxon), 77
legacy_blast  (in module pyani.pyani_tools.Dependencies attribute), 104
length  (in module pyani.pyani_orm.Genome attribute), 97
load_classes_labels()  (in module pyani.pyani_files), 94
logreport_downloaded()  (in module pyani.scripts.genbank_get_genomes_by_taxon), 77
localhash  (in module pyani.download.Hashstatus attribute), 87
make_asm_dict()  (in module pyani.pyani_download), 90
make_blastcmd_builder()  (in module pyani.pyani_anib), 81
make_job_graph()  (in module pyani.pyani_anib), 82

H
hadamard  (in module pyani.pyani_tools.ANIResults attribute), 103
has_dependencies()  (in module pyani.pyani_tools), 104
hash_genomes()  (in module pyani.scripts.subcommands.subcmd_download), 67
Hashstatus  (class in module pyani.download), 87

J
Job  (class in module pyani.pyani_jobs), 95
job  (in module pyani.scripts.subcommands.subcmd_anim.ComparisonJob attribute), 61
JobGroup  (class in module pyani.pyani_jobs), 96

K
k_complete_component_status()  (in module pyani.pyani_classify), 93

L
Label  (class in module pyani.pyani_orm), 97
label  (in module pyani.pyani_orm.Label attribute), 98
label_id  (in module pyani.pyani_orm.Label attribute), 98
label_results_matrix()  (in module pyani.pyani_tools), 104
labels  (in module pyani.pyani_orm.Genome attribute), 97
labels  (in module pyani.pyani_orm.Run attribute), 98
LabelTuple  (class in module pyani.pyani_orm), 98
last_exception()  (in module pyani.pyani_download), 90
last_exception()  (in module pyani.scripts.average_nucleotide_identity), 72
last_exception()  (in module pyani.scripts.genbank_get_genomes_by_taxon), 77
legacy_blast  (in module pyani.pyani_tools.Dependencies attribute), 104
length  (in module pyani.pyani_orm.Genome attribute), 97
load_classes_labels()  (in module pyani.pyani_files), 94
logreport_downloaded()  (in module pyani.scripts.genbank_get_genomes_by_taxon), 77
localhash  (in module pyani.download.Hashstatus attribute), 87
make_asm_dict()  (in module pyani.pyani_download), 90
make_blastcmd_builder()  (in module pyani.pyani_anib), 81
make_job_graph()  (in module pyani.pyani_anib), 82

M
make_asm_dict()  (in module pyani.pyani_download), 90
make_blastcmd_builder()  (in module pyani.pyani_anib), 81
make_job_graph()  (in module pyani.pyani_anib), 82
makeoutdir() (in module pyani.scripts), 55
makeoutdir() (in module pyani.scripts.genbank_get_genomes_by_taxon), 77
makeoutdir() (in module pyani.scripts.average_nucleotide_identity), 72
makeoutdir() (in module pyani.scripts.genbank_get_genomes_by_taxon), 77
makeoutdir() (in module pyani.scripts.average_nucleotide_identity), 72
make_sequence_fragments() (in module pyani.scripts.average_nucleotide_identity), 72
MatrixData (class in pyani.pyani_tools), 104
maxmatch (pyani.pyani_orm.Comparison attribute), 97
maxmatch (pyani.scripts.subcommands.subcmd_anim.ProgParams attribute), 62
metadata (pyani.nucmer.DeltaData attribute), 91
method (pyani.pyani_orm.Run attribute), 98
method (pyani.scripts.subcommands.subcmd_anim.RunData attribute), 63
multiprocessing_run() (in module pyani.run_multiprocessing), 105
nummer (pyani.pyani_tools.Dependencies attribute), 104

N
n_nodes (pyani.pyani_classify.Cliquesinfo attribute), 92
n_subgraphs (pyani.pyani_classify.Cliquesinfo attribute), 92
name (pyani.pyani_orm.Run attribute), 98
name (pyani.pyani_orm.Run attribute), 98
name (pyani.scripts.subcommands.subcmd_anim.RunData attribute), 63
name (pyani.scripts.subcommands.subcmd_report.ReportParams attribute), 69
NCBIDownloadException, 75, 87
nucmercmd (pyani.scripts.subcommands.subcmdAnim.ComparisonJob attribute), 61

O
organism (pyani.download.Classification attribute), 86
organism (pyani.scripts.subcommands.subcmd_download.Skipped attribute), 65
outfile (pyani.scripts.subcommands.subcmd_anim.ComparisonJob attribute), 61

P
Params (class in pyani.pyani_graphics), 95
params_mpl() (in module pyani.pyani_config), 93
parse_api_key() (in module pyani.scripts.subcommands.subcmd_download), 67
parse_blast_tab() (in module pyani.anib), 82
parse_blasttab() (in module pyani.blast), 86
parse_cmdline() (in module pyani.scripts.average_nucleotide_identity), 73
parse_cmdline() (in module pyani.scripts.genbank_get_genomes_by_taxon), 77
parse_cmdline() (in module pyani.scripts.parsers), 56
parse_delta() (in module pyani.anib), 84
passed (pyani.download.Hashstatus attribute), 87
path (pyani.pyani_orm.Genome attribute), 97
pid (pyani.scripts.subcommands.subcmd_anim.ComparisonResult attribute), 62
populate_cmdsets() (in module pyani.run_multiprocessing), 105
populate_jobset() (in module pyani.run_sge), 106
process_arguments() (in module pyani.scripts.average_nucleotide_identity), 73
process_blast() (in module pyani.anib), 82
process_deltadir() (in module pyani.anib), 85
process_formats() (in module pyani.scripts.subcommands.subcmd_report), 69

Index
update_comparison_matrices() (in module pyani.pyani_orm), 100
update_comparison_results() (in module pyani.scripts.subcommands.subcmd_anim), 63
url (pyani.scripts.subcommands.subcmd_download.Skipped attribute), 65

V
vdiff (pyani.pyani_graphics.Params attribute), 95
version (pyani.pyani_orm.Comparison attribute), 97
version (pyani.scripts.subcommands.subcmd_anim.ProgData attribute), 62

W
wait() (pyani.pyani_jobs.Job method), 95
wait() (pyani.pyani_jobs.JobGroup method), 96
write() (in module pyani.scripts.average_nucleotide_identity), 75
write_contigs() (in module pyani.scripts.genbank_get_genomes_by_taxon), 78
write_dbtable() (in module pyani.pyani_report), 101
write_distribution() (in module pyani.scripts.subcommands.subcmd_plot), 68
write_heatmap() (in module pyani.scripts.subcommands.subcmd_plot), 68
write_run_heatmaps() (in module pyani.scripts.subcommands.subcmd_plot), 68
write_styled_html() (in module pyani.pyani_report), 102
write_to_stdout() (in module pyani.pyani_report), 102