# pyhmmer

***Release 0.1.0***

**Martin Larralde**

**Dec 01, 2020**

# CONTENTS


**1 Setup** **3**

**2 Library** **5**
- 2.1 Installation . . . 5
- 2.2 Examples . . . 6
- 2.3 API Reference . . . 10
- 2.4 Contributing to pyHMMER . . . 20
- 2.5 Changelog . . . 21

**3 Indices and tables** **25**

**Python Module Index** **27**

**Index** **29**

*Cython bindings and Python interface to* HMMER3.

CHAPTER

ONE

# SETUP

Run `pip install pyhmmer` in a shell to download the latest release and all its dependencies from PyPi, or have a look at the *Installation page* to find other ways to install `pyhmmer`.

CHAPTER

TWO

# LIBRARY

## 2.1 Installation

**Note:** Wheels are provided for Linux x86-64 platforms, but other machines will have to build the wheel from the source distribution. Building `pyhmmer` involves compiling HMMER3 and Easel, which requires a C compiler to be available.

### 2.1.1 PyPi

`pyhmmer` is hosted on GitHub, but the easiest way to install it is to download the latest release from its PyPi repository. It will install all dependencies then install `pyhmmer` either from a wheel if one is available, or from source after compiling the Cython code :

```
$ pip install --user pyhmmer
```

### 2.1.2 EMBL Package Registry

You can also install `manylinux` wheels built from the latest commit that passed the unit tests. Those bleeding-edge releases are available in the GitLab Package Registry hosted on the EMBL `git` server. Just instruct `pip` to use an extra index URL as follow:

```
$ pip install --user pyhmmer --extra-index-url https://git.embl.de/api/v4/projects/
↪3638/packages/pypi/simple
```

### 2.1.3 GitHub + `pip`

If, for any reason, you prefer to download the library from GitHub, you can clone the repository and install the repository by running (with the admin rights):

```
$ pip install --user https://github.com/althonos/pyhmmer/archive/master.zip
```

**Caution:** Keep in mind this will install always try to install the latest commit, which may not even build, so consider using a versioned release instead.

### 2.1.4 GitHub + `setuptools`

If you do not want to use `pip`, you can still clone the repository and run the `setup.py` file manually, although you will need to install the build dependencies (mainly Cython):

```
$ git clone --recursive https://github.com/althonos/pyhmmer
$ cd pyhmmer
$ python setup.py build
# python setup.py install
```

**Danger:** Installing packages without `pip` is strongly discouraged, as they can only be uninstalled manually, and may damage your system.

## 2.2 Examples

### 2.2.1 Active Site Analysis

This example is adapted from the method used by AntiSMASH to annotate biosynthetic gene clusters. AntiSMASH uses profile HMMs to annotate enzymatic domains in protein sequences. By matching the amino acids in the alignment, it can then predict the product specificity of the enzyme.

In this notebook, we show how to reproduce this kind of analysis, using a PKSI Acyltransferase domain built by the AntiSMASH authors (the HMM in HMMER2 format can be downloaded from their git repository).

---

References


- Del Vecchio, F., H. Petkovic, S. G. Kendrew, L. Low, B. Wilkinson, R. Lill, J. Cortes, B. A. Rudd, J. Staunton, and P. F. Leadlay. 2003. *Active-site residue, domain and module swaps in modular polyketide synthases.* J Ind. Microbiol Biotechnol 30:489-494.
- Medema MH, Blin K, Cimermancic P, de Jager V, Zakrzewski P, Fischbach MA, Weber T, Takano E, Breitling R. *antiSMASH: rapid identification, annotation and analysis of secondary metabolite biosynthesis gene clusters in bacterial and fungal genome sequences.* Nucleic Acids Res. 2011 Jul:W339-46.


---

[1]:
```
import pyhmmer
pyhmmer.__version__
```

[1]:
```
'0.1.0'
```

#### Loading the HMM

Loading a HMMER profile is done with the `pyhmmer.plan7.HMMFile` class, which provides an iterator over the HMMs in the file. Since we only use a single HMM, we can simply use `next` to get the first (and only) `pyhmmer.plan7.HMM`.

[2]:
```
with pyhmmer.plan7.HMMFile("data/hmms/txt/PKSI-AT.hmm") as hmm_file:
    hmm = next(hmm_file)
```

### Building digitized sequences

Easel provides the code necessary to load sequences from files in common biological formats, such as GenBank or FASTA. These utilities are wrapped by the `pyhmmer.easel.SequenceFile`, which provides an iterator over the sequences in the file. Note that `SequenceFile` tries to guess the format by default, but you can force a particular format with the `format` keyword argument.

[3]:
```
with pyhmmer.easel.SequenceFile("data/seqs/PKSI.faa") as seq_file:
    sequences = [ seq.digitize(hmm.alphabet) for seq in seq_file ]
```

---

Note

The C interface of Easel allows storing a sequence in two different modes: in *text* mode, where the sequence letters are represented as individual characters (e.g. "A" or "Y"), and *digital* mode, where sequence letters are encoded as digits. To make Python programs clearer, and to allow static typecheck of the storage mode, we provide two separate classes, `TextSequence` and `DigitalSequence`, that represent a sequence stored in either of these modes.

---

`SequenceFile` yields sequences in text mode, but HMMER expects sequences in digital mode, so we must digitize them. This requires the sequence alphabet to be known, but we can just use the `Alphabet` instance stored in the `alphabet` attribute of `hmm`.

### Running a search pipeline

With the sequences and the HMM ready, we can finally run the search pipeline: it has to be initialized with an `Alphabet` instance, so that the Plan7 background model can be configured accordingly. Then, we run the pipeline in search mode, providing it one HMM, and several sequences. This method returns a `TopHits` instance that is already sorted and thresholded.

[4]:
```
pipeline = pyhmmer.plan7.Pipeline(hmm.alphabet)
hits = pipeline.search(hmm, sequences)
```

### Rendering the alignments

`Domain` instances store all the required information to report results in their `alignment` attribute. We can show the alignment between a HMM and a sequence like `hmmsearch` would as follow (using the first domain of the first hit as an example):

[5]:
```
ali = hits[0].domains[0].alignment

print(" "*3, ali.target_name.decode())
print("{:3}".format(ali.hmm_from), ali.hmm_sequence, "{:3}".format(ali.hmm_to))
print(" "*3, ali.identity_sequence)
print("{:3}".format(ali.target_from), ali.target_sequence, "{:3}".format(ali.target_
↪to))
print(" "*3, ali.hmm_name.decode())
```

```
    sp|Q9ZGI5|PIKA1_STRVZ
  1␣
↪lFpGQGsQyaGMGreLYetePVFRqalDrCaaaLrphLgfsLlevLfgdegqeeaaaslLdqTryaQPALFAvEYALArLWrSWGvePdAVlGHSvGEy
↪lpggGaMlaVraseeevrelLapyggrlsiAAvNGPrsvVvSGdaeaieallaeLeaqGirarrLkVsHAFHSplMepmldeleevlagitpraPriPl
↪308
    +FpGQG+Q+aGMG eL++++ VF++a+ +C+aaL+p++++sL +v ++ +g      a+ L++++++QP+ FAv+++LAr␣
↪W+  Gv+P+AV+GHS+GE++AA+vAG+lSL+DA+r+V  R++ ++a l+g+G+Ml+ ++se+ v e+La+++ +ls+AAvNGP␣
↪++VvSGd+ +ie+l++++ea G+rar ++V++A+HS+++e +  el+evlag++p+aPr+P++S++ G+w+t+␣
↪+ld++YW+r+lR+ V Fa+++etL+ + G+t+F+Ev++hpvLt ++  t             + la+Lrr+
```

(continues on next page)

(continued from previous page)

```
635 VFPGQGTQWAGMGAELLDSSAVFAAAMAECEAALSPYVDWSLEAVVRQAPG-----
↪APTLERVDVVQPVTFAVMVSLARVWQHHGVTPQAVVGHSQGEIAAAYVAGALSLDDAARVVTLRSKSIAAhLAGKGGMLSLALSEDAVLERLAGFD-
↪GLSVAAVNGPTATVVSGDPVQIEELARACEADGVRARVIPVDYASHSRQVEIIESELAEVLAGLSPQAPRVPFFSTLEGAWITE-
↪PVLDGGYWYRNLRHRVGFAPAVETLATDEGFTHFVEVSAHPVLTMALPGTV----------TGLATLRRD 925
    PKS-AT.tcoffee
```

You may also want to see where the domains are located in the input sequence; using the DNA feature viewer developed by the Edinburgh Genome Foundry, we can build a summary graph aligning the protein sequences to the same reference axis:

[6]:
```
from dna_features_viewer import GraphicFeature, GraphicRecord
import matplotlib.pyplot as plt

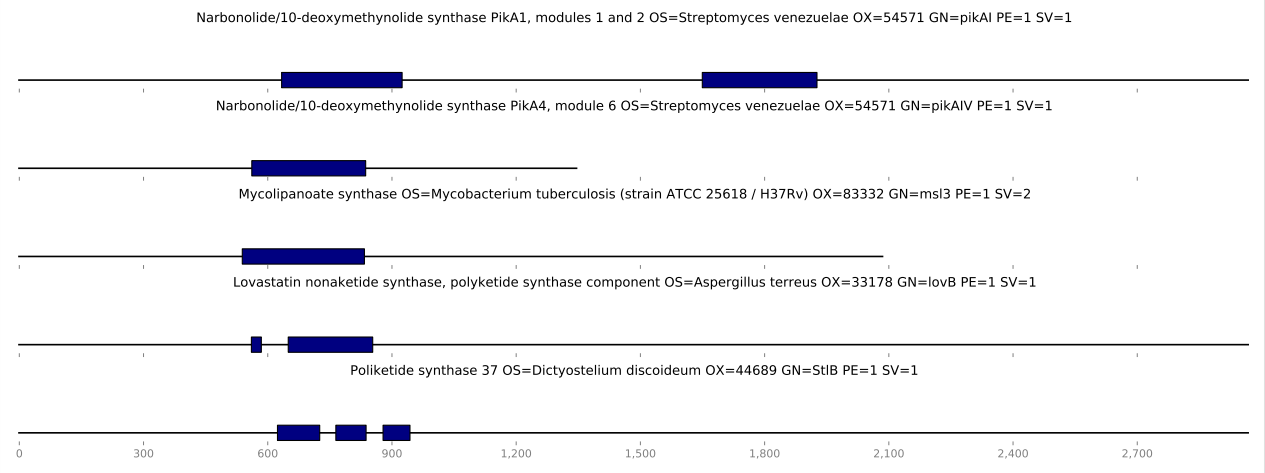
# create an index so we can retrieve a Sequence from its name
seq_index = { seq.name:seq for seq in sequences }

fig, axes = plt.subplots(nrows=len(hits), figsize=(16, 6), sharex=True)
for ax, hit in zip(axes, hits):
    # add one feature per domain
    features = [
        GraphicFeature(start=d.alignment.target_from-1, end=d.alignment.target_to)
        for d in hit.domains
    ]
    length = len(seq_index[hit.name])
    desc = seq_index[hit.name].description.decode()

    # render the feature records
    record = GraphicRecord(sequence_length=length, features=features)
    record.plot(ax=ax)
    ax.set_title(desc)

# make sure everything fits in the final graph!
fig.tight_layout()
```

```
/home/docs/checkouts/readthedocs.org/user_builds/pyhmmer/envs/v0.1.0/lib/python3.7/
↪site-packages/traitlets/traitlets.py:3036: FutureWarning: --rc={'figure.dpi': 96}␣
↪for dict-traits is deprecated in traitlets 5.0. You can pass --rc <key=value> ...␣
↪multiple times to add items to a dict.
  FutureWarning,
```

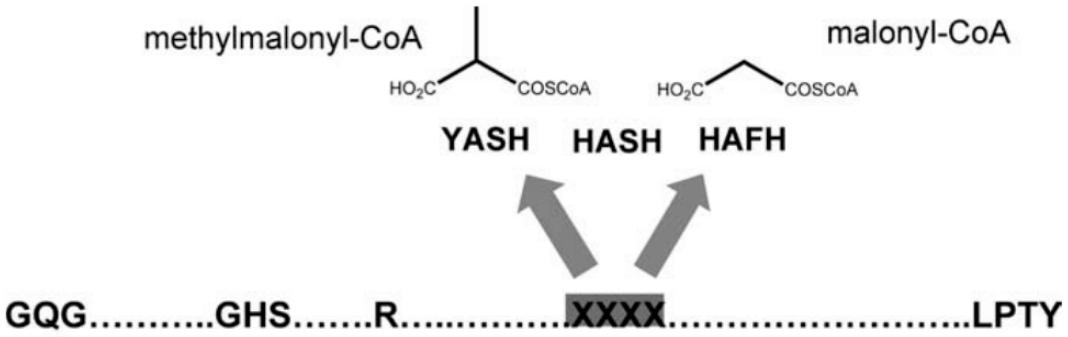


### Checking individual positions for catalytic activity

First let's define a function to iterate over an alignement; this will come in handy later. This function yields the position in the alignment (using the HMM coordinates) and the aligned amino acid, skipping over gaps in the HMM sequence.

[7]:
```
def iter_target_match(alignment):
    position = alignment.hmm_from
    for hmm_letter, amino_acid in zip(alignment.hmm_sequence, alignment.target_
↪sequence):
        if hmm_letter != ".":
            yield position, amino_acid
            position += 1
```

Now, for the final step, we want to check for the specificity of the enzyme domains; Del Vecchio *et al.* have identified two amino acids in the acyltransferase domain that once muted will decide of the enzyme specificity for either malonyl-CoA or methylmalonyl-CoA:




For this, we need to check the alignment produced by HMMER, and verify the residues of the catalytic site correspond to the ones expected by the authors. We use the function we defined previously, first to check the core amino acids are not muted, and then to check the specificity of the two remaining residues.

[8]:
```
POSITIONS   = [ 93,  94,  95, 120, 196, 198]
EXPECTED    = ['G', 'H', 'S', 'R', 'A', 'H']
SPECIFICITY = [195, 197]

for hit in hits:
    print("\nIn sequence {!r}:".format(hit.name.decode()))
    for domain in hit.domains:
        aligned = dict(iter_target_match(domain.alignment))

        print("- Found PKSI-AT domain at positions {:4} to {:4}".format(domain.ali_
↪from, domain.ali_to))
        try:
            signature = [ aligned[x] for x in POSITIONS ]
            spec = [ aligned[x] for x in SPECIFICITY ]
        except KeyError:
            print("  -> Domain likely too short")
            continue
        if signature != EXPECTED:
            print("  -> Substrate specificity unknown")
        elif spec == ["H", "F"]:
            print("  -> Malonyl-CoA specific")
        elif spec == ["Y", "S"]:
```

(continues on next page)

(continued from previous page)

```
            print("  -> Methylmalonyl-CoA specific")
        else:
            print("  -> Neither malonyl-CoA nor methylmalonyl-CoA specific")
```

```
In sequence 'sp|Q9ZGI5|PIKA1_STRVZ':
- Found PKSI-AT domain at positions  635 to  925
  -> Methylmalonyl-CoA specific
- Found PKSI-AT domain at positions 1651 to 1927
  -> Methylmalonyl-CoA specific
- Found PKSI-AT domain at positions 3181 to 3475
  -> Malonyl-CoA specific

In sequence 'sp|Q9ZGI2|PIKA4_STRVZ':
- Found PKSI-AT domain at positions  563 to  837
  -> Methylmalonyl-CoA specific

In sequence 'sp|A0A089QRB9|MSL3_MYCTU':
- Found PKSI-AT domain at positions  540 to  834
  -> Neither malonyl-CoA nor methylmalonyl-CoA specific

In sequence 'sp|Q9Y8A5|LOVB_ASPTE':
- Found PKSI-AT domain at positions  562 to  585
  -> Domain likely too short
- Found PKSI-AT domain at positions  651 to  854
  -> Neither malonyl-CoA nor methylmalonyl-CoA specific

In sequence 'sp|Q54FI3|STLB_DICDI':
- Found PKSI-AT domain at positions  625 to  726
  -> Domain likely too short
- Found PKSI-AT domain at positions  766 to  838
  -> Domain likely too short
- Found PKSI-AT domain at positions  880 to  944
  -> Domain likely too short
```

## 2.3 API Reference

### 2.3.1 Errors

Common errors and status codes for the `easel` and `hmmer` modules.

**exception** pyhmmer.errors.**AllocationError**(*MemoryError*)
A memory error that is caused by an unsuccessful allocation.

**exception** pyhmmer.errors.**UnexpectedError**(*RuntimeError*)
An unexpected error that happened in the C code.

As a user of this library, you should never see this exception being raised. If you do, please open an issue with steps to reproduce on the bug tracker, so that proper error handling can be added to the relevant part of the bindings.

**exception** pyhmmer.errors.**EaselError**(*RuntimeError*)
An error that was raised from the Easel code.

### 2.3.2 Easel

High-level interface to the Easel C library.

Easel is a library developed by the Eddy/Rivas Lab to facilitate the development of biological software in C. It is used by HMMER and Infernal.

#### Alphabet

**class** pyhmmer.easel.**Alphabet**

A biological alphabet, including additional marker symbols.

This type is used to share an alphabet to several objects in the `easel` and `plan7` modules. Reference counting helps sharing the same instance everywhere, instead of reallocating memory every time an alphabet is needed.

Use the factory class methods to obtain a default *Alphabet* for one of the three biological alphabets:

```
>>> dna = Alphabet.dna()
>>> rna = Alphabet.rna()
>>> aa  = Alphabet.amino()
```

**amino**()

Create a default amino-acid alphabet.

**dna**()

Create a default DNA alphabet.

**rna**()

Create a default RNA alphabet.

**K**

The alphabet size, counting only actual alphabet symbols.

---

**Example**

```
>>> Alphabet.dna().K
4
>>> Alphabet.amino().K
20
```

---

**Type** int

**Kp**

The complete alphabet size, including marker symbols.

---

**Example**

```
>>> Alphabet.dna().Kp
18
>>> Alphabet.amino().Kp
29
```

---

**Type** int

**symbols**
The symbols composing the alphabet.

**Example**

```
>>> Alphabet.dna().symbols
'ACGT-RYMKSWHBVDN*~'
>>> Alphabet.rna().symbols
'ACGU-RYMKSWHBVDN*~'
```

**Type** str

## Bitfield

**class** pyhmmer.easel.**Bitfield**
A statically sized sequence of booleans stored as a packed bitfield.

A bitfield is instantiated with a fixed length, and all booleans are set to `False` by default:

```
>>> bitfield = Bitfield(8)
>>> len(bitfield)
8
>>> bitfield[0]
False
```

**count**()
Count the number occurrences of `value` in the bitfield.

If no argument is given, counts the number of `True` occurences.

**Example**

```
>>> bitfield = Bitfield(8)
>>> bitfield.count(False)
8
>>> bitfield[0] = bitfield[1] = True
>>> bitfield.count()
2
```

**toggle**()
Switch the value of one single bit.

**Example**

```
>>> bitfield = Bitfield(8)
>>> bitfield[0]
False
>>> bitfield.toggle(0)
>>> bitfield[0]
True
>>> bitfield.toggle(0)
>>> bitfield[0]
False
```

### KeyHash

**class** pyhmmer.easel.**KeyHash**
A dynamically resized container to store string keys using a hash table.

### Multiple Sequence Alignment

**class** pyhmmer.easel.**MSA**
An abstract alignment of multiple sequences.

**checksum**()
Calculate a 32-bit checksum for the multiple sequence alignment.

**class** pyhmmer.easel.**TextMSA**(*MSA*)
A multiple sequence alignement stored in text mode.

**copy**()
Duplicate the text sequence alignment, and return the copy.

**digitize**()
Convert the text alignment to a digital alignment using `alphabet`.

**class** pyhmmer.easel.**DigitalMSA**(*MSA*)
A multiple sequence alignment stored in digital mode.

**alphabet**
The biological alphabet used to encode this sequence alignment to digits.

**Type** *Alphabet*, *readonly*

**copy**()
Duplicate the digital sequence alignment, and return the copy.

### Sequence

**class** pyhmmer.easel.**Sequence**
An abstract biological sequence with some associated metadata.

Easel provides two different mode to store a sequence: text, or digital. In the HMMER code, changing from one mode to another mode is done in place, which allows recycling memory. However, doing so can be confusing since there is no way to know statically the representation of a sequence.

To avoid this, `pyhmmer` provides two subclasses of the *Sequence* abstract class to maintain the mode contract: *TextSequence* and *DigitalSequence*. Functions expecting sequences in digital format, like `pyhmmer.hmmsearch`, can then use Python type system to make sure they receive sequences in the right mode. This allows type checkers such as `mypy` to detect potential contract breaches at compile-time.

**checksum**()
Calculate a 32-bit checksum for the sequence.

**clear**()
Reinitialize the sequence for re-use.

**accession**
The accession of the sequence, if any.

**Type** `str` or `None`

**description**
The description of the sequence.

**Type** bytes

**name**
The name of the sequence.

**Type** bytes

**source**
The source of the sequence, if any.

**Type** bytes

**class** pyhmmer.easel.**TextSequence**(*Sequence*)
A biological sequence stored in text mode.

**copy**()
Duplicate the text sequence, and return the copy.

**digitize**()
Convert the text sequence to a digital sequence using `alphabet`.

**class** pyhmmer.easel.**DigitalSequence**(*Sequence*)
A biological sequence stored in digital mode.

Currently, objects from this class cannot be instantiated directly. Use *TextSequence.digitize* to obtain a digital sequence from a sequence in text mode.

**alphabet**
The biological alphabet used to encode this sequence to digits.

**Type** *Alphabet*, *readonly*

**copy**()
Duplicate the digital sequence, and return the copy.

### Sequence File

**class** pyhmmer.easel.**SequenceFile**
A wrapper around a sequence file, containing unaligned sequences.

This class supports reading sequences stored in different formats, such as FASTA, GenBank or EMBL. The format of each file can be automatically detected, but it is also possible to pass an explicit format specifier when the *SequenceFile* is instantiated.

**guess_alphabet**()
Guess the alphabet of an open *SequenceFile*.

This method tries to guess the alphabet of a sequence file by inspecting the first sequence in the file. It returns the alphabet, or `None` if the file alphabet cannot be reliably guessed.

**Raises**

- **EOFError** – if the file is empty.
- **OSError** – if a parse error occurred.
- **ValueError** – if this methods is called after the file was closed.

**read**()
Read the next sequence from the file.

**Parameters**

- **skip_info** (bool) – Pass True to disable reading the sequence *metadata*, and only read the sequence *letters*. Defaults to False`.
- **skip_sequence** (bool) – Pass True to disable reading the sequence *letters*, and only read the sequence *metadata*. Defaults to False.

**Returns** *Sequence* – The next sequence in the file, or None if all sequences were read from the file.

**Raises** **ValueError** – When attempting to read a sequence from a closed file, or when the file could not be parsed.

---

**Hint:** This method allocates a new sequence, which is not efficient in case the sequences are being read within a tight loop. Use *SequenceFile.readinto* with an already initialized *Sequence* if you can to recycle the internal buffers.

---

**readinto**()

Read the next sequence from the file, using seq to store data.

**Parameters**

- **seq** (*Sequence*) – A sequence object to use to store the next entry in the file. If this sequence was used before, it must be properly reset (using the *Sequence.clear* method) before using it again with *readinto*.
- **skip_info** (bool) – Pass True to disable reading the sequence *metadata*, and only read the sequence *letters*. Defaults to False`.
- **skip_sequence** (bool) – Pass True to disable reading the sequence *letters*, and only read the sequence *metadata*. Defaults to False.

**Returns** *Sequence* – A reference to seq that was passed as an input, or None if no sequences are left in the file.

**Raises** **ValueError** – When attempting to read a sequence from a closed file, or when the file could not be parsed.

---

**Example**

Use *SequenceFile.readinto* to loop over the sequences in a file while recycling the same *Sequence* buffer:

```
>>> with SequenceFile("vendor/hmmer/testsuite/ecori.fa") as sf:
...     seq = TextSequence()
...     while sf.readinto(seq) is not None:
...         # ... process seq here ... #
...         seq.clear()
```

---

**set_digital**()

Set the *SequenceFile* to read in digital mode with alphabet.

This method can be called even after the first sequences have been read; it only affects subsequent sequences in the file.

#### Sequence / Subsequence Index

**class** pyhmmer.easel.**SSIReader**
A read-only handler for sequence/subsequence index file.

**class Entry**(*fd*, *record_offset*, *data_offset*, *record_length*)

**property data_offset**
Alias for field number 2

**property fd**
Alias for field number 0

**property record_length**
Alias for field number 3

**property record_offset**
Alias for field number 1

**class FileInfo**(*name*, *format*)

**property format**
Alias for field number 1

**property name**
Alias for field number 0

**class** pyhmmer.easel.**SSIWriter**
A writer for sequence/subsequence index files.

**add_file**()
Add a new file to the index.

**Parameters**

- **filename** (*str*) – The name of the file to register.
- **format** (*int*) – A format code to associate with the file, or *0*.

**Returns** int – The filehandle associated with the new indexed file.

### 2.3.3 Plan7

High-level interface to the Plan7 data model.

Plan7 is the model architecture used by HMMER since HMMER2.

**See also:**

Details about the Plan 7 architecture in the HMMER documentation.

### Alignment

**class** pyhmmer.plan7.**Alignment**
A single alignment of a sequence to a profile.

### Background Model

**class** pyhmmer.plan7.**Background**
The null background model of HMMER.

### Domains

**class** pyhmmer.plan7.**Domain**
A single domain in a query *Hit*.

**score**
The overall score in bits, null-corrected.

**Type** float

**class** pyhmmer.plan7.**Domains**
A sequence of domains corresponding to a single *Hit*.

### Hits

**class** pyhmmer.plan7.**Hit**
A high-scoring database hit found by the comparison pipeline.

**pre_score**
Bit score of the sequence before null2 correction.

**Type** float

**score**
Bit score of the sequence with all domains after correction.

**Type** float

**class** pyhmmer.plan7.**TopHits**
A ranked list of top-scoring hits.

*TopHits* are thresholded using the parameters from the pipeline, and are sorted by key when you obtain them from a *Pipeline* instance:

```
>>> abc = thioesterase.alphabet
>>> hits = Pipeline(abc).search(thioesterase, proteins)
>>> hits.is_sorted()
True
```

Use len to query the number of top hits, and the usual indexing notation to extract a particular hit:

```
>>> len(hits)
1
>>> hits[0].name
b'938293.PRJEB85.HG003687_113'
```

**clear**()
Free internals to allow reusing for a new pipeline run.

### HMM

**class** pyhmmer.plan7.**HMM**
A data structure storing the Plan7 Hidden Markov Model.

**write**()
Write the HMM to a file handle.

**Parameters**

- **fh** (io.IOBase) – A Python file handle, opened in binary mode (this must be the case even with `binary=False`, since the C code will emit bytes in either case).
- **binary** (bool) – Pass `False` to emit the file in ASCII mode using the latest supported HMMER format, or `True` to use the binary HMMER3 format.

**zero**()
Set all parameters to zero (including model composition).

**M**
The length of the model (i.e. the number of nodes).

**Type** int

**accession**
The accession of the HMM, if any.

**Type** bytes or None

**description**
The description of the HMM, if any.

**Type** bytes or None

**name**
The name of the HMM, if any.

**Type** bytes or None

### HMM File

**class** pyhmmer.plan7.**HMMFile**
A wrapper around a file (or database), storing serialized HMMs.

**close**()
Close the HMM file and free resources.

This method has no effect if the file is already closed.

### Pipeline

**class** pyhmmer.plan7.**Pipeline**
An HMMER3 accelerated sequence/profile comparison pipeline.

**search**()
Run the pipeline using a query HMM against a sequence database.

**Parameters**

- **hmm** (*HMM*) – The HMM object to use to query the sequence database.
- **sequences** (Iterable of *DigitalSequence*) – The sequences to query with the HMMs. Pass a SequenceFile instance in digital mode to iteratively read from disk.
- **hits** (*TopHits*, optional) – A hit collection to store results in, if any. If None, a new collection will be allocated. Use a non-None value if you are running several HMMs sequentially and want to aggregate the results, or if you are able to recycle a *TopHits* instance with *TopHits.clear*.

**Returns** *TopHits* – the hits found in the sequence database. If an instance was passed via the hits argument, it is safe to ignore the return value and to access it by reference.

**Raises** **ValueError** – When the alphabet of the current pipeline does not match the alphabet of the given HMM.

### Profile

**class** pyhmmer.plan7.**Profile**
A Plan7 search profile.

**clear**()
Clear internal buffers to reuse the profile without reallocation.

**configure**()
Configure a search profile using the given models.

**Parameters**

- **hmm** (*pyhmmer.plan7.HMM*) – The model HMM with core probabilities.
- **bg** (*pyhmmer.plan7.Background*) – The null background model.
- **L** (int) – The expected target sequence length.
- **multihit** (bool) – Whether or not to use multihit modes.
- **local** (bool) – Whether or not to use non-local modes.

**is_local**()
Returns whether or not the profile is in a local alignment mode.

**is_multihit**()
Returns whether or not the profile is in a multihit alignment mode.

**optimized**()
Convert the profile to a platform-specific optimized profile.

**class** pyhmmer.plan7.**OptimizedProfile**
An optimized profile that uses platform-specific instructions.

**is_local**()
Returns whether or not the profile is in a local alignment mode.

**write**()
Write an optimized profile to two separate files.

HMMER implements an acceleration pipeline using several scoring algorithms. Parameters for MSV (the *Multi ungapped Segment Viterbi*) are saved independently to the `fh_filter` handle, while the rest of the profile is saved to `fh_profile`.

Cython bindings and Python interface to HMMER3.

HMMER is a biological sequence analysis tool that uses profile hidden Markov models to search for sequence homologs. HMMER3 is maintained by members of the the Eddy/Rivas Laboratory at Harvard University.

`pyhmmer` is a module, implemented using the Cython language, that provides bindings to HMMER3. It directly interacts with the HMMER internals, which has several advantages over CLI wrappers like hmmer-py.

## 2.4 Contributing to pyHMMER

For bug fixes or new features, please file an issue before submitting a pull request. If the change isn't trivial, it may be best to wait for feedback.

### 2.4.1 Setting up a local repository

Make sure you clone the repository in recursive mode, so you also get the wrapped code of Easel and HMMER, which are exposed as `git` submodules:

```
$ git clone --recursive https://github.com/althonos/pyhmmer
```

### 2.4.2 Running tests

Tests are written as usual Python unit tests with the `unittest` module of the standard library. Running them requires the extension to be built locally:

```
$ python setup.py build_ext --debug --inplace
$ python -m unittest discover -vv
```

### 2.4.3 Coding guidelines

This project targets Python 3.5 or later.

Python objects should be typed; since it is not supported by Cython, you must manually declare types in type stubs (`.pyi` files). In Python files, you can add type annotations to function signatures (supported in Python 3.5) but not in variable assignments (supported from Python 3.6 onward).

#### Interfacing with C

When interfacing with C, and in particular with pointers, use assertions everywhere you assume the pointer to be non-NULL.

## 2.5 Changelog

All notable changes to this project will be documented in this file.

The format is based on Keep a Changelog and this project adheres to Semantic Versioning.

### 2.5.1 Unreleased

### 2.5.2 v0.1.0 - 2020-12-01

Initial beta release.

#### Fixed

- `TextSequence` uses the sequence argument it's given on instantiation.
- Segmentation fault in `Sequence.__eq__` caused by implicit type conversion.
- Segmentation fault on `SequenceFile.read` failure.
- Missing type annotations for the `pyhmmer.easel` module.

### 2.5.3 v0.1.0-a5 - 2020-11-28

#### Added

- `Sequence.__len__` magic method so that `len(seq)` returns the number of letters in `seq`.
- Python file-handle support when opening an `pyhmmer.plan7.HMMFile`.
- Context manager protocol to `pyhmmer.easel.SSIWriter`.
- Type annotations for `pyhmmer.easel.SSIWriter`.
- `add_alias` to `pyhmmer.easel.SSIWriter`.
- `write` method to `pyhmmer.plan7.OptimizedProfile` to write an optimized profile in binary format.
- `offsets` property to interact with the disk offsets of a `pyhmmer.plan7.OptimizedProfile` instance.
- `pyhmmer.hmmer.hmmpress` emulating the `hmmpress` binary from HMMER.
- `M` property to `pyhmmer.plan7.HMM` exposing the number of nodes in the model.

#### Changed

- Bumped vendored Easel to `v0.48`.
- Bumped vendored HMMER to `v3.3.2`.
- `pyhmmer.plan7.HMMFile` will raise an `EOFError` when given an empty file.
- Renamed `length` property to `L` in `pyhmmer.plan7.Background`.

#### Fixed

- Segmentation fault when `close` method of `pyhmmer.easel.SSIWriter` was called more than once.
- `close` method of `pyhmmer.easel.SSIWriter` not writing the index contents.

### 2.5.4 v0.1.0-a4 - 2020-11-24

#### Added

- `MSA`, `TextMSA` and `DigitalMSA` classes representing a multiple sequence alignment to `pyhmmer.easel`.
- Methods and protocol to copy a `Sequence` and a `MSA`.
- `pyhmmer.plan7.OptimizedProfile` wrapping a platform-specific optimized profile.
- `SSIReader` and `SSIWriter` classes interacting with sequence/subsequence indices to `pyhmmer.easel`.
- Exception handler using Python exceptions to report Easel errors.

#### Changed

- `pyhmmer.hmmsearch` returns an iterator of `TopHits`, with one instance per `HMM` in the input.
- `pyhmmer.hmmsearch` properly raises errors happenning in the background threads without deadlock.
- `pyhmmer.plan7.Pipeline` recycles memory between `Pipeline.search` calls.

#### Fixed

- Missing type annotations for the `pyhmmer.errors` module.

#### Removed

- Unneeded or private methods from `pyhmmer.plan7`.

### 2.5.5 v0.1.0-a3 - 2020-11-19

#### Added

- `TextSequence` and `DigitalSequence` representing a `Sequence` in a given mode.
- E-value properties to `Hit` and `Domain`.
- `TopHits` now stores a reference to the pipeline it was obtained from.
- `Pipeline.Z` and `Pipeline.domZ` properties.
- Experimental pickling support to `Alphabet`.
- Experimental freelist to `Sequence` class to avoid allocation bottlenecks when iterating on a `SequenceFile` without recycling sequence buffers.

#### Changed

- Made `Sequence` an abstract base class.
- Additional `Pipeline` parameters can be passed as keyword arguments to `pyhmmer.hmmsearch`.
- `SequenceFile.read` can now be configured to skip reading the metadata or the content of a sequence.

#### Removed

- Redundant `SequenceFile` methods.

#### Fixed

- `doctest` loader crashing on Python 3.5.
- `TopHits.threshold` segfaulting when being called without prior `Tophits.sort` call
- Unknown `format` argument to `SequenceFile` constructor not raising the right error.

### 2.5.6 v0.1.0-a2 - 2020-11-12

#### Added

- Support for compilation on PowerPC big-endian platforms.
- Type annotations and stub files for Cython modules.

#### Changed

- `distutils` is now used to compile the package, instead of calling `autotools` and letting HMMER configure itself.
- `Bitfield.count` now allows passing an argument (for compatibility with `collections.abc.Sequence`).

### 2.5.7 v0.1.0-a1 - 2020-11-10

Initial alpha release (test deployment to PyPI).

CHAPTER

THREE

# INDICES AND TABLES

- genindex
- modindex
- search

# PYTHON MODULE INDEX

## p

pyhmmer, 20
pyhmmer.easel, 11
pyhmmer.errors, 10
pyhmmer.plan7, 16

# INDEX


## A

accession (*pyhmmer.easel.Sequence attribute*), 13
accession (*pyhmmer.plan7.HMM attribute*), 18
add_file() (*pyhmmer.easel.SSIWriter method*), 16
Alignment (*class in pyhmmer.plan7*), 17
AllocationError, 10
Alphabet (*class in pyhmmer.easel*), 11
alphabet (*pyhmmer.easel.DigitalMSA attribute*), 13
alphabet (*pyhmmer.easel.DigitalSequence attribute*), 14
amino() (*pyhmmer.easel.Alphabet method*), 11

## B

Background (*class in pyhmmer.plan7*), 17
Bitfield (*class in pyhmmer.easel*), 12

## C

checksum() (*pyhmmer.easel.MSA method*), 13
checksum() (*pyhmmer.easel.Sequence method*), 13
clear() (*pyhmmer.easel.Sequence method*), 13
clear() (*pyhmmer.plan7.Profile method*), 19
clear() (*pyhmmer.plan7.TopHits method*), 17
close() (*pyhmmer.plan7.HMMFile method*), 18
configure() (*pyhmmer.plan7.Profile method*), 19
copy() (*pyhmmer.easel.DigitalMSA method*), 13
copy() (*pyhmmer.easel.DigitalSequence method*), 14
copy() (*pyhmmer.easel.TextMSA method*), 13
copy() (*pyhmmer.easel.TextSequence method*), 14
count() (*pyhmmer.easel.Bitfield method*), 12

## D

data_offset() (*pyhmmer.easel.SSIReader.Entry property*), 16
description (*pyhmmer.easel.Sequence attribute*), 13
description (*pyhmmer.plan7.HMM attribute*), 18
DigitalMSA (*class in pyhmmer.easel*), 13
DigitalSequence (*class in pyhmmer.easel*), 14
digitize() (*pyhmmer.easel.TextMSA method*), 13
digitize() (*pyhmmer.easel.TextSequence method*), 14
dna() (*pyhmmer.easel.Alphabet method*), 11
Domain (*class in pyhmmer.plan7*), 17
Domains (*class in pyhmmer.plan7*), 17

## E

EaselError, 10

## F

fd() (*pyhmmer.easel.SSIReader.Entry property*), 16
format() (*pyhmmer.easel.SSIReader.FileInfo property*), 16

## G

guess_alphabet() (*pyhmmer.easel.SequenceFile method*), 14

## H

Hit (*class in pyhmmer.plan7*), 17
HMM (*class in pyhmmer.plan7*), 18
HMMFile (*class in pyhmmer.plan7*), 18

## I

is_local() (*pyhmmer.plan7.OptimizedProfile method*), 19
is_local() (*pyhmmer.plan7.Profile method*), 19
is_multihit() (*pyhmmer.plan7.Profile method*), 19

## K

K (*pyhmmer.easel.Alphabet attribute*), 11
KeyHash (*class in pyhmmer.easel*), 13
Kp (*pyhmmer.easel.Alphabet attribute*), 11

## M

M (*pyhmmer.plan7.HMM attribute*), 18
module
    pyhmmer, 20
    pyhmmer.easel, 11
    pyhmmer.errors, 10
    pyhmmer.plan7, 16
MSA (*class in pyhmmer.easel*), 13

## N

name (*pyhmmer.easel.Sequence attribute*), 14

name (*pyhmmer.plan7.HMM attribute*), 18
name() (*pyhmmer.easel.SSIReader.FileInfo property*), 16

## O

optimized() (*pyhmmer.plan7.Profile method*), 19
OptimizedProfile (*class in pyhmmer.plan7*), 19

## P

Pipeline (*class in pyhmmer.plan7*), 19
pre_score (*pyhmmer.plan7.Hit attribute*), 17
Profile (*class in pyhmmer.plan7*), 19
pyhmmer
    module, 20
pyhmmer.easel
    module, 11
pyhmmer.errors
    module, 10
pyhmmer.plan7
    module, 16

## R

read() (*pyhmmer.easel.SequenceFile method*), 14
readinto() (*pyhmmer.easel.SequenceFile method*), 15
record_length() (*pyhmmer.easel.SSIReader.Entry property*), 16
record_offset() (*pyhmmer.easel.SSIReader.Entry property*), 16
rna() (*pyhmmer.easel.Alphabet method*), 11

## S

score (*pyhmmer.plan7.Domain attribute*), 17
score (*pyhmmer.plan7.Hit attribute*), 17
search() (*pyhmmer.plan7.Pipeline method*), 19
Sequence (*class in pyhmmer.easel*), 13
SequenceFile (*class in pyhmmer.easel*), 14
set_digital() (*pyhmmer.easel.SequenceFile method*), 15
source (*pyhmmer.easel.Sequence attribute*), 14
SSIReader (*class in pyhmmer.easel*), 16
SSIReader.Entry (*class in pyhmmer.easel*), 16
SSIReader.FileInfo (*class in pyhmmer.easel*), 16
SSIWriter (*class in pyhmmer.easel*), 16
symbols (*pyhmmer.easel.Alphabet attribute*), 11

## T

TextMSA (*class in pyhmmer.easel*), 13
TextSequence (*class in pyhmmer.easel*), 14
toggle() (*pyhmmer.easel.Bitfield method*), 12
TopHits (*class in pyhmmer.plan7*), 17

## U

UnexpectedError, 10

## W

write() (*pyhmmer.plan7.HMM method*), 18
write() (*pyhmmer.plan7.OptimizedProfile method*), 19

## Z

zero() (*pyhmmer.plan7.HMM method*), 18