

PyCogent Cookbook draft

Patrick Yannul, Kristian Rother, Gavin Huttley, Rob Knight

This is a set of working examples for using of the PyCogent bioinformatics Python library (pycogent.sf.net), which I wanted to see on the web.

1. Handling Sequences

1.1 Create a DNA sequence object

```
>>> from cogent import DNA
>>> my_seq = DNA.makeSequence("AGTACACTGGT")
>>> my_seq.CodonAlphabet()
('CTT', 'ACC', 'ACA', 'ACG', 'ATC', 'ATA', 'AGG', 'CCT', 'AGC',
'AGA', 'ATT', 'CTG', 'CTA', 'ACT', 'CCG', 'AGT', 'CCA', 'CCC',
'TAT', 'GGT', 'CGA', 'CGC', 'CGG', 'GGG', 'GGA', 'GGC', 'TAC',
'CGT', 'GTA', 'GTC', 'GTG', 'GAG', 'GTT', 'GAC', 'ATG', 'AAG',
'AAA', 'AAC', 'CTC', 'CAT', 'AAT', 'CAC', 'CAA', 'CAG', 'TGT',
'TCT', 'GAT', 'TTT', 'TGC', 'TGG', 'TTC', 'TCG', 'TTA', 'TTG',
'TCC', 'GAA', 'TCA', 'GCA', 'GCC', 'GCG', 'GCT')
```

```
>>> my_seq
DnaSequence(AGTACAC... 11)

>>> print my_seq
AGTACACTGGT

>>> str(my_seq)
'AGTACACTGGT'
```

1.2 Inverting & transcribing a DNA sequence

```
>>> from cogent import DNA
>>> my_seq = DNA.makeSequence("AGTACACTGGT")
>>> print my_seq.complement()
TCATGTGACCA

>>> print my_seq.reversecomplement()
ACCAGTGTACT

>>> print my_seq.toRna()
AGUACACUGGU
```

1.3 Comparing complementarity:

```
>>> from cogent import DNA
>>> a = DNA.makeSequence("AGTACACTGGT")
>>> a.canPair(a.complement())
False
```

```
>>> a.canPair(a.reversecomplement())
True
```

1.4 Joining two DNA sequences

```
>>> from cogent import DNA
>>> my_seq = DNA.makeSequence("AGTACACTGGT")
>>> a = DNA.makeSequence("AGTACACTGGT")
>>> print my_seq + a
AGTACACTGGTAGTACACTGGT
```

1.5 Slicing DNA sequences

```
>>> from cogent import DNA
>>> my_seq = DNA.makeSequence("AGTACACTGGT")
>>> print my_seq[1:6]
GTACA
```

1.6 Converting to Fasta format

```
>>> from cogent import DNA
>>> my_seq = DNA.makeSequence("AGTACACTGGT")
>>> print my_seq.toFasta()
>0
AGTACACTGGT
```

1.7 Changing the Name of a sequence

```
>>> from cogent import DNA
>>> my_seq = DNA.makeSequence("AGTACACTGGT")
>>> my_seq = 'my_gene'
>>> print my_seq.toFasta()
>my_gene
AGTACACTGGT
```

1.8 Creating a protein sequence

```
>>> from cogent import PROTEIN
>>> p = PROTEIN.makeSequence('THISISAPRQTEINSEQUENCE')
```

1.9 Creating a sequence with a name

```
>>> from cogent import PROTEIN
>>> p = PROTEIN.makeSequence('THISISAPRQTEIN', 'myProtein')
```

1.10 Creating a general sequence object

```
>>> from cogent.core.sequence import Sequence
>>> seq_without_name = Sequence('ACDEF', 'Name')
>>> print seq.toFasta()
```

2. Parsing files with many sequences

Example files:

http://biopython.org/DIST/docs/tutorial/examples/ls_orchid.fasta

http://biopython.org/DIST/docs/tutorial/examples/ls_orchid.gbk

2.1 Reading a FASTA file with DNA sequences

```
>>> from cogent import LoadSeqs
>>> seq = LoadSeqs('ls_orchid.fasta', aligned=False)
>>> print seq
<cogent.core.alignment.SequenceCollection object at ...>
```

2.2 Loading protein sequences in a FASTA file

```
>>> from cogent import LoadSeqs
>>> seq = LoadSeqs('file_name', moltype=PROTEIN, format='FASTA',
aligned=False)
```

2.3 Loading FASTA sequences from an open file or list of lines

```
>>> from cogent.parse.fasta import FastaParser
>>> f=open('BcnI.fasta')
>>> seq = [sequence for sequence in FastaParser(f)]
```

2.4 Loading DNA sequences from a GenBank file

```
>>> from cogent import LoadSeqs
>>> seq = LoadSeqs('ls_orchid.gbk', aligned=False)
>>> seq.items()[0]
('Z78533', ByteSequence(CGTAACA... 740))
```

2.5 Accessing single sequences in a SequenceCollection object

```
>>> from cogent import LoadSeqs
>>> seq = LoadSeqs('ls_orchid.gbk', aligned=False)
>>> seq.items()[0]
('gi|2765658|emb|Z78533.1|CIZ78533 C.irapeanum 5.8S rRNA gene and
ITS1 and ITS2 DNA', ByteSequence(CGTAACA... 740))

>>> seq.items()[1]
('gi|2765657|emb|Z78532.1|CCZ78532 C.californicum 5.8S rRNA gene
and ITS1 and ITS2 DNA', ByteSequence(CGTAACA... 753))

>>> seq.items()[0:2]
[(('gi|2765658|emb|Z78533.1|CIZ78533 C.irapeanum 5.8S rRNA gene and
ITS1 and ITS2 DNA', ByteSequence(CGTAACA... 740)), ('gi|2765657|
emb|Z78532.1|CCZ78532 C.californicum 5.8S rRNA gene and ITS1 and
ITS2 DNA', ByteSequence(CGTAACA... 753))]
```

2.6 Using a SequenceCollection like a dictionary

```
>>> from cogent import LoadSeqs
>>> seq = LoadSeqs('ls_orchid.gbk', aligned=False)
>>> seq.keys()
..
>>> seq.values()
..
>>> seq.items()
..
>>> k = seq.keys[0]
>>> print k
..
>>> seq.getSeq(k)
```

2.7 Converting a SequenceCollection to FASTA format

```
>>> from cogent import LoadSeqs
>>> seq = LoadSeqs('ls_orchid.gbk', aligned=False)
>>> fasta_data = seq.toFasta()
```

2.8 Creating a SequenceCollection object from a FASTA string

```
>>> from cogent.core.alignment import SequenceCollection
>>> sc = SequenceCollection(fasta_data)
```

3. Parsing sequence alignments

3.1 Loading an alignment from a Fasta file

Example file: align.fasta

```
>>> from cogent.core.alignment import Alignment
>>> fasta_file = open('align.fasta')
>>> ali = Alignment(fasta_file.read())
```

3.2 Creating an Alignment object from a SequenceCollection

```
>>> from cogent.core.alignment import Alignment
>>> seq = LoadSeqs('align.fasta', aligned=False)
>>> ali = Alignment(seq)
>>> fasta_1 = seq.toFasta()
>>> fasta_2 = ali.toFasta()
>>> fasta_1 == fasta_2
True
```

3.3 Converting an alignment to FASTA format

```
>>> from cogent.core.alignment import Alignment
>>> seq = LoadSeqs('align.fasta', aligned=False)
>>> ali = Alignment(seq)
```

```
>>> fasta_align = ali.toFasta()
```

3.4 Converting an alignment into Phylip format

```
>>> from cogent.core.alignment import Alignment
>>> seq = LoadSeqs('align.fasta', aligned=False)
>>> ali = Alignment(seq)

>>> phylip_file, name_dictionary = ali.toPhylip()
```

3.5 Convert an alignment to a list of strings

```
>>> from cogent.core.alignment import Alignment
>>> seq = LoadSeqs('align.fasta', aligned=False)
>>> ali = Alignment(seq)

>>> string_list = ali.todict().values()
```

3.6 Getting a single column from an alignment

```
>>> from cogent.core.alignment import Alignment
>>> seq = LoadSeqs('align.fasta', aligned=False)
>>> ali = Alignment(seq)

>>> column_four = ali[3]
```

3.7 Getting some columns as a sub-alignment

```
>>> from cogent.core.alignment import Alignment
>>> seq = LoadSeqs('align.fasta', aligned=False)
>>> ali = Alignment(seq)

>>> sub_ali = ali[50:70]
```

3.8 Removing some sequences from the alignment

```
>>> from cogent.core.alignment import Alignment
>>> seq = LoadSeqs('align.fasta', aligned=False)
>>> ali = Alignment(seq)

>>> ali2 = ali.takeSeqs(ali.Names[2], ali.Names[0])
```

3.9 Calculating gap fractions for each column in an alignment

```
>>> from cogent.core.alignment import Alignment
>>> seq = LoadSeqs('align.fasta', aligned=False)
>>> ali = Alignment(seq)

>>> for column in ali.iterPositions():
```

```

>>> ungapped = filter(lambda x:x=='-', column)
>>> gap_fraction = len(ungapped)*1.0/len(column)
>>> print gap_fraction

```

3.10 Getting all variable positions from an alignment

```

>>> from cogent.core.alignment import Alignment
>>> seq = LoadSeqs('align.fasta', aligned=False)
>>> ali = Alignment(seq)

>>> for index in ali.variablePositions():
>>>     column = ali[index]
>>>     print column

```

if you wanted to get the variable positions back as an alignment you could use the filtered method, eg for a DNA alignment

```

>>> just_variable_aln = aln.filtered(lambda x: len(set(x)) > 1)

```

you can also specify a motif_length to filtered which will then treat the alignment as an alignment of tuples with motif_length. For instance say you had a protein coding sequence and you wanted only the variable codons this could be done as:

```

>>> just_variable_aln = aln.filtered(lambda x: len(set(x)) > 1,
motif_length=3)

```

3.11 Remove all gaps from an alignment in FASTA format

```

>>> from cogent.core.alignment import Alignment
>>> fasta_file = open('align.fasta')
>>> ali = Alignment(fasta_file.read())

>>> seq_coll = ali.degap()
>>> seq_coll.toFasta()

```

3.12 Getting the third sequence from an Alignment as a Sequence object

```

>>> from cogent.core.alignment import Alignment
>>> seq = LoadSeqs('align.fasta', aligned=False)
>>> ali = Alignment(seq)

>>> sequence = ali.getSeq(ali.Names[2])

```

3.13 Parsing a Phylip sequence file into an Alignment object

defunct

```

>>> from cogent.parse.phylip import get_align_for_phylip
>>> ali = get_align_for_phylip(phylip_file.readlines())

```

4 Connecting with biological databases

EUtils is a web service offered by the NCBI to access the sequence, literature and other databases by a special format of URLs. PyCogent offers an interface to construct the URLs and retrieve the results in text format.

4.1 Retrieving PubMed abstracts from NCBI via EUtils

```
>>> from cogent.db.ncbi import EUtils
>>> e = EUtils(db='pubmed', rettype='brief')
>>> result = e['Michalsky Preissner'].read()

>>> e = EUtils(db='pubmed', rettype='abstract')
>>> result = e['Michalsky Preissner'].read()
```

4.2 Retrieving PubMed abstracts via PMID

```
>>> from cogent.db.ncbi import EUtils
>>> e = EUtils(db='pubmed', rettype='abstract')
>>> result = e['14983078'].read()
```

4.3 Fetching FASTA or Genpept sequences from NCBI using GI's

```
>>> from cogent.db.ncbi import EFetch
>>> EFetch(id='1234567,459567', rettype='fasta').read()
>>> EFetch(id='1234567,459567', rettype='genpept').read()
```

4.4 Retrieving GenPept files from NCBI via EUtils

```
>>> from cogent.db.ncbi import EUtils
>>> e = EUtils(numseqs=100, db='protein', rettype='genpept')
>>> result = e["lysyl tRNA-synthetase"[ti] AND bacteria[orgn]]
>>> print result.read()
```

5 Parsing GenBank files

```
>>> from cogent.parse.genbank import parse
>>> gb = parse(open('ls_orchid.gbk'))
>>> print gb.Name
>>> print gb.Info
>>> print gb.Info['features']
```

5.1 Retrieving and parsing GenBank entries

```
>>> from cogent.db.ncbi import EUtils
>>> e = EUtils(numseqs=100, db='protein', rettype='genpept')
>>> result = e["lysyl tRNA-synthetase"[ti] AND bacteria[orgn]]
>>> gb = parse(result)
```

6 Running a pairwise Needleman-Wunsch-Alignment:

```
>>> from cogent.align.algorithm import nw_align
>>> seq1 = 'AKSAMITNY'
>>> seq2 = 'AKHSAMMIT'
>>> print nw_align(seq1,seq2)
```

other parameters of nw_align, and other functions of algorithm
may be used to change the matrix, gap penalties etc.

7 Parsing BLAST

The built-in BLAST9 parser handles files with results in the format:

```
ece:Z4181 ece:Z4181 100.00 110 0 0 1 110 1 110 3e-
47 187
```

This works for output files from BLAST, PSI-BLAST, and MEGABLAST and can be used like this:

```
>>> from cogent.parse.blast import MinimalBlastParser9
>>> lines = open(blast_outfile)
>>> for record in MinimalBlastParser9(lines):
>>>     print record
(There is some method for converting the result table into the
according formats, but I dont know how to use it.)
```

8 Protein structures

8.1 Retrieving PDB structures from RCSB

```
>>> from cogent.db.pdb import Pdb
>>> p = Pdb()
>>> pdb_file = open(p['1cse'])
```

8.2 Calculating euclidean distances between atoms

```
>>> from numpy import array
>>> from cogent.maths.geometry import distance
>>> a1 = array([1.0, 2.0, 3.0])
>>> a2 = array([1.0, 4.0, 9.0])
>>> distance(a1,a2)
```

9 RNA structures

9.1 Calculate base pairs with RnAView

```
>>> from cogent.app.rnaview import RnaView
>>> rna_prog = RnaView()
>>> result = rna_prog(pdb_filename)
```

9.2 Calculate base pairs with RNaView

```
>>> from cogent.parse.rnaview import RnaviewParser
>>> bp_stats = result['bp_stats']
>>> bpairs = result['base_pairs']
>>> bp_dict = RnaviewParser(bpairs)
>>> print bp_dict
>>> for bp in bp_dict['BP']:
>>>     up = [bp.Up.ChainId, bp.Up.ResId, bp.Up.ResName,
bp.Up.RnaViewSeqPos]
>>>     down = [bp.Down.ChainId, bp.Down.ResId, bp.Down.ResName,
bp.Down.RnaViewSeqPos]
>>>     annotation = [bp.Edges, bp.Orientation, bp.Conformation,
bp.Saenger, bp.isWC(), bp.isWobble()]
```

9.3 Energy parameters for base pairs

```
>>> from cogent.data.energy_params import ENTHALPY_PARAMS, ENTROPY_PARAMS
>>> ENTHALPY_PARAMS.gcInit
>>> ENTHALPY_PARAMS.atInit
>>> ENTHALPY_PARAMS.nearestNeighbors
>>> ENTHALPY_PARAMS.symCorrection
>>> ENTROPY_PARAMS.gcInit
>>> ENTROPY_PARAMS.atInit
>>> ENTROPY_PARAMS.nearestNeighbors
>>> ENTROPY_PARAMS.symCorrection
```

10 Genetic code

10.1 Translate DNA sequences

```
>>> from cogent.core import genetic_code
>>> code = genetic_code.DEFAULT
>>> code.translate("TTTGCAAAC")
This uses the standard Nuclear code, several other codes can be accessed in
>>> genetic_code.GeneticCodes
(how to access their names?)
```

11 Running ClustalW

(under construction)

(uses
<http://biopython.org/DIST/docs/tutorial/examples/opuntia.fasta>)

```
>>> from cogent.app.clustalw import Clustalw
>>> app = Clustalw()
>>> result = app('opuntia.fasta')['Align']
>>> result
<open file 'opuntia.aln', mode 'r' at 0x1783de8>
>>> print ''.join(result.readlines()[:11])
CLUSTAL W (1.83) multiple sequence alignment
```

gi|6273285|gb|AF191659.1|AF191

```

TATACATTAAAGAAGGGGGATGCGGATAAATGGAAAGGCGAAAGAAAGAA
gi|6273284|gb|AF191658.1|AF191
TATACATTAAAGAAGGGGGATGCGGATAAATGGAAAGGCGAAAGAAAGAA
gi|6273287|gb|AF191661.1|AF191
TATACATTAAAGAAGGGGGATGCGGATAAATGGAAAGGCGAAAGAAAGAA
gi|6273286|gb|AF191660.1|AF191
TATACATAAAAGAAGGGGGATGCGGATAAATGGAAAGGCGAAAGAAAGAA
gi|6273290|gb|AF191664.1|AF191
TATACATTAAAGGAGGGGGATGCGGATAAATGGAAAGGCGAAAGAAAGAA
gi|6273289|gb|AF191663.1|AF191
TATACATTAAAGGAGGGGGATGCGGATAAATGGAAAGGCGAAAGAAAGAA
gi|6273291|gb|AF191665.1|AF191
TATACATTAAAGGAGGGGGATGCGGATAAATGGAAAGGCGAAAGAAAGAA

```

***** *

```

>>> all_records = LoadSeqs('opuntia.aln')
>>> print 'description:', all_records.getSeqNames()[0]; print
'sequence:', all_records.Seqs[1][:50], '...'
description: gi|6273285|gb|AF191659.1|AF191
sequence: TATACATTAAAGAAGGGGGATGCGGATAAATGGAAAGGCGAAAGAAAGAA ...

```

6.4.3

```

>>> consensus = all_records.IUPACConsensus()
>>> print ''.join(consensus)
TATACAT?AAAG?
AGGGGGATGCGGATAAATGGAAAGGCGAAAGAAAGAAAAAATGAATCTAAATGATATA?
GATTCCACTATGTAAGGCTTTTGAATCATATCATAAAAGACAATGTAATAAAGCATGAATACAGAT
TCACACATAATTATCTGATATGAATCTATTCATAGAAAAAGAAAAAGTAAGAGCCTCCGGCCAA
TAAAGACTAAGAGGGTTGGCTCAAGAACAAGTTCATTAAGAGCTCCATTGTAGAATTCAGACCTA
ATCATTAATCAAGAAGCGATGGGAACGATGTAATCCATGAATACAGAAGATTCAATTGAAAAAGAT
CCTA?????CATTGG?
AAGGATGGCGGAACGAACCAGAGACCAATTCATCTATTCTGAAAAGTGATAAACTAATCCTATAAA
ACTAAAATAGATATTGAAAGAGTAAATATTCGCCCGCGAAAATTCCTTTTTT?TT?
AAATTGCTCATATTTT?TTTTAGCAATGCAATCTAATAAAATATATCTATACAAAAAA?A?
AGACAAACTATATATATA?????????ATATATTT??AATT?CCTTATATA?
CCAAATATAAAAATATCTAATAAATTAGATGAATATCAAAGAATC?ATTGATTTAGT?
TATTATTAATGTATAT?TTAATTCAATATTATTATTCTATTTCATTTTTTATTTCATTTTCAA?
TTTATAATATATTAATCTATATATTAATTTA?
AATTCTATTCTAATTCGAATTCAATTTTTTAAATATTTCATATTCAATTAAAATTGAAATTTTTTCAT
TCGCGAGGAGCCGGATGAGAAGAACTCTCATGTCCGGTTCTGTAGTAGAGATGGAATTAAGAAAA
AACCATCAACTATAACCCCAA?AGAACCAGA

```

6.4.4

```

>>> my_pssm=all_records.scoreMatrix()
>>> my_pssm[1]['A']
7.0

```