Welcome to this little Tutorial for the Bioroebe Project.

This page will attempt to demonstrate and explain the **available options** for the **Bioroebe project**.

The project here is vaguely similar to the older **BioRuby** project, but there are some minor and major differences between these two projects.

As an **example** - the BioRoebe project will heavily make use of **YAML files**.

Take the different codon tables for instance. These codon tables for different organisms are stored in different .yml files (if they have been ported yet that is). The yaml file 1.yml is the standard codon table for the Eukaryotes.

BioRuby on the other hand, uses hardcoded hash tables in a .rb file.

Both approaches have their pros and cons - I personally much prefer yaml files, also due to portability (you can re-use the yaml file, but how can you re-use hardcoded tables in .rb files?), so I opted to go that way. But if people want to use a hash instead, they can do so, too - see the **API** for codon tables lateron.

The BioRoebe project also features an **interactive REPL** (read, eval, print, loop), a shell that can easily evaluate user input. The **BioShell**.

The interactive BioShell can be used to input various different commands "live". This allows one to apply commands / instructions onto datasets that originate from the **life sciences**.

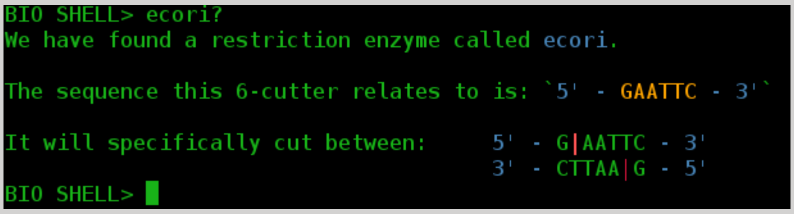
If the last character in the user input supplied is the '?' character, then we will assume this to be a **restriction enzyme**. This is mostly a convenience feature.

Example for this:

ecori?

This will also show where exactly we will cut within this target sequence. It will show both the forward strand and the complementary strand, including the site where it cuts via a red vertical bar.

Let's have a look at an image to demonstrate how this looks, from the commandline:



```
BIO SHELL> ecori?
We have found a restriction enzyme called ecori.

The sequence this 6-cutter relates to is: `5' - GAATTC - 3'`

It will specifically cut between:      5' - G|AATTC - 3'
                                       3' - CTTAA|G - 5'

BIO SHELL> █
```

You can also get a list of **available 8-cutters**, that is, enzymes that **leave an overhang of 8 nucleotides**.

Example:

8-cutters

You can view all **available metabolic pathways**, such as the glycolysis, via:

pathways?

To find out the Alu element sequence, do:

alu?

To calculate the **melting temperature**:

melting_temperature?

Sequence Objects

You can build Sequence objects by doing something like this:

```
sequence_object = Bioroebe::Sequence.new( sequence: "aaaatgggggggggggccccgtt",
alphabet: :dna )
```

or

```
sequence_object = Bioroebe.sequence( sequence: "aaaatgggggggggggccccgtt", alphabet:
:dna )
```

This is similar to BioPerl, by the way. If you want to, you can also use Bioroebe::Seq rather than Bioroebe::Sequence. I personally prefer the longer variant but both ways are doable.

Usage examples:

```
Bioroebe::Sequence.new "aaaatgggggggggggccccgtt"
Bioroebe::Sequence.new "aaaatgggggggggggccccgtt", :upcase # This variant will upcase
the input.
```

Let's look at a slightly more complicated example for creating such a sequence object.

```
sequence_object = Bioroebe::Sequence.new( seq: "aaaatgggggggggggccccgtt", display_id:
"#12345", desc: "example 1", alphabet: "dna" )
puts sequence_object.seq
puts sequence_object.sequence?
```

You can also cut this sequence with a restriction enzyme.

Example:

```
sequence.cut_with_enzyme('EcoRI')
```

Help

You can view the help options by doing:

```
help
```

Since this can be quite long, you can also look at the individual subsections. In order to look at all entries that start with c, do this:

```
help c
```

You can also specify a range of help options via the - character. See:

```
help g-k
help g-m
```

Helper Classes

The class **SanitizeNucleotideSequence** allows you to format input that is like this:

```
10 AGTA
20 TTGC
```

It will get rid of all the numbers, and yield to you a simple string without any " " characters or newlines.

This will be useful for **FASTA format files**.

A more specific example will follow next:

```
Bioroebe::SanitizeNucleotideSequence.new(" 10 AGTA \n 20 TTGC").result?
```

From within the BioShell itself, you can trigger this by doing:

```
sanitize_nucleotide_sequence " 10 AGTA \n 20 TTGC"
```

This will also assign the sequence to the main DNA sequence in use.

Shell Prompt

You can set a **custom prompt**, via the keywords "prompt" or "set_prompt".

To display the **current working directory**, do:

```
prompt pwd
```

To revert to the old default again, do this:

```
prompt REVERT
prompt revert
prompt DEFAULT
prompt default
```

If you do not want to set any prompt, do:

```
prompt none
```

Fasta Input

Bioroebe understands the **FASTA format**.

If you wish to do multi-line fasta input as part of the Bioshell, that is input that will include newlines, then you currently have to use either of the following commands:

```
multi-line
multiline
multi_line
multi_input
```

Pick any of these. You can terminate the input by using __.

You can also use "fasta" for reading .fasta files. Or doing single line input.

At a later time, this may change, but for now, please use these two distinct ways to read in a fasta file.

If you want to read in some fasta file, you can use this command:

```
pfasta
parse_fasta
```

Example:

```
pfasta foobar.fasta
pfasta foobar.fa
```

If you do not provide input, the bioshell will first try to find any .fasta file in this directory, then any .fa file. If neither exists, we will use a default input. This should allow to test some things, hence why this feature exists.

Note that multiline actually also allows you to assign multiline input from normal, regular sequences too. Simply start to paste in your sequence.

Also note that if you paste a Genbank accession number into it, one that starts with a > token, then we will simply strip this away. This allows you to more easily copy/paste data into the BioShell.

As of May 2016 you can simply input an URL to a NCBI entry and the BioShell will try to download the sequence.

You can also input something such as:

```
NC_000866.4_Enterobacteria_phage_T4.fasta
```

We will also look locally if this fasta sequence exists.

Translate into amino acids

You can translate any given DNA sequence into the corresponding **amino acids** by issuing:

```
translate
trans
```

Simply add the DNA sequence as argument to translate. If you omit it, the BioShell will attempt to use the main DNA sequence.

translate ATGCCCCGCTGA # => MPR*

Note that we will also show, for the first frame, the specific position.

You can do so too via:

frame1
frame2
frame3

This will show the respective frame at hand.

Also note that you can do the reverse again, by issuing:

backtrack

This will attempt to use the most likely codon, in order to deduce the corresponding DNA sequence.

backtrack uses a **codon usage table** which gives the frequency of usage of each codon for each amino acid.

For each amino acid in the input sequence given, the corresponding **most frequently occuring codon** is used in the nucleic acid sequence that is output.

Codon Table

A codon table is a human-made mapping that keeps track of which particular codon will be translated into the corresponding aminoacid at the ribosome, during translation. They can be species-specific, that is, in different organisms, different codons may code for different aminoacids - or be interpreted as a stop codon.

You can designate a specific codon table to use via the following API:

Bioroebe.set_codon_table()

And you can query the codon table in use via:

Bioroebe.codon_table?

We will follow the **NCBI convention** for the codon table, so using 1 will use the eukaryote table, for instance (this is the default anyway).
Note that you can also use a symbol instead, such as :human or :humans, which should be easier to remember.

You can get a list of all registered codon_tables via:

show_all_codon_tables

From the Bioshell.

If you only want to show the headers, do this:

codon_tables_headers
codontablesheaders

You can also obtain the codons that code for a specific aminoacid by simply typing the name of this aminoacid, and a trailing question mark. This currently (Mai 2016) only works for aminoacids in german.

glycin?

The above will tell us some things about the aminoacid **Glycin**.

Start Codon

By default, the most commonly used start codon will be ATG (respectively AUG in the mRNA).

Sometimes another start codon may be used. In this case, we need to have the Bioroebe project use that other codon too.

You can modify the start codon in the Bioshell, by doing:

```
set_start_codon GUG
```

This would use GUG as the new start codon. (We will convert to DNA form automatically).

You can also show both start and stop codons simultaneously, by issuing:

```
start_stop
```

NCBI

The NCBI provides a lot of information related to different organisms.

You can search for NCBI entries via the file **ncbi.rb**. This file resides in the path bioroebe/ncbi/ncbi.rb

You can also download NCBI files, that is, usually fasta files.

Let's provide some examples here.

If you know the ID, you can simply download this fasta file in the bioshell:

```
download_fasta 296010862
dfasta NC_000913.3
```

This will download the fasta sequence of the id 296010862

You can batch download from instructions given in files.

For instance, simply do:

```
dfasta /FASTA_URLS
```

If the file exists, we will grab its content.

Uniprot

You can fetch data from **Uniprot**, such as by issuing:

```
unitprot_fetch
```

Truncate output

DNA/RNA sequences can become very long and then become quite difficult to view, read and handle on the commandline.

It is possible to truncate output that is too long. By default, we will always do so, but you can toggle this behaviour by doing:

```
Bioroebe.do_not_truncate
```

The above instruction will toggle the truncate behaviour to not truncate, ever.

If you need to do so within the bioshell, this is the way:

```
no_truncate
```

Or simply

```
truncate
```

This will toggle, like a switch.

compseq

You can use **compseq** to compare a sequence to "itself". That is, we will show how many dinucleotides can be found in the given sequence and whether this is "unexpected" or not. This is similar to the Emboss compseq tool.

Remember, this may be useful for when you need to find CpG islands, which can possibly be noticed by the amount of distribution of CG dinucleotides.

Usage example:

```
compseq
```

Working with Aminoacids

You can assign some aminoacids via:

```
setaa LLLTLTLT
```

Then, you can query the positions via:

```
positions?
```

This will display the amino acids in a chunked, numbered manner.

Deduce an aminoacid sequence

You can deduce an amino acid sequence.

The keyword is "deduce" or "ded", and some aliases.

Example how to use this:

```
Lys - Ser - Pro - Ser - Leu - Asn - Ala - Ala - Lys
Lys - Val - His - His - Leu - Met - Ala - Ala - Lys
```

This will show the corresponding RNA codons that can possibly code for these.

AT and GC content

You can find out the AT or GC content of a DNA sequence.

Do either of these:

```
at_content?
gc_content?
```

Or shorter:

```
at?
atcontent
gc?
gccontent
```

You can use this method to obtain the GC content of a sequence:

```
Bioroebe.gc_content("ATCG")
```

Shuffling the DNA/RNA string

Via

```
shuffle
```

you can randomly rearrange the main DNA/RNA string.

Pubmed

If you input

```
pubmed
```

then we will go to pubmed.

Molmass

You can query the molecular mass for an aminoacid by issuing:

```
molmasse glycine
```

If you want to get an overview over all aminoacids and their respective mass, do this:

```
show_aminoacids_mass_table
```

CCAAT

You can try to find CCAAT sequences by issuing:

```
ccaat?
```

TaxID

You can find out the taxonomic ID correlating to a name or vice versa.

Presently, we prefer to handle the ID number itself.

Example:

```
taxid? 9606
```

This would show that 9606 corresponds to Homo sapiens.

Generate DNA

You can generate random DNA strings by issuing this code:

```
x = Bioroebe.random_dna 50 # =>
"AGACATCCGGCTTGGATACCTCATAAGTCATATCAGCATCGTCGGACATT"
```

The number tells us how many nucleotides to generate.

The Hydropathy index

You can display the hydropathy index for aminoacids.

Simply issue:

```
hydropathy?
```

Weight of Nucleotides

You can calculate the total weight of a nucleotide sequence.

Simply issue the following command:

```
weight?
```

Do not forget to first **assign** a DNA string, though.

Or to randomly grab one nucleotide:

random 1; weight?

DNA to mRNA conversion

You can "convert" a DNA string into the corresponding mRNA.

In order to do so, try any of this:

```
to_mrna
tomrna
to_mRNA
mrna
```

```
random 9
```

Generating some DNA of length 9.
Setting DNA sequence to `ATGGTAGAC`.

5' - ATGGTAGAC - 3'

```
tomrna
```

5' - AUGGUAGAC - 3'

Start Codons

A start codon is where the ribosome will start to translate the mRNA into the corresponding polypeptide chain.

Different organisms may use slightly different start codons.

The most common one, normally, is the start codon AUG (or, in the DNA, ATG).

The code ::Bioroebe.start_codon? will point to this main start codon in use. You can redefine this too.

The value currently defaults to:

ATG

You can also show all translated ORFs. There are several ways how to do so, but a simple one is the following:

```
showorf
```

Since as of June 2016, you can also align all ORFs of the main sequence. Simply issue the following command for this:

```
align_ORFS
```

Oligo Frequency

You can analyze the oligo frequencies.

First, assing some sequence. Now do:

two
three

This will show the distribution of the oligos.

Stop Codons

To showcase all stop codons, do:

```
default_stop_codons?
```

You can also show and colourize all existing stop codons in a mRNA.

In order to do this, do:

```
stop?
```

❗ You must have assigned a sequence prior to this of course.

Find

To find a subsequence, do:

```
find ATG
```

We can only find something this pattern is part of the string.

Nucleotide Table

You can display the nucleotide table via:

```
nucleotide_table?
```

You can also query a subsequence from this.

Example:

```
125-250
```

This would show the subsequence from 125 to 250.

If you have assigned a nucleotide string, you can also specifically manipulate certain positions.

Example:

```
658-660 = CCA
```

If you input only nucleotides into the BioShell, such as AGGTGCC, then hit enter, we will also assume that you wish to assign a new DNA sequence.

Example:

```
GTTGCAACAACCTACAGATGCGTTCAGTCTATCTATATACAAGAGGAAGAATACAA
```

Piped

You can add the | character in display at every third position of a nucleotide. This is mostly just a visual display aid, on the commandline or the BioShell.

Invocation example:

```
piped
```

transeq

You can convert a DNA sequence into an aminoacid sequence by doing this:

```
transeq
```

Align two different sequences

You can compare two different sequences via **sscompare**.

This will then show an alignment between these two different sequences.

```
sscompare
```

This is how it may look like:

You can also use the command "align". This one works a bit differently.

The following example shows that:

```
setseq1 CAGGTCCC
setseq3 GGGGGCAC
align 1 3
```

Another image follows next, it may explain what happens there:

As you may be able to see, the output is a bit different. Via "align", we will show in red which ones do NOT match. Via sscompare, we will show via a red dot which ones will not match, whereas a (green) vertical tab will indicate which nucleotides will match.

CpG islands

You can show whether there may be CpG islands.

```
CpG
CpG?
```

GenBank

You can parse or generate GenBank formats.

The class that will generate GenBank file formats is called class GenbankFlatFileFormatGenerator. Not a very simple name that one is!

The basic use is to pass the DNA sequence into it. Then it will output a file that can be stored.

You can also "generate" this sequence from within the BioShell.

```
to_genbank
togenbank
togen
```

If you wish to find out the current GenBank version, do either of the following:

```
current_genbank_version?
current_genbank_version
genbank_version?
genbank_version
```

Colourize Aminoacids

You can colourize some aminoacids in red colour, at the least on the commandline.

In order to do so, you first must tell the BioShell which aminoacid you want to colourize. This can be done like so:

```
cola K
```

This would colourize Lysine. K is the one amino acid letter for Lysine.

Next, you can test whether this works. A simply way is to ask for the Ubiquitin sequence.

You can do this like so:

```
ubi?
```

You can also uncolourize it again:

uncola K

Or just reset to the initial state again, via **reset**.

Colours

This is a subsection about colours, both in regards to the BioShell but also for other classes in the bioroebe project.

There is a default "colourize" tag in the method rev().

By default, this will be in green.

You can assign another colour by doing this:

```
pick_colour
pcolour
```

Or another variant.

This will currently use Konsole colours, but in theory, we could also use plain and simple ansi colours for now. Make sure that the colour exists though.

If you dislike the colours, you can disable them, by issuing this command:

```
disable colours
disable_colours
no colours
```

American English spelling should also work.

If you wish to re-enable them, do:

```
enable colours
use colours
```

Either will work.

You can also designate another default highlight colour to use. The highlight colour is the colour that will be used to highlight important sequences.

```
set_highlight_colour violet
set_highlight_colour random
```

Hamming distance

You can calculate the Hamming distance by doing:

```
hamming
ATTATTATTATTATTATTATTATTATTATTATTATTATTATTATTATTATTATTATTATTATTATTATTATTATTATTATTATTATTATTATTATTATTTTAT
ATTATTATTATTATTATTATTATTATTATTATTATTATTATTATTATTATTATTATTATTATTATTATTATTATTATTATTATTATTATTATTATTA
```

Adding and removing nucleotides

You can add and remove nucleotides from the 3-prime end of a string.

Add 3 nucleotides:

```
+3
```

Remove 3 nucleotides:

```
-3
```

Note that you can also use names such as add or chop in order to remove nucleotides. The above + and - just exist as shortcuts.

Show both DNA strands

In order to show both DNA strands, do:

```
both
double
```

Print aminoacid table

You can print the aminoacid table.

```
print_aminoacid_table
print_aa_table
patable
```

Padding

By default, all strings/sequences output will be padded with 2 space characters on the left side.

You can check this by doing:

```
padding?
```

You can set another padding by doing either:

```
set_padding 0
setpadding 0
```

The number given will be how many space characters we will use.

Random Inserts

You can randomly insert some sequence into your target DNA.

```
rinsert ATGCC
```

The above will insert this 5-nucleotide sequence somewhere into your DNA string.

First nucleotide

You can change the first nucleotide of a sequence by doing this:

```
first A
```

The argument should be the nucleotide you wish to see appear on the first position there.

Taxonomy

You can find out how many species are registered in the NCBI Taxonomy database, by issuing this command in the BioShell:

```
n_species?
nspecies
```

This will call the class method called **Taxonomy.report_n_species**.

Set name of a gene

You can set a name of your gene, in the BioShell, via:

```
set_name foo
```

The argument will be the name of your target gene in question.

This may be useful at a later point, if you want to keep track of different target sequences.

Chunked display

You can show chunked display via:

```
chunked_display
```

May be easier to assign a sequence before using that method.

Colour schemes

You can test the available colour schemes by issuing something such as:

```
test_colour_scheme
```

You can also do so for the main nucleotide sequence, by issuing:

```
colour_scheme
```

You can also create a test-demo page, via:

```
colour_tests
```

Replay

Sometimes you may have to store your input history. This can be done by:

```
save_history
```

This will create a local file. You can, if this file exists, also use it to "replay" input history. This will allow you to simulate that you already did input these specific instructions into the BioShell.

The reason as to why this exists is simple - I needed a quicker way to redo the very same sequence.

```
replay file
```

A configuration option also allows us to log all inputted commands into a file. By defaut, I have set this on - if you wish to disable it, you can modify the file configuration/configuration.rb

Phosphorylation Sites

You can colourize phosphorylation sites in your amino acid sequence, within the BioShell, if you issue this instruction:

```
psites?
phosphorylation_sites?
```

This will feedback some information as to where phosphorylation sites can be found, within any given DNA sequence or aminoacid sequence.

PolyA Tail

You can add a PolyA tail to a mRNA sequence by doing:

```
polyY
```

Note that for now, this will only append 250 A on the 3-prime end of the RNA in question. In the long run, we will try to make this more accurate. (Statement here made in June 2016)

Chromosome Table

You can print out a chromosome table by issuing this command:

chromosome_table

GUI

We can use some ruby-gtk widgets. These are not very advanced though.

In general, you can start them by prefacing the word gui_ and then the name of the class.

For instance:

gui_restriction_enzymes

Will start the GUI widget for restriction enzymes.

Restriction Enzymes

You can digest your DNA sequence. Provided that you have a target DNA sequence, you can run a digest by doing this:

digest KpnI?
digest EcoRI

This will show the fragments that are generated.

Palindromes

You can generate palindromes by issuing this command:

generate_palindrome 20

You can also scan for palindromes by issuing this:

discover_all_palindromes

Xorg Buffer

You can set the main DNA sequence to the xorg buffer on linux.

To copy the DNA sequence into the Xorg Buffer, issue the following command:

to_buffer

Mutations

You can mutate the DNA nucleotide sequence.

Example:

mutate_position 5 C
mutate_position 4 A

Here we will mutate the nucleotide at position 5 to C or the one at position 4 to A.

You can also mutate the aminoacid sequence. The following example shows that.

mutate_aminoacid_position 5

cutseq

You can cut out any sequence. There are two basic ways how to do so:

cutseq 5 6

This will remove nucleotide 5, 6 and 7. Remember that we start to count at **nucleotide 1**.

A similar syntax is possible:

cutseq 5 +3

This is the same as above but +3 means that we add this number to 5, so it is equivalent to:

cutseq 5 8

Another way is to use the interactive variant. This will ask you for the start and end position.

cutseq

- Oligo Frequency
- Stop Codons
- Find
- Nucleotide Table
- Piped
- transeq
- Align two different sequences
- CpG islands
- GenBank
- Colourize Aminoacids
- Colours
- Hamming distance
- Adding and removing nucleotides
- Show both DNA strands
- Print aminoacid table
- Padding
- Random Inserts
- First nucleotide
- Taxonomy
- Set name of a gene
- Chunked display
- Colour schemes
- Replay
- Phosphorylation Sites
- PolyA Tail
- Chromosome Table
- GUI
- Restriction Enzymes
- Palindromes
- Xorg Buffer
- Mutations
- cutseq
-

Links

../lib/bioroebe/www/bioroebe.cgi

Chaos Game representation of Gene Structure