BEAST$_{d}ocsDocumentation$

Release latest

May 04, 2020

Contents

1 Description **3**

2 Usage **5**

3 Contribute **7**

4 Contents **9**

4.1 Usage 9

5 Search **21**

BEAST: Bacterial Epigenomics Analysis SuiTe

CHAPTER 1

Description

With only a few thousand bacterial methylomes published to date, it is becoming increasingly evident that epigenetic regulation of gene expression is highly prevalent across bacterial species. Despite the exciting prospects for studying epigenetic regulation, our ability to comprehensively analyze bacterial epigenomes is limited by a bottleneck in integratively characterizing methylation events, methylation motifs, transcriptomic data, and functional genomics data. In this regard, we provided the first comprehensive comparative analysis of a large collection of epigenomes in a single bacterial species, as well as a detailed roadmap that can be used by the scientific community to leverage the current status quo of epigenetic analyses (Oliveira et al., 2019, Nat. Microbiology). BEAST includes a set of R and wrapper shell scripts that allow to reconstitute the major analyses steps of this publication. A detailed step-by-step tutorial and links to an executable repository and computational reproducibility platform (will be submitted soon) are provided at each section below.

CHAPTER 2

Usage

For a guide on each manuscript section and corresponding wrapper scripts, see *Usage*.

CHAPTER 3

Contribute

- Issue tracker: GitHub
- Source code: GitHub

CHAPTER 4

Contents

4.1 Usage

4.1.1 Section 1

Motif refining

Fundamental dependencies

- Recent version of R

Example usage

For convenience, install EDirect as described here.

For testing purposes the user may download four test *C. difficile* FASTA files (all 36 genomes are available under Bioproject PRJNA448390):

```
$ esearch -db nucleotide -query CP028530.1 | efetch -format fasta | awk -v v1=020711 '
↪{if($0~">") print ">"v1; else print $0}' > 020711.fas
$ esearch -db nucleotide -query CP028525.1 | efetch -format fasta | awk -v v1=020482 '
↪{if($0~">") print ">"v1; else print $0}' > 020482.fas
$ esearch -db nucleotide -query CP028524.1 | efetch -format fasta | awk -v v1=020477 '
↪{if($0~">") print ">"v1; else print $0}' > 020477.fas
$ esearch -db nucleotide -query CP028529.1 | efetch -format fasta | awk -v v1=020709 '
↪{if($0~">") print ">"v1; else print $0}' > 020709.fas
```

Then download the corresponding **motif_summary.csv** and **modifications.csv** methylation files:

```
$ https://submit.ncbi.nlm.nih.gov/ft/byid/fjjp6a7k/motif_summary_020711.csv
$ https://submit.ncbi.nlm.nih.gov/ft/byid/xgibymgy/modifications_020711.csv
```

(continues on next page)

(continued from previous page)

```
$ https://submit.ncbi.nlm.nih.gov/ft/byid/rvupzqfb/motif_summary_020482.csv
$ https://submit.ncbi.nlm.nih.gov/ft/byid/nf0yhhue/modifications_020482.csv
$ https://submit.ncbi.nlm.nih.gov/ft/byid/1bxrewy4/motif_summary_020477.csv
$ https://submit.ncbi.nlm.nih.gov/ft/byid/umkrolrg/modifications_020477.csv
$ https://submit.ncbi.nlm.nih.gov/ft/byid/aeyzpzfk/motif_summary_020709.csv
$ https://submit.ncbi.nlm.nih.gov/ft/byid/j9evun0v/modifications_020709.csv
```

- A separate folder should be created for each genome to be analyzed. Each folder should contain 3 files: the genome FASTA file (renamed as genome.fasta), and the output files of the SMRT pipeline (renamed as **modifications.csv** and **motif_summary.csv** files). An output folder should also be created.
- Manually add the paths to each of the above folders in **Parameter_File.txt**. Mean methylation motif fraction is already given for a sample of 50 common motifs. This motif list can be tailored as desired.

Install the latest release of R, then get the latest version of Bioconductor by starting R and entering the commands:

```
> if (!requireNamespace("BiocManager", quietly = TRUE))
     install.packages("BiocManager")
  BiocManager::install()
```

```
# Install the Biostrings package:
> BiocManager::install("Biostrings")
```

```
# Install the dependent R packages:
> install.packages(c("data.table","stringi","plyr","nnet","seqinr","parallel","R.utils
↪","ggplot2","gridExtra","reshape2","foreach","RSQLite","gplots"))
```

then run:

```
$ Rscript Motif_Refine.R Parameter_File.txt
```

This program will take a genome fasta file and SMRT-seq kinetic data (**modifications.csv** and **motif_summary.csv**) and output a list of refined (trimmed) methylated motifs.

One data directory will be created for each genome folder. This folder will have output files for each motif indicated in the **Parameter_File.txt**. The latter include one **exp.csv** and one **con.csv** for motif statistics respectively for the leading and lagging strands, graphical distribution of scores and IPD ratios for each base of the motif, and **expdat2.csv** with kinetic information for all possible motif combinations.

4.1.2 Section 2

Motif count exceptionalities using Markov models

Fundamental dependencies

- R'MES
- SeqKit

Example usage

```
$ ./GO_Abundance.sh <*.fasta> <motif> <word_length> <markov_model_order>
↪<approximation method>
```

This wrapper script takes a genome FASTA file and computes the statistical over/under-representation of a given motif under Markov models of order N.

<word_length>: length l of the word to search

<markov_model_order>: From 1 to a maximum of l-2

<approximation_method>: gauss (gaussian), compoundpoisson (Poisson)

For example: ./GO_Abundance.sh 020711.fas CAAAAA 6 4 compoundpoisson

will output a **020711.motif.txt** with 7,627 total CAAAAA motifs detected in the chromosome, and a **020711.RMES.txt** with all observed and expected frequencies of CAAAAA under a Markov model of order 4.

Multi-Scale Representation (MSR) of methylation motifs

Fundamental dependencies

- SeqKit
- MSR standalone executable (msr_runtime_SIGNAL and wrapper run_msr_runtime_SIGNAL.sh). Can be found here.
- MATLAB Compiler Runtime (MCR) version 8.1 (R2013a). Can be found here.

Example usage

```
$ ./GO_MSR.sh <*.fasta> <motif> <mcr_directory> <parameter_file> <MSR_output_filename␣
↪(same as in parameter file)>
```

This wrapper script takes (1) a genome FASTA file and (2) a methylation motif, builds a signal file, runs the MSR pipeline, parses, and plots its output.

4.1.3 Section 3

Conservation of methylation motifs across genomes

Fundamental dependencies

- ProgressiveMauve
- SeqKit
- convertAlignment.pl
- Bedtools
- jvarkit
- VCFtools

Example usage

```
$ ./GO_ConsVar.sh <minimal length of LCB> <number of genomes to align> <species_
↪prefix> <MAUVE_DIR> <motif>
```

This wrapper performs multiple whole-genome alignment and computes orthologous (conserved/variable) and non-orthologous methylation motifs across genomes. The list of genome FASTA files should be placed in the same folder as **GO_ConsVar.sh**.

For example, to align the two dummy *C. difficile* genomes provided in **Section 1** with a minimum length of local collinear blocks (LCB) of 50 bp and compute conserved and variable CAAAAA motifs:

./GO_ConsVar.sh 50 2 CDIF /path/to/mauve/ CAAAAA

It outputs 4 main files:

CDIF.Indels.txt: All orthologous variable motif positions harbouring indels.

CDIF.SNPs.txt: All orthologous variable motif positions harbouring SNPs.

CDIF.Conserved.txt: All orthologous conserved motif positions.

CDIF.NonOrthologous.txt: All non-orthologous positions. Usually are found within MGEs.

4.1.4 Section 4

TFBS mapping

Fundamental dependencies

- Recent version of R
- Bioconductor
- MAST (MEME Suite)

Example usage

Install the latest release of R, then get the latest version of Bioconductor by starting R and entering the commands:

```
> if (!requireNamespace("BiocManager", quietly = TRUE))
     install.packages("BiocManager")
  BiocManager::install()
```

```
# Install the Biostrings package:
> BiocManager::install("Biostrings")
```

```
# Install the dependent R package:
> install.packages("knitr")
```

Then run:

```
$ ./GO_TFBS.sh <TFBS_multifasta> <*.fasta> <TF_name>
```

This wrapper takes a TFBS multifasta, computes a PSSM, and corresponding TFBS hits in a given DNA sequence in FASTA format. For example, to compute a hit list of XylR TFBSs in one of the dummy *C. difficile* FASTA files given in **Section 1**:

./GO_TFBS.sh XylR.fasta 020711.fas XylR

The TFBS multifasta for XylR is provided as example. The output file XylR_TFBS.txt will contain all raw TFBS hits.

TSS mapping

Fundamental dependencies

- Samtools
- IGVtools
- Parseq
- GSL

Example usage

For illustrative purposes, the user may download some test FASTQ files (available under Bioproject PRJNA445308):

```
# The SRR10297679 test sample corresponds to the MTase mutant at a sporulation stage␣
↪(10.5 h after sporulation induction):
$ fastq-dump --outdir /output/dir/ -I --split-files SRR10297679
```

and the reference chromosome of *C. difficile* 630:

```
$ esearch -db nucleotide -query AM180355.1 | efetch -format fasta | awk -v v1=CD630 '
↪{if($0~">") print ">"v1; else print $0}' > CDIFF630_0.fasta
```

The above FASTQ files can then be converted to BAM format:

```
$ bwa index CDIFF630_0.fasta
$ bwa mem -t 8 CDIFF630_0.fasta SRR10297679_1.fastq SRR10297679_2.fastq > SRR10297679.
↪sam
$ samtools view -bS SRR10297679.sam > SRR10297679.bam
```

Finally, we can run Parseq:

```
$ ./GO_TSS.sh SRR10297679 CDIFF630_0.chrom.sizes CDIFF630_0.fasta <counts_folder_path>
↪ <results_folder_path> <parseq_Parameters_folder_path>
```

This wrapper runs the Parseq program in both DNA strands for reconstruction of the transcriptional landscape from RNA-Seq data, and infers TSSs from abrupt shifts in transcription levels.

Several files will be produced.

In the "…/counts/" folder:

A **Parameters_initial** file;

An **ORFs<strand>** regions file for each chromosome;

–//–

In the "…/results/" folder:

Particles files where each line represents a transcription profile sample;

Particles_struct file - each line is a local bias profile sample;

Expression_<strand>.wig - expression level average along the chromosomes;

Trans_<strand>_<expression threshold>.wig - transcription probability at position resolution accounting only for expression level above a user given expression threshold;

breakpoints_3_5_<strand>_<expression threshold>.bed - breakpoints with local cumulative probability above threshold;

4.1.5 Section 5

Differential gene expression analysis

Fundamental dependencies

- Recent version of R
- Java
- BWA
- AdapterRemoval
- Trimmomatic
- SortMeRNA
- Samtools
- featureCounts

Example usage

For illustrative purposes, the user may download some test FASTQ files (all available under Bioproject PRJNA445308):

```
# The SRR10297679 test sample corresponds to the MTase mutant at a sporulation stage␣
↪(10.5 h after sporulation induction):
$ fastq-dump --outdir /output/dir/ -I --split-files SRR10297679
```

```
# Install DESeq2 in R:
> BiocManager::install("DESeq2")
```

```
# Install the R packages:
> install.packages(c("DESeq2","RColorBrewer","gplots","ggplot2","rgl","pheatmap"))
```

then:

```
$ ./GO_GetCounts.sh <*.fastq1 file> <*.fastq2 file> <*.fasta reference file> <*.SAF␣
↪annotation file> <PATH to SILVA rRNA_databases folder> <PATH to SILVA index folder>
↪<PATH to Trimmomatic.jar> <PATH to adapters *.fa> <file prefix>
```

This wrapper performs RNA-seq paired-end read cleaning and mapping for differential expression analysis. It takes as input the FASTQ files, a reference FASTA file for read mapping, and a SAF annotation file. Mapping does not take into consideration multi-mapping and multi-overlapping reads

The SAF annotation format (example provided for *C. difficile* 630) has five required columns, including GeneID, Chr, Start, End and Strand. These columns can be in any order. More columns can be included in the annotation. Columns are tab-delimited. Column names are case insensitive. GeneID column may contain integers or character strings. Chromosomal names included in the Chr column must match those used included in the mapping results, otherwise reads will fail to be assigned.

The adapters file (example provided) is a multifasta containing adapter sequences identified in the FASTQ files, for example, via the AdapterRemoval tool:

```
$ AdapterRemoval --identify-adapters --file1 <*.fastq1 file> --file2 <*.fastq2 file>
```

DE analysis is performed by DESeq2 through the dependent **GO_DE.R** script. For this purpose, each ***.count.txt** file (corresponding to each triplicate of treated and untreated conditions) should be concatenated into a single file (here provided as an example **count_file.csv**). Also, a **colData.csv** file explicitly mentioning which columns are treated/untreated is needed (example provided).

```
$ Rscript GO_DE.R count_file.csv colData.csv
```

This produces the following output files:

outinfo.txt: outputs for each gene, information on non-normalised and normalised read counts, log2FC, P value, and FDR.

MyData.csv: Subset of outinfo.txt containing log2FC, standard error, P value, and FDR.

plotMA.pdf and **plotMAShrunk.pdf**: Regular and shrunken MA plots (log2FC vs mean of normalized read counts).

PCA.pdf and **dendrogram.pdf**: Builds a dendrogram to evaluate sample clustering and a 2D PCA plot to check for potential outliers.

4.1.6 Section 6

Gene flux analysis - Horizontal Gene Transfer (HGT)

Fundamental dependencies

- Java
- Count

Example usage

```
$ ./GO_HGT.sh <Pan_genome_matrix.csv> <Newick_tree> <Species_prefix> <Posterior_gain_
↪probability>
```

This wrapper runs Count to perform ancestral reconstruction and infer family and lineage specific characteristics along the evolutionary tree. It takes as input a pan-genome matrix file (example provided for 45 *C. difficile* genomes), which can be obtained, for example, with Roary. It also requires the species tree in Newick format (example provided). The user is also required to specify the posterior gain probability for the family sizes at inner nodes [0-1].

The final output file **Species_prefix.Gains.out** contains the sum of gene families acquired at each tip of the phylogenetic tree.

Gene flux analysis - Homologous Recombination (HR)

Fundamental dependencies

- Recent version of R
- ClonalFrameML

- convertAlignment.pl
- RAxML

Example usage

```
# Install the R packages:
> install.packages(c("PopGenome","ape"))
```

then run:

```
$ ./GO_HR.sh <ordered_core_alignment> <Newick_tree> <Species_prefix>
```

This wrapper runs ClonalFrameML given an ordered core genome alignment and corresponding phylogenetic tree in Newick format. The ordered core genome alignment can be extracted from a progressiveMauve alignment using stripSubsetLCBs as described here.

Let's look as an example. Briefly, starting in the directory where the data resides (and using the test genomes from **Section 1**):

```
$ progressiveMauve --output=full_alignment.xmfa 020482.fas 020711.fas 020477.fas␣
↪020709.fas
$ stripSubsetLCBs full_alignment.xmfa full_alignment.xmfa.bbcols core_alignment.xmfa␣
↪500
```

The first command constructs a multiple genome alignment of the four genomes. The second command strips out variable regions from the alignment to leave only core alignment blocks longer than 500 nt.

Then, we need to build the core-genome phylogenetic tree in Newick format:

```
$ perl convertAlignment.pl -i core_alignment.xmfa -o core.conv.fas -f fasta -g xmfa -c
$ raxmlHPC-PTHREADS-AVX -s core.conv.fas -n best-CDIF -m GTRGAMMA -j -p 12345 -# 20
$ raxmlHPC-PTHREADS-AVX -s core.conv.fas -n CDIF -m GTRGAMMA -j -# 100 -b 12345 -p␣
↪67890
$ raxmlHPC-PTHREADS-AVX -f b -m GTRGAMMA -s core.conv.fas -z RAxML_bootstrap.CDIF -t␣
↪RAxML_bestTree.best-CDIF -n finalboot-CDIF
```

The first raxml command will generate 20 ML trees on distinct starting trees and also print the tree with the best likelihood to a file called **RAxML_bestTree.best-CDIF**.

Now we will want to get support values for this tree, so for the second raxml command we provide a bootstrap random number seed via -b 12345 and the number of bootstrap replicates we want to compute via -# 100. This will print a file called **RAxML_bootstrap.CDIF**.

We can use the latter two to draw bipartitions on the best ML tree as follows. The third raxml command will produce the files **RAxML_bipartitions.CDIF** (support values assigned to nodes) and **RAxML_bipartitionsBranchLabels.CDIF** (support values assigned to branches of the tree).

Finally we can compute HR:

```
$ ./GO_HR.sh core.conv.fas RAxML_bipartitions.CDIF CDIF
```

The output files include:

CDIF.smout.* for the standard model run;

CDIF.pbmout.* for the per-branch model run (recombination parameters are estimated per branch);

Core_Sites.txt a list of core sites in the alignment;

A pdf with a graphical representation of the HR landscape.

4.1.7 Section 7

CRISPR detection

Fundamental dependencies

- Java
- CRT

Example usage

```
$ ./GO_CRISPRs.sh <*.fasta> <CRT_filename.jar> <minNR> <minRL> <maxRL> <minSL> <maxSL>
↪ <searchWL>

minNR: minimum number of repeats a CRISPR must contain; default 3
minRL: minimum length of a CRISPR's repeated region; default 19
maxRL: maximum length of a CRISPR's repeated region; default 38
minSL: minimum length of a CRISPR's non-repeated region (or spacer region); default 19
maxSL: maximum length of a CRISPR's non-repeated region (or spacer region); default 48
searchWL: length of search window used to discover CRISPRs; (range: 6-9)
```

This wrapper runs CRT on a FASTA file and parses the output file (**.crispr_raw**) into a tab delimited output (**.crispr_parsed**).

For example, running it on 020477.fas:

```
$ ./GO_CRISPRs.sh 020477.fas CRT1.2-CLI.jar 3 19 38 19 48 8
```

returns **020477.crispr_raw** and **020477.crispr_parsed** files with 9 putative CRISPRs found.

Prophage and Integron detection

Fundamental dependencies

- Phage Finder and corresponding third party dependencies
- EDirect
- Integron Finder

Example usage

```
$ ./GO_Prophages_Integrons.sh <genome_accession_number> <prefix>
```

This wrapper searches prophages and integrons in full genome sequences using Phage Finder and IntegronFinder.

For prophages, it outputs:

***.tab file**: a tab-delimited file containing (contig_id, size of the genome, G+C% content of the genome, 5' end of the phage region, 3' end of the phage region, size of region in bp, label (small, medium, large), region type (prophage, integrated element, degenerate), sequence of attR, sequence of attL, name of integration target, G+C% of region, 5' feat_name or locus name, 3' feat_name or locus name, # integrase HMM hits, # core_HMM hits, # above noise core_HMM hits, # lytic gene HMM hits, # tail HMM hits, # Mu HMM hits, orientation of the prophage based on orientation of the target or the position of the integrase, the distance from att site to integrase, and the number of genes in the region;

***.seq file**: a file in FASTA format containing the DNA sequence of each gene within the phage region;

A subdirectory with:

phage_phinder_<id>.log: a log file recording Phage_Finder progress;

phgraph file: an XGRAPH plot of the phage regions;

phreport file: a tab-delimited report file that shows (coordinate incremented by the step size, # hits per window, and the feat_name or locus name of the hits);

By default, integron_finder will output 3 files under **Results_Integron_Finder_mysequences**:

mysequences.integrons: A file with all integrons and their elements detected in all sequences in the input file;

mysequences.summary: A summary file with the number and type of integrons per sequence;

integron_finder.out: A copy of standard output;

Running BEAST via Docker

For a quicker and more streamlined use of BEAST, and due to its large number of dependencies, the user may consider running it via Docker. Docker is a tool that uses OS-level virtualization, thus making it easier to run applications through capsules (containers). A step-by-step tutorial is presented below:

- Install Docker on your computer following the instructions provided here.
- Now, let us export capsules pertaining to the BEAST pipeline, and run them on our local machine. These capsules are stored on the Code Ocean platform, and can be found at each of the following links (missing ones will be updated soon) :

Section 1 - Motif Refining

Section 2 - Motif Count Exceptionalities using Markov Models

Section 2 - Multi-Scale Representation (MSR) of Methylation Motifs

Section 3 - Conservation of Methylation Motifs across Genomes

Section 4 - TFBS Mapping

Section 4 - TSS Mapping

Section 5 - Differential Gene Expression Analysis

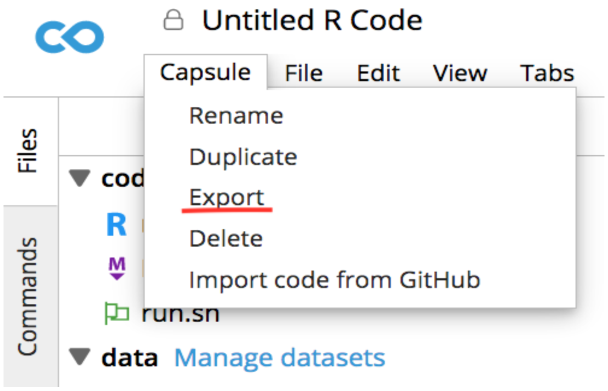
Section 6 - Gene Flux Analysis - Homologous Recombination (HR)

Section 6 - Gene Flux Analysis - Horizontal Gene Transfer (HGT)

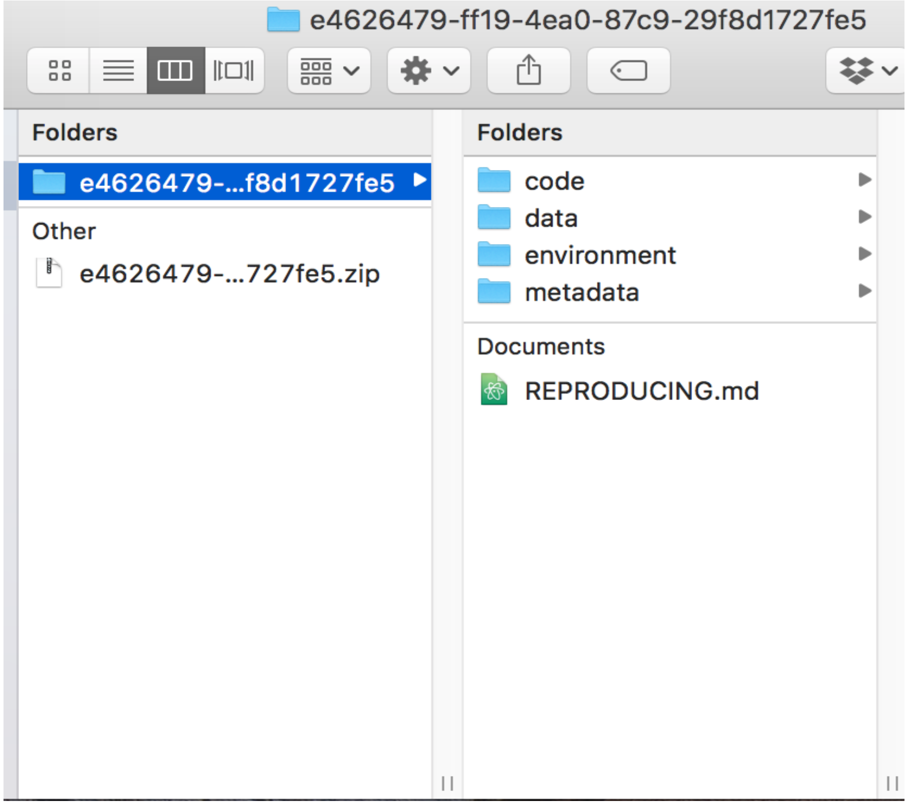
Section 7 - CRISPR detection

Section 7 - Prophage and Integron detection

To download any of these capsules, click the 'Capsule' tab in the menu and select 'Export':



This will prompt a download screen where you can download the environment template, metadata, code, and, optionally, the data (the latter is needed if you would like to run a test example). After unzipping, the user will see something like the following (this screenshot comes from a mac):



REPRODUCING.md contains specific instructions for how to reproduce the capsule's results locally, with notes on the necessary prerequisites and commands. If you have downloaded a published capsule, this document will point to the preserved Docker image in our registry.

/code has the capsule's code, and /data has the capsule's data.

/metadata has a file called metadata.yml.

The /environment folder contains, at a minimum, a file called Dockerfile. Dockerfile is the recipe for rebuilding the capsule's computational environment locally. A postInstall script may be present as well.

- Finally, let's open a command-line terminal (or Power Shell in Windows), and, inside the recently downloaded folder, simply copy / paste the following commands where the large alpha-numeric tag corresponds to the name of the container recently downloaded, (here exemplified for Section 2 - Motif Count Exceptionalities using Markov Models):

```
$ cd environment && docker build . --tag 78bdb7a2-6832-4167-b5e8-fd8f925e67e5; cd ..
```

```
$ docker run --rm \
--workdir /code \
--volume "$PWD/data":/data \
--volume "$PWD/code":/code \
--volume "$PWD/results":/results \
78bdb7a2-6832-4167-b5e8-fd8f925e67e5 ./run.sh
```

- Output files are automatically placed in the Results folder. The container may be re-run using a different initial dataset. For this, the user just needs to replace the example files found within the /data folder with a different dataset of its own.

To re-run the capsule with alternative initial starting conditions, the user just need to edit the run.sh script, or simply replace it by the corresponding one liner given in the corresponding section above, changing the desired parameters. Again, taking the same capsule as example, we could test, for example, the exceptionality of the motif GATC in a user-provided yourfile.fasta according to a second order Markov model using:

```
$ docker run --rm \
--workdir /code \
--volume "$PWD/data":/data \
--volume "$PWD/code":/code \
--volume "$PWD/results":/results \
78bdb7a2-6832-4167-b5e8-fd8f925e67e5 ./GO_Abundance.sh yourfile.fasta GATC 4 2␣
↪gaussian
```

CHAPTER 5

Search

- search