Website
https://bitbucket.org/schultzmattd/methylpy/wiki/Home
https://bitbucket.org/schultzmattd/methylpy/overview

Installation
Enter the directory where you would like to install methylpy and run
"git clone https://bitbucket.org/schultzmattd/methylpy.git"

Dependencies
numpy
scipy
cutadapt
samtools
bowtie and/or bowtie2
picard (Only picard.jar is needed)

To install numpy and scipy, the easiest way is to install anaconda. If you have the methylpy cloned and have these dependencies installed, the last step is to include methylpy into python library search path list. Supposing that your methylpy clone is at /YOUR/PATH/methylpy/, then you can include the command below in your ~/.bashrc file:

"export PYTHONPATH=/YOUR/PATH/methylpy/:$PYTHONPATH"

Also, make sure the paths to cutadapt, samtools, bowtie and/or bowtie2 are included in PATH. Otherwise, you will have to pass these path to methylpy functions when running the pipeline.

Checking
To check whether your installation is successful, try:

"
python2.7
>>>import methylpy.call_mc
>>>import methylpy.DMRfind
"

If you get no error, there is a good chance that methylpy is correctly installed. However, these commands do not check the dependencies.

Running methylpy for paired-end MethylC-seq read mapping
Step -1 Set directory for temporary files

Before running methylpy, please set the directory for storing temporary files to be a directory with large space by adding below command to ~/.bashrc file:

export TMPDIR=/YOUR/TMP/DIR/

Step 0 Prepare (converted) reference genome for mapping

Include the sequence of lambda phage genome (as control) into the reference genome fasta file. For example:

cat mm10.fa chrL.fa > mm10_with_lambda.fa

chrL.fa can be downloaded from http://neomorph.salk.edu/yupeng/share/chrL.fa

Step 1 Build bowtie/bowtie2 index for converted genome

Build the bowtie/bowtie2 index for bisulfite-converted genome. Note that the genome index used for WGBS data processing is different from the one used for ChIP-seq or genome sequence data processing.

```
---------------------------------------------------------------------------------------------
#Start of the script

from methylpy.call_mc import build_ref
#fasta file(s) of genome
#Specifying multiple files like
input_files=['chr1.fa','chr2.fa',...,'chrY.fa','chrL.fa'] also #work
input_files=['mm10_with_lambda.fa']

#Prefix of output files
output='mm10'

#Build bowtie2 index
build_ref(input_files,output,bowtie2=True)

#Build bowtie index
#build_ref(input_files,output,bowtie2=False)

#End of the script
---------------------------------------------------------------------------------------------
```

Step 2.1 Create script to run methylpy (paired-end data)

Create a run_call_mc.py file like for each sample:

```
---------------------------------------------------------------------------------------------
#Start of the script: run_call_mc.py

from methylpy.call_mc import run_methylation_pipeline_pe

#Sample name
sample="Test_WGBS"

#Path to fastq files
fastq_R1 = ["lib1/*_1.fastq"," lib2/*_1.fastq"] #read 1
```

```
fastq_R2 = ["lib1/*_2.fastq"," lib2/*_2.fastq"] #read 2

#Assoicate fastq files with corresponding library names
libraries=["lib1","lib2"]

#Genome (take mm10 as example)
f_ref = "/home/users/mm10/mm10_f"
r_ref = "/home/users/mm10/mm10_r"
ref_fasta = "/home/users/mm10/mm10.fa"

#Number of processors
num_procs = 8

#Adapter to trim
adapter_seq_R1='AGATCGGAAGAGCACACGTCTGAAC'
adapter_seq_R2='AGATCGGAAGAGCGTCGTGTAGGGA'

#Control (unmethylated lambda genome) to estimate bisulfite conversion
efficiency #which will be measured by bisulfite non-conversion rate
m_control="chrL:"

run_methylation_pipeline_pe(
                    read1_files = fastq_R1,
                    read2_files = fastq_R2,
                    libraries=libraries,
                    sample=sample,
                    forward_reference = f_ref,
                    reverse_reference = r_ref,
                    reference_fasta = ref_fasta,
                    unmethylated_control=m_control,
                    path_to_samtools="",
                    path_to_bowtie="",
                    num_procs=num_procs,
                    trim_reads=True,
                    path_to_cutadapt="",
                    adapter_seq_R1= adapter_seq_R1,
                    adapter_seq_R2= adapter_seq_R2,
                    sig_cutoff=0.01,
                    binom_test=True,
                    bh=True,
                    sort_mem='1G',
                    path_to_MarkDuplicates=”/YOURPATH/picard/”
)
#End of script
---------------------------------------------------------------------------------------
```

Step 2.2 Create script to run methylpy (for single-end data)

```
#Start of the script: run_call_mc.py
from methylpy.call_mc import run_methylation_pipeline

#Sample name
sample="Test_WGBS"

#Path to fastq files
files=["lib1/*.fastq","lib2/*.fastq"]

#Assoicate fastq files with corresponding library names
libraries=["lib1","lib2"]

#Genome (take mm10 as example)
f_ref = "/home/users/mm10/mm10_f"
r_ref = "/home/users/mm10/mm10_r"
ref_fasta = "/home/users/mm10/mm10.fa"

#Number of processors
num_procs = 8

#Adapter to trim
adapter = 'AGATCGGAAGAGCTCGTATGCC'

#Control (unmethylated lambda genome) to estimate bisulfite conversion
efficiency which will be measured by bisulfite non-conversion rate
m_control="chrL:"

run_methylation_pipeline(
                        files=files,
                        libraries=libraries,
                        sample=sample,
                        forward_reference = f_ref,
                        reverse_reference = r_ref,
                        reference_fasta = ref_fasta,
                        unmethylated_control=m_control,
                        path_to_samtools="",
                        path_to_bowtie="",
                        num_procs=num_procs,
                        path_to_cutadapt ="",
                        adapter_seq=adapter_seq,
                        sig_cutoff=0.01,
                        binom_test=True,
                        bh=True,
                        sort_mem='1G',
```

```
                path_to_MarkDuplicates="/YOURPATH/picard/"
            )

    #End of script
    ----------------------------------------------------------------------------------------
```

Step 3 Running methylpy and read output

Copy the script to output directory and run it with command "python2.7 run_call_mc.py > log 2> err". Printed output will be store in the log file and error messages will be store in the err file. If everything is correct, the allc files (final output from methylpy) will be created in the directory. allc files are tab-delimited files containing seven columns, which correspond to chromosome, position, strand, methylation class, methylation count, total coverage, and methylation call.

Step 4 Remove some temporary files to free up space

Remove *mpileup* files in the output directory. These intermediate files in total take usually huge space (hundred gigabytes) and they can be regenerated using bam file

Step 5 Calling differentially methylated regions

You can use the below scripts to get regions with different methylation patterns in different samples.

```
    ----------------------------------------------------------------------------------------
    #Start of the script: run_call_mc.py

    from methylpy.DMRfind import DMRfind

    #Samples included in the comparison
    samples = ["Test_WGBS_1"," Test_WGBS_2"]

    #Regions included in the DMR analysis
    chrom = map(str,range(1,20))
    chrom.extend('X')
    chrom.extend('Y')
    region_dict={}
    for chr in chrom:
       region_dict[chr]=[0,5000000000000000]

    #methylation type
    mc_type=['CGN']

    #Number of processors
    num_procs=4

    path_to_allc = "/path/where/you/store/all/allc files/"
```

```
output_prefix = "DMR_CG_Test_WGBS"

DMRfind(
        mc_type=mc_type,
        region_dict=region_dict,
        samples=samples,
        path_to_allc= path_to_allc,
        num_sims=3000,
        num_sig_tests=100,
        use_mc_status = False,
        num_procs = num_procs,
        save_result = output_prefix,
        dmr_max_dist=250)
#End of script
```

Hard drive space and memory requirement

Hard drive space

usually space that is three times of the size of all uncompressed fastq files is enough for running the pipeline.

Memory

methylpy is memory efficient but the sort function (from linux system) could potentially occupy memory. This can be control by the “sort_mem” option (see below). For example, if a node has 24G memory and 16 CPUs, the “sort_mem” option should be set to be “1G” if all 16 CPUs are used. In that case, methylpy will take a most 16 * 1G = 16G memory. Please set the value of sort_mem according to the capacity of your computer/cluster.

Help information

To get help information, please run

“

```
python2.7
>>>from methylpy.call_mc import run_methylation_pipeline_pe
>>>help(run_methylation_pipeline_pe)
```

“

Appendix – Help information about run_methylation_pipeline_pe

run_methylation_pipeline_pe(read1_files, read2_files, libraries, sample, forward_reference, reverse_reference, reference_fasta,unmethylated_control, quality_version='1.8', path_to_samtools='', path_to_bowtie='', bowtie_options=[], num_procs=1, trim_reads=True, path_to_cutadapt='', adapter_seq_R1='AGATCGGAAGAGCACACGTCTGAAC', adapter_seq_R2='AGATCGGAAGAGCGTCGTGTAGGGA', max_adapter_removal=None, overlap_length=None, zero_cap=None, error_rate=None, min_qual_score=10, min_read_len=30, sig_cutoff=0.01, min_cov=0, binom_test=True, bh=False, keep_temp_files=False, num_reads=-1,

save_space=True, bowtie2=True, sort_mem='500M', path_to_output='', in_mem=False, min_base_quality=1, path_to_MarkDuplicates=False)

read1_files and read2_files are lists of fastq files of the forward and reverse reads
from paired-end bisulfite sequencing data, which you'd like to run through the pipeline.
The length of read1_files and read2_files should be the same. Also, Files in there two
lists should be ordered such that the forward reads for a particular read set are in the
same position of read1_files as the reverse reads are in the read2_files.
(i.e. the elements in each of these lists are paired)
Note that globbing is supported here (i.e., you can use * in your paths)

libraries is a list of library IDs (in the same order as the files list) indiciating which
libraries each set of fastq files belong to. If you use a glob, you only need to indicate
the library ID for those fastqs once (i.e., the length of files and libraries should be
the same)

sample is a string indicating the name of the sample you're processing. It will be included
in the output files.

forward_reference is a string indicating the path to the forward strand reference created by
build_ref

reverse_reference is a string indicating the path to the reverse strand reference created by
build_ref

reference_fasta is a string indicating the path to a fasta file containing the sequences
you used for mapping
input is the path to a bam file that contains mapped bisulfite sequencing reads

unmethylated_control is the name of the chromosome/region that you want to use to estimate
the non-conversion rate of your sample, or the non-conversion rate you'd like to use.
Consequently, control is either a string, or a decimal
If control is a string then it should be in the following format: "chrom:start-end".
If you'd like to specify an entire chromosome simply use "chrom:"

quality_version is either an integer indicating the base offset for the quality scores or a float indicating
which version of casava was used to generate the fastq files.

path_to_samtools is a string indicating the path to the directory containing your
installation of samtools. Samtools is assumed to be in your path if this is not
provided

path_to_bowtie is a string indicating the path to the folder in which bowtie resides. Bowtie
is assumed to be in your path if this option isn't used

bowtie_options is a list of strings indicating options you'd like passed to bowtie2 (or bowtie)
(default for bowtie2: "-X 1000 -k 2 --no-mixed --no-discordant --no-overlap")
(default for bowtie: "-X 1000 -S -k 1 -m 1 --best --strata --chunkmbs 64 -n 1")

num_procs is an integer indicating how many num_procs you'd like to run this function over

trim_reads is a boolean indicating that you want to have reads trimmed by cutadapt. This
option is currently not compatible with "in_mem" mode.

path_to_cutadapt is the path to the cutadapt execuatable. Otherwise this is assumed to be in your path.

adapter_seq_R1:
Sequence of an adapter that was ligated to the 3' end of read 1. The adapter itself and anything that follows is trimmed.

adapter_seq_R2:
Sequence of an adapter that was ligated to the 3' end of read 2. The adapter itself and anything that follows is trimmed.

max_adapter_removal indicates the maximum number of times to try to remove adapters. Useful when an adapter gets appended multiple times.

overlap_length is the minimum overlap length. If the overlap between the read and the adapter is shorter than LENGTH, the read is not modified. This reduces the no. of bases trimmed purely due to short random adapter matches.

zero_cap causes negative quality values to be set to zero (workaround to avoid segmentation faults in BWA).

error_rate is the maximum allowed error rate (no. of errors divided by the length of the matching region) (default: 0.1)

min_qual_score allows you to trim low-quality ends from reads before adapter removal. The algorithm is the same as the one used by BWA (Subtract CUTOFF from all qualities; compute partial sums from all indices to the end of the sequence; cut sequence at the index at which the sum is minimal).

min_read_len indicates the minimum length a read must be to be kept. Reads that are too short even before adapter removal are also discarded. In colorspace, an initial primer is not counted. It is not recommended to change this value in paired-end processing because it may results in the situation that one of the two reads in a pair is discarded. And this will lead to error.

sig_cutoff is a float indicating the adjusted p-value cutoff you wish to use for determining whether or not a site is methylated

min_cov is an integer indicating the minimum number of reads for a site to be tested.

binom_tests indicates that you'd like to use a binomial test, rather than the alternative method outlined here https://bitbucket.org/schultzmattd/methylpy/wiki/Methylation%20Calling

keep_temp_files is a boolean indicating that you'd like to keep the intermediate files generated by this function. This can be useful for debugging, but in general should be left False.

num_reads is an integer indicating how many reads you'd like to process at a time. Breaking up the processing this way can reduce the size of some of the intermediate files that are created. Use this if you're strapped for space.

save_space indicates whether or not you'd like to perform read collapsing right after mapping or once all the libraries have been mapped. If you wait until after everything has been mapped, the collapsing can be parallelized. Otherwise the collapsing will have to be done serially. The trade-off is that you must keep all the mapped files around, rather than deleting them as they are processed, which can take up a considerable amount of space. It's safest to set this to True.

bowtie2 specifies whether to use the bowtie2 aligner instead of bowtie

sort_mem is the parameter to pass to unix sort with -S/--buffer-size command

in_mem: does not write files as part of run_mapping, keeps all input/output in memory (This mode is unavailable now)

path_to_output is the path to a directory where you would like the output to be stored. The default is the same directory as the input fastqs.

min_base_quality is an integer indicating the minimum PHRED quality score for a base to be included in the mpileup file (and subsequently to be considered for methylation calling)

path_to_MarkDuplicates is the path to MarkDuplicates jar from picard. Default is false indicating that you don't want to use this jar for duplication removal