# User Manual for **Sigma** V1.0.2

***Tae-Hyuk Ahn, Juanjuan Chai and Chongle Pan***

**October, 1, 2014**

## Table of Contents

# Introduction

## What is Sigma?

Identification of microbes and accurate estimation of their relative abundance are crucial subjects in metagenomic analysis. Existing taxonomic classification methods are unsuitable for metagenomic biosurveillance due to the following three factors. First, reference genomes including complete genomes of pathogenic bacteria are required. Second, strain-level genome identification from the same species is essential. Third, statistical confidence evaluation of the metagenomic analysis is recommended. We developed Sigma (**S**train-level **I**nference of **G**enomes from **M**etagenomic **A**nalysis), a novel sequence similarity-based approach for strain-level identification of genomes from metagenomic analysis for biosurveillance. Nucleotide-level alignments analysis using maximum likelihood estimation empowers Sigma to estimate the relative abundances and likelihood ratios of each genome accurately given a list of reference genomes. Hybrid parallel architecture of the Sigma allows its computation to be scalable from desktops to supercomputers for different sizes of metagenomic reads and reference datasets.

## How to cite Sigma?

T.-H. Ahn J. Chai and C. Pan, Sigma: Strain-level Inference of Genomes from Metagenomic Analysis for Biosurveillance, *Bioinformatics*, 2014.

# Obtaining and Installing the Program

## Prerequisites:  Install Bowtie2 and SAMtools.

The inputs for Sigma are a database of reference genomes and metagenomic sequences. In the first step, Sigma uses a short-read alignment algorithm, Bowtie 2, to align all metagenomic reads onto every reference genome, allowing up to a given number of mismatches per read. Bowtie2 outputs alignment results in SAM (Sequence Alignment/Map) format. To minimize disk usage and space, Sigma converts the SAM format to BAM (Binary form of the SAM) format on the memory buffer using SAMtools. Current Sigma was experimented with Bowtie2 2.1.0 and SAMtools 0.1.19.

Sigma was designed to work with any other short-read alignment algorithm that uses FASTQ/FASTA formats for read input and SAM/BAM formats for alignment output, but we strongly recommend to users to use Bowtie2.

After you download and install the latest stable version of Bowtie2 and SAMtools, user should update system path or provide the installed directories of the tools in the Sigma configuration file.

## Install Sigma

Binaries were built and tested in the Intel architectures (x86_64) running Linux/Unix environment. Sigma package was developed using C++. C++ source codes were compiled and linked by g++ (gcc version 4.7.2).

Please download the last version of Sigma at http://sigma.omicsbio.org/download/. You can decompress it from the command line

```
$ tar xzvf sigma-version.tar.gz
```

After you extract the tar ball, please check usage with `–h` option. For example,

```
$ cd Sigma/bin
$ ./sigma -h
```

To run Sigma in anywhere without providing location of Sigma, user needs to add the Sigma directory into PATH environment variable. For example, if user uses a bash shell, open ~/.basrch and add the below line.

```
export PATH=/home/user/Sigma:$PATH
```

## Building Sigma from source codes

Building Sigma from source requires a GNU-like environment with GCC, GNU Make and other basics. Try it

```
$ cd Sigma/src
$ make
```

If you fail to make the binary, then you should install IPOPT by yourself. Sigma uses the primal-dual interior point NLP method implemented in the IPOPT library to solve the probabilistic model. Ipopt (Interior Point OPTimizer, pronounced eye-pea-Opt) is a software package for large-scale nonlinear optimization. User should download and install the IPOPT package, and provide library of IPOPT into the Sigma. Sigma was experimented with the latest stable version IPOPT 3.11.

Download the IPOPT compressed source at http://www.coin-or.org/download/source/Ipopt/ and install it. Please refer the IPOPT document to install it. IPOPT requires external packages that are not included in the IPOPT source code distribution. Below steps is what we did:

1. Download IPOPT source package.
2. Extract the package.

```
$ tar xzvf sigma-version-src.tar.gz
$ mv Ipopt-x.y.z CoinIpopt
```

3. Download external code.

```
$ cd $IPOPTDIR/ThirdParty/Blas
$ ./get.Blas
$ cd ../Lapack
$ ./get.Lapack
$ cd ../Mumps
$ ./get.Mumps
```

HSL and Mumps linear solver libraries are compatible with Sigma. The Sigma binary, however, only contains Mumps library for licensing issue of HSL. User can install personal license HSL library for using IPOPT in Sigma.

4. Install IPOPT

```
$ mkdir $IPOPTDIR/build
$ cd $IPOPTDIR/build
$ ../configure
$ make
$ make install
```

5. Provide IPOPT path to Sigma. Open the Sigma Makefile and update two fields; `INCL` and `LDFLAGS`. The `INCL` path should be `$IPOPTDIR/build/include/coin`. You can get the library path from `$IPOPTDIR/build/share/coin/doc/Ipopt/ipopt_addlibs_cpp.txt.` Example of Makefile as below:

```
############
# Makefile #
############

#########################################################
# Please change values based on your system environment #
#########################################################

# C++ Compiler
CC = g++

# Open MPI Compiler
MPICC = mpiCC

# C++ Compile Flag
CFLAGS = -c -Wall

# GCC OpenMP Flag
OMPFLAG = -fopenmp

# Static build
STFLAG = -static

# Include directories
INCL = -I/home/thm/Software/CoinIpopt/build/include/coin

# Library
LDFLAGS = -L/home/thm/Software/CoinIpopt/build/lib64 -
L/usr/lib64/gcc/x86_64-suse-linux/4.7 -L/usr/lib64/gcc/x86_64-suse-
linux/4.7/../../../../lib64 -L/lib/../lib64 -L/usr/lib/../lib64 -
L/usr/lib64/gcc/x86_64-suse-linux/4.7/../../../../x86_64-suse-linux/lib
-L/usr/lib64/gcc/x86_64-suse-linux/4.7/../../..
-lipopt -llapack -ldl -lcoinmumps -lfblas -lpthread -lgfortran -lm -
lquadmath
```

After you update the Sigma Makefile, you can build the binaries from the command line

```
$ make
```

It is computationally intensive to align hundreds of millions metagenomic reads into thousands of genomes. Sigma provides MPI (Message Passing Interface) modules, a parallel computation communication feature to distribute the computation across nodes in distributed-memory systems from small clusters to supercomputers. To use this feature, the system should have MPI protocol such as MPICH or OpenMPI. We tested Sigma in both MPI features. If you have MPI protocol in your system, just build Sigma MPI binaries from the command line

```
$ make mpi
```

Make sure your shell search path includes the directory containing `mpicxx` or `mpiCC` to compile Sigma MPI. You should update the `MPIFLAG` based on your MPI compiler wrapper in the Makefile.

# Sigma Configuration File

## Create configuration file for each experiment

Sigma provides a configuration file to help users easily setup input/output and parameters for running Sigma. We recommend users to create the configuration file for each experiment to keep the history of the experiment. We provide a template of the configuration file in the package. For example, if you want to make a directory for an experiment “run01”, then

```
$ mkdir run01
$ cp $SigmaDIR/sigma_config.cfg ./run01/
```

## Configuration file setting

The configuration file follows the INI format that is a simple text file with a basic structure composed of “sections” and “properties”.  Every key has a *name* and a *value*, delaminated by an equals sign (=). [] is used for section name, e.g., [Section Name]. Pounds (#) at the beginning of the line indicate a comment. Comment lines are ignored.

For example, if you want to provide the specific paths of prerequisite software, then setup the configuration file as below.

```
#################################
##### Parameters for Sigma #####
#################################

#-----------------------------#
# Program selectiton and path #
#-----------------------------#
[Program_Info]

# Provide bowtie2 and samtools software directory path.
# If you don't know the exact path, just comment the below line with #.
# Then Sigma will search the programs automatically from env path.
Bowtie_Directory=/home/user/software/bowtie2-2.1.0
Samtools_Directory=/home/user/software/samtools-0.1.9
```

The current configuration file has 5 sections, [Program_Info], [Data_Info], [Bowtie_Search], [Model Probability], and [Statistics]. The keys and values for each section of the configuration are described in the each section of the manual.

# Aligning Metagenomic Reads onto Reference Genome Database

## How to prepare reference database?

The reference genome database for Sigma may contain dozens to thousands genomes. In this study, the reference genome databases were constructed from RefSeq downloaded from NCBI ftp site (ftp://ftp.ncbi.nlm.nih.gov/genomes/Bacteria/all.fna.tar.gz).

```
$ mkdir RefSeq
$ cd RefSeq
$ wget ftp://ftp.ncbi.nlm.nih.gov/genomes/Bacteria/all.fna.tar.gz
$ tar xzvf all.fna.tar.gz
```

In November 2013, NCBI RefSeq database includes 2731 bacterial and archaeal complete genomes (2.7GB compressed, 9.1GB decompressed). Sigma uses a hierarchical structure for the reference genome database as set up by RefSeq:  a root directory containing many sub-directories of FASTA files. Each sub-directory corresponds to a genome and the directory names are used by Sigma as the genome names because they are unique identifiers. The FASTA files in a sub-directory correspond to multiple chromosomes and plasmids of a genome. Genomes can be added or removed from the reference genome database by simply adding or removing their sub-directories in the root directory. The root directory is specified in the configuration file of Sigma.

```
#-------------------------#
# Data directory and path #
#-------------------------#
[Data_Info]

# Reference genome directory
#   - required database hierarchy:
#   [database directory] - [genome directory] - [fasta file]
#                        - [genome directory] - [fasta file]
#                                             - [fasta file]
#                        - [genome directory] - [fasta file]
#                                             - [fasta file]
#                             ...
Reference_Genome_Directory=/home/user/database/refseq
```

## Quality control of metagenomic reads

We recommend users to preprocess the reads to ensure data quality. For example, HMP WGS reads were preprocessed by identifying and masking human reads, removing duplicated reads, and trimming low quality bases. Detailed description for the preprocessing of the HMP datasets is available. NGS QC Toolkit, FASTX-Toolkit, and sickle can be used for quality processing.

## How to provide metagenomic reads to Sigma?

User needs to provide the (preprocessed) metagenomic reads set in the configuration file. Sigma automatically detects FASTQ /FASTA format from the input data. If you have paired-end input, you should provide each mate file in the Sigma configuration file. The current Sigma cannot work with both paired-end inputs and single-end inputs. So, user should provide either paired-end inputs or single-end inputs. Multiple input sets should be separated by comma (,), e.g., HMP01_pe_1.fq,HMP02_pe_1.fq.

```
# Provide metagenome NGS read(s) path using comma separation files.
# You should select only one: paired-end read(s) OR single end read(s).
# You should comment the unselected option
#
# For paired-end reads
Paired_End_Reads_1=/home/user/database/HMP/HMP01_pe_1.fq
Paired_End_Reads_2=/home/user/database/HMP/HMP01_pe_2.fq
#
# For single-end reads
# Single_End_Reads=
```

## Index reference genomes before aligning reads to genomes

Before aligning metagenomic reads onto reference genomes using Bowtie2, the reference genomes should be indexed using Bowtie2 indexer (`bowtie2-build`) for fast and memory-efficient aligning. If the reference genomes are indexed once, users do not need to re-index the genomes again. The indexed files are generated in each genome directory. The basename of the index is the reference genome name.

Bowtie2 does not support parallel indexing of the genomes. Sigma provides two types of parallelization indexing wrappers to scale up from multi-core workstations to distributed-memory clusters. One (`sigma-index-genomes`) is for multi-core computers using multi-processing, and the other (`sigma-index-genomes-mpi`) is for distributed-memory clusters using MPI protocol.

### sigma-index-genomes

```
$ sigma-index-genomes -h

[Usage]
  sigma-index-genomes [options] -c <config file path> -w <working
directory>

[Inputs]
  1. config file path (default: sigma_config.cfg)
    - if config file is not specified, the program will search it in the
working directory
    - include bowtie search options and more
  2. working directory (default: current running directory)
    - if working_directory is not specified, the program will work in the
current directory
    - results will be generated in working directory

[Options]
  -h/--help
  -v/--version
  -p/--multi-processes <int>    # number of multi-processes (default: 1)

[Outputs]
  Bowtie2 index files for each genome will be generated in the genome
directory
```

### sigma-index-genomes-mpi

```
$ sigma-index-genomes-mpi -h

[Usage]
  sigma-index-genomes-mpi [options] -c <config file path> -w <working
directory>

[Inputs]
  1. config file path (default: sigma_config.cfg)
  2. working directory (default: current running directory)

[Options]
  -h/--help
  -v/--version

[Outputs]
  Bowtie2 index files for each genome will be generated in the genome
directory
```

To run the MPI program, you can use "`mpirun`" or "`mpiexec`" wrapper with "`-np`" or "`-n`"option for setting number of processes. The executable wrapper is different for your MPI environment of the system, so please contact your system administrator to learn how to run MPI programs.

This is an example how to run the HMP01 experiment (working directory is `/home/user/test/HMP01` and the configuration file for the experiment is in the working directory) with `sigma-index-genomes-mpi` using 10 processes.

```
$ mpirun -np 10 sigma-index-genomes-mpi -w /home/user/test/HMP01
```

## Aligning read to genomes

Reads are aligned against the reference genome database using Bowtie2. User may specify the maximum number of mismatches per read, the range of the inter-mate distance, and number of threads for the Bowtie2 alignment in the Sigma configuration file.

```
[Bowtie_Search]

# Maximum count of mismatches for one read alignment. (Default: 3)
Maximum_Mismatch_Count=3

# The minimum fragment length (insert-size) for valid paired-end
alignments. (Default: 0)
Minimum_Fragment_Length=0

# The maximum fragment length (insert-size) for valid paired-end
alignments. (Default: 500)
Maximum_Fragment_Length=1000

# Number of threads for running one bowtie task. (Default: 1)
Bowtie_Threads_Number=4
```

Sigma also provides two types of parallelization aligning wrappers; one (`sigma-align-reads`) is for multi-core computers using multi-processing, and the other (`sigma-align-reads-mpi`) is for distributed-memory clusters using MPI protocol. Sigma uses a simple dynamic load balancing strategy in both multi-processing and MPI to maximize throughput and minimize wall-clock computing time.

### sigma-align-reads

```
$ sigma-align-reads -h

[Usage]
  sigma-align-reads [options] -c <config file path> -w <working directory>

[Inputs]
  1. config file path (default: sigma_config.cfg)
  2. working directory (default: current running directory)

[Options]
  -h/--help
  -v/--version
  -p/--multi-processes <int>    # number of multi-processes (default: 1)

[Outputs]
  bam format aligned results are reported in each output genome directory
```

Bowite2 supports parallel searching using threads (pthreads library). Multi-threads has speed-up scalability limit that is different to systems, but usually 8 to 16 threads in the current technology. Multi-processes overcome the limit of the multi-threads especially when a system has a shared memory with many cores.

For example, if your system has 8 cores, you may set the Bowtie2 threads number in the config file

```
# Number of threads for running one bowtie task. (Default: 1)
Bowtie_Threads_Number=8
```

and run aligning wrapper without `-p` option.

```
$ sigma-align-reads -w /home/user/test/HMP01
```

If your system has 48 cores, you may set the Bowtie2 threads number in the config file

```
# Number of threads for running one bowtie task. (Default: 1)
Bowtie_Threads_Number=8
```

and run aligning wrapper with `-p` option.

```
$ sigma-align-reads -p 8 -w /home/user/test/HMP01
```

This command executes 8 multiple aligning processes simultaneously using 8 threads for each process. Therefore, 48 cores can be used most efficiently to get maximum speed-up scalability and minimum wall-clock time.

### sigma- align-reads -mpi

```
$ sigma-align-reads-mpi -h

[Usage]
  sigma-align-reads-mpi [options] -c <config file path> -w <working
directory>

[Inputs]
  1. config file path (default: sigma_config.cfg)
  2. working directory (default: current running directory)

[Options]
  -h/--help
  -v/--version

[Outputs]
  bam format aligned results are reported in each output genome directory
```

Bowtie2 outputs alignment results in SAM (Sequence Alignment/Map) format. To minimize disk usage and space, Sigma converts the SAM format to BAM (Binary form of the SAM) format on the memory buffer using SAMtools. The alignment output directory "`sigma_alignments_output`" is generated in the working directory.

# Genome Identification using Maximum Likelihood Estimation

## How to build a probabilistic model and solve it?

Sigma parses the alignment results and generates the Q matrix (probabilistic model). The probabilistic model is described in the Results section of the paper in detail. The relative abundances of genomes are estimated from the Q matrix using maximum likelihood estimation (MLE) implemented in C++. Sigma solves the nonlinear programming (NLP) problem for MLE using the Ipopt library. Most of the computing time for NLP is spent in the objective function evaluation step, which involves non-trivial calculation on the large Q matrix. To speed up the calculation, we parallelized the objective functional evaluation step using multi-threading with OpenMP (Open Multi-Processing).

## How to run Sigma?

Before you run the Sigma, you may check the configuration file for setting model probability such as mismatch probability for one base pair (Default: 0.05 equals 5%) and minimum relative abundance rate to report (0.01 %).

```
[Model_Probability]

# Mismatch probability for one base pair. (Default: 0.05 equals 5%)
Mismatch_Probability=0.05

# Minimum relative abundance rate (%) to report (Default: 0.01)
Minimum_Relative_Abundance = 0.01
```

You can run the Sigma with `-t` (multi-threads) option.

```
$ sigma -t 16 -w /home/user/test/HMP01
```

We do not provide MPI version for this module.

### sigma

```
$ sigma -h

[Usage]
  sigma [options] -c <config file path> -w <working directory>

[Inputs]
  1. config file path (default: sigma_config.cfg)
  2. working directory (default: current running directory)

[Options]
  -h/--help
  -v/--version
  -t/--multi-threads <int>    # number of threads (default: 1)

[Outputs]
  sigma_out.qmatrix.txt
  sigma_out.gvector.txt
  sigma_out.gvector.html
  sigma_out.ipopt.txt
```

Sigma is a wrapper to call two modules: `sigma-build-model` and `sigma-solve-model`. If you only need the probabilistic matrix model (Q-matrix), just run the `sigma-build-model`.

### sigma-build-model

```
$ sigma-build-model -h

[Usage]
  sigma-build-model [options] -c <config file path> -w <working directory>

[Inputs]
  1. config file path (default: sigma_config.cfg)
  2. working directory (default: current running directory)

[Options]
  -h/--help
  -v/--version

[Outputs]
  sigma_out.qmatrix.txt
```

Then you can solve the model by `sigma-solve model` module. You can input the Q-matrix directly.

### sigma-solve-model

```
$ sigma-solve-model -h

[Usage]
  sigma-solve-model [options] -c <config file path> -w <working directory>

[Inputs]
  1. config file path (default: sigma_config.cfg)
  2. working directory (default: current running directory)

[Options]
  -h/--help
  -v/--version
  -i/--input-qmatrix <string> # provide q-matrix filename directly
  -t/--multi-threads <int>    # number of threads (default: 1)

[Outputs]
  sigma_out.gvector.txt
  sigma_out.gvector.html
  sigma_out.ipopt.txt
```

## Sigma output

Finally, Sigma summarizes results in two types of formats: a HTML format for result visualization and a text format for further data analysis.

- sigma_out.gvector.txt: relative abundance estimation output (text format)
- sigma_out.gvector.html: relative abundance estimation output (html format)

The Sigma outputs provide genome alignment results, estimated relative abundances, and percentage chances of genomes. Sigma also provides internal procedure output: sigma_out.qmatrix.txt for probabilistic model text file and sigma_out.ipopt.txt for IPOPT results. Users usually do not need to open these internal outputs.

## Statistical Confidence Measure

### What is Bootstrapping?

The bootstrap confidence interval estimation measures the uncertainty caused by the stochastic sampling process of metagenomic sequencing. Sigma generates a bootstrap Q matrix by randomly taking reads from the original Q matrix with replacement. Sigma then performs MLE using the bootstrap Q matrix. This process is iterated for many times to generate a set of bootstrap estimates, which are used to calculate a percentile confidence interval and relative standard deviation. Sigma distributes the parallel processing of bootstrap samples on a cluster using MPI/OpenMP.

### How to run Bootstrapping?

Users can set the number of bootstrapping iterations in the Sigma configuration file.

```
[Statistics]

# Number of iterations for bootstrapping. (Default: 100)
Bootstrap_Iteration_Number=200
```

You can run the Sigma bootstrapping with `-t` (multi-threads) and/or `-p` (multi-processes) options for multi-processing. Multi-threads (`-t`) option is used for solving IPOPT (`sigma-solve-model`) in parallel, and multi-processes (`-p`) is used for running bootstrapping iterations in parallel.

```
$ sigma-bootstrap -t 8 -p 8 -w /home/user/test/HMP01
```

#### sigma-bootstrap

```
$ sigma-bootstrap -h

[Usage]
  sigma-bootstrap [options] -c <config file path> -w <working directory>

[Inputs]
  1. config file path (default: sigma_config.cfg)
  2. working directory (default: current running directory)

[Options]
  -h/--help
  -v/--version
  -p/--multi-processes <int>  # number of multi-processes (default: 1)
  -t/--multi-threads <int>    # number of threads (default: 1)

[Outputs]
  sigma_out.stat_bootstrap.txt
```

We also provides MPI version for the bootstrapping.  Multi-threads (`-t`) option also can be used in the MPI mode. This example will launch 50 MPI tasks, each with 8 threads, total 400 cores required.

```
$ mpirun -np 50 sigma-bootstrap-mpi -t 8 -w /home/user/test/HMP01
```

### sigma-bootstrap-mpi

```
$ sigma-bootstrap-mpi -h

[Usage]
  sigma-bootstrap-mpi [options] -c <config file path> -w <working
directory>

[Inputs]
  1. config file path (default: sigma_config.cfg)
  2. working directory (default: current running directory)

[Options]
  -h/--help
  -v/--version
  -t/--multi-threads <int>    # number of threads (default: 1)

[Outputs]
    sigma_out.stat_bootstrap.txt
```

## Bootstrapping results

Bootstrapping reports average, STD, upper confidence bound, and lower confidence bound for the estimated relative abundance and percentage chance.

## What is likelihood ratio test (Jackknife)?

Sigma performs likelihood ratio tests on a target genome with a relative abundance above a user-defined threshold. The log-likelihood of the null hypothesis (i.e. the target genome is absent) is calculated using a Q matrix that does not contain the target genome. The log-likelihood of the alternative hypothesis (i.e. the target genome is present) is the same as calculated from the original Q matrix. The log-likelihood of the null hypothesis should be smaller than that of the alternative hypothesis, because the target genome has been estimated to have a non-zero relative abundance. A p-value is calculated using the likelihood ratio test by assuming a chi-square distribution for the likelihood ratio test statistic.  Sigma can perform likelihood ratio tests for many target genomes in parallel on a cluster using MPI/OpenMP.

## How to run likelihood ratio test?

You can run the Sigma likelihood ratio tests with `-t` (multi-threads) and/or `-p` (multi-processes) options for multi-processing. Multi-threads (`-t`) option is used for solving IPOPT (`sigma-solve-model`) in parallel, and multi-processes (`-p`) is used for running all reported genomes by Sigma in parallel.

```
$ sigma-jackknife -t 8 -p 8 -w /home/user/test/HMP01
```

### sigma-jackknife

```
$ sigma-jackknife -h

[Usage]
  sigma-bootstrap [options] -c <config file path> -w <working directory>

[Inputs]
  1. config file path (default: sigma_config.cfg)
  2. working directory (default: current running directory)

[Options]
  -h/--help
  -v/--version
  -p/--multi-processes <int>  # number of multi-processes (default: 1)
  -t/--multi-threads <int>    # number of threads (default: 1)

[Outputs]
  sigma_out .stat_jackknife_relative_abundance_estimation.txt
  sigma_out .stat_jackknife_percentage_scaled_estimation.txt
```

We also provides MPI version for the likelihood ratio test.  Multi-threads (`-t`) option also can be used in the MPI mode. This example will launch 10 MPI tasks, each with 8 threads, total 800 cores required.

```
$ mpirun -np 10 sigma-jackknife-mpi -t 8 -w /home/user/test/HMP01
```

### sigma-jackknife-mpi

```
$ sigma-jackknife-mpi -h

[Usage]
  sigma-jackknife-mpi [options] -c <config file path> -w <working
directory>

[Inputs]
  1. config file path (default: sigma_config.cfg)
  2. working directory (default: current running directory)

[Options]
  -h/--help
  -v/--version
  -t/--multi-threads <int>    # number of threads (default: 1)

[Outputs]
  sigma_out .stat_jackknife_relative_abundance_estimation.txt
sigma_out .stat_jackknife_percentage_scaled_estimation.txt
```

## Likelihood ratio test results

Log scale likelihood ratio for each genome is reported. A *p*-value can be calculated using the likelihood ratio test by assuming a chi-square distribution for the likelihood ratio test statistic.

# Variant Calling

## Why does variant calling require in metagenomics analysis?

Sigma enables strain variant calling by assigning metagenomic reads to their most likely reference genomes.

When an outbreak spreads or a pathogen re-emerges, it is important to determine the sequence variations between the genomes of this pathogen in the field-collected samples and its reference genomes in the database. This allows accurate reconstruction of the transmission sources of the outbreak and the evolutionary history of the pathogen. Thus, biosurveillance further requires information on the single-nucleotide polymorphisms (SNPs) of a detected pathogen to distinguish different variant populations. Identification of variants is straightforward for isolate genome sequencing by mapping all reads to a reference genome. However, in metagenome sequencing, one cannot simply map all metagenomic reads to a reference genome, because some of the aligned reads may originate from different microorganisms, which could introduce false variations. Sigma solves this problem by assigning each read to its most likely originating genome based on the Sigma probabilistic model, which allows subsequent variant calling by variants calling software such as SAMTools.

## How to run variant calling?

Update Sigma configuration file to provide the target genome for variant calling.

```
#---------------------------------#
# Parameters for variants calling #
#---------------------------------#
[Variants_Calling]
Filtering_Genome_Name=Escherichia_coli_O104_H4_2011C_3493_uid176127
```

Then run as below:

```
$ sigma-target-reads [options] -c <config file path> -w <working directory>
```

The filtered reads output for the target genome should be reported in the genome directory with name as below:

```
genome_basename.filtered.bam
```

Then follow variant calling procedures of SAMtools (http://samtools.sourceforge.net/mpileup.shtml). We also support a script to filter out ambiguous variants.

```
vcf_filter.py:

  [Usage]
    vcf_filter.py [options] -i <input vcf file> -o <output vcf file>

  [Options]
    -q <int or float> : filter out below quality (phred scaled score)
                        Ex) -q 20 is phred-20 (1% error) quality
    -f <float>        : filter out below allele frequency (0.0-1.0)
                        Ex) -f 0.5 filter out a case that has 4 alleles out
of 10 reads depth
    -m                : report homozygous (FQ value is negative from
samtools)
    -h/--help
    -v/--version
```