SigProfiler: Framework for Deciphering Mutational Signatures from Mutational Catalogues of Cancer Genomes
Version: 1.9
February 19, 2018

INTRODUCTION
The purpose of this document is to provide a brief and essential guide for using SigProfiler (formerly known as the Wellcome Trust Sanger Institute [WTSI]'s framework) for deciphering signatures of mutational processes from catalogues of cancer genomes. Detailed explanation of SigProfiler's theoretical model and computational framework is available in our manuscript entitled "Deciphering signatures of mutational processes operative in human cancer" by Alexandrov *et al.*, Cell Reports, Volume 3, Issue 1, 246-259:

https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3588146/

Further, three examples and two input files are provided to better illustrate how to apply SigProfiler to mutational catalogues of cancer genomes. Please note that SigProfiler allows identifying mutational signatures in both a direct manner and a hierarchical manner (see examples later in the document). The approach underlying the direct manner identification of mutational signatures can be found in the manuscript "Deciphering signatures of mutational processes operative in human cancer", while the approach underlying the hierarchical manner is briefly described in the manuscript "Landscape of somatic mutations in 560 breast cancer whole-genome sequences":

https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4910866/

In addition to SigProfiler, a tool for assigning mutational signatures in single samples is also provided. The tool is known as SigProfilerSingleSample and its source code as well as an example for running it can be found in the *tools* folder. The algorithm underlying SigProfilerSingleSample is briefly described in our manuscript entitled "Clock-like mutational processes in human somatic cells" by Alexandrov *et al.*, Nature Genetics, Volume 47, Issue 12, 1402-1407:

https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4783858/

PREREQUISITES
SigProfiler and all additional tools are written in MATLAB and require the following packages with the specified (or newer) versions:

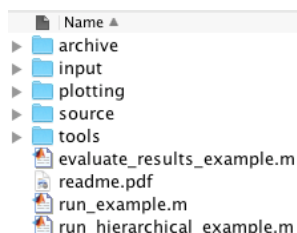|	
MATLAB	9.3.0.713579 (R2017b)
Parallel Computing Toolbox	Version 6.11 (R2017b)
Global Optimization Toolbox	Version 3.4.3 (R2017b)
Optimization Toolbox	Version 8.0 (R2017b)
Bioinformatics Toolbox	Version 4.9 (R2017b)
Statistics and Machine Learning Toolbox	Version 11.2 (R2017b)

Please note that MATLAB and the parallel computing toolbox are essential for running SigProfiler's core functionality. In addition to the parallel computing toolbox, the global optimization toolbox and the optimization toolbox are required for running SigProfilerSingleSample. The other two toolboxes are desirable but the code for deciphering mutational signatures and the code for assigning mutational signatures to an individual sample could be executed without them as other freely available packages have been leveraged in the appropriate places.

Accurately deciphering signatures of mutational processes is computationally intensive. SigProfiler is usually executed on a computational cluster (or a computational farm) with at least 100 nodes. Further, the three provided examples make the assumption that the default parallel cluster has already been preconfigured (please refer to MATLAB's documentation for configuring a default cluster). The code will make use of all available workers for the default cluster.

By default, SigProfiler uses the nonnegative matrix factorization (NMF) solver from (Brunet et al., PNAS, 2004, 12, 4164-4169), which is based on the multiplicative update algorithm (Lee and Seung, 1999, Nature 401, 788-791). However, if the Statistics and Machine Learning Toolbox is available, the provided NMF solver (i.e., nnmf) could be used instead and it will generally produce faster results. Additional freely available NMF solvers based on the multiplicative update algorithm as well as other algorithms are also provided. In principle, all solvers (with the appropriate options) converge to almost identical solutions and the main difference between the algorithms is the required, for code execution, CPU time and memory.

FOLDER STRUCTURE

When first downloaded, SigProfiler contains five folders, three example files, and this readme file. The source folder includes all code related to deciphering signatures of mutational processes including several nonnegative matrix factorization solvers. The plotting folder contains all source code related to plotting mutational signatures with and without strand bias as well as a plot that could be used for identifying the number of operative mutational signatures. The input folder contains MATLAB (i.e., *.mat) files for the given examples each containing a set of mutational catalogues of cancer genomes (described below). The archive folder contains previous versions of SigProfiler. The tools folder contains any additional tools for examining mutational signatures, for example, a tool for examining individual samples: SigProfilerSingleSample. After execution of the *run_example.m*, the code will also create an output folder structure (shown on the left) and a temp folder. The output folder structure generates a folder for the analyzed dataset and *full*, *skinny*, and *summary* subfolders. Each of the subfolders contains MATLAB (i.e., *.mat) files the results of executing SigProfiler. The results files in the

full folder contain data for all iterations. These files usually require a lot of storage (i.e., 100+ gigabytes) when examining a large dataset. The results files in the *skinny* folder contain data only for the average mutational signatures and the mutations they generate in individual samples. The result file in the *summary* folder provides summary information (i.e., average reconstruction, average stability, *etc.*) across the examined dataset. The *temp* folder is used to store sub-matrix input files that are subsequently leveraged by the hierarchical example. Please also note that running a hierarchical analysis will generate sub-sub-folders for each layer of the analysis.

INPUT FILE FORMAT

Name ▲	Value
cancerType	'WTSI BRCA Genomes'
originalGenomes	*96x21 double*
sampleNames	*21x1 cell*
subtypes	*96x1 cell*
types	*96x1 cell*

An input file is a MATLAB (i.e., *.mat) file that contains a set of mutational catalogues and metadata information about the cancer type, and the mutational types, subtypes, etc. for which these mutational catalogues have been defined. For example, the provided **21_WTSI_BRCA_whole_genome_substitutions.mat** is shown in the Figure of this section. The file contains the following fields:

- **cancerType** – string describing the type of samples in the file.
- **sampleNames** – a list of strings in which each element corresponds to the name of the analyzed sample.
- **types** – a list of strings in which each element corresponds to the name of the mutational types for which the catalogues have been defined.
- **subtypes** – a list of strings in which each element corresponds to the name of the mutational subtype for which the catalogues have been defined. Note that additional fields could be added if more classes of mutational types are being examined (e.g., strand bias).
- **originalGenomes** – an array containing mutational catalogues of cancer genomes with size <samples> by <mutational types> in which each element corresponds to the number of mutations per sample per mutational type and its subtype.

Please note that an input file could contain more fields but the fields listed above are required for SigProfiler to examine the provided mutational catalogues.

DESCRIPTION OF PROVIDED EXAMPLES

The provided examples perform 10 iterations per available core. Please note that there is an expectation of at least 1,000 iterations (*i.e.*, 100 available cores) and this number should be adjusted accordingly to the available nodes, otherwise it is possible that the identified mutational signatures are not accurate. Please note that only one of the examples generates plots, while the remaining two are used to generate these results in a MATLAB format (*.mat files).

run_example.m: This example illustrates deciphering mutational signatures in a direct manner from both: (i) a set of mutational catalogues derived from 21 breast cancer genomes; and (ii) a set of mutational catalogues derived from 100 breast cancer exomes with a third mutational subtype (*i.e.*, strand bias).

run_hierarchical_example.m: This example illustrates deciphering mutational signatures in a hierarchical manner. The example requires executing *run_example.m* and identifying the optimal number of mutational signatures in the dataset. The parameters are set for identifying a second layer of mutational signatures in 21 breast cancer genomes. Provided that there are enough samples, the example can be modified for analyzing deeper hierarchical layers.

evaluate_results_example.m: This example illustrates identifying the number of mutational processes operative in an examined dataset. The default option show results for the set of 21 breast cancer genomes, while simply changing the file names will allow examining the results for the 100 breast cancer exomes. This example requires running *run_example.m*.

CONTACT INFORMATION
Please address any queries or bug reports to Ludmil B. Alexandrov at l2alexandrov@ucsd.edu.