tugMedi 1.0.31

–*tu*mor-*g*enome *Medi*cal– simulator

2025/06/23

Contents

1 Quick start guide

1.1 Run

1. Load the library in R.

   `library(tugMedi)`

2. Copy the test data of this package under the current directory.

   `copy_test_data("./tugMedi.1.0/")`

   You will get Test1/, Test2, ... under the current directory.

3. Move into Test1/ and run a test.

   `setwd("Test1/")`

   `source("test1.R")`

   You will get a renewed Output/ under this test directory.

- Alternatively, you can `q()` after `copy_test_data()`.

  Then, after moving into the test directory, on your OS,

  `Rscript --vanilla --slave test1.R`

* Note that this is just an artificial example for explanatory purposes. **The inputs and outputs do not have any biological meanings**.

1.2 test1.R

Please see test1.R, and edit it when you understand.

1.2.1 Simulation

```
start_simulation( input_files = 'Input/DATA/FILES.txt',
                  saveTo      = 'Output/Results.sim.01.RDS',
                  seed        = NA
                 )
```

This command starts a simulation. Input data are defined through option `input_files`. Please see the section of Input data for the details of input data. RDS data are saved to `saveTo` for the second simulation, where usually drug intervention is performed. Drug intervention is explained below. You can change the random seed through `seed`.

1.2.2 Generators

These commands randomly generate parameter values in specific manners for Approximate Bayesian computation later. You can comment out these commands and manually set parameter values, too.

```
write_prm(       input = './Input/ForGenerators/parameters.parameters.txt')
write_CF(        input = './Input/ForGenerators/CF.parameters.txt')
write_weights(   input = './Input/ForGenerators/hallmark_weights.parameters.txt' )
write_cloneinit( input  = './Input/ForGenerators/cloneinit.parameters.txt' )
```

```
write_Rint(      input = './Input/ForGenerators/Rint.parameters.txt' )
write_EF.Rint(   input = './Input/ForGenerators/EF.Rint.parameters.txt' )
# Only this depends on write_prm() result
write_EF.Rother( input = './Input/ForGenerators/EF.Rother.parameters.txt' )
```

The inputs of all these functions are independent except that of `write_EF.Rother()`, which depends on the output of `write_prm()`. So, please put `write_EF.Rother()` after `write_prm()`.

See the section of Generators for the details of these functions.

1.2.3 Post processes

These commands take in the outputs of the simulation and perform post processes such as the calculation of VAF (variant allele frequency).

```
write_VAF(   input = './Input/ForPosts/VAF.parameters.txt' )
write_TMB(   input = './Input/ForPosts/TMB.parameters.txt' )
write_evals( input = './Input/ForPosts/evals.parameters.txt' )
write_realTime_clone( input = './Input/ForPosts/realTime.parameters.txt' )
```

`write_TMB()` and `write_evals()` use the output of `write_VAF()`.

See the section of Post processes for the details of these functions.

1.3 Drug intervention in test1.R

```
start_simulation( first       = F,
                  input_files = 'Input/DATA/FILES.drgInt.txt',
                  loadFrom    = 'Output/Results.sim.01.RDS',
                  saveTo      = 'Output/Results.sim.02.RDS',
                  seed        = NA,
                  drug_int_param = drug_int_param
                 )
```

One can simulate how moleculary targeted drugs take effect on cancer cells with this command. Because drug intervention simulation usually takes advantage of the results of the first simulation, option `first` is `F`. The results of the first simulation are loaded with `loadFrom`, and the results of drug intervention simulation are saved to `saveTo`. As with the first simulation, input data are defined through `input_files` and the random seed can be set with `seed`.

To continue from the first simulation, it is necessary to update the first RDS file before the second simulation by the command:

```
update_RDS_from_output( in.file  = 'Input/DATA/FILES.txt',
                        rds.file = 'Output/Results.sim.01.RDS' )
```

By this, simulation parameters can be changed through the input data at the second simulation. Typically, you may extend `control.censor_cell_number` (the censor [program stop] of the maximum cell number) and `control.censor_time_step` (maximum simulation time step) in the input data for drug intervention. (In this example, the second simulation command reads `input_files = 'Input/DATA/FILES.drgInt.txt'` to use parameters.drgInt.txt, an input file different from parameters.txt, used in the first simulation.) The second simulation starts from the next to the last time step of the first simulation. The last state of clones can be changed through editing an output file named cloneout.txt. See Chapter 11 for further details.

`drug_int_param` in the second simulation command is set as follows:

```
drug_int_param <- list( 'kill_prob' = 0.5,
                        'block_prob'= 0.0,
                        'gene' = c('KRAS', 'APC') ) # Any of them is a target
```

Cells with any of the genes indicated by `'gene'` in the dysfunctional state are the targets of "kill" or "block" operations. The kill operation kills cells at the rate of `'kill_prob'` per simulation time step. The block operation blocks the dysfunctional state of the genes, i.e., revert the dysfunctional state ('1', indicating the dysfunction) to the normal state ('0', the normal state) of the genes in the program. This blocking is performed on cells randomly selected at the rate of `'block_prob'` per simulation time step.

1.4 Outputs

1.4.1 Output/

You will get a new Output/, which has:

- cloneout.txt: Simulation result of cancer-cell evolution.
- cloneout.realTime.txt: Estimated real time is added to cloneout.txt.
- Results.sim.01.RDS: RDS data for the first simulation.
- Results.sim.02.RDS: RDS data for the second (drug intervention) simulation.
- Mutations/: Data on point mutations and CNAs are stored.
- VAF/: VAF data are stored.
- Info/: Other information is stored.

(The older Output/ is automatically renamed with a time stamp.)

See the section of Output data for the details of these outputs.

1.4.2 Cancer-cell evolution

In short, you can first see cloneout.txt. Simulation time step represented in column Time increases with the rows. Column N_primary, meaning the number of cells in the primary tumor, and N_metastatic, the number of cells in the metastatic tumor, may increase with the time or become 0 at some time point, the latter of which means extinction. Column 'ID', i.e., clone ID, shows various clones over the time, which means intra-tumor heterogeneity (ITH).

Columns genesMalfunc and genesWmutsNoExp show genes in the malfunctional state and genes with mutations not causing the malfunction, respectively. The latter genes are mostly passenger mutations, but the situation may be complex if the genes are defined as recessive. Under the recessive, such mutations can be driver mutations because recessive genes need to have driver mutations on both parental chromosomes to cause the malfunction.

Columns PointMut_ID and CNA_ID show the IDs of point mutations and CNAs, respectively. They are driver or passenger mutations. The detailed information is stored under Mutations/.

Different clones have different mutations (point mutations and CNAs). But, if clones are metastatic clones as shown in column Type, clones with the same mutations may appear because primary clones that keep the same mutations seed metastatic clones at different time steps, as distinguishable in column Birth_time.

Different clones have different trial probabilities (columns d to im) and different hallmark variable values (Hd to Hb).

1.4.3 Drug intervention simulation

In cloneout.txt, you may see a decrease of N_primary or N_metastatic at last time steps. This results from the drug intervention simulation. Only clones with specified target genes decrease as shown in column N_cells, which represents the number of cells in each clone.

2 Quick start guide for parallel computing

2.1 Run

For parallel computing, let us assume that GNU parallel is available (for `qsub`, see the chapter of Approximate Bayesian Computation).

After `copy_test_data()` explained at the previous chapter, move into the test directory of Test2/ on your OS. Then,

1. Change the newline code of the files according to your OS.
   - Newline code for Linux or Win
2. Grant execute permission to the following script.
   `chmod ugo+x abc.run.sh`
3. Run a test.
   `bash mcr.cmdsAll.bash`

You will see a small parallel computing of 10 simulations as an example. Then, ABC is automatically processed and again 10 simulations run in parallel with parameters selected by ABC.

Outputs are explained below.

* Note that this is just an artificial example for explanatory purposes. **The inputs and outputs do not have any biological meanings**.

2.1.1 What mcr.cmdsAll.bash is

Just a series of simple commands as follows. “mcr” stands for macro.

```
bash mcr.input.bash
bash mcr.remove.bash


Rscript abc.a.01_make_list.R abc.01_config.R
bash abc.b_run_parallel.sh
Rscript abc.c_make_data.R abc.01_config.R
Rscript abc.d_exec_calc.R abc.01_config.R


Rscript --vanilla --slave ./test2.3_eval.R &


Rscript abc.a.02_make_list.R abc.02_config.R
bash abc.b_run_parallel.sh
Rscript abc.c_make_data.R abc.02_config.R
Rscript abc.d_exec_calc.R abc.02_config.R


Rscript --vanilla --slave ./test2.4_eval.R
```

Of course, you can execute each command one-by-one, which will be useful for debugging. We will explain these commands below.

2.1.2 The first two macros

mcr.input.bash

Just prepare input data:

```
rm -r Input
cp -r Input.TCGA_7903 Input
```

mcr.remove.bash

Just remove previous settings:

```
rm -r run_list.txt
rm -r log/
rm -r work/

rm -r ABC1/
rm -r ABC2/
rm -r old/
```

2.1.3 ABC part

The following commands correspond to the ABC part. The ABC settings are controlled with `abc.*_config.R`. See the section of Approximate Bayesian Computation (ABC) for the details.

```
Rscript abc.a.01_make_list.R abc.01_config.R
bash abc.b_run_parallel.sh
Rscript abc.c_make_data.R abc.01_config.R
Rscript abc.d_exec_calc.R abc.01_config.R

Rscript abc.a.02_make_list.R abc.02_config.R
bash abc.b_run_parallel.sh
Rscript abc.c_make_data.R abc.02_config.R
Rscript abc.d_exec_calc.R abc.02_config.R
```

2.1.4 Evaluation part

In this part, it is evaluated how close simulated VAFs are to observed VAFs across simulation replications. See Evaluation scores at the section of Post processes for the details.

```
Rscript --vanilla --slave ./test2.3_eval.R &

Rscript --vanilla --slave ./test2.4_eval.R
```

2.2 Outputs

You will see work/, and ABC1/ and ABC2/ as outputs.

2.2.1 work/

- You will see `first/` and `TCGA-55-7903-01A-11D-2167-08/`.
  - Under each directory, you will see `000001/`, `000002/`, . . . , each of which is a simulation replication.
  - Each simulation replication has the same content as would be performed in a single execution.
- `first/` has results before ABC.
- `TCGA-55-7903-01A-11D-2167-08/` has results after ABC, where parameter values selected by ABC are used.

2.2.2 ABC1/ and ABC2/

- ABC1/ and ABC2/ are the results of ABC before and after parameter selection, respectively.
- Under each directory, you will see `TCGA-55-7903-01A-11D-2167-08/`. Under this, you will see files.
  - Files of `evals.*.txt` are related to the evaluation (fit between observed and simulation VAFs).
  - The other files are related to ABC.
  - See relevant sections for the details.
- Here, three types of files will be useful.
  - *evals.stats.{pre,post}.txt*
    - This records scores of the evaluation.
  - *TOLSel_rejection_tumor_content=*_tol=*.txt*
    - This stores the IDs (directory names) of the simulation replications that are selected by ABC.
  - *ABC_dist_tumor_content=*.txt*
    - This stores the ABC distance for each replication.

2.3 Selected parameter values

- In each simulation replication, generators generate random values. Then, simulations are performed. The ABC process selects simulation replications with good fit to observed data and stores the directory names of the selected replications in `TOLSel_rejection_tumor_content=*_tol=*.txt`.
  - From this list, simulation replications are re-sampled into `work/TCGA-55-7903-01A-11D-2167-08/`. Then, without using generators this time, simulations are performed.
- In other words, selected parameter values can be tracked to simulation replications in `work/first/` with the directory names listed in `ABC1/`'s `TOLSel_rejection_tumor_content=*_tol=*.txt`.
  - (Meanwhile, `ABC2/`'s `TOLSel_rejection_tumor_content=*_tol=*.txt`
    correspond to
    `work/TCGA-55-7903-01A-11D-2167-08/`)
- In our experience, several rounds of ABC were necessary to obtain good fit we satisfied with.

3 Input data

3.1 Input/

3.1.1 CF.txt

This file defines compaction factors.

Columns

- Hallmark: Hallmark name. Either of ‘apoptosis’, ‘growth’, ‘immortalization’, or ‘invasion’.
  - The compaction factor of angiogenesis is mathematically equivalent to Fb in parameter.txt. Thus, it is unnecessary to specify it here.
- CompactionFactor: Value of the compaction factor, ranging from 0 to 1.

3.1.2 cloneinit.txt

This file defines the initial state of a clone or clones.

Columns

- cloneID: ID to distinguish a clone from another
- Ncells: Number of cells in the initial clone
- Type: Mutation type. Either of ‘pom’, ‘del’, or ‘dup’.
  - pom is point mutation, del is deletion of CNA, and dup is duplication of CNA.
  - Poms correspond to SNVs and indels in reality.
    * Note that short deletions and insertions in reality are not dels or dups, but poms.
- Gene: Gene name where the mutation resides.
- Chr: Chromosome number where the mutation resides.
- Chr_stt: Chromosomal starting position of the mutation.
- Chr_end: Chromosomal ending position of the mutation. NA for pom.
- Pchr: Parental chromosome number 1 or 2 (either paternal or maternal) where the mutation resides.
  - This information is used to decide if the mutation is expressed according to the recessive or dominant mode of the gene.
- DrvPss: ‘drv’ or ‘pss’, which represents the mutation is driver or passenger, respectively. Usually, ‘drv’ is used.
  * If an initial clone does not have any mutations, set NA for the columns from Type to DrvPss. But this is not a usual use case.

3.1.3 hallmark_weights.txt

This file defines hallmark weights for each gene.

Columns

- Hallmark: Hallmark name. Either of 'apoptosis', 'growth', 'immortalization', or 'invasion', or 'angiogenesis'.
- Gene: Gene name.
- Weight: Weight values.
  - The values will be normalized such that the summation across genes for a hallmark is 1.

3.1.4 parameters.txt

This file defines parameters.

Parameters

- m0: Mutation rate of a pom per bp per cell division.
- uo: Rate at which a gene defined as oncogene is made dysfunctional by the occurrence of a pom.
- us: Same as above for a suppressor.
- dN: Cell division rate of a normal cell per (simulation) time step.
- kN: Rate of constant cell death per time step. NA can be used, where a value that derives the equilibrium state between cell division and cell death is calculated and given.
- sN: The gain parameter for a sigmoid function that determines the rate of cell death by apoptosis.
- K_N: Carrying capacity in logistic growth.
- Fb: Expansion order (10^{Fb}) to K_N due to angiogenesis.
- ctmax: The residual count of cell divisions to the maximum count of the replication (Hayflick) limit

- m_dup: Mutation rate of a dup per bp per cell division.
- m_del: Same as above for a del.
- ave_len_dup: Average chromosomal length of a dup.
  - The individual lengths of dups are randomly selected by an exponential distribution with this average.
- ave_len_del: Same as above for a del.
- uo_dup: Rate at which a gene defined as oncogene is made dysfunctional by the occurrence of a dup.
- us_dup: Same as above for suppressor. Usually, it may be 0.
- uo_del: Same as above for oncogene by the occurrence of a del. Usually, it may be 0.

- us_del: Same as above for suppressor.
- Zim: Probability of the success of invasion/metastasis in the proportional metastatic model.

- tumbler_for_metastasis_trial: Switch to turn on/off the invasion/metastasis trial.
- tumbler_for_apoptosis_trial: Same as above for the apoptosis trial.
- tumbler_for_immortalization_trial: Same as above for the immortalization trial.
- tumbler_for_angiogenesis_trial: Same as above for the angiogenesis trial.
- control.censor_cell_number: Maximun cell number at which a program run stops.
- control.censor_time_step: Same as above for maximum simulation time step.
- control.censor_real_time: Same as above for maximum real time.
- control.monitor: Switch to turn on/off monitoring a program run.

3.1.5 Rint.txt

This file defines regions of interest, where driver mutations occur and passenger mutations can occur.

Columns

- Gene: Gene name, with which positional information in chromosomes is extracted from Input/DATA/CCDS.current.my.txt.
- CDS_ID: Same as above for coding sequence ID.
- Type: ‘s’, ‘o’, or ‘dn’, which represents the gene is suppressor, oncogene, or dominant negative, respectively.

3.1.6 Rother.txt

This file defines other regions than regions of interest. Only passenger mutations occur.

Columns

- Gene: Same as in Rint.txt.
- CDS_ID: Same as in Rint.txt.

3.1.7 EF.Rint.txt

This file defines mutations to occur in Rint by the EF algorithm.

Columns

- Time_step: Simulation time step at which the mutation is inserted into a clone.
- Mutation: ID to distinguish a mutation from another.

- Condition: The mutation is inserted into a clone with mutation IDs indicated in this column.
  - Multiple mutations are represented by the comma-separated such as M1,M2.
  - NA is used for no condition (any clone).
- Type: Mutation type. Either of ‘pom’, ‘del’, or ‘dup’.
- Gene: Gene name where the mutation occurs.
  - Multiple gene names are possible in the comma-separated form (e.g., ‘gene1,gene2’) for a CNA.
- Chr: Chromosome number where the mutation occurs.
- Chr_stt: Chromosomal starting position of the mutation.
- Chr_end: Chromosomal ending position of the mutation. NA for pom.
- Pchr: Parental chromosome number 1 or 2 (either paternal or maternal) where the mutation occurs.
  - This information is used to decide if the mutation is expressed according to the recessive or dominant mode of the gene
- DrvPss: ‘drv’ or ‘pss’, which represents the mutation is driver or passenger, respectively. Usually, ‘drv’ is used.

3.1.8 EF.Rother.txt

This file defines mutations to occur in Rother by the EF algorithm.

Columns

- Waiting_division: Cell division count at which the mutation is inserted into a clone. Namely, “waiting division” analogy to waiting time in the queueing theory.
- Mutation: Same as in EF.Rint.txt.
- Type: Same as in EF.Rint.txt.
- Gene: Same as in EF.Rint.txt.

3.2 Input/DATA

Files more fixed are stored under this directory.

3.2.1 CCDS.current.my.txt

This file provides the chromosomal positions of exons and introns, edited from CCDS.current.txt.

CCDS.current.txt is consensus CDS database from CCDS database at the National Center for Biotechnology Information, storing data on the positions of consensus CDSs along chromosomes.

This file, CCDS.current.my.txt, is edited from the original CCDS file such that genes in each chromosome are connected for the convenience of computation.

3.2.2 FILES.txt

This file defines the paths of input and output files.

3.2.3 FIXED_PARAMETERS.txt

This file defines parameters more fixed than parameters.txt and ones under test.

Parameters

- compaction_factor: Switch to turn on/off whether the compaction factor is multiplied to hallmark weights.
- kappa: Time-scaling parameter to convert trial probabilities into Poisson
  lambdas.
- tumbler_for_event_enforcement: Switch to turn on/off the EF algorithm.
- metastatic_model: Metastatic model.
  - Now ‘proportional_metastatic’ only (‘threshold_metastatic’ is under test).
- growth: Growth model for clones in primary tumor.
  - Either of ‘logistic’ or ‘exponential’.
- trial$spec: Probability distribution with which the trials are conducted.
  - Now ‘pois’ (Poisson distribution) only.
- meta.addNcells$spec: Probability distribution with which the number of cells that metastasize together as a group is sampled.
  - Now ‘pois’ (Poisson distribution) only.
- meta.addNcells$lambda: The lambda parameter for the distribution above.
- meta.loc$sig: Under test.
- meta.loc$sig[1,]: Under test.
- meta.loc$sig[2,]: Under test.
- meta.loc$sig[3,]: Under test.

3.2.4 GENE_TYPE.txt

This file defines the relationship between gene types and dysfunction rate types, and the dominant/recessive modes.

Columns

- type: Gene type. Either of ‘o’, ‘s’, or ‘dn’, which represent oncogene, suppressor, or dominant negative, respectively.

- u.pom: Dysfunction rate type assigned to the gene type for a pom.
- u.del: Same as above for a del.
- u.dup: Same as above for a dup.
- mode.pom: Genetic mode assigned to the gene type for a pom. Either of 'D', 'R', or 'N', which represent dominant, recessive, or not expressed (no effect on phenotype), respectively.
- mode.del: Same as above for a del.
- mode.dup: Same as above for a dup.

3.3 Samples/

This folder stores observed data of VAF (variant allele frequency).

3.3.1 samples.Rint.txt

- Mutation data of which the columns are selected from the MAF files of TCGA data, except that the t_vaf column is added with t_alt_count / t_ref_count, which are the original MAF columns.
- The columns of gene name ('Hugo_Symbol'), sample ID ('Tumor_Sample_Barcode'), and VAF ('t_vaf') are necessary.
- The other columns are only for readability.
- The order of columns can be changed.

- These mutations and VAFs are compared with simulated mutations and VAFs in Rother.

3.3.2 samples.Rother.txt

- Same as in samples.Rint.txt for Rother.

3.4 ForGenerators/

See the section of Generators.

3.5 ForPosts/

See the section of Post processes.

4 Output data

4.1 Output/

4.1.1 cloneout.txt

This file contains the simulation results of clones at each time step.

Columns

- Time: Simulation time step.
- AvgOrIndx: ‘avg’ or index number. ‘avg’ is for a line with averaged values across different lines at the same time step. An index number shows the index of a clone within the current time step.
- ID: ID of a clone.
- Parent_ID: Parental clone ID of the clone.
- Birth_time: Time step of the clone’s birth.
- Type: ‘’primary, ’metastatic’, or ‘normal’, which represent the clone is of primary tumor, metastatic tumor, or normal tissue, respectively.

- N_cells: The number of cells in this clone.
- N_primary: The number of primary tumor cells at this time step.
- N_metastatic: Same as above for metastatic tumor cells.
- N_normal_intact: Same as above for normal intact cells (normal cells without mutations).
- N_normal_speckled: Same as above for normal speckled cells (normal cells with passenger mutations).

- ct: The counter of cell divisions for the clone. It equals the mean across all cells in the clone.
- d: Cell division rate.
- k: Constant cell death rate.
- a: The rate of cell death due to apoptosis.
- i: Probability of the effect of immortalization, namely that cells avoid the replication (Hayflick) limit to divide.
- im: Probability of trying invasion/metastasis.
- K: Carrying capacity for the primary tumor in the logistic growth.
- Nmax: Theoretically maximal number of primary tumor cells, namely inverse of K.
- Hd: Value of the hallmark variable “Growth / Anti-growth”.
- Ha: Same as above for “Apoptosis”.
- Hi: Same as above for “Immortalization”.
- Him: Same as above for “Invasion / Metastasis”.
- Hb: Same as above for “Angiogenesis”.

- mutden: Density of passenger poms per bp.
- total_divIdx: Total number of cell divisions.
- genesDysfunc: Dysfunctional genes.
- genesWmutsNoExp: Genes with mutations not expressed, i.e., mutations not causing the dysfunctional state. Such mutations are passenger mutations, or driver mutations on either parental chromosome in the recessive genetic mode.
- PointMut_ID: pom ID the clone has. This ID corresponds to pom ID in pointMutations_B.txt and pointMutations_A.txt.
- CNA_ID: Same as above for CNA ID and CNAs.txt.

4.2 Output/Mutations/

4.2.1 pointMutations_B.txt and pointMutations_A.txt

The files contain data on poms (point mutations) of the variant allele B (_B.txt) and the original allele A (_A.txt), respectively.

Columns

- PointMut_ID: ID of a pom, which corresponds to the allele B or A.
- Parental_1or2: Either of the two parental chromosomes where the pom resides.
- Chr: Name of a chromosome where the pom resides.
- Ref_pos: Reference position of the pom. The reference position is on the coordinate system of the human reference genome.
- Phys_pos: Physical position of the pom. The physical length of a (parental) chromosome is extended or shrunk by CNA duplications or deletions, respectively.
- Delta: Difference between the reference and physical positions.
- Copy_number: Copy number of the allele B or A of the pom.
- Gene_name: Name of a gene where the pom resides.
- MalfunctionedByPointMut: Logical indicator of whether or not the gene is made dysfunctional by the pom.
- Ovlp_CNA_ID: CNA ID in CNAs.txt that overlaps with the pom on the chromosome.

4.2.2 CNAs.txt

This file contains data on CNAs.

Columns

- CNA_ID: ID of a CNA.
- Parental_1or2: Same as in pointMutations_B.txt and pointMutations_A.txt for CNA.
- dupOrdel: Indicator of dup (duplication) or del (deletion) for CNA.
- Chr: Same as in pointMutations_B.txt and pointMutations_A.txt for CNA.
- Ref_start: Reference position of the CNA start.
- Ref_end: Reference position of the CNA end.
- Gene_names: Name(s) of a gene(s) where the CNA resides.
- MalfunctionedByCNA: Same as in pointMutations_B.txt for CNA.

4.3 Output/Info/

This directory contains additional information.

4.3.1 used.*.txt

Input data that once went through the program, which are useful for confirmation.

Notable columns in used.hallmark_weights.txt

- weight_woCF: Weight parameter values without multiplied by compaction factors.
- weight_used: Weight parameter values used.

4.3.2 monitor_*.txt

Monitoring files for debugging and development.

4.3.3 log.txt

Log file.

4.3.4 Other files

See the section of Post processes.

4.4 Output/VAF/

See the section of Post processes.

5 Generators

Generators randomly generate parameter values used as input data for simulation.

5.1 Generation of CF.txt

5.1.1 Input

5.1.1.1 CF.skeleton.txt

- The output file is generated based on this skeleton.
- Parameter values in this skeleton are overwritten with random values generated in the way specified in CF.parameters.txt.

5.1.1.2 CF.parameters.txt

Parameters

- skeleton: File path to the skeleton.
- out: File path to the output.

rtrunc() based generation

- dist$*hallmark_name*$spec: spec of rtrunc() in the truncdist library of R.
  - For example, beta can be used.
  - If NULL, the non-informative beta distribution is used.
- dist$*hallmark_name*$a: Same as above for a
- dist$*hallmark_name*$b: Same as above for b
- dist$*hallmark_name*$*option:* Same as above for *option*
  * The italic text means a variable

sample() based generation

- dist$*hallmark_name*$spec: sample
- dist$*hallmark_name*$x: x of sample()
- dist$*hallmark_name*$prob: prob of sample()
  * replace = T

5.1.2 Run

```
write_CF( input = './Input/ForGenerators/CF.parameters.txt' )
```

This command inputs CF.parameters.txt and outputs CF.txt.

5.2 Generation of hallmark_weights.txt

5.2.1 Input

5.2.1.1 hallmark_weights.skeleton.txt

- The output file is generarated based on this skeleton.
- Parameter values in this skeleton are overwritten with random values generated in the way specified in hallmark_weights.parameters.txt.

5.2.1.2 hallmark_weights.parameters.txt

Parameters

- skeleton: File path to the skeleton.
- out: File path to the output.

Dirichlet distribution based generation

- dist$*hallmark_name*$spec: Either of 'dirichlet', 'dirichlet.mode', or 'dirichlet.mean'.
  - 'dirichlet': rdirichlet() in the dirmult library of R is used to generate random numbers based on the alpha parameters.
  - 'dirichlet.mode': The mode of the Dirichlet distribution is calculated from the alpha parameters, and is used deterministically.
  - 'dirichlet.mean': Same as above for the mean.
- dist$*hallmark_name*$alpha.column: Indicate which column in the skeleton is used for the alpha parameters of the Dirichlet distribution
  - If NULL, the non-informative Dirichlet distribution is used.

5.2.2 Run

```
write_weights( input = './Input/ForGenerators/hallmark_weights.parameters.txt' )
```

This command inputs hallmark_weights.parameters.txt and outputs hallmark_weights.txt.

5.3 Generation of parameters.txt

5.3.1 Input

5.3.1.1 parameters.skeleton.txt

- The output file is generarated based on this skeleton.
- Parameter values in this skeleton are overwritten with random values generated in the way specified in parameters.parameters.txt.

5.3.1.2 parameters.parameters.txt

Parameters

- skeleton: File path to the skeleton.
- out: File path to the output.

rtrunc() based generation

- dist$*parameter_name*$spec: spec of rtrunc() in the truncdist library of R.
- dist$*parameter_name*$a: Same as above for a
- dist$*parameter_name*$b: Same as above for b
- dist$*parameter_name*$*option:* Same as above for *option*

sample() based generation

- dist$*parameter_name*$spec: sample
- dist$*parameter_name*$x: x of sample()
- dist$*parameter_name*$prob: prob of sample()
  * replace = T

5.3.2 Run

```
write_prm( input = './Input/ForGenerators/parameters.parameters.txt' )
```

This command inputs parameters.parameters.txt and outputs parameters.txt.

5.4 Generation of EF.Rint.txt

** Now this only generates values in the 'Time_step' and 'Pchr' columns.*

5.4.1 Input

5.4.1.1 EF.Rint.skeleton.txt

- The output file is generarated based on this skeleton.
- Parameter values in this skeleton are overwritten with random values generated in the way specified in EF.Rint.parameters.txt.

5.4.1.2 EF.Rint.parameters.txt

Parameters

- skeleton: File path to the skeleton.
- out: File path to the output.

rtrunc() based generation

- dist$*column_name*$spec: spec of rtrunc() in the truncdist library of R.
  - Without prior knowledge, unif may be used for the Time_step column.
  - Then, norm may be used.

- dist$*column_name*$a: Same as above for a
- dist$*column_name*$b: Same as above for b
- dist$*column_name*$*option:* Same as above for *option*
  - Different values can be given by indicating mutation IDs in vector, such as c(M2=5, M3=20), where M2 and M3 are mutation IDs in the skeleton.

sample() based generation

- dist$*column_name*$spec: sample
- dist$*column_name*$x: x of sample()
- dist$*column_name*$prob: prob of sample()
  * replace = T

5.4.2 Run

```
write_EF.Rint( input = './Input/ForGenerators/EF.Rint.parameters.txt' )
```

This command inputs EF.Rint.parameters.txt and outputs EF.Rint.txt.

5.5 Generation of EF.Rother.txt

5.5.1 Input

5.5.1.1 EF.Rother.parameters.txt

Parameters

- input.prms: File path to parameters.txt, which is used as input data.
- input.Rother: File path to Rother.txt, which is used as input data.
- input.CCDSdatabase: File path to CCDS.current.my.txt, which is used as input data.
- out: File path to the output.
- CNA_presence: whether or not to generate CNAs (dels and dups)
- N_mut: Number of mutations to be generated

rtrunc() based generation

- dist$EEL$spec: spec of rtrunc() in the truncdist library of R.
  - exp, gamma, and weibull are usable.
    * The scale parameter of gamma and weibull is scaled to keep the mean the same as exp's.
- dist$EEL$*option:* Same as above for *option*

5.5.2 Run

```
write_EF.Rother( input = './Input/ForGenerators/EF.Rother.parameters.txt' )
```

This command inputs EF.Rother.parameters.txt and outputs EF.Rother.txt.

5.6 Generation of cloneinit.txt

5.6.1 Input

5.6.1.1 cloneinit.skeleton.txt

- The output file is genererated based on this skeleton.
- For the output file, passenger poms are added according to cloneinit.parameters.txt.

5.6.1.2 cloneinit.parameters.txt

Parameters

- skeleton: File path to the skeleton.
- out: File path to the output.
- input.Rother: File path to Rother.txt, which is used as input data.
- input.CCDSdatabase: File path to CCDS.current.my.txt, which is used as input data.

sample() based generation

The number of poms to add to *cloneID* in the skeleton is randomly generated. Added poms are passenger poms in genes that are randomly selected from Rother in proportion to the gene sizes. The positions are also randomly determined.

- pom[['*cloneID*']]$dist$spec: sample
- pom[['*cloneID*']]$dist$x: x of sample(). x must be a vector of integers to indicate the number.
- pom[['*cloneID*']]$dist$prob: prob of sample()
  * replace = T

5.6.2 Run

```
write_cloneinit( input = './Input/ForGenerators/cloneinit.parameters.txt' )
```

This command inputs cloneinit.parameters.txt and outputs cloneinit.txt.

5.7 Generation of Rint.txt

5.7.1 Input

5.7.1.1 Rint.skeleton.txt

- The output file is generarated based on this skeleton.
- Parameter values in this skeleton are overwritten with random values generated in the way specified in Rint.parameters.txt.

5.7.1.2 Rint.parameters.txt

Parameters

- skeleton: File path to the skeleton.
- out: File path to the output.

sample() based generation

The value of the Type column for *gene* in the skeleton is randomly replaced.

- dist$Type[['*gene*']]$spec: sample
- dist$Type[['*gene*']]$x: x of sample(). x must be a vector of gene types such as 's', 'o', and 'dn'.
- dist$Type[['*gene*']]$prob: prob of sample()
  * replace = T

5.7.2 Run

```
write_Rint( input = './Input/ForGenerators/Rint.parameters.txt' )
```

This command inputs Rint.parameters.txt and outputs Rint.txt.

6 Approximate Bayesian Computation (ABC)

You can perform parallel execution and subsequent ABC. See scripts/prmEst.ABC/abc.* in the gitHub repository for the scripts and examples.

6.1 Scripts

6.1.1 For parallele execution

- **abc.a.XX__make__list.R**: Make a job list
- **abc.b__run__{parallel, qsub}.sh**: Run the jobs in parallel with GNU parallel or qsub

6.1.2 For abc() of library abc

- **abc.c_make_data.R**: Make data for ABC from simulation results
- **abc.d_exec_calc.R**: Calculate ABC from the ABC data

6.2 Example

```
# First ABC round
# You need to have Input/ and R/ here.
# You need to chmod abc.run{,_qsub}.sh to be executable.
# You may need to change the newline code (for Linux's or Win's) of abc.*.
cp scripts/ABC/abc.*.{sh,R} .
cp -a inst/extdata/Test1/Input .
Rscript abc.a.01_make_list.R abc.01_config.R
bash abc.b_run_parallel.sh
Rscript abc.c_make_data.R abc.01_config.R
Rscript abc.d_exec_calc.R abc.01_config.R

# Second ABC round, using parameters selected by the first ABC
Rscript abc.a.02_make_list.R abc.02_config.R
bash abc.b_run_parallel.sh
Rscript abc.c_make_data.R abc.02_config.R
Rscript abc.d_exec_calc.R abc.02_config.R
```

6.3 abc.a.XX_make_list.R

6.3.1 Inputs

- `abc.XX_config.R`: Configuration file explained below in **ABC configuration file**.
- Internally uses a script specified by `run_script`, such as `abc.01_run.R`, in the configuration file.
  - This script is run in parallel. Change this script according to your purpose.
- Internally uses `abc.run{,_qsub}.sh`.

6.3.2 Outputs

- `run_list.txt`: Job list
- `work/*/`: Output directory for parallel execution. Can change this name in the configuration file.
  - Under this directory, replications are made as `000001/`, `000002/`, ...

6.4 abc.b_run_{parallel, qsub}.sh

6.4.1 Input

- Internally uses `run_list.txt`.

6.4.2 Outputs

- Simulation outputs are output under the replication directories under `work/*/`.
- `log/`: Log directory

6.4.3 Notes

- This script is simple. You may need to change option parameters written in this script (especially, _qsub.sh) for parallel computation.

6.5 abc.c_make_data.R and abc.d_exec_calc.R

6.5.1 Input

- `abc.XX_config.R`: Configuration file explained below in **ABC configuration file**.

6.5.2 Outputs

- Described below in **ABC output directory**.

6.6 ABC configuration file

# Environment setting

- base_dir: your working directory.

6.6.1 For parallel execution

- replicates: number of replications to simulate
- work_dir: output directory for parallel execution
- run_script: this script will be run in parallel with Input/ and R/.

6.6.2 For ABC

# Output

- ABC_dir: output directory for ABC results, as described below in **ABC output directory**.

# Simulation VAFs

- VAF_files_glob: glob pattern to get simulation VAFs

# Observation VAFs

- rint_file: path to samples.Rint.txt
- rother_file: path to samples.Rother.txt

- rother_regex: regular expression to select other regions/genes from rother_file
- samples: sample IDs to select samples from rint_file and rother_file
- sample_cols: column numbers to indicate sample, gene, and vaf in rint_file and rother_file

# Parameters for simulation results

- tumor_contents: assumed tumor contents
- min_vaf: VAFs below this value are ignored as LOD (limit of detection).

# Parameters for ABC

- abc.tol: tolerance rate of abc() in library abc
- abc.method: ABC method of abc() in library abc

# Inputs for next round ABC

- seleted_file: path to an output file, TOLSel_…, which lists simulation replications selected by ABC.
  - See below **ABC output directory**.
- generator_dir: path to the directory where the selected replications exist.
  - From this directory, the selected replications are copied in the bootstrap method to be ready for next round ABC.

6.6.3 Example 1

```
## Your working directory with R/, Input/, and abc.run{,_qsub}.sh.
base_dir <- './'


## For parallele execution =======================================
replicates <- 5
work_dir <- "./work/first"
run_script <- './abc.01.run.R'


## For ABC ========================================================
## Output
ABC_dir <- './ABC1'

## Sim VAFs
VAF_files_glob <- 'work/first/*/Output/VAF/VAF.txt'

## Obs VAFs
rint_file    <- 'Input/Samples/samples.Rint.txt'
rother_file  <- 'Input/Samples/samples.Rother.txt'
rother_regex <- "^_.*"
samples      <- c('TCGA-AZ-6608-01A-11D-1835-10')
sample_cols  <- c(6, 1, 11) ## sample, gene, vaf
```

```
##
tumor_contents <- c(1.00, 0.8, 0.6, 0.4)
min_vaf        <- 0.1

##
abc.tol    <- 0.1
abc.method <- 'rejection'
```

6.6.4 Example 2

```
## Similar to the example above
...

## Selection based on ABC
selected_file <- 'ABC1/TCGA-AZ-6608-01A-11D-1835-10/TOLSel_rejection_tumor_content=0.40_tol=0.100.txt'
generator_dir <- 'work/first/%06d'
```

6.7 ABC output directory

- Outputs by abc.c_make_data.R

```
<ABC>/<Sample Name>:
- param_mat.txt: Parameters        , data for abc() of library abc
- sumstat_*.txt: Summary statistics, data for abc() of library abc
- target_*.txt:  Target            , data for abc() of library abc
```

- Outputs by abc.d_exec_calc.R

```
<ABC>/<Sample Name>:
- TOLSel_rejection_*.txt: Selected simulation replications,         by abc()
- ABC_dist_*.txt: Distance between observation and each simulation, by abc()
```

6.8 Bayesian optimization

To obtain a point estimate from a joint posterior distribution implied by points selected through ABC, scripts/prmEst.BayesianOptim/* in the gitHub repository may be useful.

The scripts estimate the optimal point (a combination of parameter values) with shortest ABC distance.

7 Post processes

7.1 VAF

To get VAF (variant allele frequency).

7.1.1 Input

7.1.1.1 VAF.parameters.txt

Parameters

- input.cloneout: File path to cloneout.txt, which is used as input data.
- input.pom: File path to pointMutations_B.txt, which is used as input data.
- input.pomA: File path to pointMutations_A.txt, which is used as input data.
- input.cna: File path to CNAs.txt, which is used as input data.
- output.VAF: File path to the output, VAF.txt.
- output.ForVAF: File path to the output, ForVAF.txt.
- time: Simulation time step at which VAFs are calculated.
  - Multiple time steps can be input in vector, such as c(1, 10, 35).
  - Last time step is always output. Time steps over last time step are ignored.
  - Time steps before the first time step of a simulation are ignored.
- tumor_content: Assumed tumor content.
  - Multiple tumor contents can be input in vector, such as c(1.0, 0.8, 0.6).

7.1.2 Run

```
write_VAF( input = './Input/ForPosts/VAF.parameters.txt' )
```

This command inputs VAF.parameters.txt and outputs VAF.txt and ForVAF.txt.

7.1.3 Output

7.1.3.1 Output/VAF/VAF.txt

Columns

- Time: Simulation time step at which a VAF is calculated.
- tumor_content: Tumor content.
- PointMut_ID: ID of the pom, which is the same as in pointMutations_B.txt.
- Chr: Chromosome where the pom for the VAF is located.
- site: Chromosomal position where the pom is located.
- gene: Name of gene where the pom is located.
- VAF_primary: VAF in the primary tumor.
- VAF_metastatic: Same as above for the metastatic tumor.

7.1.3.2 Output/VAF/ForVAF.txt

This file contains more detailed information to calculate VAF.

Columns

- Time: Simulation time step
- PointMut_ID: ID of the pom, which corresponds to variant allele B, in contrast to original allele A.
- Parental_1or2: Indicates either of the two parental chromosomes where the pom is located.
- Chr: Chromosome name where the pom is located.
- Ref_pos: Reference position of the pom. This position is on the coordinate system of the human reference genome.
- Phys_pos: Physical position of the pom. The physical length of a parental chromosome is extended or shrunk by CNA dups or dels, respectively.
- Delta: Difference between the reference and physical positions.
- Copy_number_A: Copy number of allele A.
- Copy_number_B: Copy number of allele B.
- Gene_name: Name of a gene where the pom is located.

- N_speckled_normal: Number of speckled normal cells with the pom.
- N_primary: Same as above for primary tumor cells.
- N_metastatic: Same as above for metastatic tumor cells.
- N_spekled_normal_total: Total number of speckled normal cells.
- N_primary_total: Same as above for primary tumor cells.
- N_metastatic_total: Same as above for metastatic tumor cells.

7.2 TMB

To get TMB (tumor mutation burden).

7.2.1 Input

7.2.1.1 TMB.parameters.txt

Parameters

- input.Rother: File path to Rother.txt, which is used as input data.
- input.VAF: File path to VAF.txt, which is used as input data.

- input.CCDSdatabase: File path to CCDS.current.my.txt, which is used as input data.
- out: File path to the output.
- VAF.type: Column name of VAF type in VAF.txt to calculate TMB.
- VAF.LOD: Limit of detection (LOD) of VAF. Poms with VAFs below this value are neglected to calculate TMB.

7.2.2 Run

```
write_TMB( input = './Input/ForPosts/TMB.parameters.txt' )
```

This command inputs TMB.parameters.txt and outputs TMB.txt.

7.2.3 Output

7.2.3.1 Output/Info/TMB.txt

Columns

- Time: Simulation time step at which TMB is calculated.
  - All time steps in VAF.txt are shown.
- tumor_content: Tumor content.
  - All tumor contents in VAF.txt are shown.
- TMBvaf*LOD*%: TMB, the number of poms per Mb, at the LOD of *LOD* value, where the italic text represents a variable.
  - For example, TMBvaf5% represents TMB at the LOD of 5%.

7.3 Real time

To convert simulation time into real time.

7.3.1 Input

7.3.1.1 realTime.parameters.txt

Parameters

- input.cloneout: File path to cloneout.txt, which is used as input data.
- out: File path to the output.
- outUnit: Either of 'year', 'month', or 'day', which is used as real time unit in the output.
- col.replace: Whether to replace the column of simulation time with that of real time in the output.

*** Conversion is based on any of tumor VDT (classical, volume doubling time), tumor volume, or tumor diameter. Please choose one.**

VDT based conversion

- VDT$n1: Number of simulated cells at simulation time 1.
- VDT$n2: Same as above for simulation time 2.
- VDT$t1: Simulation time 1.
- VDT$t2: Simulation time 2.
- VDT$VDT.days: Known VDT in day, such as 200 (days) for primary colon cancer.

Volume based conversion

- realTumorSize$V1.mm3: Tumor volume in mm^3 at real time 1.
- realTumorSize$V2.mm3: Same as above for real time 2.
- realTumorSize$Ntype: Column name indicating the number of simulated cells in cloneout.txt, such as 'N_primary' and 'N_metastatic'.
- realTumorSize$deltaT.days: Real time 2 minus real time 1 in day.

Diameter based conversion

- realTumorSize$LD1.mm: Long diameter of tumor in mm at real time 1.
- realTumorSize$SD1.mm: Same as above for short diameter.
- realTumorSize$LD2.mm: Long diameter of tumor in mm at real time 2.
- realTumorSize$SD2.mm: Same as above for short diameter.
- realTumorSize$Ntype: Same as in Volume based conversion.
- realTumorSize$deltaT.days: Same as in Volume based conversion.

*** It is *un*necessary to write VDT or realTumorSize when the conversion rate of time units, cnvRate, is given in the function as follows:** `write_realTime_clone( ..., cnvRate = tau )`

7.3.2 Run

```
tau <- write_realTime_clone( input = './Input/ForPosts/realTime.parameters.txt' )
```

This command inputs realTime.parameters.txt, and outputs cloneout.realTime.txt and returns the conversion rate. You can use the obtained conversion rate for successive simulations:

```
write_realTime_clone( input = './Input/ForPosts/realTime.parameters.drg.txt', cnvRate =
tau )
```

7.3.3 Output

7.3.3.1 Output/cloneout.realTime.txt

Columns

- Time.*outUnit*: Real time in unit of *outUnit*, where the italic text represents a variable. For example, Time.year represents real time in year.
  - This column is added right to the Time column in cloneout.txt.
  - If col.replace in realTime.parameters.txt is T, the Time column is replaced with this column.

7.4 Evaluation scores

To evaluate simulated VAFs at **last** simulation time step, based on observed VAFs. Multiple simulation replications are used, typically, after parallel computation.

7.4.1 Input

7.4.1.1 evals.parameters.txt

Parameters

- obs$sample: Sample (case/patient) ID in observed data.
- obs$Rint$file: File name of observed data on genes in Rint (regions of interest).
- obs$Rint$col.gene: Index of the column to indicate gene name in the above file.
- obs$Rint$col.sample: Same as above for sample ID.
- obs$Rint$col.VAF: Same as above for VAF.
- obs$Rother$file: File name of observed data on genes in Rother (regions other than those of interest).
- obs$Rother$col.gene: Index of the column to indicate gene name in the above file.
- obs$Rother$col.sample: Same as above for sample ID.
- obs$Rother$col.VAF: Same as above for VAF.
- obs$Rother$glob: Glob pattern to capture the names of Rother genes to rename as the same name.
  - Rother genes are **indistinguishable** regions where only passenger mutations reside.

- sim$glob: Glob pattern to capture the files of VAF.txt.
- sim$type: Column name of VAF type in VAF.txt for evaluation.
- sim$tc: Tumor contents present in VAF.txt.
  - Multiple tumor contents can be input in vector.

- sim$n.rep: Number of simulation replications you performed.
  - Please set it to the number of simulation replications you initially intended. For example, you perform 1000 simulations but may get only 950 VAF.txt files due to 50 program stops. In this case, please set it to 1000, not 950.
  - VAFs of zero values are supplied for the 50 stopped replications, like "penalty". Bad settings of parameter values often cause program stops and can be penalized in this way.
- sim$Rother$glob: Glob pattern to capture the names of Rother genes to rename as the same name.

- LOD: Limit of detection (LOD) of VAF. VAFs below this value are **converted to 0** in evaluation.
- out.stats: File path to the output, evals.stats.txt.
- out.obs_sims: File path to the output, evals.obs_sims.txt.
- survivorToo: TRUE or FALSE to output statistics that are calculated only for survivors (excluding extinct clones), too.
  - If T, an additional file is output in a modified file name of out.stats with 'survivor'.
  - out.stats itself has statistics calculated for survivors **PLUS** extinct clones.

7.4.2 Run

```
write_evals( input = './Input/ForPosts/evals.parameters.txt' )
```

This command inputs evals.parameters.txt and outputs evals.stats.txt and evals.obs_sims.txt.

7.4.3 Output

7.4.3.1 Output/Info/evals.stats.txt

Columns

- sample: Sample ID.
- tumor_content: Tumor content.
  - Only tumor contents included in VAF.txt are shown.
- id: ID of evaluated VAF.
- obs: Observed value.
- sim.mean: Mean of simulated data.
- sim.med: Median of simulated data.
- sim.sd: Standard deviation of simulated data.
- sim.Q1: First quartile of simulated data.
- sim.Q3: Third quartile of simulated data.
- ME: Mean error.
- RMSE: Root mean squared error.

7.4.3.2 Output/Info/evals.obs_sims.txt

This file shows original data used for evaluation, which is usable for confirmation and for other evaluation methods.

Columns

- sample: Sample ID.
- tumor_content: Tumor content.
- id: ID of evaluated VAF.
- obs: Observed VAF value.
- sim.*replication*: Simulated VAF value of *replication*-th simulation replication.
  - out.*replication*: out is noted for columns over sim$n.rep.

8 Preparing input data

It is better to prepare input data from templates or those you have already had than from the scratch. Below, we will make notes on input data.

8.1 Input/

8.1.1 CF.txt

- It would be better to set the compaction factor of the division rate as, for example, around 0.1-0.3 to see the growth of cancer cells in a reasonable computation time at a reasonable time resolution.
- The compaction factor of invasion would be a small, for example, 1e-2.

8.1.2 cloneinit.txt

- Do not set the same position for different poms. We assume the infinite-site model.
- When you want to make a suppressor gene dysfunctional, it may need to set a dummy del of driver at one- or a few-base last of the last exon.
  - Because of the recessive model
  - Because such a last base or bases of a deletion do not much affect the chromosomal coordinate.

8.1.3 EF.Rint.txt

- Check if the values of Time_step and Condition are those you intend.
  - If Time_step of a mutation conditioned on another mutation is smaller than that of the other mutation, the simulation will usually fail.

8.1.4 EF.Rother.txt

- Check if Type and Gene are what you intend.

8.1.5 Rint.txt

- Check if the genes are what you intend.
- Check if the CDS_ID and Type are what you intend.

8.1.6 Rother.txt

- Check if the genes are the same as those you intend in EF.Rother.txt.

8.1.7 hallmark_weights.txt

- Usually, the values are just generated from the generator.

8.1.8 parameters.txt

- If you switch off tumblers, you do not need to care about parameters related to the tumblers.
- Perform fine tuning with generators and ABC if necessary.

8.1.8.1 For individual parameters

- m0: You can find typical values in literature, such as 1e-9.
- uo: Can be estimated from the following idea: poms of gain-of-function tend to concentrate at a few positions in a gene, of which the total size of the exons is roughly in the scale of 1 kb. Missense (non-synonymous) mutations may be possible. For example, 1/1 kb * (maximally) 2/3 in codons ~ a scale of 1e-4.
- us: Can be estimated from the following idea: poms of loss-of-function tend to scatter over a gene. Non-silent, especially truncating mutations at the upper part of a gene are likely. For example, 3 / 64 in the codon table in the upper ~ a scale of 1e-2.
- dN: Because this represents how many cells are expected to didive in one simulation time unit (when kappa = 1), the values will be determined only from experiences in simulations.
  - When kN = NA, small values such as <1e-2 lead to stable simulation results.
- kN: NA will be convenient, which balances the division and death rates of **normal** (not tumor) cells.
  - If you want to include the effects of unknown drivers on the cell division rate, you can set the value of kN (or dN) so that dN - kN > 0 (as dummy).
- sN: Determined only from experiences in simulations. Mathematically, graphs of the sigmoid curves showed that around 10 would be effective.
- K_N: Because this is the carrying capacity without angiogenesis, this would be roughly 1e9. With angiogenesis, this would be roughly 1e12.
- Fb: Because this is the expansion rate by angiogenesis, you can find typical values in literature, such as 1e3.
- ctmax: The replication (Hayflick) limit is roughly 50 or around. Because cells have already experienced divisions until the transform into tumor, this would be roughly 50 minus 30-40 or around.

- m_dup: See Processing observed data.
- m_del: See Processing observed data.
- ave_len_dup: See Processing observed data.
- ave_len_del: See Processing observed data.
- uo_dup: Biologically, this would be a high value, such as 0.8.
- us_dup: Perhaps, 0.
- uo_del: Perhaps, 0.
- us_del: Biologically, this would be a high value, such as 0.8.
- Zim: Biologically, this would be an extremely small value, such as 1e-6.

- control.censor_cell_number: Typically, set it as a clinically-detectable size, such as 1e9. For tumor of 10 mm in the diameter, it would be roughly 1e9.
- control.censor_time_step: Depends on you. It would be good to set an arbitrary large number, where simulation time only depends on control.censor_cell_number.
- control.censor_real_time: Depends on you.
- control.monitor: Depends on you.

8.2 ForGenerators/

8.2.1 Each skeleton file

- Confirm each skeleton file first. Previous settings may unintentionally remain.

8.2.2 CF.parameters.txt

- No notes now

8.2.3 EF.Rint.parameters.txt

- Perhaps, you many need to change the values of hyper-parameters for Time_step.
  - If VAFs for a gene are higher than expected especially in post ABC, you may change hyper-parameters related to the mean of Time_step to lower.
    - Vice versa for a VAF value lower than expected.

8.2.4 EF.Rother.parameters.txt

- N_mut is expected to be proportional to TMB.

8.2.5 Rint.parameters.txt

- Check if the genes are what you intend when you use this generator.

8.2.6 cloneinit.parameters.txt

- Check if the range of the number of poms (pom[['*cloneID*']]$dist$x) is what you intend.

8.2.7 hallmark_weights.parameters.txt

- Perhaps, the first choice would be the non-informative Dirichlet distributions:
  - dist$*$spec: dirichlet
  - dist$*$alpha.column: NULL

8.2.8 parameters.parameters.txt

- It would be better to make a small change at one time.
  - e.g., to focus only on m0 and dN in limited ranges.

8.3 ForPosts/

8.3.1 VAF.parameters.txt

- Check if time is what you intend.
  - If you want only to see VAFs at the simulation end, set a large number or a number equal or more than control.censor_time_step.
  - If you want to see the dynamics of VAFs, you can set multiple values you want to monitor.

8.3.2 TMB.parameters.txt

- Check if VAF.LOD (and VAF.type) is what you intend.

8.3.3 evals.parameters*.txt

- In particular, check if sim$n.rep is set as you intend
  and if LOD is appropriately set.

- Check if obs${Rother, Rint}$col.* indicate the correct column numbers in your observed data.
- Check if the three glob patterns ({obs, sim}$Rother$glob and sim$glob) are what you intend.
- Check if sim$type and sime$tc are what you intend.

- When you get an error, check the presence of the **directory** indicated by out.*.

8.3.4 realTime.parameters.txt

8.3.4.1 When you use VDT

- Check if VDT$VDT.days is what you intend.
- Check if the **number of simulated tumor cells** in the **time zone** is roughly in an exponential growth.
  - The **number of simulated tumor cells**: Check the number of primary or metastatic cells in simulations, depending on the cell type you intend
  - The **time zone**: indicated by VDT${t1, t2}
  - Check the rough number of VDT${n1, n2}
  - Shift the time zone if it greatly differs from an exponential growth.

8.4 DATA/

8.4.1 CCDS.current.my.txt

- Check if the bld version of CCDS.current.my.txt matches with that of observed data.

  The given CCDS.current.my.txt is a copy of either CCDS.current.Hs37.my.txt (Bld. 37) or CCDS.current.Hs38.my.txt (Bld. 38).
- When it does not match, copy a right one under this directory.

9 Preparing observed data

Observed data are not required by simulation itselt, but they are necessary for ABC and evaluation, and useful for setting simulation inputs.

9.1 See observed SNV/indel data you have

For ABC and evaluation, it is necessary to format observed data into samples.{Rint, Rother}.txt, which only have to have the columns of gene name, sample name, and VAFs.

- *You can change the names of samples.{Rint, Rother}.txt, but here we use these names for convenience.*

You can format them as follows.

9.1.1 0) Confirm if observed SNVs/indels are tumor-specific.

- To see cancer cells evolving from normal cells, SNVs/indels must be tumor-specific (must be somatic, not including germline mutations).

9.1.2 1) Add the VAF column

- Calculate observed VAF if it's absent, and add the column.
  - e.g., VAF = t_alt_count / (t_alt_count + t_ref_count) in the MAF format.

9.1.3 2) Select columns

- Keep the columns of gene name and sample name.
  - The column numbers of the three columns (gene name, sample name, VAF) will be required later.
  - The column order does not matter.
- Keep other columns useful for interpretation. These columns are not necessary to run the program, but will be informative later.
  - For the MAF format, we usually select the columns of Hugo_Symbol, Chromosome, Start_Position, End_Position, Variant_Type, Tumor_Sample_Barcode, HGVSc, HGVSp_Short, t_ref_count, and t_alt_count, adding t_vaf (calculated VAF).

9.1.4 3) Select samples

- You can limit samples if the file size is large. A sample among them will be later used for analysis.

9.1.5 4-a) Set samples.Rint.txt – regions of interest

- From the file of subsection 3), select rows with possible driver genes you think in the column of gene name.
    - You can sort genes with VAFs. Generally, well-known tumor-related genes with high VAFs are likely to be drivers.
    - It is likely that genes with low VAFs close to LOD have little influence, if any.
    - You can later evaluate how much your hypothesis is likely by the model comparison technique of ABC.

      ** If there are multiple SNVs/indels for a possible driver gene, you may select one with the highest VAF. The others can be threw away or kept as passengers.*

9.1.6 4-b) Set samples.Rother.txt – other regions

- From the file of subsection 3), select rows with possible passenger genes you think in the column of gene name.
    - You can sort chromosome numbers and then keep rows until you think the number of passengers is enough (e.g., around 10 to 20 at the first trial).
- You may need to change the gene names to artificial gene names to match with artificial gene names in Input/Rother.txt.

9.2 See observed CNA data if you have

Observed CNA data can be useful for setting simulation input values.

** If you do not have, you may apply data of the same cancer type you have already had to the estimates below as priors.*

You can set them as follows.

9.2.1 1) Estimation of m_del and m_dup

** m_del and m_dup are in Input/parameters.txt.*

- Count SNVs/indels across autosomes.
- Count dels and dups across autosomes.
    - You may need to see a histogram of logR values (associated with copy numbers)
      and then decide a cut-off value to determine del and dup.
- Take the ratio of the number of dels to that of SNVs/indels, which equals to the ratio of m_del to m0.
- Apply the same to dups.
- Then, you can get the estimates of m_del and m_dup to a specific m0.

9.2.2 2) Estimation of ave_len_del and ave_len_dup

** ave_len_del and ave_len_dup are in Input/parameters.txt.*

- Get the median lengths of dels and dups.
  - Mean is not robust to outliers.
- Calculate the median / ln(2), which equals to ave_len_{del, dup} under the assumption of an exponential distribution.
  - You can compare this value with the direct mean.

9.2.3 3) Check the presence of dels and dups in genes you specified as regions of interest.

- Then, you may get information useful for setting dels and dups in EF.Rint.txt and cloneinit.txt.

10 Analyses

10.1 1) First, check the points below for a single simulation

10.1.1 See VAF.txt

- Simulated VAFs are close to observed ones?
  Note that differences of values by 5-10% often happen due to the observation errors of VAFs in NGS.
- The number of VAFs are close to observed ones?
- Check if the range of tumor contents is set as you intend.

10.1.2 Check LOD of VAFs

- LOD is appropriately set? For example, 10% or 5%?
  Then, do you have an enough number of observed VAFs?

10.2 2) Checkpoints for simulation results after ABC

10.2.1 First

- See the evals.stats file for post ABC (evals.stats.post.txt) to check if the summary statistics are close to observed ones.

10.2.2 When you find something wrong

- Check stderr.txt in simulation replications.
- Go to the folder of an arbitrary simulation replication.
  - Check if the result is consistent with that from a single simulation without using ABC.
  - Check the presence of ___tmp.addPnt___ and the consistency of chromosomes and the positions.
- See the log from post.eval.R on the screen to check if enough VAF.txt files are globbed.

10.2.3 Check how many simulation replications survived extinction

- At last time step, a simulation replication reached the number of cells you intended?
  - Can be checked with a LINUX command pipeline combined with `xargs (ls |xargs -i ...)` and `tail`
    - For `xargs`, you need to stand under the directory where `ls` shows replications such as named 000001/, 000002/, . . .
- If the proportion of survivors is below ~80% after ABC, something may be wrong.

10.3 3) After getting through the checkpoints

10.3.1 Tumor contents

- You may need to determine on a tumor content for analyses.
  - Simply, you can use evaluation scores such as ME (mean error) in the evaluation file to select which tumor content leads to a good-fit with observed data.
  - Also, you can use `postpr()`, which performs model selection of ABC. Refer to the section below.

10.3.2 Posteriors and statistics

- Aggregate the values of parameters and of variables of interest. Then, see the distributions.
  - Values can be aggregated over replications with `xargs` (`ls |xargs -i ...`) or with a simple script to get the distributions.

10.4 4) For ensemble plots

We provide scripts we internally use to perform the followings in scripts/analyses/analyses. in the gitHub repository for your reference.*

10.4.1 1. Evaluate the data fit

- You need to confirm if simulated VAFs are close to observed VAFs on average.
  - You can use evals.stats.post.txt.

10.4.2 2. Tune virtual drug intervention

- You may need to change the parameter values of virtual drug intervention. With parameter values changed, you can re-run the drug intervention part only, not simulating the entire time steps.
  - Can be re-run with `xargs` with ‘`Rscript --vanilla --slave`’ and an R script with `start_simulation( first = F, ... )` having the changed parameters of drug intervention.
    - For `xargs`, you need to stand under the directory where `ls` shows replications such as named 000001/, 000002/, . . .

10.4.3 3. Adjust VDT when you use VDT

- You may need to adjust the time scaling of VDT when you use VDT for the real time conversion.
    1. Make an ensemble plot WITHOUT time calibration, as described below.
        - Because it is hard to handle negative values.
    2. Find the range of a exponential growth of tumor.
    3. Reset VDT${n1, n2, t1, t2} in realTime.parameters.txt.
- Run the conversion of simulation time into real time for each replication.
    - Can be run with `xargs` with '`Rscript --vanilla --slave`' and an R script with `write_realTime_clone()`.

10.4.4 4. Real time for virtual drug intervention

- You may need to convert the simulation time unit into the real time for simulation replications after you tune virtual drug intervention.
    - Can be run with `xargs` with '`Rscript --vanilla --slave`' and an R script with `write_realTime_clone()`.

10.4.5 5. Draw ensemble plots

- You can make the ensemble plots of 1) tumor growth from simulation replications of cloneout.txt, 2) the timings of pom insertions from those of EF.Rint.txt, and 3) TMB from those of TMB.txt.
    - An ensemble plot is a plot in which statistics such as the mean and percentiles over replications are plotted against time.
    - It is necessary to calibrate time so that the observed time is set to 0, and the past and future are measured with negative and positive values, respectively.
        1. Identify replications in which the number of cancer cells reached the number you specified in simulation settings.
        2. For those replications, subtract the last time of a simulation from every time so that time is calibrated as above.
    - You may first utilize `xargs` (`ls |xargs -i ...`) to make a table of raw data across replications, and then use R to get statistics and draw an ensemble plot.
        * To avoid multi-modal plots, it may be better to aggregate data for the same input or at least for the same order of magnitude in inputs, especially of dN and the compaction factor of growth.
    - For example, the first command to make a table may be the following, though you may need to adjust this to your computer environments:

```
ls | xargs -i sh -c 'test -e {}Output/cloneout.realTime.txt && echo {}' | xargs
-i sh -c 'echo -n -e "{}\t"; tail -n 1 {}Output/cloneout.realTime.txt' |cut
-f1,2,9|awk '$3 > 1e9' |xargs -n 3 sh -c 'awk -F"\t" "{print FILENAME, \$1-${1},
\$8, \$1, \$2, ${1}, ${2} }" ${0}Output/cloneout.realTime.txt ${0}Output/cloneout.realTime.drg
|grep -v avg|sed 's/ /\t/g'|cut -f1-4|grep -v -P '\tN_primary' |sort |uniq |sort
-k1,1 -k2,2n -k3,3nr |less
```

- You can make the ensemble plots of 1) VAFs from simulation replications of VAF.txt and 2) subpopulations from those of cloneout.txt.

  - To draw these is more difficult than the ensemble plots above, because of multiple entities such as multiple poms and subpopulations.
  - You may need to sort VAFs or N_cells at each time in the same replication and then rename ids with the ordered numbers. Statistics are calculated over the ids. For example, top 1s across replications at each time are used for the calculation of statistics. “‘

10.4.6 6. Draw ensemble plots of virtual drug intervention

- For a table for ensemble plots, you can **concatenate** an output file that has the results of virtual drug intervention **to** the output file of the first simulation without drug intervention in each simulation replication.
  - The concatnation is examplified in the LINUX command above, where `awk` takes two files: `awk ... ${0}Output/cloneout.txt ${0}Output/cloneout.drg.txt'`
  - Thus, time with virtual drug intervention is measured with positive values.

10.5 4) When you have different statistical models

For example, when you start simulations with different initial states or when you try different tumor contents.

Different statistical models can be evaluated also in the basic ABC procedures if different states are randomly generated from priors. But when you define different states manually, you can use the following method:

- `postpr()` in the vignette of library(abc).

11 Sequence of simulations

11.1 test3.R

The sequence of simulations can be used for emulation of the combined target therapy as a sequential drug intervention with different drugs and treatments.

Please see tugMedi.1.0/test3.R example with a sequence of simulations, and edit it under your way. This example shows how one may manage the simulation flow changing parameters set in a sequence of simulations. The input, output, generators, and files' formats are described in the corresponding chapters. Here, only the organization of the sequential set of simulations is explained.

11.1.1 First simulation

After generators there is a first simulation (parameter ‘first’ is TRUE by default, so it can be skipped):

```
start_simulation( first       = T,
                  input_files = 'Input/DATA/FILES.txt',
                  saveTo      = 'Output/Results.sim.01.RDS',
                  seed        = NA
                 )
```

This command starts a first simulation where usually cancer clones appear and developed. Input data are defined through option `input_files`. Please see the section of Input data for the details of input data. RDS data are saved to `saveTo` for the second simulation, where usually drug intervention is performed. Just after the first simulation the post calculations follow with commands:

```
write_VAF(   input = './Input/ForPosts/VAF.parameters.txt' )
write_TMB(   input = './Input/ForPosts/TMB.parameters.txt' )
write_evals( input = './Input/ForPosts/evals.parameters.txt' )
```

Next several simulations are about drug intervention.

11.2 Drug intervention stages

11.2.1 First drug intervention

The second simulation starts from the next to the last time step of the first simulation. Before the continue of the first simulation, it is necessary to update the first RDS file before the second simulation by the command:

```
update_RDS_from_output( in.file = 'Input/DATA/FILES.txt',
                        rds.file = 'Output/Results.sim.01.RDS' )
```

By this, pool of clones can be updated in the `rds.file` from an output file (`cloneout.txt`). Other input parameters can be changed in other input files in the exactly the same way as for the first simulation using the another list of input files. For example, the second simulation command reads `input_files = 'Input/DATA/FILES.drg1.txt'` to use parameters.drgInt.txt, an input file different from parameters.txt, used in the first simulation. Thus, the simulation parameters can be changed through the input data at the second simulation. Typically, you may extend `control.censor_cell_number` (the censor [program stop] of the maximum cell number) and `control.censor_time_step` (maximum simulation time step) in the input data for drug intervention.

The `drug_int_param` in the second simulation command is set as follows:

```
drug_int_param <- list( 'kill_prob' = 0.5,
                        'block_prob'= 0.0,
                        'gene' = c('KRAS', 'APC') ) # Any of them is a target
```

And implementation of the second simulation can be done by:

```
system.time(
    start_simulation( first       = F,
                      input_files = 'Input/DATA/FILES.drg1.txt',
                      loadFrom    = 'Output/Results.sim.01.RDS',
                      saveTo      = 'Output/Results.sim.02.RDS',
                      seed        = NA,
                      drug_int_param = drug_int_param
                      #                  drug_int_param = NULL
    )
)
```

Because drug intervention simulation usually takes advantage of the results of the first simulation, option `first` is `F`. The results of the first simulation are loaded with `loadFrom`, and the results of drug intervention simulation are saved to `saveTo`. As with the first simulation, input data are defined through `input_files` and the random seed can be set with `seed`.

11.2.2 Next drug intervention

The next stage of drug intervention is implemented by the same way:

```
update_RDS_from_output( in.file = 'Input/DATA/FILES.drg1.txt',
                        rds.file = 'Output/Results.sim.02.RDS' )

drug_int_param <- list( 'kill_prob' = 0.2,
                        'block_prob'= 0.3,
                        'gene' = c('KRAS', 'APC') ) # Any of them is a target

system.time(
    start_simulation( first       = F,
                      input_files = 'Input/DATA/FILES.drg2.txt',
                      loadFrom    = 'Output/Results.sim.02.RDS',
                      saveTo      = 'Output/Results.sim.03.RDS',
                      seed        = NA,
                      drug_int_param = drug_int_param
    )
)
write_realTime_clone( input = './Input/ForPosts/realTime.parameters.txt' )
```

The difference with the first drug intervention is only in the files' names where the input files are from output of the previous simulation:

`rds.file = 'Output/Results.sim.02.RDS'` `loadFrom    = 'Output/Results.sim.02.RDS'` `input_files = 'Input/DATA/FILES.drg2.txt'`

And also, `drug_int_param` is changed to represent another target drug.

This procedure can be repeated many times under condition of the change of files' names.

11.3 Outputs

11.3.1 Output/

You will get a new Output/, which has:

- cloneout.txt: Simulation result of cancer-cell evolution.
- cloneout.realTime.txt: Estimated real time is added to cloneout.txt.
- Results.sim.01.RDS: RDS data for the first simulation.
- Results.sim.02.RDS: RDS data for the second (first drug intervention) simulation.
- Results.sim.03.RDS: RDS data for the third (second drug intervention) simulation.
- Mutations/: Data on point mutations and CNAs are stored.
- VAF/: VAF data are stored.
- Info/: Other information is stored.

See the section of Output data for the details of these outputs.