#### 部分核心程序代码

1. MosaicHunter贝叶斯判别模型的C语言版本实现

1-1. 程序包流程框架图（2014年1月版本）

generate\_beta\_log10\_val\_file

beta\_log10\_val.tsv

Yyx\_real\_log10lik\_from\_baseQ

real\_log10lik\_from\_baseQ.tsv

Yyx\_estimate\_mosaic\_AF\_from\_baseQ

mosaic\_AF\_estimate.tsv

Yyx\_genotype\_log10lik\_with\_precalc\_beta

Yyx\_individual\_genotyper

genotype\_posterior.tsv

原始数据：  
ID, ref\_baseQ, alt\_baseQ

Yyx\_trio\_genotyper

trio\_genotype\_posterior.tsv

prior\_allele\_frequency.tsv

dbSNP MAF

黄岳程序整理得到

genotype\_log10lik.tsv

1-2. Yyx\_individual\_genotyper.base\_change\_relative\_rate.v0\_2.c

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*

\* \* This program is just used to compute individual genotype posterior

\* based on 2 files: individual genotype likelihood and prior,

\* which is pretty simple, and to suit the recent framework

\* \* <base\_change\_relative\_rate> set the relative rate of base change

\* relative to mosaic rate

\* \* <genotype\_log10lik> and <log10\_prior> should be sorted

\* on the first column ID(chr:pos:refBase:altBase)

\*

\* Version: 0.2.0 (2014-06-09)\n");

\* Author: Adam Yongxin Ye @ CBI\n");

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <stdlib.h>

#include <stdio.h>

#include <math.h>

#include <string.h>

#include <assert.h>

#include <ctype.h>

#include <float.h>

#include <unistd.h>

#define MIN(X,Y) ((X) < (Y) ? (X) : (Y))

#define MAX(X,Y) ((X) > (Y) ? (X) : (Y))

#define POW10(x) (pow(10.0, x))

#define LOGZERO -1e100

#define LINE\_LEN 1000

#define YYX\_EOF\_STR "EOF\_Yyx\_EOF\_Yyx\_EOF"

double mosaic\_rate = 1e-7;

double minus\_1\_AF = 0.002;

double minus\_2\_AF = 1e-4;

char \*chr\_pos\_delim;

char \*chr\_order[1000];

int chr\_order\_len = 0;

char individual\_sex[5];

double base\_change\_matrix[16]; // row\_idx \* 4 + col\_idx, row and col 0-A,1-C,2-G,3-T

double log10\_add\_pow10(double a, double b);

int compare\_two\_ID\_chr\_pos(const char \*IDa, const char \*IDb);

int base2idx(char base);

double \*normalize\_population\_ACGT\_allele\_frequency(double \*population\_ACGT\_allele\_frequency, double minus\_1\_AF, double minus\_2\_AF);

double \*individual\_log10\_prior(const char \*chr\_type, char individual\_sex, const char \*individual\_ref\_alt\_base, const double \*population\_ACGT\_allele\_frequency, double mosaic\_rate, double \*tmp\_4\_double\_vec);

double \*yyx\_calc\_posterior(const double \*log10\_prior\_vec, const double \*log10\_likelihood\_vec, double \*tmp\_four\_double\_vec, double base\_change\_relative\_rate);

char get\_max\_AF\_base(const double \*population\_ACGT\_allele\_frequency, char base\_excluded);

char get\_higher\_AF\_base(const double \*population\_ACGT\_allele\_frequency, char base1, char base2);

int individual\_genotyper\_process\_func(int num\_str, const char \*\*each\_str);

int match\_and\_process\_firstColumnSorted\_files(int num\_files, char \*\*filenames, int (\*compare\_two\_ID)(const char \*IDa, const char \*IDb), int (\*process\_func)(int num\_str, const char \*\*each\_str), int process\_error\_should\_stop );

/\*\*

\* @brief Computes log10(10^(a) + 10^(b))

\*/

double log10\_add\_pow10(double a, double b)

{

/\* in R code:

tmp\_max <- max(vec)

log10(sum(10^(vec-tmp\_max))) + tmp\_max

\*/

if (a > b) {

return a + log10(1+POW10(b-a));

} else {

return b + log10(1+POW10(a-b));

}

}

/\* end of log10\_add\_pow10() \*/

/\*\*

\* @brief The compare function for sorting ID(chr\_pos)

\*

\* compare ID, make YYX\_EOF\_STR the largest

\* if ID has '\_', cut as chr\_pos, str\_cmp for chr and int\_cmp for pos

\* otherwise, str\_cmp ID

\*/

int compare\_two\_ID\_chr\_pos(const char \*IDa, const char \*IDb)

{

char modifiable\_IDa[LINE\_LEN];

char modifiable\_IDb[LINE\_LEN];

char \*chr\_a;

char \*chr\_b;

char \*pos\_str\_a;

char \*pos\_str\_b;

long pos\_a, pos\_b;

int chr\_idx\_a, chr\_idx\_b;

strcpy(modifiable\_IDa, IDa);

strcpy(modifiable\_IDb, IDb);

if(strcmp(IDa, YYX\_EOF\_STR)==0){

if(strcmp(IDb, YYX\_EOF\_STR)==0){

return 0;

}else{ // IDb != YYX\_EOF\_STR

return +1; // make YYX\_EOF\_STR the largest

}

}else{ // IDa != YYX\_EOF\_STR

if(strcmp(IDb, YYX\_EOF\_STR)==0){

return -1; // make YYX\_EOF\_STR the largest

}else{ // IDb != YYX\_EOF\_STR

if(strcmp(chr\_pos\_delim, "")!=0){

pos\_str\_a = modifiable\_IDa;

chr\_a = strsep(&pos\_str\_a, chr\_pos\_delim);

sscanf(pos\_str\_a, "%ld", &pos\_a);

pos\_str\_b = modifiable\_IDb;

chr\_b = strsep(&pos\_str\_b, chr\_pos\_delim);

sscanf(pos\_str\_b, "%ld", &pos\_b);

if(strcmp(chr\_a,chr\_b)==0){

return pos\_a - pos\_b;

}else{

for(chr\_idx\_a=0; chr\_idx\_a<chr\_order\_len; chr\_idx\_a++){

if(strcmp(chr\_a, chr\_order[chr\_idx\_a])==0){

break;

}

}

for(chr\_idx\_b=0; chr\_idx\_b<chr\_order\_len; chr\_idx\_b++){

if(strcmp(chr\_b, chr\_order[chr\_idx\_b])==0){

break;

}

}

return chr\_idx\_a - chr\_idx\_b;

}

}else{

return strcmp(IDa, IDb);

}

}

}

}

/\* end of compare\_two\_ID\_chr\_pos() \*/

/\*\*

\* @brief a test process\_func()

\*/

int test\_process\_func(int num\_str, const char \*\*each\_str)

{

int fi;

for(fi=0; fi<num\_str; fi++){

printf("%d\t%s\n", fi, each\_str[fi]);

}

printf("\n");

return 0;

}

/\* end of test\_process\_func() \*/

/\*\*

\* @brief array idx2base[] and function base2idx()

\*/

const char \*idx2base = "ACGT.";

int base2idx(char base){

if(base=='A'){

return 0;

}else if(base=='C'){

return 1;

}else if(base=='G'){

return 2;

}else if(base=='T'){

return 3;

}else{

return 4;

}

}

/\* end of base2idx() \*/

/\*\*

\* @brief normalize 4 allele frequency, so that they sum to 1

\*

\* input population\_ACGT\_allele\_frequency will be modified

\*

\* in fact, this is used to stablize mosaic\_rate (1e-7)

\*/

double \*normalize\_population\_ACGT\_allele\_frequency(double \*population\_ACGT\_allele\_frequency, double minus\_1\_AF, double minus\_2\_AF)

{

int bi;

for(bi=0; bi<4; bi++){

if(population\_ACGT\_allele\_frequency[bi]>=0){

// good, do nothing

}else if(population\_ACGT\_allele\_frequency[bi]==-1){

population\_ACGT\_allele\_frequency[bi] = minus\_1\_AF;

}else if(population\_ACGT\_allele\_frequency[bi]==-2){

population\_ACGT\_allele\_frequency[bi] = minus\_2\_AF;

}else{

fprintf(stderr, "Warning: unknown ref population allele frequency code '%f', I use '-2' instead\n", population\_ACGT\_allele\_frequency[bi]);

population\_ACGT\_allele\_frequency[bi] = minus\_2\_AF;

}

}

return population\_ACGT\_allele\_frequency;

}

/\* end of normalize\_population\_ACGT\_allele\_frequency() \*/

/\*\*

\* @brief calculate individual's log10 prior based on population allele frequency

\*

\* you should first call normalize\_population\_ACGT\_allele\_frequency() to code '-1', '-2' and normalize

\*

\* output is log10 prior for ref-hom (or ref-hemi), het (or LOGZERO), alt-hom (or alt-hemi), mosaic

\*/

double \*individual\_log10\_prior(const char \*chr\_type, char individual\_sex, const char \*individual\_ref\_alt\_base, const double \*population\_ACGT\_allele\_frequency, double mosaic\_rate, double \*tmp\_4\_double\_vec)

{

int individual\_ref\_base\_idx = base2idx(individual\_ref\_alt\_base[0]);

int individual\_alt\_base\_idx = base2idx(individual\_ref\_alt\_base[1]);

double ref\_AF = population\_ACGT\_allele\_frequency[individual\_ref\_base\_idx];

double alt\_AF = population\_ACGT\_allele\_frequency[individual\_alt\_base\_idx];

double log10\_ref\_AF;

log10\_ref\_AF = log10(ref\_AF/(ref\_AF+alt\_AF)); // normalize of the 2 allele/base

double log10\_alt\_AF;

log10\_alt\_AF = log10(alt\_AF/(ref\_AF+alt\_AF)); // normalize of the 2 allele/base

tmp\_4\_double\_vec[3] = log10(mosaic\_rate);

double log10\_not\_mosiac = log10(1-mosaic\_rate);

if(chr\_type[0]=='A' || (chr\_type[0]=='X' && individual\_sex=='F') ){ // autosome or (chrX and female

tmp\_4\_double\_vec[0] = 2\*log10\_ref\_AF + log10\_not\_mosiac;

tmp\_4\_double\_vec[1] = log10(2)+log10\_ref\_AF+log10\_alt\_AF + log10\_not\_mosiac;

tmp\_4\_double\_vec[2] = 2\*log10\_alt\_AF + log10\_not\_mosiac;

}else if(chr\_type[0]=='X' && individual\_sex=='M'){ // chrX and male

tmp\_4\_double\_vec[0] = log10\_ref\_AF + log10\_not\_mosiac;

tmp\_4\_double\_vec[1] = LOGZERO;

tmp\_4\_double\_vec[2] = log10\_alt\_AF + log10\_not\_mosiac;

}else if(chr\_type[0]=='Y' && individual\_sex=='M'){ // chrY and male

tmp\_4\_double\_vec[0] = log10\_ref\_AF + log10\_not\_mosiac;

tmp\_4\_double\_vec[1] = LOGZERO;

tmp\_4\_double\_vec[2] = log10\_alt\_AF + log10\_not\_mosiac;

}else if(chr\_type[0]=='Y' && individual\_sex=='F'){ // chrY and female

// for chrY and female, set all prior = 1

// which genotyping results should be then filtered

tmp\_4\_double\_vec[0] = 0;

tmp\_4\_double\_vec[1] = 0;

tmp\_4\_double\_vec[2] = 0;

tmp\_4\_double\_vec[3] = 0;

}else{

fprintf(stderr, "Warning: unknown chr\_type '%s' in individual\_log10\_prior()\n", chr\_type);

}

return tmp\_4\_double\_vec;

}

/\* end of individual\_log10\_prior() \*/

/\*\*

\* @brief Computes genotype state posterior

\*

\* calculate posterior as R code below:

\*

ref\_hom\_posterior=ref\_hom\_prior\*ref\_hom\_likelihood

het\_posterior=het\_prior\*het\_likelihood

alt\_hom\_posterior=alt\_hom\_prior\*alt\_hom\_likelihood

somatic\_posterior=somatic\_prior\*somatic\_likelihood

sum\_posterior=ref\_hom\_posterior+het\_posterior+alt\_hom\_posterior+somatic\_posterior

c(ref\_hom\_posterior/sum\_posterior,het\_posterior/sum\_posterior,alt\_hom\_posterior/sum\_posterior,somatic\_posterior/sum\_posterior)

\*/

double \*yyx\_calc\_posterior(const double \*log10\_prior\_vec, const double \*log10\_likelihood\_vec, double \*tmp\_four\_double\_vec, double base\_change\_relative\_rate)

{

double alt\_hom\_log10\_posterior = log10\_prior\_vec[2] + log10\_likelihood\_vec[2];

double ref\_hom\_log10\_posterior = log10\_prior\_vec[0] + log10\_likelihood\_vec[0];

double log10\_posterior\_normalization\_constant = log10\_add\_pow10(alt\_hom\_log10\_posterior, ref\_hom\_log10\_posterior);

double het\_log10\_posterior = log10\_prior\_vec[1] + log10\_likelihood\_vec[1];

log10\_posterior\_normalization\_constant = log10\_add\_pow10(log10\_posterior\_normalization\_constant, het\_log10\_posterior);

double mosaic\_log10\_posterior = log10\_prior\_vec[3] + log10\_likelihood\_vec[3] + log10(base\_change\_relative\_rate);

log10\_posterior\_normalization\_constant = log10\_add\_pow10(log10\_posterior\_normalization\_constant, mosaic\_log10\_posterior);

tmp\_four\_double\_vec[0] = ref\_hom\_log10\_posterior - log10\_posterior\_normalization\_constant;

tmp\_four\_double\_vec[1] = het\_log10\_posterior - log10\_posterior\_normalization\_constant;

tmp\_four\_double\_vec[2] = alt\_hom\_log10\_posterior - log10\_posterior\_normalization\_constant;

tmp\_four\_double\_vec[3] = mosaic\_log10\_posterior - log10\_posterior\_normalization\_constant;

return tmp\_four\_double\_vec;

}

/\* end of yyx\_calc\_posterior() \*/

/\*\*

\* @brief find out base has highest pop\_AF excluding the base\_excluded

\*/

char get\_max\_AF\_base(const double \*population\_ACGT\_allele\_frequency, char base\_excluded){

double max\_AF = -10;

char max\_AF\_base = '.';

int bi;

for(bi=0; bi<4; bi++){

if(idx2base[bi]!=base\_excluded && population\_ACGT\_allele\_frequency[bi]>max\_AF){

max\_AF = population\_ACGT\_allele\_frequency[bi];

max\_AF\_base = idx2base[bi];

}

}

return max\_AF\_base;

}

/\* end of get\_max\_AF\_base() \*/

/\*\*

\* @brief find out which base has higher pop\_AF

\* (output base1 when equal)

\*/

char get\_higher\_AF\_base(const double \*population\_ACGT\_allele\_frequency, char base1, char base2){

if(population\_ACGT\_allele\_frequency[base2idx(base1)]>=population\_ACGT\_allele\_frequency[base2idx(base2)]){

return base1;

}else{

return base2;

}

}

/\* end of get\_higher\_AF\_base() \*/

/\*\*

\* @brief the process\_func() for individual genotyper

\*

\* input must be 1(genotype\_log10lik) + 1(prior\_population\_allele\_frequency) = 2 strings

\*

\* read in data, construct CPD, calculate joint distribution (posterior) , and finally marginalize

\*/

int individual\_genotyper\_process\_func(int num\_str, const char \*\*each\_str)

{

if(num\_str!=2){

fprintf(stderr, "Error: the num\_str for individual\_genotyper\_process\_func() must be 2\n");

return -1;

}

// test\_process\_func(num\_str, each\_str);

char ID[1000] = "";

char chr\_pos[1000] = "";

char \*ID\_ptr;

char \*chr;

char \*pos\_str;

char \*refBase;

char \*altBase;

double genotype\_log10lik[4];

char input\_ref\_alt\_base[10];

char ref\_alt\_base[10];

char chr\_type[10];

double pop\_ACGT\_AF[5];

// above four: [0] for ref-hom, [1] for het, [2] for alt-hom, [3] for mosaic

int i;

double posterior\_log10[4];

// read in each\_str[0]: genotype\_log10lik

if(strcmp(each\_str[0],"")==0){

genotype\_log10lik[0] = 1;

genotype\_log10lik[1] = 1;

genotype\_log10lik[2] = 1;

genotype\_log10lik[3] = 1;

strcpy(input\_ref\_alt\_base, "..");

}else{

sscanf(each\_str[0], "%s\t%lf\t%lf\t%lf\t%lf", ID, &genotype\_log10lik[0], &genotype\_log10lik[1], &genotype\_log10lik[2], &genotype\_log10lik[3]);

ID\_ptr = ID;

chr = strsep(&ID\_ptr, chr\_pos\_delim);

pos\_str = strsep(&ID\_ptr, chr\_pos\_delim);

refBase = strsep(&ID\_ptr, chr\_pos\_delim);

altBase = strsep(&ID\_ptr, chr\_pos\_delim);

strcpy(chr\_pos, chr);

strcat(chr\_pos, chr\_pos\_delim);

strcat(chr\_pos, pos\_str);

input\_ref\_alt\_base[0] = refBase[0];

input\_ref\_alt\_base[1] = altBase[0];

input\_ref\_alt\_base[2] = '\0';

}

// read in each\_str[1]: prior\_population\_allele\_frequency

if(strcmp(each\_str[1],"")==0){

if(strcmp(chr\_pos, "")==0){

fprintf(stderr, "Warning: no chr\_pos info, and no valid ID for chr\_pos, I just skipped it\n");

return 0;

}

strcpy(chr\_type, "A");

pop\_ACGT\_AF[0] = -2;

pop\_ACGT\_AF[1] = -2;

pop\_ACGT\_AF[2] = -2;

pop\_ACGT\_AF[3] = -2;

pop\_ACGT\_AF[4] = -10;

// assign prior's ref base to individual's ref

if(input\_ref\_alt\_base[0]!='.'){

pop\_ACGT\_AF[base2idx(input\_ref\_alt\_base[0])] = 1;

fprintf(stderr, "Warning: no prior info for chr\_pos = '%s', and I judge ref = individual's ref '%c'\n", chr\_pos, input\_ref\_alt\_base[0]);

}else{

fprintf(stderr, "Warning: no prior info and without any known base for chr\_pos = '%s', I just skip it\n", chr\_pos);

return 0;

}

}else{

sscanf(each\_str[1], "%s\t%s\t%lf\t%lf\t%lf\t%lf", chr\_pos, chr\_type, &pop\_ACGT\_AF[0], &pop\_ACGT\_AF[1], &pop\_ACGT\_AF[2], &pop\_ACGT\_AF[3]);

if(chr\_type[0]!='A' && chr\_type[0]!='X' && chr\_type[0]!='Y'){

fprintf(stderr, "Warning: cannot recognize chr\_type '%c', I just treat it 'A' (autosome)\n", chr\_type[0]);

strcpy(chr\_type, "A");

}

pop\_ACGT\_AF[4] = -10;

}

// assign ref/alt base if necessary (troublesome)

if(input\_ref\_alt\_base[0]=='.'){ // no individual's ref/alt base

ref\_alt\_base[0] = get\_max\_AF\_base(pop\_ACGT\_AF, '.');

ref\_alt\_base[1] = get\_max\_AF\_base(pop\_ACGT\_AF, ref\_alt\_base[0]);

}else if(input\_ref\_alt\_base[1]=='.'){ // only individual's ref base, no alt base

ref\_alt\_base[0] = input\_ref\_alt\_base[0];

ref\_alt\_base[1] = get\_max\_AF\_base(pop\_ACGT\_AF, ref\_alt\_base[0]);

}else{ // we know individual's ref/alt base

ref\_alt\_base[0] = input\_ref\_alt\_base[0];

ref\_alt\_base[1] = input\_ref\_alt\_base[1];

}

ref\_alt\_base[2] = '\0';

// calculate log10 prior

normalize\_population\_ACGT\_allele\_frequency(pop\_ACGT\_AF, minus\_1\_AF, minus\_2\_AF);

double prior\_log10[4];

individual\_log10\_prior(chr\_type, individual\_sex[0], ref\_alt\_base, pop\_ACGT\_AF, mosaic\_rate, prior\_log10);

// parse chr\_pos, and select the corresponding base\_change\_relative\_rate

int row\_idx = base2idx(ref\_alt\_base[0]);

int col\_idx = base2idx(ref\_alt\_base[1]);

double base\_change\_relative\_rate = 1;

if(row\_idx>3 || col\_idx>3){

fprintf(stderr, "Warning: refBase(%c) or altBase(%c) not in {A,C,G,T} for chr\_pos = '%s', so I just set relative\_rate=1\n", ref\_alt\_base[0], ref\_alt\_base[1], chr\_pos);

}else{

base\_change\_relative\_rate = base\_change\_matrix[row\_idx\*4+col\_idx];

}

// compute posterior by multiplying prior and likelihood

yyx\_calc\_posterior(prior\_log10, genotype\_log10lik, posterior\_log10, base\_change\_relative\_rate);

// output format: ID(chr\_pos) + chr\_type + refBase/altBase + 4 genotype posterior

printf("%s\t%s", chr\_pos, chr\_type);

// ref/alt base

printf("\t%c/%c", ref\_alt\_base[0], ref\_alt\_base[1]);

if(strcmp(ref\_alt\_base, input\_ref\_alt\_base)!=0){

printf("(%c/%c)", input\_ref\_alt\_base[0], input\_ref\_alt\_base[1]);

}

if(individual\_sex[0]=='F' && chr\_type[0]=='Y'){

printf("\t0\t0\t0\t0");

}else{

for(i=0; i<4; i++){ // genotype

if(posterior\_log10[i] < -1e8){

printf("\t%.2e", posterior\_log10[i]);

}else{

printf("\t%.5f", posterior\_log10[i]);

}

}

}

printf("\n");

return 0;

}

/\* end of individual\_genotyper\_process\_func() \*/

/\*\*

\* @brief a wrapped framework to process matched line from sorted files

\*

\* open and read in each line from several sorted files, cut the first column as ID,

\* compare ID from each file (by compare\_two\_ID() ),

\* get the min\_ID and put "" to those unmatched,

\* send these strings (may contain '\n') to process\_func() for computing and outputing

\*

\* If process\_func() returns not 0, show warning and proceed when process\_error\_should\_stop=0

\* or stop, report error when process\_error\_should\_stop=1

\*/

int match\_and\_process\_firstColumnSorted\_files(int num\_files, char \*\*filenames, int (\*compare\_two\_ID)(const char \*IDa, const char \*IDb), int (\*process\_func)(int num\_str, const char \*\*each\_str), int process\_error\_should\_stop )

{

size\_t malloc\_line\_len = LINE\_LEN;

const char \*null\_str = "";

FILE \*\*fileHandles;

if((fileHandles = malloc((num\_files)\*sizeof(FILE \*)))==NULL){

fprintf(stderr, "Error: cannot allocate memory for fileHandles at %d\n", \_\_LINE\_\_);

return 1;

}

int fi\_0, fi\_1;

int has\_error = 0;

for(fi\_0=0; fi\_0<num\_files; fi\_0++){

if((fileHandles[fi\_0] = fopen(filenames[fi\_0], "r"))==NULL){

has\_error = 1;

fprintf(stderr, "Error: cannot open file '%s' for input\n", filenames[fi\_0]);

// break;

}

}

int \*should\_read\_next\_line = malloc((num\_files)\*sizeof(int));

char \*\*input\_each\_str = malloc((num\_files)\*sizeof(char\*));

int input\_str\_len;

char \*\*each\_str = malloc((num\_files)\*sizeof(char\*));

char \*\*each\_ID\_str = malloc((num\_files)\*sizeof(char\*));

if(should\_read\_next\_line==NULL || each\_str==NULL || input\_each\_str==NULL || each\_ID\_str==NULL){

fprintf(stderr, "Error: cannot allocate memory for should\_read\_next\_line, each\_str, input\_each\_str or each\_ID\_str at %d\n", \_\_LINE\_\_);

has\_error = 1;

}

if(!has\_error){

for(fi\_1=0; fi\_1<num\_files; fi\_1++){

should\_read\_next\_line[fi\_1] = 1;

if((input\_each\_str[fi\_1] = malloc((malloc\_line\_len)\*sizeof(char)))==NULL){

has\_error = 1;

fprintf(stderr, "Error: cannot allocate memory for input\_each\_str[%d]\n", fi\_1);

// break;

}

if((each\_ID\_str[fi\_1] = malloc((malloc\_line\_len)\*sizeof(char)))==NULL){

has\_error = 1;

fprintf(stderr, "Error: cannot allocate memory for each\_ID\_str[%d]\n", fi\_1);

// break;

}

}

char \*min\_ID\_str;

int not\_all\_eof = 1;

int process\_return\_value;

if(!has\_error){

while(not\_all\_eof){

// read in new line

for(fi\_1=0; fi\_1<num\_files; fi\_1++){

if(should\_read\_next\_line[fi\_1]==1){

if(strcmp(each\_ID\_str[fi\_1], YYX\_EOF\_STR)==0){

continue;

}

while((input\_str\_len = getline(&input\_each\_str[fi\_1], &malloc\_line\_len, fileHandles[fi\_1]))!=-1){

if(input\_str\_len>0 && strcmp(input\_each\_str[fi\_1], "")!=0 && strcmp(input\_each\_str[fi\_1], "\n")!=0){

break;

}

}

if(input\_str\_len==-1){

strcpy(each\_ID\_str[fi\_1], YYX\_EOF\_STR);

}else{

sscanf(input\_each\_str[fi\_1], "%s", each\_ID\_str[fi\_1]);

}

}

}

// get min\_ID\_str

min\_ID\_str = each\_ID\_str[0];

for(fi\_1=1; fi\_1<num\_files; fi\_1++){

if(compare\_two\_ID(each\_ID\_str[fi\_1],min\_ID\_str)<0){

min\_ID\_str = each\_ID\_str[fi\_1];

}

}

if(strcmp(min\_ID\_str, YYX\_EOF\_STR)==0){

not\_all\_eof = 0;

continue;

}

// determine each\_str and should\_read\_next\_line

for(fi\_1=0; fi\_1<num\_files; fi\_1++){

if(compare\_two\_ID(each\_ID\_str[fi\_1],min\_ID\_str)==0){

each\_str[fi\_1] = input\_each\_str[fi\_1];

should\_read\_next\_line[fi\_1] = 1;

}else{

each\_str[fi\_1] = (char \*)null\_str;

should\_read\_next\_line[fi\_1] = 0;

}

}

// call process\_func

process\_return\_value = (\*process\_func)(num\_files, (const char \*\*)each\_str);

if(process\_return\_value!=0){

if(process\_error\_should\_stop){

fprintf(stderr, "Error: process\_func returns %d\n", process\_return\_value);

break;

}else{

fprintf(stderr, "Warning: process\_func returns %d\n", process\_return\_value);

}

}

}

}

for(fi\_0=num\_files-1;fi\_0>=0;fi\_0--){

if(each\_ID\_str[fi\_0]!=NULL){

free(each\_ID\_str[fi\_0]);

}

}

}

free(should\_read\_next\_line);

free(each\_str);

free(input\_each\_str);

free(each\_ID\_str);

for(fi\_0=num\_files-1;fi\_0>=0;fi\_0--){

if(fileHandles[fi\_0]!=NULL){

fclose(fileHandles[fi\_0]);

}

}

free(fileHandles);

if(!has\_error){

return 0;

}else{

return -1;

}

}

/\* end of match\_and\_process\_firstColumnSorted\_files() \*/

int main(int argc, char\* argv[])

{

char usage\_str[10000];

sprintf(usage\_str, "\nUsage: %s <chr\_pos\_delim> <chr\_order.list> \n", argv[0]);

strcat(usage\_str, " <genotype\_log10lik> <prior\_allele\_frequency>\n");

strcat(usage\_str, " <individual\_sex> [mosaic\_rate] [minus\_1\_AF] [minus\_2\_AF] [base\_change\_relative\_rate]\n");

strcat(usage\_str, "\n");

strcat(usage\_str, " \* This program is just used to compute individual genotype posterior\n");

strcat(usage\_str, " based on 2 files: individual genotype likelihood and prior,\n");

strcat(usage\_str, " which is pretty simple, and to suit the recent framework\n");

strcat(usage\_str, " \* <base\_change\_relative\_rate> set the relative rate of base change\n");

strcat(usage\_str, " relative to mosaic rate\n");

strcat(usage\_str, " \* <genotype\_log10lik> and <log10\_prior> should be sorted\n");

strcat(usage\_str, " on the first column ID(chr:pos:refBase:altBase)\n");

strcat(usage\_str, "\n");

strcat(usage\_str, " <chr\_pos\_delim> : the char in ID separate chr and pos, eg. ':'\n");

strcat(usage\_str, " use \"\" for just strcmp on ID\n");

strcat(usage\_str, "\n");

strcat(usage\_str, " <chr\_order.list> : a file contains chr order\n");

strcat(usage\_str, " only consider the first column\n");

strcat(usage\_str, "\n");

strcat(usage\_str, " <genotype\_log10lik> format:\n");

strcat(usage\_str, " ID(chr:pos:refBase:altBase) + log10 likelihood P(o|G)\n");

strcat(usage\_str, " for genotype (ref-hom, het, alt-hom, mosaic)\n");

strcat(usage\_str, " for each site on each line,\n");

strcat(usage\_str, " can be generated by 'Yyx\_genotype\_log10lik\_with\_precalc\_beta'\n");

strcat(usage\_str, "\n");

strcat(usage\_str, " <prior\_allele\_frequency> format: (6 columns)\n");

strcat(usage\_str, " ID(chr:pos) + A/X/Y (autosome/X/Y)\n");

strcat(usage\_str, " + population allele frequency (A,C,G,T)\n");

strcat(usage\_str, " for each site on each line,\n");

strcat(usage\_str, "\n");

strcat(usage\_str, " <individual\_sex>: individual's sex M/F (male/female)\n");

strcat(usage\_str, "\n");

strcat(usage\_str, " [mosaic\_rate]: mosaic rate\n");

strcat(usage\_str, " the probability to see a site is a mosaic site in an individual\n");

strcat(usage\_str, " [ default = 1e-7 ]\n");

strcat(usage\_str, "\n");

strcat(usage\_str, " [minus\_1\_AF]: allele frequency for code '-1'\n");

strcat(usage\_str, " the allele frequency for that in dbSNP but no MAF information\n");

strcat(usage\_str, " [ default = 0.002 (1/500) ]\n");

strcat(usage\_str, "\n");

strcat(usage\_str, " [minus\_2\_AF]: allele frequency for code '-2'\n");

strcat(usage\_str, " the allele frequency for that not found in dbSNP\n");

strcat(usage\_str, " [ default = 1e-4 ]\n");

strcat(usage\_str, "\n");

strcat(usage\_str, " [base\_change\_relative\_rate] format:\n");

strcat(usage\_str, " 4 \* 4 table ('\\t' separated),\n");

strcat(usage\_str, " the relative rate of change occurance,\n");

strcat(usage\_str, " relative to the mosaic prior\n");

strcat(usage\_str, " from row(A,C,G,T) to col(A,C,G,T)\n");

strcat(usage\_str, " [ default: all = 1 ]\n");

strcat(usage\_str, "\n");

strcat(usage\_str, " output (stdout) format: (7 columns, separated by '\\t')\n");

strcat(usage\_str, " ID(chr:pos) + A/X/Y (autosome/X/Y) + refBase/altBase\n");

strcat(usage\_str, " log10 posterior (ref-hom, het, alt-hom, mosaic)\n");

strcat(usage\_str, " for each site on each line\n");

strcat(usage\_str, "\n");

strcat(usage\_str, "Version: 0.2.0 (2014-06-09)\n");

strcat(usage\_str, "Author: Adam Yongxin Ye @ CBI\n");

strcat(usage\_str, "\n");

char \*chr\_order\_filename;

char \*base\_change\_filename = NULL;

char \*\*filenames;

if(argc<5+1 || argc>9+1){

fprintf(stderr, "%s", usage\_str);

return 1;

}

chr\_pos\_delim = argv[1];

chr\_pos\_delim[1] = '\0';

chr\_order\_filename = argv[2];

filenames = argv+3;

if(argc>5){

sscanf(argv[5], "%s", individual\_sex);

if(strcmp(individual\_sex,"M")!=0 && strcmp(individual\_sex,"F")!=0){

fprintf(stderr, "Error: cannot recognize individual's sex = '%c'\n", individual\_sex[0]);

return 2;

}

}

if(argc>6){ sscanf(argv[6], "%lf", &mosaic\_rate); }

if(argc>7){ sscanf(argv[7], "%lf", &minus\_1\_AF); }

if(argc>8){ sscanf(argv[8], "%lf", &minus\_2\_AF); }

if(argc>9){ base\_change\_filename = argv[9];}

int has\_error = 0;

size\_t tmp\_line\_len = LINE\_LEN;

char \*tmp\_line = NULL;

if((tmp\_line = malloc((tmp\_line\_len)\*sizeof(char)))==NULL){

fprintf(stderr, "Error: cannot allocate memory for tmp\_line\n");

has\_error = 1;

}

int return\_value = -1;

// read in base\_change file

// double base\_change\_matrix[16]; // row\_idx \* 4 + col\_idx, row and col 0-A,1-C,2-G,3-T

if(base\_change\_filename != NULL){

FILE \*base\_change\_fileHandle;

char \*ptr\_for\_strsep;

if(!has\_error){

if((base\_change\_fileHandle = fopen(base\_change\_filename, "r"))==NULL){

fprintf(stderr, "Error: cannot open file '%s' for input\n", base\_change\_filename);

has\_error = 1;

}else{

int row\_idx = 0;

int col\_idx;

while((getline(&tmp\_line, &tmp\_line\_len, base\_change\_fileHandle))!=-1){

if(strcmp(tmp\_line, "")==0){

continue;

}

ptr\_for\_strsep = tmp\_line;

for(col\_idx=0; col\_idx<4; col\_idx++){

base\_change\_matrix[row\_idx\*4+col\_idx] = atof(ptr\_for\_strsep);

strsep(&ptr\_for\_strsep, "\t");

}

row\_idx++;

if(row\_idx>=4){

break;

}

}

fclose(base\_change\_fileHandle);

}

}

}else{ // base\_change\_file not specified, so use default: all = 1

int row\_idx, col\_idx;

for(row\_idx=0; row\_idx<4; row\_idx++){

for(col\_idx=0; col\_idx<4; col\_idx++){

base\_change\_matrix[row\_idx\*4+col\_idx] = 1;

}

}

}

// read in chr\_order file

if(!has\_error){

FILE \*chr\_order\_fileHandle;

if((chr\_order\_fileHandle = fopen(chr\_order\_filename, "r"))==NULL){

fprintf(stderr, "Error: cannot open file '%s' for input\n", chr\_order\_filename);

has\_error = 1;

}

if(!has\_error){

while((getline(&tmp\_line, &tmp\_line\_len, chr\_order\_fileHandle))!=-1){

if(strcmp(tmp\_line, "")==0){

continue;

}

if((chr\_order[chr\_order\_len] = malloc((tmp\_line\_len)\*sizeof(char)))==NULL){

fprintf(stderr, "Error: cannot allocate memory for chr\_order[%d]\n", chr\_order\_len);

has\_error = 1;

break;

}

sscanf(tmp\_line, "%s", chr\_order[chr\_order\_len]);

chr\_order\_len++;

}

fclose(chr\_order\_fileHandle);

if(!has\_error){

return\_value = match\_and\_process\_firstColumnSorted\_files(2, filenames, compare\_two\_ID\_chr\_pos, individual\_genotyper\_process\_func, 1 );

}

for(chr\_order\_len--; chr\_order\_len>=0; chr\_order\_len--){

free(chr\_order[chr\_order\_len]);

}

}else{

}

}else{

}

if(tmp\_line != NULL){

free(tmp\_line);

tmp\_line = NULL;

}

return return\_value;

}

1-3. Yyx\_trio\_genotyper.v0\_2\_2.c

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Copyright (C) 2013 Center for Bioinformatics, Peking University, China

\*

\* This program is used to integrate trio genotype likelihood,

\* parents' prior, and parents to child CPD (support 4 bases),

\* to get joint posterior distribution

\* and then marginalize to get everyone's genotype posterior

\* Input includes 3 genotype log10lik files, 2 parents' mosaic AF, and prior

\* all 6 files should be sorted on the first column ID(chr:pos...)

\*

\* Version: 0.2.2 (2014-05-20)\n");

\* Author: Adam Yongxin Ye @ CBI\n");

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <stdlib.h>

#include <stdio.h>

#include <math.h>

#include <string.h>

#include <assert.h>

#include <ctype.h>

#include <float.h>

#include <unistd.h>

#define MIN(X,Y) ((X) < (Y) ? (X) : (Y))

#define MAX(X,Y) ((X) > (Y) ? (X) : (Y))

#define POW10(x) (pow(10.0, x))

#define LOGZERO -1e100

#define LINE\_LEN 1000

#define YYX\_EOF\_STR "EOF\_Yyx\_EOF\_Yyx\_EOF"

double de\_novo\_rate = 1e-8;

double mosaic\_rate = 1e-7;

double minus\_1\_AF = 0.002;

double minus\_2\_AF = 1e-4;

char \*chr\_pos\_delim;

char \*chr\_order[1000];

int chr\_order\_len = 0;

char child\_sex[5];

double log10\_add\_pow10(double a, double b);

int compare\_two\_ID\_chr\_pos(const char \*IDa, const char \*IDb);

int base2idx(char base);

double \*parent\_to\_child\_CPD\_log10lik(const char \*chr\_type, const char \*father\_ref\_alt\_base, const char \*mother\_ref\_alt\_base, const char \*child\_ref\_alt\_base, double father\_mosaic\_AF, double mother\_mosaic\_AF, double de\_novo\_rate, double mosaic\_rate, double \*tmp\_64\_double\_vec);

double \*normalize\_population\_ACGT\_allele\_frequency(double \*population\_ACGT\_allele\_frequency, double minus\_1\_AF, double minus\_2\_AF);

double \*parent\_log10\_prior(const char \*chr\_type, char parent\_sex, const char \*parent\_ref\_alt\_base, const double \*population\_ACGT\_allele\_frequency, double mosaic\_rate, double \*tmp\_4\_double\_vec);

char get\_max\_AF\_base(const double \*population\_ACGT\_allele\_frequency, char base\_excluded);

char get\_higher\_AF\_base(const double \*population\_ACGT\_allele\_frequency, char base1, char base2);

int trio\_genotyper\_process\_func(int num\_str, const char \*\*each\_str);

int match\_and\_process\_firstColumnSorted\_files(int num\_files, char \*\*filenames, int (\*compare\_two\_ID)(const char \*IDa, const char \*IDb), int (\*process\_func)(int num\_str, const char \*\*each\_str), int process\_error\_should\_stop );

/\*\*

\* @brief Computes log10(10^(a) + 10^(b))

\*/

double log10\_add\_pow10(double a, double b)

{

/\* in R code:

tmp\_max <- max(vec)

log10(sum(10^(vec-tmp\_max))) + tmp\_max

\*/

if (a > b) {

return a + log10(1+POW10(b-a));

} else {

return b + log10(1+POW10(a-b));

}

}

/\* end of log10\_add\_pow10() \*/

/\*\*

\* @brief The compare function for sorting ID(chr\_pos)

\*

\* compare ID, make YYX\_EOF\_STR the largest

\* if ID has '\_', cut as chr\_pos, str\_cmp for chr and int\_cmp for pos

\* otherwise, str\_cmp ID

\*/

int compare\_two\_ID\_chr\_pos(const char \*IDa, const char \*IDb)

{

char modifiable\_IDa[LINE\_LEN];

char modifiable\_IDb[LINE\_LEN];

char \*chr\_a;

char \*chr\_b;

char \*pos\_str\_a;

char \*pos\_str\_b;

long pos\_a, pos\_b;

int chr\_idx\_a, chr\_idx\_b;

strcpy(modifiable\_IDa, IDa);

strcpy(modifiable\_IDb, IDb);

if(strcmp(IDa, YYX\_EOF\_STR)==0){

if(strcmp(IDb, YYX\_EOF\_STR)==0){

return 0;

}else{ // IDb != YYX\_EOF\_STR

return +1; // make YYX\_EOF\_STR the largest

}

}else{ // IDa != YYX\_EOF\_STR

if(strcmp(IDb, YYX\_EOF\_STR)==0){

return -1; // make YYX\_EOF\_STR the largest

}else{ // IDb != YYX\_EOF\_STR

if(strcmp(chr\_pos\_delim, "")!=0){

pos\_str\_a = modifiable\_IDa;

chr\_a = strsep(&pos\_str\_a, chr\_pos\_delim);

pos\_a = 0;

if(pos\_str\_a == NULL){

fprintf(stderr, "Warning: something (chr\_pos\_ID) may be wrong in compare\_two\_ID\_chr\_pos() with input IDa='%s'\n", IDa);

}else{

sscanf(pos\_str\_a, "%ld", &pos\_a);

}

pos\_str\_b = modifiable\_IDb;

chr\_b = strsep(&pos\_str\_b, chr\_pos\_delim);

pos\_b = 0;

if(pos\_str\_b == NULL){

fprintf(stderr, "Warning: something (chr\_pos\_ID) may be wrong in compare\_two\_ID\_chr\_pos() with input IDb='%s'\n", IDb);

}else{

sscanf(pos\_str\_b, "%ld", &pos\_b);

}

if(strcmp(chr\_a,chr\_b)==0){

return pos\_a - pos\_b;

}else{

for(chr\_idx\_a=0; chr\_idx\_a<chr\_order\_len; chr\_idx\_a++){

if(strcmp(chr\_a, chr\_order[chr\_idx\_a])==0){

break;

}

}

for(chr\_idx\_b=0; chr\_idx\_b<chr\_order\_len; chr\_idx\_b++){

if(strcmp(chr\_b, chr\_order[chr\_idx\_b])==0){

break;

}

}

return chr\_idx\_a - chr\_idx\_b;

}

}else{

return strcmp(IDa, IDb);

}

}

}

}

/\* end of compare\_two\_ID\_chr\_pos() \*/

/\*\*

\* @brief a test process\_func()

\*/

int test\_process\_func(int num\_str, const char \*\*each\_str)

{

int fi;

for(fi=0; fi<num\_str; fi++){

printf("%d\t%s\n", fi, each\_str[fi]);

}

printf("\n");

return 0;

}

/\* end of test\_process\_func() \*/

/\*\*

\* @brief array idx2base[] and function base2idx()

\*/

const char \*idx2base = "ACGT.";

int base2idx(char base){

if(base=='A'){

return 0;

}else if(base=='C'){

return 1;

}else if(base=='G'){

return 2;

}else if(base=='T'){

return 3;

}else{

return 4;

}

}

/\* end of base2idx() \*/

/\*\*

\* @brief Calculate CPD log10lik given chr\_type, child\_sex, father, mother's ref/alt base and mosaic AF

\*

\* Output: tmp\_64\_double\_vec[i\*16+j\*4+k] = P(G\_child=k | G\_father=i, G\_mother=j)

\* i for father's genotype, j for mother's genotype, and k for child's genotype

\* 0=ref-hom, 1=het, 2=alt-hom, 3=mosaic

\*/

double \*parent\_to\_child\_CPD\_log10lik(const char \*chr\_type, const char \*father\_ref\_alt\_base, const char \*mother\_ref\_alt\_base, const char \*child\_ref\_alt\_base, double father\_mosaic\_AF, double mother\_mosaic\_AF, double de\_novo\_rate, double mosaic\_rate, double \*tmp\_64\_double\_vec)

{

// chr\_type should be "A", "X", or "Y"

// all three \*\_\_ref\_alt\_base are two-char strings (without '\0')

// child\_sex should be "M" or "F"

// construct CPD from parents to child

int i,j,k; // i,j,k each 0-3, for father, mother, child's genotype

int bi, bj; // bi: base\_idx 0-3

// tmp\_64\_double\_vec[i\*16+j\*4+k] for CPD log10lik

int father\_ref\_base\_idx = base2idx(father\_ref\_alt\_base[0]);

int father\_alt\_base\_idx = base2idx(father\_ref\_alt\_base[1]);

int mother\_ref\_base\_idx = base2idx(mother\_ref\_alt\_base[0]);

int mother\_alt\_base\_idx = base2idx(mother\_ref\_alt\_base[1]);

int child\_ref\_base\_idx = base2idx(child\_ref\_alt\_base[0]);

int child\_alt\_base\_idx = base2idx(child\_ref\_alt\_base[1]);

double father\_base\_prob[16];

double mother\_base\_prob[16];

// above two parents' genotype to allele/base [i\*4+bi] [j\*4+bj]

i = 0; // father ref-hom

for(bi=0; bi<4; bi++){

if(bi==father\_ref\_base\_idx){

father\_base\_prob[i\*4+bi] = 1 - de\_novo\_rate;

}else{

father\_base\_prob[i\*4+bi] = de\_novo\_rate / 3;

}

}

i = 1; // father het

for(bi=0; bi<4; bi++){

if(bi==father\_ref\_base\_idx || bi==father\_alt\_base\_idx){

father\_base\_prob[i\*4+bi] = (double)(1) / 2 - de\_novo\_rate / 3;

}else{

father\_base\_prob[i\*4+bi] = de\_novo\_rate / 3;

}

}

i = 2; // father alt-hom

for(bi=0; bi<4; bi++){

if(bi==father\_alt\_base\_idx){

father\_base\_prob[i\*4+bi] = 1 - de\_novo\_rate;

}else{

father\_base\_prob[i\*4+bi] = de\_novo\_rate / 3;

}

}

i = 3; // father mosaic

for(bi=0; bi<4; bi++){

if(bi==father\_ref\_base\_idx){

father\_base\_prob[i\*4+bi] = (1 - father\_mosaic\_AF) \* (1 - de\_novo\_rate / 3 \* 2);

}else if(bi==father\_alt\_base\_idx){

father\_base\_prob[i\*4+bi] = father\_mosaic\_AF \* (1 - de\_novo\_rate / 3 \* 2);

}else{

father\_base\_prob[i\*4+bi] = de\_novo\_rate / 3;

}

}

i = 0; // mother ref-hom

for(bi=0; bi<4; bi++){

if(bi==mother\_ref\_base\_idx){

mother\_base\_prob[i\*4+bi] = 1 - de\_novo\_rate;

}else{

mother\_base\_prob[i\*4+bi] = de\_novo\_rate / 3;

}

}

i = 1; // mother het

for(bi=0; bi<4; bi++){

if(bi==mother\_ref\_base\_idx || bi==mother\_alt\_base\_idx){

mother\_base\_prob[i\*4+bi] = (double)(1) / 2 - de\_novo\_rate / 3;

}else{

mother\_base\_prob[i\*4+bi] = de\_novo\_rate / 3;

}

}

i = 2; // mother alt-hom

for(bi=0; bi<4; bi++){

if(bi==mother\_alt\_base\_idx){

mother\_base\_prob[i\*4+bi] = 1 - de\_novo\_rate;

}else{

mother\_base\_prob[i\*4+bi] = de\_novo\_rate / 3;

}

}

i = 3; // mother mosaic

for(bi=0; bi<4; bi++){

if(bi==mother\_ref\_base\_idx){

mother\_base\_prob[i\*4+bi] = (1 - mother\_mosaic\_AF) \* (1 - de\_novo\_rate / 3 \* 2);

}else if(bi==mother\_alt\_base\_idx){

mother\_base\_prob[i\*4+bi] = mother\_mosaic\_AF \* (1 - de\_novo\_rate / 3 \* 2);

}else{

mother\_base\_prob[i\*4+bi] = de\_novo\_rate / 3;

}

}

// gamate combination

double normalize\_non\_mosaic\_constant;

if(chr\_type[0]=='A' || (chr\_type[0]=='X' && child\_sex[0]=='F') ){ // autosome or (chrX and child==female)

for(i=0; i<4; i++){ // father's genotype

for(j=0; j<4; j++){ // mother's genotype

k = 0; // child's genotype = ref-hom

bi = child\_ref\_base\_idx;

bj = child\_ref\_base\_idx;

tmp\_64\_double\_vec[i\*16+j\*4+k] = log10(father\_base\_prob[i\*4+bi])+log10(mother\_base\_prob[j\*4+bj]);

normalize\_non\_mosaic\_constant = tmp\_64\_double\_vec[i\*16+j\*4+k];

k = 1; // child's genotype = het

bj = child\_alt\_base\_idx;

tmp\_64\_double\_vec[i\*16+j\*4+k] = log10\_add\_pow10(log10(father\_base\_prob[i\*4+bi])+log10(mother\_base\_prob[j\*4+bj]),log10(father\_base\_prob[i\*4+bj])+log10(mother\_base\_prob[j\*4+bi]));

normalize\_non\_mosaic\_constant = log10\_add\_pow10(normalize\_non\_mosaic\_constant, tmp\_64\_double\_vec[i\*16+j\*4+k]);

k = 2; // child's genotype = alt-hom

bi = child\_alt\_base\_idx;

tmp\_64\_double\_vec[i\*16+j\*4+k] = log10(father\_base\_prob[i\*4+bi])+log10(mother\_base\_prob[j\*4+bj]);

normalize\_non\_mosaic\_constant = log10\_add\_pow10(normalize\_non\_mosaic\_constant, tmp\_64\_double\_vec[i\*16+j\*4+k]);

// normalize non-mosaic genotype, so that sum of P(G\_child|G\_father,G\_mother) = 1 - mosaic rate

normalize\_non\_mosaic\_constant = log10(1-mosaic\_rate) - normalize\_non\_mosaic\_constant;

for(k=0; k<3; k++){

tmp\_64\_double\_vec[i\*16+j\*4+k] += normalize\_non\_mosaic\_constant;

}

k = 3; // child's genotype = mosaic

tmp\_64\_double\_vec[i\*16+j\*4+k] = log10(mosaic\_rate);

}

}

}else if(chr\_type[0]=='X' && child\_sex[0]=='M'){ // chrX and child==male

for(i=0; i<4; i++){ // father's genotype (useless for chrX and child==male)

for(j=0; j<4; j++){ // mother's genotype

k = 0; // child's genotype = ref-hom (ref-hemizygous)

bj = child\_ref\_base\_idx;

tmp\_64\_double\_vec[i\*16+j\*4+k] = log10(mother\_base\_prob[j\*4+bj]) + log10(1-mosaic\_rate);

normalize\_non\_mosaic\_constant = tmp\_64\_double\_vec[i\*16+j\*4+k];

k = 1; // child's genotype = het (unnormal, prob~=0)

tmp\_64\_double\_vec[i\*16+j\*4+k] = LOGZERO;

normalize\_non\_mosaic\_constant = log10\_add\_pow10(normalize\_non\_mosaic\_constant, tmp\_64\_double\_vec[i\*16+j\*4+k]);

k = 2; // child's genotype = alt-hom (alt-hemizygous)

bj = child\_alt\_base\_idx;

tmp\_64\_double\_vec[i\*16+j\*4+k] = log10(mother\_base\_prob[j\*4+bj]) + log10(1-mosaic\_rate);

normalize\_non\_mosaic\_constant = log10\_add\_pow10(normalize\_non\_mosaic\_constant, tmp\_64\_double\_vec[i\*16+j\*4+k]);

// normalize non-mosaic genotype, so that sum of P(G\_child|G\_father,G\_mother) = 1 - mosaic rate

normalize\_non\_mosaic\_constant = log10(1-mosaic\_rate) - normalize\_non\_mosaic\_constant;

for(k=0; k<3; k++){

tmp\_64\_double\_vec[i\*16+j\*4+k] += normalize\_non\_mosaic\_constant;

}

k = 3; // child's genotype = mosaic

tmp\_64\_double\_vec[i\*16+j\*4+k] = log10(mosaic\_rate);

}

}

}else if(chr\_type[0]=='Y' && child\_sex[0]=='M'){ // chrY and child==male

for(i=0; i<4; i++){ // father's genotype

for(j=0; j<4; j++){ // mother's genotype (useless for chrY and child==male)

k = 0; // child's genotype = ref-hom (ref-hemizygous)

bi = child\_ref\_base\_idx;

tmp\_64\_double\_vec[i\*16+j\*4+k] = log10(father\_base\_prob[i\*4+bi]) + log10(1-mosaic\_rate);

normalize\_non\_mosaic\_constant = tmp\_64\_double\_vec[i\*16+j\*4+k];

k = 1; // child's genotype = het (unnormal, prob~=0)

tmp\_64\_double\_vec[i\*16+j\*4+k] = LOGZERO;

normalize\_non\_mosaic\_constant = log10\_add\_pow10(normalize\_non\_mosaic\_constant, tmp\_64\_double\_vec[i\*16+j\*4+k]);

k = 2; // child's genotype = alt-hom (alt-hemizygous)

bi = child\_alt\_base\_idx;

tmp\_64\_double\_vec[i\*16+j\*4+k] = log10(father\_base\_prob[i\*4+bi]) + log10(1-mosaic\_rate);

normalize\_non\_mosaic\_constant = log10\_add\_pow10(normalize\_non\_mosaic\_constant, tmp\_64\_double\_vec[i\*16+j\*4+k]);

// normalize non-mosaic genotype, so that sum of P(G\_child|G\_father,G\_mother) = 1 - mosaic rate

normalize\_non\_mosaic\_constant = log10(1-mosaic\_rate) - normalize\_non\_mosaic\_constant;

for(k=0; k<3; k++){

tmp\_64\_double\_vec[i\*16+j\*4+k] += normalize\_non\_mosaic\_constant;

}

k = 3; // child's genotype = mosaic

tmp\_64\_double\_vec[i\*16+j\*4+k] = log10(mosaic\_rate);

}

}

}else if(chr\_type[0]=='Y' && child\_sex[0]=='F'){ // chrY and child==female

// for chrY and child==female, all CPD = 1

// thus, if no sequencing data for child, she will got posterior 1:1:1:1, which should be then filtered in the outer calling function

for(i=0; i<4; i++){ // father's genotype

for(j=0; j<4; j++){ // mother's genotype

for(k=0; k<4; k++){ // child's genotype

tmp\_64\_double\_vec[i\*16+j\*4+k] = 1;

}

}

}

}else{

fprintf(stderr, "Warning: unknown chr\_type '%s' in parent\_to\_child\_CPD\_log10lik()\n", chr\_type);

}

return tmp\_64\_double\_vec;

}

/\* end of parent\_to\_child\_CPD\_log10lik() \*/

/\*\*

\* @brief normalize 4 allele frequency, so that they sum to 1

\*

\* input population\_ACGT\_allele\_frequency will be modified

\*

\* in fact, this is used to stablize mosaic\_rate (1e-7)

\*/

double \*normalize\_population\_ACGT\_allele\_frequency(double \*population\_ACGT\_allele\_frequency, double minus\_1\_AF, double minus\_2\_AF)

{

int bi;

for(bi=0; bi<4; bi++){

if(population\_ACGT\_allele\_frequency[bi]>=0){

// good, do nothing

}else if(population\_ACGT\_allele\_frequency[bi]==-1){

population\_ACGT\_allele\_frequency[bi] = minus\_1\_AF;

}else if(population\_ACGT\_allele\_frequency[bi]==-2){

population\_ACGT\_allele\_frequency[bi] = minus\_2\_AF;

}else{

fprintf(stderr, "Warning: unknown ref population allele frequency code '%f', I use '-2' instead\n", population\_ACGT\_allele\_frequency[bi]);

population\_ACGT\_allele\_frequency[bi] = minus\_2\_AF;

}

}

return population\_ACGT\_allele\_frequency;

}

/\* end of normalize\_population\_ACGT\_allele\_frequency() \*/

/\*\*

\* @brief calculate parents log10 prior based on population allele frequency

\*

\* you should first call normalize\_population\_ACGT\_allele\_frequency() to code '-1', '-2' and normalize

\*

\* output is log10 prior for ref-hom (or ref-hemi), het (or LOGZERO), alt-hom (or alt-hemi), mosaic

\*/

double \*parent\_log10\_prior(const char \*chr\_type, char parent\_sex, const char \*parent\_ref\_alt\_base, const double \*population\_ACGT\_allele\_frequency, double mosaic\_rate, double \*tmp\_4\_double\_vec)

{

int parent\_ref\_base\_idx = base2idx(parent\_ref\_alt\_base[0]);

int parent\_alt\_base\_idx = base2idx(parent\_ref\_alt\_base[1]);

double ref\_AF = population\_ACGT\_allele\_frequency[parent\_ref\_base\_idx];

double alt\_AF = population\_ACGT\_allele\_frequency[parent\_alt\_base\_idx];

double log10\_ref\_AF;

log10\_ref\_AF = log10(ref\_AF/(ref\_AF+alt\_AF)); // normalize of the 2 allele/base

double log10\_alt\_AF;

log10\_alt\_AF = log10(alt\_AF/(ref\_AF+alt\_AF)); // normalize of the 2 allele/base

tmp\_4\_double\_vec[3] = log10(mosaic\_rate);

if(chr\_type[0]=='A' || (chr\_type[0]=='X' && parent\_sex=='F') ){ // autosome or (chrX and female

tmp\_4\_double\_vec[0] = 2\*log10\_ref\_AF;

tmp\_4\_double\_vec[1] = log10(2)+log10\_ref\_AF+log10\_alt\_AF;

tmp\_4\_double\_vec[2] = 2\*log10\_alt\_AF;

}else if(chr\_type[0]=='X' && parent\_sex=='M'){ // chrX and male

tmp\_4\_double\_vec[0] = log10\_ref\_AF;

tmp\_4\_double\_vec[1] = LOGZERO;

tmp\_4\_double\_vec[2] = log10\_alt\_AF;

}else if(chr\_type[0]=='Y' && parent\_sex=='M'){ // chrY and male

tmp\_4\_double\_vec[0] = log10\_ref\_AF;

tmp\_4\_double\_vec[1] = LOGZERO;

tmp\_4\_double\_vec[2] = log10\_alt\_AF;

}else if(chr\_type[0]=='Y' && parent\_sex=='F'){ // chrY and female

// for chrY and female, set all prior = 1

// which genotyping results should be then filtered

tmp\_4\_double\_vec[0] = 0;

tmp\_4\_double\_vec[1] = 0;

tmp\_4\_double\_vec[2] = 0;

tmp\_4\_double\_vec[3] = 0;

}else{

fprintf(stderr, "Warning: unknown chr\_type '%s' in parent\_log10\_prior()\n", chr\_type);

}

return tmp\_4\_double\_vec;

}

/\* end of parent\_log10\_prior() \*/

/\*\*

\* @brief find out base has highest pop\_AF excluding the base\_excluded

\*/

char get\_max\_AF\_base(const double \*population\_ACGT\_allele\_frequency, char base\_excluded){

double max\_AF = -10;

char max\_AF\_base = '.';

int bi;

for(bi=0; bi<4; bi++){

if(idx2base[bi]!=base\_excluded && population\_ACGT\_allele\_frequency[bi]>max\_AF){

max\_AF = population\_ACGT\_allele\_frequency[bi];

max\_AF\_base = idx2base[bi];

}

}

return max\_AF\_base;

}

/\* end of get\_max\_AF\_base() \*/

/\*\*

\* @brief find out which base has higher pop\_AF

\* (output base1 when equal)

\*/

char get\_higher\_AF\_base(const double \*population\_ACGT\_allele\_frequency, char base1, char base2){

if(population\_ACGT\_allele\_frequency[base2idx(base1)]>=population\_ACGT\_allele\_frequency[base2idx(base2)]){

return base1;

}else{

return base2;

}

}

/\* end of get\_higher\_AF\_base() \*/

/\*\*

\* @brief the process\_func() for trio genotyper

\*

\* input must be 3(genotype\_log10lik) + 2(mosaic\_AF\_estimate) + 1(prior\_population\_allele\_frequency) = 6 strings

\*

\* read in data, construct CPD, calculate joint distribution (posterior) , and finally marginalize

\*/

int trio\_genotyper\_process\_func(int num\_str, const char \*\*each\_str)

{

if(num\_str!=6){

fprintf(stderr, "Error: the num\_str for trio\_genotyper\_process\_func() must be 6\n");

return -1;

}

char ID[1000] = "";

char chr\_pos[1000] = "";

char \*ID\_ptr;

char \*chr;

char \*pos\_str;

char \*refBase;

char \*altBase;

double father\_genotype\_log10lik[4];

double mother\_genotype\_log10lik[4];

double child\_genotype\_log10lik[4];

// above four: [0] for ref-hom, [1] for het, [2] for alt-hom, [3] for mosaic

char input\_father\_ref\_alt\_base[10];

char input\_mother\_ref\_alt\_base[10];

char input\_child\_ref\_alt\_base[10];

char father\_ref\_alt\_base[10];

char mother\_ref\_alt\_base[10];

char child\_ref\_alt\_base[10];

char chr\_type[10];

double pop\_ACGT\_AF[5];

// above: [0-3] for A C G T, and [4] for '.'

int i,j,k;

double mosaic\_AF\_estimate[3]; // [0] for father, [1] for mother // [2] for child (removed, useless)

double tmp1;

double CPD\_64\_vec[64];

double joint\_64\_vec[64];

// read in each\_str[0]: father\_genotype\_log10lik

if(strcmp(each\_str[0],"")==0){

if(!(strcmp(each\_str[5],"")==0 && strcmp(chr\_pos, "")==0)){

fprintf(stderr, "Warning: no corresponding input father's genotype log10lik for chr\_pos='%s', so I use 1:1:1:1 instead\n", chr\_pos);

}

father\_genotype\_log10lik[0] = 1;

father\_genotype\_log10lik[1] = 1;

father\_genotype\_log10lik[2] = 1;

father\_genotype\_log10lik[3] = 1;

strcpy(input\_father\_ref\_alt\_base, "..");

}else{

sscanf(each\_str[0], "%s\t%lf\t%lf\t%lf\t%lf", ID, &father\_genotype\_log10lik[0], &father\_genotype\_log10lik[1], &father\_genotype\_log10lik[2], &father\_genotype\_log10lik[3]);

ID\_ptr = ID;

chr = strsep(&ID\_ptr, chr\_pos\_delim);

pos\_str = strsep(&ID\_ptr, chr\_pos\_delim);

refBase = strsep(&ID\_ptr, chr\_pos\_delim);

altBase = strsep(&ID\_ptr, chr\_pos\_delim);

strcpy(chr\_pos, chr);

strcat(chr\_pos, chr\_pos\_delim);

strcat(chr\_pos, pos\_str);

input\_father\_ref\_alt\_base[0] = refBase[0];

input\_father\_ref\_alt\_base[1] = altBase[0];

input\_father\_ref\_alt\_base[2] = '\0';

}

// read in each\_str[1]: mother\_genotype\_log10lik

if(strcmp(each\_str[1],"")==0){

if(!(strcmp(each\_str[5],"")==0 && strcmp(chr\_pos, "")==0)){

fprintf(stderr, "Warning: no corresponding input mother's genotype log10lik for chr\_pos='%s', so I use 1:1:1:1 instead\n", chr\_pos);

}

mother\_genotype\_log10lik[0] = 1;

mother\_genotype\_log10lik[1] = 1;

mother\_genotype\_log10lik[2] = 1;

mother\_genotype\_log10lik[3] = 1;

strcpy(input\_mother\_ref\_alt\_base, "..");

}else{

sscanf(each\_str[1], "%s\t%lf\t%lf\t%lf\t%lf", ID, &mother\_genotype\_log10lik[0], &mother\_genotype\_log10lik[1], &mother\_genotype\_log10lik[2], &mother\_genotype\_log10lik[3]);

ID\_ptr = ID;

chr = strsep(&ID\_ptr, chr\_pos\_delim);

pos\_str = strsep(&ID\_ptr, chr\_pos\_delim);

refBase = strsep(&ID\_ptr, chr\_pos\_delim);

altBase = strsep(&ID\_ptr, chr\_pos\_delim);

strcpy(chr\_pos, chr);

strcat(chr\_pos, chr\_pos\_delim);

strcat(chr\_pos, pos\_str);

input\_mother\_ref\_alt\_base[0] = refBase[0];

input\_mother\_ref\_alt\_base[1] = altBase[0];

input\_mother\_ref\_alt\_base[2] = '\0';

}

// read in each\_str[2]: child\_genotype\_log10lik

if(strcmp(each\_str[2],"")==0){

if(!(strcmp(each\_str[5],"")==0 && strcmp(chr\_pos, "")==0)){

fprintf(stderr, "Warning: no corresponding input child's genotype log10lik for chr\_pos='%s', so I use 1:1:1:1 instead\n", chr\_pos);

}

child\_genotype\_log10lik[0] = 1;

child\_genotype\_log10lik[1] = 1;

child\_genotype\_log10lik[2] = 1;

child\_genotype\_log10lik[3] = 1;

strcpy(input\_child\_ref\_alt\_base, "..");

}else{

sscanf(each\_str[2], "%s\t%lf\t%lf\t%lf\t%lf", ID, &child\_genotype\_log10lik[0], &child\_genotype\_log10lik[1], &child\_genotype\_log10lik[2], &child\_genotype\_log10lik[3]);

ID\_ptr = ID;

chr = strsep(&ID\_ptr, chr\_pos\_delim);

pos\_str = strsep(&ID\_ptr, chr\_pos\_delim);

refBase = strsep(&ID\_ptr, chr\_pos\_delim);

altBase = strsep(&ID\_ptr, chr\_pos\_delim);

strcpy(chr\_pos, chr);

strcat(chr\_pos, chr\_pos\_delim);

strcat(chr\_pos, pos\_str);

input\_child\_ref\_alt\_base[0] = refBase[0];

input\_child\_ref\_alt\_base[1] = altBase[0];

input\_child\_ref\_alt\_base[2] = '\0';

}

// read in each\_str[3]: father mosaic AF

if(strcmp(each\_str[3],"")==0){

if(!(strcmp(each\_str[5],"")==0 && strcmp(chr\_pos, "")==0)){

fprintf(stderr, "Warning: no corresponding input father's mosaic AF for chr\_pos='%s', so I use 0.5 instead\n", chr\_pos);

}

mosaic\_AF\_estimate[0] = 0.5;

}else{

sscanf(each\_str[3], "%s\t%lf\t%lf", ID, &mosaic\_AF\_estimate[0], &tmp1);

}

// read in each\_str[4]: mother mosaic AF

if(strcmp(each\_str[4],"")==0){

if(!(strcmp(each\_str[5],"")==0 && strcmp(chr\_pos, "")==0)){

fprintf(stderr, "Warning: no corresponding input mother's mosaic AF for chr\_pos='%s', so I use 0.5 instead\n", chr\_pos);

}

mosaic\_AF\_estimate[1] = 0.5;

}else{

sscanf(each\_str[4], "%s\t%lf\t%lf", ID, &mosaic\_AF\_estimate[1], &tmp1);

}

// read in each\_str[5]: prior\_population\_allele\_frequency

if(strcmp(each\_str[5],"")==0){

if(strcmp(chr\_pos, "")==0){

fprintf(stderr, "Warning: no chr\_pos info, and no valid ID for chr\_pos, I just skipped it\n");

return 0;

}

strcpy(chr\_type, "A");

pop\_ACGT\_AF[0] = -2;

pop\_ACGT\_AF[1] = -2;

pop\_ACGT\_AF[2] = -2;

pop\_ACGT\_AF[3] = -2;

pop\_ACGT\_AF[4] = -10;

// assign prior's ref base to child's ref or parents' ref

if(input\_child\_ref\_alt\_base[0]!='.'){

pop\_ACGT\_AF[base2idx(input\_child\_ref\_alt\_base[0])] = 1;

fprintf(stderr, "Warning: no prior info for chr\_pos = '%s', and I judge ref = child's ref '%c'\n", chr\_pos, input\_child\_ref\_alt\_base[0]);

}else if(input\_father\_ref\_alt\_base[0]!='.' || input\_mother\_ref\_alt\_base[0]!='.'){

if(input\_father\_ref\_alt\_base[0]!='.' && input\_mother\_ref\_alt\_base[0]!='.'){

if(input\_father\_ref\_alt\_base[0]==input\_mother\_ref\_alt\_base[0]){

pop\_ACGT\_AF[base2idx(input\_father\_ref\_alt\_base[0])] = 1;

fprintf(stderr, "Warning: no prior info for chr\_pos = '%s', and I judge ref = father|mother's ref '%c'\n", chr\_pos, input\_father\_ref\_alt\_base[0]);

}else{

// ambiguity for father's ref and mother's ref

pop\_ACGT\_AF[base2idx(input\_mother\_ref\_alt\_base[0])] = 1;

fprintf(stderr, "Warning: no prior info for chr\_pos = '%s', and I judge ref = mother's ref '%c' (with ambiguity, prefer mother)\n", chr\_pos, input\_mother\_ref\_alt\_base[0]);

}

}else if(input\_father\_ref\_alt\_base[0]!='.'){

pop\_ACGT\_AF[base2idx(input\_father\_ref\_alt\_base[0])] = 1;

fprintf(stderr, "Warning: no prior info for chr\_pos = '%s', and I judge ref = father's ref '%c'\n", chr\_pos, input\_father\_ref\_alt\_base[0]);

}else{

pop\_ACGT\_AF[base2idx(input\_mother\_ref\_alt\_base[0])] = 1;

fprintf(stderr, "Warning: no prior info for chr\_pos = '%s', and I judge ref = mother's ref '%c'\n", chr\_pos, input\_mother\_ref\_alt\_base[0]);

}

}else{ // father, mother, child all ref = '.'

fprintf(stderr, "Warning: no prior info and without any known base for chr\_pos = '%s', I just skip it\n", chr\_pos);

return 0;

}

}else{

sscanf(each\_str[5], "%s\t%s\t%lf\t%lf\t%lf\t%lf", chr\_pos, chr\_type, &pop\_ACGT\_AF[0], &pop\_ACGT\_AF[1], &pop\_ACGT\_AF[2], &pop\_ACGT\_AF[3]);

if(chr\_type[0]!='A' && chr\_type[0]!='X' && chr\_type[0]!='Y'){

fprintf(stderr, "Warning: cannot recognize chr\_type '%c', I just treat it 'A' (autosome)\n", chr\_type[0]);

strcpy(chr\_type, "A");

}

pop\_ACGT\_AF[4] = -10;

}

// assign ref/alt base if necessary (troublesome)

char tmpBase1, tmpBase2;

// first assign child's base

if(input\_child\_ref\_alt\_base[0] == '.'){ // no child's ref/alt base

if(input\_father\_ref\_alt\_base[0] == input\_mother\_ref\_alt\_base[0]){ // father's ref == mother's ref

if(input\_father\_ref\_alt\_base[0] != '.'){ // ref != '.'

child\_ref\_alt\_base[0] = input\_father\_ref\_alt\_base[0];

if(input\_father\_ref\_alt\_base[1] == input\_mother\_ref\_alt\_base[1]){ // father's alt == mother's alt

if(input\_father\_ref\_alt\_base[1] != '.'){ // alt != '.'

child\_ref\_alt\_base[1] = input\_father\_ref\_alt\_base[1];

}else{ // alt == '.'

child\_ref\_alt\_base[1] = get\_max\_AF\_base(pop\_ACGT\_AF, child\_ref\_alt\_base[0]);

}

}else{ // father's alt != mother's alt

// compare pop\_AF of father's alt and mother's alt, (prefer mother)

child\_ref\_alt\_base[1] = get\_higher\_AF\_base(pop\_ACGT\_AF, input\_mother\_ref\_alt\_base[1],input\_father\_ref\_alt\_base[1]);

}

}else{ // both parents = ./.

child\_ref\_alt\_base[0] = get\_max\_AF\_base(pop\_ACGT\_AF, '.');

child\_ref\_alt\_base[1] = get\_max\_AF\_base(pop\_ACGT\_AF, child\_ref\_alt\_base[0]);

}

}else{ // father's ref != mother's ref

if(input\_father\_ref\_alt\_base[0] == input\_mother\_ref\_alt\_base[1]){ // father's ref == mother's alt

if(input\_father\_ref\_alt\_base[0] != '.'){ // != '.'

child\_ref\_alt\_base[0] = input\_father\_ref\_alt\_base[0];

// compare pop\_AF of father's alt and mother's ref, (prefer mother)

child\_ref\_alt\_base[1] = get\_higher\_AF\_base(pop\_ACGT\_AF, input\_mother\_ref\_alt\_base[0],input\_father\_ref\_alt\_base[1]);

}else{ // father = ./. , mother's ref should not be '.' logically

if(input\_mother\_ref\_alt\_base[0] != '.'){

child\_ref\_alt\_base[0] = input\_mother\_ref\_alt\_base[0];

child\_ref\_alt\_base[1] = get\_max\_AF\_base(pop\_ACGT\_AF, child\_ref\_alt\_base[0]);

}else{

fprintf(stderr, "Warning: this place should not be visited: father, mother, child ref == '.'\n");

return 0;

}

}

}else if(input\_mother\_ref\_alt\_base[0] == input\_father\_ref\_alt\_base[1]){ // mother's ref == father's alt

if(input\_father\_ref\_alt\_base[1] != '.'){ // != '.'

child\_ref\_alt\_base[0] = input\_father\_ref\_alt\_base[1];

// compare pop\_AF of father's ref and mother's alt, (prefer mother)

child\_ref\_alt\_base[1] = get\_higher\_AF\_base(pop\_ACGT\_AF, input\_mother\_ref\_alt\_base[1],input\_father\_ref\_alt\_base[0]);

}else{ // mother = ./. , father's ref should not be '.' logically

if(input\_father\_ref\_alt\_base[0] != '.'){

child\_ref\_alt\_base[0] = input\_father\_ref\_alt\_base[0];

child\_ref\_alt\_base[1] = get\_max\_AF\_base(pop\_ACGT\_AF, child\_ref\_alt\_base[0]);

}else{

fprintf(stderr, "Warning: this place should not be visited: father, mother, child ref == '.'\n");

return 0;

}

}

}else if(input\_father\_ref\_alt\_base[1] == input\_mother\_ref\_alt\_base[1]){ // father's alt == mother's alt

if(input\_father\_ref\_alt\_base[1] != '.'){ // != '.'

child\_ref\_alt\_base[0] = input\_father\_ref\_alt\_base[1];

// compare pop\_AF of father's ref and mother's ref, (prefer mother)

child\_ref\_alt\_base[1] = get\_higher\_AF\_base(pop\_ACGT\_AF, input\_mother\_ref\_alt\_base[0],input\_father\_ref\_alt\_base[0]);

}else{ // parents' alt == '.'

// compare pop\_AF of father's ref and mother's ref, (prefer mother)

if(pop\_ACGT\_AF[base2idx(input\_father\_ref\_alt\_base[0])]>pop\_ACGT\_AF[base2idx(input\_mother\_ref\_alt\_base[0])]){

child\_ref\_alt\_base[0] = input\_father\_ref\_alt\_base[0];

if(input\_mother\_ref\_alt\_base[0]!='.'){

child\_ref\_alt\_base[0] = input\_mother\_ref\_alt\_base[0];

}else{

child\_ref\_alt\_base[1] = get\_max\_AF\_base(pop\_ACGT\_AF, child\_ref\_alt\_base[0]);

}

}else{

child\_ref\_alt\_base[0] = input\_mother\_ref\_alt\_base[0];

if(input\_father\_ref\_alt\_base[0]!='.'){

child\_ref\_alt\_base[0] = input\_father\_ref\_alt\_base[0];

}else{

child\_ref\_alt\_base[1] = get\_max\_AF\_base(pop\_ACGT\_AF, child\_ref\_alt\_base[0]);

}

}

}

}else{ // father's ref/alt != mother's ref/alt

if(input\_father\_ref\_alt\_base[0]=='.'){ // father = ./. , mother = x/y

strcpy(child\_ref\_alt\_base, input\_mother\_ref\_alt\_base);

}else if(input\_mother\_ref\_alt\_base[0]=='.'){ // mother = ./. , father = x/y

strcpy(child\_ref\_alt\_base, input\_father\_ref\_alt\_base);

}else if(input\_father\_ref\_alt\_base[1]=='.'){ // father = x/., mother = y/?

// compare pop\_AF of mother's ref and alt, (prefer ref)

tmpBase1 = get\_higher\_AF\_base(pop\_ACGT\_AF, input\_mother\_ref\_alt\_base[0],input\_mother\_ref\_alt\_base[1]);

// compare pop\_AF of mother's higher and father's ref, (prefer mother)

child\_ref\_alt\_base[0] = get\_higher\_AF\_base(pop\_ACGT\_AF, tmpBase1,input\_father\_ref\_alt\_base[0]);

if(child\_ref\_alt\_base[0]==tmpBase1){

child\_ref\_alt\_base[1] = input\_father\_ref\_alt\_base[0];

}else{

child\_ref\_alt\_base[1] = tmpBase1;

}

}else if(input\_mother\_ref\_alt\_base[1]=='.'){ // mother = x/., father = y/?

// compare pop\_AF of father's ref and alt, (prefer ref)

tmpBase1 = get\_higher\_AF\_base(pop\_ACGT\_AF, input\_father\_ref\_alt\_base[0],input\_father\_ref\_alt\_base[1]);

// compare pop\_AF of father's higher and mother's ref, (prefer mother)

child\_ref\_alt\_base[0] = get\_higher\_AF\_base(pop\_ACGT\_AF, input\_mother\_ref\_alt\_base[0],tmpBase1);

if(child\_ref\_alt\_base[0]==tmpBase1){

child\_ref\_alt\_base[1] = input\_mother\_ref\_alt\_base[0];

}else{

child\_ref\_alt\_base[1] = tmpBase1;

}

}else{ // father = x1/x2, mother = y1/y2

// compare pop\_AF of father's ref and alt, (prefer ref)

tmpBase1 = get\_higher\_AF\_base(pop\_ACGT\_AF, input\_father\_ref\_alt\_base[0],input\_father\_ref\_alt\_base[1]);

// compare pop\_AF of mother's ref and alt, (prefer ref)

tmpBase2 = get\_higher\_AF\_base(pop\_ACGT\_AF, input\_mother\_ref\_alt\_base[0],input\_mother\_ref\_alt\_base[1]);

// compare pop\_AF of mother's higher and father's higher, (prefer mother)

child\_ref\_alt\_base[0] = get\_higher\_AF\_base(pop\_ACGT\_AF, tmpBase2,tmpBase1);

if(child\_ref\_alt\_base[0]==tmpBase1){

child\_ref\_alt\_base[1] = tmpBase2;

}else{

child\_ref\_alt\_base[1] = tmpBase1;

}

}

}

}

}else if(input\_child\_ref\_alt\_base[1] == '.'){ // only know child's ref base, no child's alt base

child\_ref\_alt\_base[0] = input\_child\_ref\_alt\_base[0];

if(child\_ref\_alt\_base[0]==input\_father\_ref\_alt\_base[0] || child\_ref\_alt\_base[0]==input\_father\_ref\_alt\_base[1] || child\_ref\_alt\_base[0]==input\_mother\_ref\_alt\_base[0] || child\_ref\_alt\_base[0]==input\_mother\_ref\_alt\_base[1]){ // child's ref matches at least one of parents' bases

if(child\_ref\_alt\_base[0]==input\_father\_ref\_alt\_base[0] || child\_ref\_alt\_base[0]==input\_father\_ref\_alt\_base[1]){ // child's ref matches one of father's bases

if(child\_ref\_alt\_base[0]==input\_mother\_ref\_alt\_base[0] || child\_ref\_alt\_base[0]==input\_mother\_ref\_alt\_base[1]){ // child's ref also matches one of mother's bases

if(child\_ref\_alt\_base[0]==input\_father\_ref\_alt\_base[0]){ // child's ref == father's ref

if(child\_ref\_alt\_base[0]==input\_mother\_ref\_alt\_base[0]){ // child's ref == mother's ref

// compare pop\_AF of mother's alt and father's alt, (prefer mother)

child\_ref\_alt\_base[1] = get\_higher\_AF\_base(pop\_ACGT\_AF, input\_mother\_ref\_alt\_base[1],input\_father\_ref\_alt\_base[1]);

if(child\_ref\_alt\_base[1]=='.'){

child\_ref\_alt\_base[1] = get\_max\_AF\_base(pop\_ACGT\_AF, child\_ref\_alt\_base[0]);

}

}else{ // child's ref == mother's alt

// compare pop\_AF of mother's ref and father's alt, (prefer mother)

child\_ref\_alt\_base[1] = get\_higher\_AF\_base(pop\_ACGT\_AF, input\_mother\_ref\_alt\_base[0],input\_father\_ref\_alt\_base[1]);

if(child\_ref\_alt\_base[1]=='.'){

child\_ref\_alt\_base[1] = get\_max\_AF\_base(pop\_ACGT\_AF, child\_ref\_alt\_base[0]);

}

}

}else{ // child's ref == father's alt

if(child\_ref\_alt\_base[0]==input\_mother\_ref\_alt\_base[0]){ // child's ref == mother's ref

// compare pop\_AF of mother's alt and father's ref, (prefer mother)

child\_ref\_alt\_base[1] = get\_higher\_AF\_base(pop\_ACGT\_AF, input\_mother\_ref\_alt\_base[1],input\_father\_ref\_alt\_base[0]);

if(child\_ref\_alt\_base[1]=='.'){

child\_ref\_alt\_base[1] = get\_max\_AF\_base(pop\_ACGT\_AF, child\_ref\_alt\_base[0]);

}

}else{ // child's ref == mother's alt

// compare pop\_AF of mother's ref and father's ref, (prefer mother)

child\_ref\_alt\_base[1] = get\_higher\_AF\_base(pop\_ACGT\_AF, input\_mother\_ref\_alt\_base[0],input\_father\_ref\_alt\_base[0]);

if(child\_ref\_alt\_base[1]=='.'){

child\_ref\_alt\_base[1] = get\_max\_AF\_base(pop\_ACGT\_AF, child\_ref\_alt\_base[0]);

}

}

}

}else{ // child's ref not in mother's bases

child\_ref\_alt\_base[1] = get\_higher\_AF\_base(pop\_ACGT\_AF, input\_mother\_ref\_alt\_base[0],input\_mother\_ref\_alt\_base[1]);

if(child\_ref\_alt\_base[1]=='.'){

child\_ref\_alt\_base[1] = get\_max\_AF\_base(pop\_ACGT\_AF, child\_ref\_alt\_base[0]);

}

}

}else{ // child's ref not in father's bases, so logically should match one of mother's bases

child\_ref\_alt\_base[1] = get\_higher\_AF\_base(pop\_ACGT\_AF, input\_father\_ref\_alt\_base[0],input\_father\_ref\_alt\_base[1]);

if(child\_ref\_alt\_base[1]=='.'){

child\_ref\_alt\_base[1] = get\_max\_AF\_base(pop\_ACGT\_AF, child\_ref\_alt\_base[0]);

}

}

}else{ // child's ref not match any one of parents' bases

tmpBase1 = get\_higher\_AF\_base(pop\_ACGT\_AF, input\_father\_ref\_alt\_base[0],input\_father\_ref\_alt\_base[1]);

tmpBase2 = get\_higher\_AF\_base(pop\_ACGT\_AF, input\_mother\_ref\_alt\_base[0],input\_mother\_ref\_alt\_base[1]);

child\_ref\_alt\_base[1] = get\_higher\_AF\_base(pop\_ACGT\_AF, tmpBase2,tmpBase1);

if(child\_ref\_alt\_base[1]=='.'){

child\_ref\_alt\_base[1] = get\_max\_AF\_base(pop\_ACGT\_AF, child\_ref\_alt\_base[0]);

}

}

}else{ // we know child's ref/alt base

// strcpy(child\_ref\_alt\_base, input\_child\_ref\_alt\_base);

child\_ref\_alt\_base[0] = input\_child\_ref\_alt\_base[0];

child\_ref\_alt\_base[1] = input\_child\_ref\_alt\_base[1];

}

child\_ref\_alt\_base[2] = '\0';

// then assign parents' base

if(input\_father\_ref\_alt\_base[0] == '.'){ // no father's ref/alt base

strcpy(father\_ref\_alt\_base, child\_ref\_alt\_base);

}else if(input\_father\_ref\_alt\_base[1] == '.'){ // no father's alt base

if(input\_father\_ref\_alt\_base[0]==child\_ref\_alt\_base[0]){ // father's ref == child's ref

strcpy(father\_ref\_alt\_base, child\_ref\_alt\_base);

}else if(input\_father\_ref\_alt\_base[0]==child\_ref\_alt\_base[1]){ // father's ref == child's alt

father\_ref\_alt\_base[0] = child\_ref\_alt\_base[1];

father\_ref\_alt\_base[1] = child\_ref\_alt\_base[0];

}else{ // father's ref base != any of child's base

father\_ref\_alt\_base[0] = input\_father\_ref\_alt\_base[0];

father\_ref\_alt\_base[1] = get\_higher\_AF\_base(pop\_ACGT\_AF, child\_ref\_alt\_base[0],child\_ref\_alt\_base[1]);

}

}else{ // we know father's ref/alt base

// strcpy(father\_ref\_alt\_base, input\_father\_ref\_alt\_base);

father\_ref\_alt\_base[0] = input\_father\_ref\_alt\_base[0];

father\_ref\_alt\_base[1] = input\_father\_ref\_alt\_base[1];

}

father\_ref\_alt\_base[2] = '\0';

if(input\_mother\_ref\_alt\_base[0] == '.'){ // no mother's ref/alt base

strcpy(mother\_ref\_alt\_base, child\_ref\_alt\_base);

}else if(input\_mother\_ref\_alt\_base[1] == '.'){ // no mother's alt base

if(input\_mother\_ref\_alt\_base[0]==child\_ref\_alt\_base[0]){

strcpy(mother\_ref\_alt\_base, child\_ref\_alt\_base);

}else if(input\_mother\_ref\_alt\_base[0]==child\_ref\_alt\_base[1]){

mother\_ref\_alt\_base[0] = child\_ref\_alt\_base[1];

mother\_ref\_alt\_base[1] = child\_ref\_alt\_base[0];

mother\_ref\_alt\_base[2] = '\0';

}else{ // mother's ref base != any of child's base

mother\_ref\_alt\_base[0] = input\_mother\_ref\_alt\_base[0];

mother\_ref\_alt\_base[1] = get\_higher\_AF\_base(pop\_ACGT\_AF, child\_ref\_alt\_base[0],child\_ref\_alt\_base[1]);

}

}else{ // we know mother's ref/alt base

// strcpy(mother\_ref\_alt\_base, input\_mother\_ref\_alt\_base);

mother\_ref\_alt\_base[0] = input\_mother\_ref\_alt\_base[0];

mother\_ref\_alt\_base[1] = input\_mother\_ref\_alt\_base[1];

}

mother\_ref\_alt\_base[2] = '\0';

// calculate parents' log10 prior

normalize\_population\_ACGT\_allele\_frequency(pop\_ACGT\_AF, minus\_1\_AF, minus\_2\_AF);

double father\_prior\_log10[4];

double mother\_prior\_log10[4];

parent\_log10\_prior(chr\_type, 'M', father\_ref\_alt\_base, pop\_ACGT\_AF, mosaic\_rate, father\_prior\_log10);

parent\_log10\_prior(chr\_type, 'F', mother\_ref\_alt\_base, pop\_ACGT\_AF, mosaic\_rate, mother\_prior\_log10);

// construct CPD from parents to child

parent\_to\_child\_CPD\_log10lik(chr\_type, father\_ref\_alt\_base, mother\_ref\_alt\_base, child\_ref\_alt\_base, mosaic\_AF\_estimate[0], mosaic\_AF\_estimate[1], de\_novo\_rate, mosaic\_rate, CPD\_64\_vec);

// printf("CPD matrix:\n");

// for(i=0; i<4; i++){ // father's genotype

// for(j=0; j<4; j++){ // mother's genotype

// printf("i = %d, j = %d:\t", i,j);

// for(k=0; k<4; k++){ // child's genotype

// printf("%.2e ", CPD\_64\_vec[i\*16+j\*4+k]);

// }

// printf("\n");

// }

// }

// calculate joint distribution by multiplying 3 genotype lik, 2 prior and 1 CPD

double joint\_unnormalized\_total = LOGZERO;

for(i=0; i<4; i++){ // father's genotype

for(j=0; j<4; j++){ // mother's genotype

for(k=0; k<4; k++){ // child's genotype

joint\_64\_vec[i\*16+j\*4+k] = father\_genotype\_log10lik[i] + mother\_genotype\_log10lik[j] + child\_genotype\_log10lik[k] + father\_prior\_log10[i] + mother\_prior\_log10[j] + CPD\_64\_vec[i\*16+j\*4+k];

joint\_unnormalized\_total = log10\_add\_pow10(joint\_unnormalized\_total, joint\_64\_vec[i\*16+j\*4+k]);

}

}

}

// printf("joint matrix:\n");

// for(i=0; i<4; i++){ // father's genotype

// for(j=0; j<4; j++){ // mother's genotype

// printf("i = %d, j = %d\n", i,j);

// for(k=0; k<4; k++){ // child's genotype

// printf("%.5f ", CPD\_64\_vec[i\*16+j\*4+k]);

// }

// printf("\n");

// }

// }

// marginalize to get posterior for father, mother, child's genotype

double father\_posterior[4] = {LOGZERO,LOGZERO,LOGZERO,LOGZERO};

double mother\_posterior[4] = {LOGZERO,LOGZERO,LOGZERO,LOGZERO};

double child\_posterior[4] = {LOGZERO,LOGZERO,LOGZERO,LOGZERO};

for(i=0; i<4; i++){ // father's genotype

for(j=0; j<4; j++){ // mother's genotype

for(k=0; k<4; k++){ // child's genotype

father\_posterior[i] = log10\_add\_pow10(father\_posterior[i], joint\_64\_vec[i\*16+j\*4+k]);

mother\_posterior[j] = log10\_add\_pow10(mother\_posterior[j], joint\_64\_vec[i\*16+j\*4+k]);

child\_posterior[k] = log10\_add\_pow10(child\_posterior[k], joint\_64\_vec[i\*16+j\*4+k]);

}

}

}

// output format: ID(chr:pos) + chr\_type + father's (refBase/altBase + 4 genotype posterior) + mother's + child's

printf("%s\t%s", chr\_pos, chr\_type);

// father's ref/alt base

printf("\t%c/%c", father\_ref\_alt\_base[0], father\_ref\_alt\_base[1]);

if(strcmp(father\_ref\_alt\_base, input\_father\_ref\_alt\_base)!=0){

printf("(%c/%c)", input\_father\_ref\_alt\_base[0], input\_father\_ref\_alt\_base[1]);

}

// father's genotype posterior

for(i=0; i<4; i++){ // father's genotype

if(i==0){

printf("\t");

}else{

printf(",");

}

if(father\_posterior[i]-joint\_unnormalized\_total < -1e8){

printf("%.2e", father\_posterior[i]-joint\_unnormalized\_total);

}else{

printf("%.5f", father\_posterior[i]-joint\_unnormalized\_total);

}

}

// mother's ref/alt base

printf("\t%c/%c", mother\_ref\_alt\_base[0], mother\_ref\_alt\_base[1]);

if(strcmp(mother\_ref\_alt\_base, input\_mother\_ref\_alt\_base)!=0){

printf("(%c/%c)", input\_mother\_ref\_alt\_base[0], input\_mother\_ref\_alt\_base[1]);

}

// mother's genotype posterior

if(chr\_type[0]=='Y'){

printf("\t0,0,0,0");

}else{

for(j=0; j<4; j++){ // mother's genotype

if(j==0){

printf("\t");

}else{

printf(",");

}

if(mother\_posterior[j]-joint\_unnormalized\_total < -1e8){

printf("%.2e", mother\_posterior[j]-joint\_unnormalized\_total);

}else{

printf("%.5f", mother\_posterior[j]-joint\_unnormalized\_total);

}

}

}

// child's ref/alt base

printf("\t%c/%c", child\_ref\_alt\_base[0], child\_ref\_alt\_base[1]);

if(strcmp(child\_ref\_alt\_base, input\_child\_ref\_alt\_base)!=0){

printf("(%c/%c)", input\_child\_ref\_alt\_base[0], input\_child\_ref\_alt\_base[1]);

}

// child's genotype posterior

if(child\_sex[0]=='F' && chr\_type[0]=='Y'){

printf("\t0,0,0,0");

}else{

for(k=0; k<4; k++){ // child\_posterior's genotype

if(k==0){

printf("\t");

}else{

printf(",");

}

if(child\_posterior[k]-joint\_unnormalized\_total < -1e8){

printf("%.2e", child\_posterior[k]-joint\_unnormalized\_total);

}else{

printf("%.5f", child\_posterior[k]-joint\_unnormalized\_total);

}

}

}

printf("\n");

return 0;

}

/\* end of trio\_genotyper\_process\_func() \*/

/\*\*

\* @brief a wrapped framework to process matched line from sorted files

\*

\* open and read in each line from several sorted files (skip blank line)

\* cut the first column as ID,

\* compare ID from each file (by compare\_two\_ID() ),

\* get the min\_ID and put "" to those unmatched,

\* send these strings (may contain '\n') to process\_func() for computing and outputing

\*

\* If process\_func() returns not 0, show warning and proceed when process\_error\_should\_stop=0

\* or stop, report error when process\_error\_should\_stop=1

\*/

int match\_and\_process\_firstColumnSorted\_files(int num\_files, char \*\*filenames, int (\*compare\_two\_ID)(const char \*IDa, const char \*IDb), int (\*process\_func)(int num\_str, const char \*\*each\_str), int process\_error\_should\_stop )

{

size\_t malloc\_line\_len = LINE\_LEN;

const char \*null\_str = "";

FILE \*\*fileHandles;

if((fileHandles = malloc((num\_files)\*sizeof(FILE \*)))==NULL){

fprintf(stderr, "Error: cannot allocate memory for fileHandles at %d\n", \_\_LINE\_\_);

return 1;

}

int fi\_0, fi\_1;

int has\_error = 0;

for(fi\_0=0; fi\_0<num\_files; fi\_0++){

if((fileHandles[fi\_0] = fopen(filenames[fi\_0], "r"))==NULL){

has\_error = 1;

fprintf(stderr, "Error: cannot open file '%s' for input\n", filenames[fi\_0]);

// break;

}

}

int \*should\_read\_next\_line = malloc((num\_files)\*sizeof(int));

char \*\*input\_each\_str = malloc((num\_files)\*sizeof(char\*));

int input\_str\_len;

char \*\*each\_str = malloc((num\_files)\*sizeof(char\*));

char \*\*each\_ID\_str = malloc((num\_files)\*sizeof(char\*));

if(should\_read\_next\_line==NULL || each\_str==NULL || input\_each\_str==NULL || each\_ID\_str==NULL){

fprintf(stderr, "Error: cannot allocate memory for should\_read\_next\_line, each\_str, input\_each\_str or each\_ID\_str at %d\n", \_\_LINE\_\_);

has\_error = 1;

}

if(!has\_error){

for(fi\_1=0; fi\_1<num\_files; fi\_1++){

should\_read\_next\_line[fi\_1] = 1;

if((input\_each\_str[fi\_1] = malloc((malloc\_line\_len)\*sizeof(char)))==NULL){

has\_error = 1;

fprintf(stderr, "Error: cannot allocate memory for input\_each\_str[%d]\n", fi\_1);

// break;

}

if((each\_ID\_str[fi\_1] = malloc((malloc\_line\_len)\*sizeof(char)))==NULL){

has\_error = 1;

fprintf(stderr, "Error: cannot allocate memory for each\_ID\_str[%d]\n", fi\_1);

// break;

}

}

char \*min\_ID\_str;

int not\_all\_eof = 1;

int process\_return\_value;

if(!has\_error){

while(not\_all\_eof){

// read in new line

for(fi\_1=0; fi\_1<num\_files; fi\_1++){

if(should\_read\_next\_line[fi\_1]==1){

if(strcmp(each\_ID\_str[fi\_1], YYX\_EOF\_STR)==0){

continue;

}

while((input\_str\_len = getline(&input\_each\_str[fi\_1], &malloc\_line\_len, fileHandles[fi\_1]))!=-1){

if(input\_str\_len>0 && strcmp(input\_each\_str[fi\_1], "")!=0 && strcmp(input\_each\_str[fi\_1], "\n")!=0){

break;

}

}

if(input\_str\_len==-1){

strcpy(each\_ID\_str[fi\_1], YYX\_EOF\_STR);

}else{

sscanf(input\_each\_str[fi\_1], "%s", each\_ID\_str[fi\_1]);

}

}

}

// get min\_ID\_str

min\_ID\_str = each\_ID\_str[0];

for(fi\_1=1; fi\_1<num\_files; fi\_1++){

if(compare\_two\_ID(each\_ID\_str[fi\_1],min\_ID\_str)<0){

min\_ID\_str = each\_ID\_str[fi\_1];

}

}

if(strcmp(min\_ID\_str, YYX\_EOF\_STR)==0){

not\_all\_eof = 0;

continue;

}

// determine each\_str and should\_read\_next\_line

for(fi\_1=0; fi\_1<num\_files; fi\_1++){

if(compare\_two\_ID(each\_ID\_str[fi\_1],min\_ID\_str)==0){

each\_str[fi\_1] = input\_each\_str[fi\_1];

should\_read\_next\_line[fi\_1] = 1;

}else{

each\_str[fi\_1] = (char \*)null\_str;

should\_read\_next\_line[fi\_1] = 0;

}

}

// call process\_func

process\_return\_value = (\*process\_func)(num\_files, (const char \*\*)each\_str);

if(process\_return\_value!=0){

if(process\_error\_should\_stop){

fprintf(stderr, "Error: process\_func returns %d\n", process\_return\_value);

break;

}else{

fprintf(stderr, "Warning: process\_func returns %d\n", process\_return\_value);

}

}

}

}

for(fi\_0=num\_files-1;fi\_0>=0;fi\_0--){

if(each\_ID\_str[fi\_0]!=NULL){

free(each\_ID\_str[fi\_0]);

}

}

}

free(should\_read\_next\_line);

free(each\_str);

free(input\_each\_str);

free(each\_ID\_str);

for(fi\_0=num\_files-1;fi\_0>=0;fi\_0--){

if(fileHandles[fi\_0]!=NULL){

fclose(fileHandles[fi\_0]);

}

}

free(fileHandles);

if(!has\_error){

return 0;

}else{

return -1;

}

}

/\* end of match\_and\_process\_firstColumnSorted\_files() \*/

int main(int argc, char\* argv[])

{

char usage\_str[10000];

sprintf(usage\_str, "\nUsage: %s <chr\_pos\_delim> <chr\_order.list> \n", argv[0]);

strcat(usage\_str, " <father\_genotype\_log10lik> \n");

strcat(usage\_str, " <mother\_genotype\_log10lik> <child\_genotype\_log10lik> \n");

strcat(usage\_str, " <father\_mosaic\_AF> <mother\_mosaic\_AF> \n");

strcat(usage\_str, " <prior\_allele\_frequency> <child\_sex> \n");

strcat(usage\_str, " [de\_novo\_rate] [mosaic\_rate] [minus\_1\_AF] [minus\_2\_AF]\n");

strcat(usage\_str, "\n");

strcat(usage\_str, " \* This program is used to integrate trio genotype likelihood,\n");

strcat(usage\_str, " parents' prior, and parents to child CPD (support 4 bases),\n");

strcat(usage\_str, " to get joint posterior distribution\n");

strcat(usage\_str, " and then marginalize to get everyone's genotype posterior\n");

strcat(usage\_str, " \* Input includes 3 genotype log10lik files, 2 parents' mosaic AF, and prior\n");

strcat(usage\_str, " all 6 files should be sorted on the first column ID(chr:pos...)\n");

strcat(usage\_str, "\n");

strcat(usage\_str, " <chr\_pos\_delim> : the char in ID separate chr and pos, eg. ':'\n");

strcat(usage\_str, " use \"\" for just strcmp on ID\n");

strcat(usage\_str, "\n");

strcat(usage\_str, " <chr\_order.list> : a file contains chr order\n");

strcat(usage\_str, " only consider the first column\n");

strcat(usage\_str, "\n");

strcat(usage\_str, " <\*\_genotype\_log10lik> format: (5 columns)\n");

strcat(usage\_str, " ID(chr:pos:refBase:altBase) + log10 likelihood P(o|G)\n");

strcat(usage\_str, " for genotype (ref-hom, het, alt-hom, mosaic)\n");

strcat(usage\_str, " for each site on each line,\n");

strcat(usage\_str, " can be generated by 'Yyx\_genotype\_log10lik\_with\_precalc\_beta'\n");

strcat(usage\_str, "\n");

strcat(usage\_str, " <\*\_mosaic\_AF> format: (>=2 columns)\n");

strcat(usage\_str, " ID(chr:pos:refBase:altBase) + AF estimate + ...\n");

strcat(usage\_str, " can be generated by 'Yyx\_estimate\_mosaic\_AF\_from\_baseQ'\n");

strcat(usage\_str, "\n");

strcat(usage\_str, " <prior\_allele\_frequency> format: (6 columns)\n");

strcat(usage\_str, " ID(chr:pos) + A/X/Y (autosome/X/Y)\n");

strcat(usage\_str, " + population allele frequency (A,C,G,T)\n");

strcat(usage\_str, " for each site on each line,\n");

strcat(usage\_str, "\n");

strcat(usage\_str, " <child\_sex>: child's sex M/F (male/female)\n");

strcat(usage\_str, "\n");

strcat(usage\_str, " [de\_novo\_rate]: de novo mutation rate\n");

strcat(usage\_str, " the probability to see an alt allele in gamate when the parent is ref-hom\n");

strcat(usage\_str, " [ default = 1e-8 ]\n");

strcat(usage\_str, "\n");

strcat(usage\_str, " [mosaic\_rate]: mosaic rate\n");

strcat(usage\_str, " the probability to see a site is a mosaic site in an individual\n");

strcat(usage\_str, " [ default = 1e-7 ]\n");

strcat(usage\_str, "\n");

strcat(usage\_str, " [minus\_1\_AF]: allele frequency for code '-1'\n");

strcat(usage\_str, " the allele frequency for that in dbSNP but no MAF information\n");

strcat(usage\_str, " [ default = 0.002 (1/500) ]\n");

strcat(usage\_str, "\n");

strcat(usage\_str, " [minus\_2\_AF]: allele frequency for code '-2'\n");

strcat(usage\_str, " the allele frequency for that not found in dbSNP\n");

strcat(usage\_str, " [ default = 1e-4 ]\n");

strcat(usage\_str, "\n");

strcat(usage\_str, " output (stdout) format: (8 columns, separated by '\\t')\n");

strcat(usage\_str, " ID(chr:pos) + A/X/Y (autosome/X/Y) + \n");

strcat(usage\_str, " (refBase/altBase + log10 posterior) for father, mother, child,\n");

strcat(usage\_str, " each of log10 posterior columns contains 4 values\n");

strcat(usage\_str, " (ref-hom, het, alt-hom, mosaic), separated by ','\n");

strcat(usage\_str, "\n");

strcat(usage\_str, "Version: 0.2.2 (2014-05-20)\n");

strcat(usage\_str, "Author: Adam Yongxin Ye @ CBI\n");

strcat(usage\_str, "\n");

char \*chr\_order\_filename;

char \*\*filenames;

if(argc<9+1 || argc>13+1){

fprintf(stderr, "%s", usage\_str);

return 1;

}else{

chr\_pos\_delim = argv[1];

chr\_pos\_delim[1] = '\0';

chr\_order\_filename = argv[2];

filenames = argv+3;

if(argc>9){

sscanf(argv[9], "%s", child\_sex);

if(strcmp(child\_sex,"M")!=0 && strcmp(child\_sex,"F")!=0){

fprintf(stderr, "Error: cannot recognize child's sex = '%c'\n", child\_sex[0]);

return 2;

}

}

if(argc>10){ sscanf(argv[10], "%lf", &de\_novo\_rate); }

if(argc>11){ sscanf(argv[11], "%lf", &mosaic\_rate); }

if(argc>12){ sscanf(argv[12], "%lf", &minus\_1\_AF); }

if(argc>13){ sscanf(argv[13], "%lf", &minus\_2\_AF); }

}

// read in chr\_order file

int has\_error = 0;

size\_t tmp\_line\_len = LINE\_LEN;

char \*tmp\_line;

if((tmp\_line = malloc((tmp\_line\_len)\*sizeof(char)))==NULL){

fprintf(stderr, "Error: cannot allocate memory for tmp\_line\n");

has\_error = 1;

}

int return\_value = -1;

if(!has\_error){

FILE \*chr\_order\_fileHandle;

if((chr\_order\_fileHandle = fopen(chr\_order\_filename, "r"))==NULL){

fprintf(stderr, "Error: cannot open file '%s' for input\n", chr\_order\_filename);

has\_error = 1;

}

if(!has\_error){

while((getline(&tmp\_line, &tmp\_line\_len, chr\_order\_fileHandle))!=-1){

if(strcmp(tmp\_line, "")==0){

continue;

}

if((chr\_order[chr\_order\_len] = malloc((tmp\_line\_len)\*sizeof(char)))==NULL){

fprintf(stderr, "Error: cannot allocate memory for chr\_order[%d]\n", chr\_order\_len);

has\_error = 1;

break;

}

sscanf(tmp\_line, "%s", chr\_order[chr\_order\_len]);

chr\_order\_len++;

}

fclose(chr\_order\_fileHandle);

if(!has\_error){

return\_value = match\_and\_process\_firstColumnSorted\_files(6, filenames, compare\_two\_ID\_chr\_pos, trio\_genotyper\_process\_func, 1 );

// fprintf(stderr, "[DEBUG] here 4\n");

}

for(chr\_order\_len--; chr\_order\_len>=0; chr\_order\_len--){

free(chr\_order[chr\_order\_len]);

}

return return\_value;

}else{

return -1;

}

}else{

return -1;

}

}

2. 数值计算嵌合突变等位基因比例（AF）可信区间的R包 yyxMosaicHunter

2-1. Rcpp的C代码

#include <Rcpp.h>

using namespace Rcpp;

// [[Rcpp::export]]

LogicalVector yyx\_is\_within\_range\_\_c(NumericVector vec, NumericVector range){

long n = vec.size();

LogicalVector out(n);

double range\_min = min(range);

double range\_max = max(range);

for(long i = 0; i < n; ++i) {

out[i] = vec[i] >= range\_min && vec[i] <= range\_max;

}

return out;

}

/\*\*\* R

#yyx\_is\_within\_range\_\_c(rep(1:5,5), c(2.5,2,3,4.5))

stopifnot(all( yyx\_is\_within\_range\_\_c(rep(1:5,5), c(2.5,2,3,4.5)) == yyx\_is\_within\_range(rep(1:5,5), c(2.5,2,3,4.5)) ))

#microbenchmark(

# yyx\_is\_within\_range(rep(1:5,5), c(2.5,2,3,4.5)),

# yyx\_is\_within\_range\_\_c(rep(1:5,5), c(2.5,2,3,4.5))

#)

\*/

// [[Rcpp::export]]

double yyx\_log\_add\_exp\_\_c(double a, double b){

if(a >= b){

return a + log(1 + exp(b - a));

}else{

return b + log(1 + exp(a - b));

}

}

/\*\*\* R

yyx\_log\_add\_exp\_\_c(-1, -2)

stopifnot(all( yyx\_log\_add\_exp\_\_c(-1, -2) == yyx\_log\_sum\_exp(c(-1, -2)) ))

#microbenchmark(

# yyx\_log\_add\_exp\_\_c(-1, -2),

# yyx\_log\_sum\_exp(c(-1, -2))

#)

\*/

// [[Rcpp::export]]

double yyx\_log\_sum\_exp\_\_c(NumericVector vec){

long n = vec.size();

double vec\_max = max(vec);

if(vec\_max == INFINITY){

return vec\_max;

}

double temp = 0.0;

for(long i = 0; i < n; ++i) {

temp += exp(vec[i] - vec\_max);

}

return vec\_max + log(temp);

}

/\*\*\* R

#yyx\_log\_sum\_exp\_\_c(c(-(1:5), -(10:1)))

stopifnot(all( yyx\_log\_sum\_exp\_\_c(c(-(1:5), -(10:1))) - yyx\_log\_sum\_exp(c(-(1:5), -(10:1))) < .Machine$double.eps ))

#yyx\_log\_sum\_exp\_\_c(c(Inf, 0, 1))

stopifnot(all( yyx\_log\_sum\_exp\_\_c(c(Inf, 0, 1)) == yyx\_log\_sum\_exp(c(Inf, 0, 1)) ))

#microbenchmark(

# yyx\_log\_sum\_exp(c(-(1:5), -(10:1))),

# yyx\_log\_sum\_exp\_\_c(c(-(1:5), -(10:1))),

# times = 1000L

#)

\*/

double log\_one\_minus\_exp\_P\_\_c(double log\_prob){

return log(1 - exp(log\_prob));

}

// [[Rcpp::export]]

NumericVector yyx\_log\_P\_o\_given\_r\_q\_\_c(NumericVector ref\_baseQ\_vec, NumericVector alt\_baseQ\_vec){

long ref\_count = ref\_baseQ\_vec.size();

long alt\_count = alt\_baseQ\_vec.size();

long depth = ref\_count + alt\_count;

double ln\_10 = log(10);

NumericVector ref\_log\_error\_rate\_vec = ln\_10 \* -ref\_baseQ\_vec/10;

NumericVector alt\_log\_error\_rate\_vec = ln\_10 \* -alt\_baseQ\_vec/10;

NumericVector ans(depth+1);

// ans idx=0:depth

// P(o|r,q) as function of r(real\_alt\_count)=0:depth

double this\_ref\_alt\_log\_prob[2];

ans[0] = 0;

long i = 0;

long j, k;

for(k = 0; k < ref\_count; ++k){

++i;

this\_ref\_alt\_log\_prob[1] = ref\_log\_error\_rate\_vec[k];

this\_ref\_alt\_log\_prob[0] = log\_one\_minus\_exp\_P\_\_c(ref\_log\_error\_rate\_vec[k]);

ans[i] = ans[i-1] + this\_ref\_alt\_log\_prob[1];

for(j = i - 1; j >= 1; --j){

ans[j] = yyx\_log\_add\_exp\_\_c(ans[j] + this\_ref\_alt\_log\_prob[0], ans[j-1] + this\_ref\_alt\_log\_prob[1]);

}

ans[0] = ans[0] + this\_ref\_alt\_log\_prob[0];

}

for(k = 0; k < alt\_count; ++k){

++i;

this\_ref\_alt\_log\_prob[0] = alt\_log\_error\_rate\_vec[k];

this\_ref\_alt\_log\_prob[1] = log\_one\_minus\_exp\_P\_\_c(alt\_log\_error\_rate\_vec[k]);

ans[i] = ans[i-1] + this\_ref\_alt\_log\_prob[1];

for(j = i - 1; j >= 1; --j){

ans[j] = yyx\_log\_add\_exp\_\_c(ans[j] + this\_ref\_alt\_log\_prob[0], ans[j-1] + this\_ref\_alt\_log\_prob[1]);

}

ans[0] = ans[0] + this\_ref\_alt\_log\_prob[0];

}

return ans;

}

/\*\*\* R

#yyx\_log\_P\_o\_given\_r\_q(c(10,20,30,40,50,+Inf), c(20,30,40))

#yyx\_log\_P\_o\_given\_r\_q\_\_c(c(10,20,30,40,50,+Inf), c(20,30,40))

stopifnot(all( yyxNA2FALSE(yyx\_log\_P\_o\_given\_r\_q(c(10,20,30,40,50,+Inf), c(20,30,40)) - yyx\_log\_P\_o\_given\_r\_q\_\_c(c(10,20,30,40,50,+Inf), c(20,30,40))) < .Machine$double.eps^0.9 ))

stopifnot(all( yyx\_log\_P\_o\_given\_r\_q(rep(20, 800), rep(30,200)) - yyx\_log\_P\_o\_given\_r\_q\_\_c(rep(20, 800), rep(30,200)) < .Machine$double.eps^0.7 ))

\*/

// [[Rcpp::export]]

NumericVector yyx\_log\_P\_r\_given\_theta\_depth\_\_c(double theta, long depth){

NumericVector ans(depth + 1);

if(theta==0){

std::fill(ans.begin(), ans.end(), -INFINITY);

ans[0] = 0;

}else if(theta==1){

std::fill(ans.begin(), ans.end(), -INFINITY);

ans[depth] = 0;

}else{

double log\_theta = log(theta);

double log\_1\_theta = log(1 - theta);

for(long i = 0; i <= depth; ++i){

ans[i] = i \* log\_theta + (depth - i) \* log\_1\_theta;

}

}

return ans;

}

/\*\*\* R

#yyx\_log\_P\_r\_given\_theta\_depth\_\_c(0, 100)

#yyx\_log\_P\_r\_given\_theta\_depth\_\_c(1, 100)

#yyx\_log\_P\_r\_given\_theta\_depth\_\_c(0.5, 100)

#yyx\_log\_P\_r\_given\_theta\_depth\_\_c(0.1, 100)

stopifnot(all( yyx\_log\_P\_r\_given\_theta\_depth\_\_c(0, 100) - yyx\_log\_P\_r\_given\_theta\_depth(0, 100) < .Machine$double.eps^0.9 ))

stopifnot(all( yyx\_log\_P\_r\_given\_theta\_depth\_\_c(1, 100) - yyx\_log\_P\_r\_given\_theta\_depth(1, 100) < .Machine$double.eps^0.9 ))

stopifnot(all( yyx\_log\_P\_r\_given\_theta\_depth\_\_c(0.5, 100) - yyx\_log\_P\_r\_given\_theta\_depth(0.5, 100) < .Machine$double.eps^0.9 ))

stopifnot(all( yyx\_log\_P\_r\_given\_theta\_depth\_\_c(0.1, 100) - yyx\_log\_P\_r\_given\_theta\_depth(0.1, 100) < .Machine$double.eps^0.8 ))

\*/

// [[Rcpp::export]]

double yyx\_log\_P\_o\_given\_theta\_q\_\_c(double theta, NumericVector log\_P\_o\_given\_r\_q\_vec){

long depth = log\_P\_o\_given\_r\_q\_vec.size() - 1;

NumericVector log\_P\_r\_given\_theta\_depth\_vec = yyx\_log\_P\_r\_given\_theta\_depth\_\_c(theta, depth);

return yyx\_log\_sum\_exp\_\_c( log\_P\_o\_given\_r\_q\_vec + log\_P\_r\_given\_theta\_depth\_vec );

}

/\*\*\* R

log\_P\_o\_given\_r\_q\_vec <- yyx\_log\_P\_o\_given\_r\_q\_\_c(rep(20, 80), rep(30,20))

stopifnot(all(

yyx\_log\_P\_o\_given\_theta\_q(0.5, log\_P\_o\_given\_r\_q\_vec=log\_P\_o\_given\_r\_q\_vec)

-

yyx\_log\_P\_o\_given\_theta\_q\_\_c(0.5, log\_P\_o\_given\_r\_q\_vec)

< .Machine$double.eps^0.9 ))

\*/

// [[Rcpp::export]]

NumericVector yyx\_loglikelihood\_theta\_vec\_\_c(NumericVector ref\_baseQ\_vec, NumericVector alt\_baseQ\_vec, NumericVector theta\_vec){

NumericVector log\_P\_o\_given\_r\_q\_vec = yyx\_log\_P\_o\_given\_r\_q\_\_c(ref\_baseQ\_vec, alt\_baseQ\_vec);

// NumericVector theta\_vec(knots);

// for(long i = 0; i < knots; ++i){

// theta\_vec[i] = double(i) / knots;

// }

long knots = theta\_vec.size();

NumericVector loglikelihood\_vec(knots);

for(long i = 0; i < knots; ++i){

loglikelihood\_vec[i] = yyx\_log\_P\_o\_given\_theta\_q\_\_c(theta\_vec[i], log\_P\_o\_given\_r\_q\_vec);

}

return loglikelihood\_vec;

}

/\*

yyx\_loglikelihood\_theta\_splinefun\_\_call\_c <- function(ref\_baseQ\_vec, alt\_baseQ\_vec, knots=1001, ...){

theta\_vec <- seq(0,1, length.out=knots)

loglikelihood\_vec <- yyx\_loglikelihood\_theta\_vec\_\_c(ref\_baseQ\_vec, alt\_baseQ\_vec, theta\_vec)

stopifnot(length(theta\_vec)==length(loglikelihood\_vec))

simple\_outlier\_check\_list <- yyx\_loglik\_simple\_outlier\_idx(loglikelihood\_vec)

theta\_vec\_normal<- theta\_vec[simple\_outlier\_check\_list$normal\_idx]

loglikelihood\_vec\_normal <- loglikelihood\_vec[simple\_outlier\_check\_list$normal\_idx]

stopifnot(length(theta\_vec\_normal)==length(loglikelihood\_vec\_normal))

yyx\_restrict\_x\_interval\_function(splinefun(theta\_vec\_normal, loglikelihood\_vec\_normal, ...), c(0,1), -Inf)

}

\*/

2.2. R代码

##### define functions

### general functions

#' Is each element within the specified range?

#'

#' \code{yyx\_is\_within\_range} returns a bool vector indicating

#' whether each element of input vector is within the specified range

#'

#' @param vec A numeric vector.

#' @param range A vector specify the wanted range.

#' @return A bool vector (can be used as index vector),

#' indicating whether vec[index] is within range.

#'

#' @seealso \code{yyx\_is\_within\_range\_\_c} for Rcpp version

#' @family general functions

#'

#' @examples

#' yyx\_is\_within\_range(1:5, c(2,4.5))

#' # FALSE TRUE TRUE TRUE FALSE

#' yyx\_is\_within\_range(c(1980,1985,1990,1995), 1982:1992)

#' # FALSE TRUE TRUE FALSE

#' yyx\_is\_within\_range\_\_c(1:5, c(2,4.5))

#' # FALSE TRUE TRUE TRUE FALSE

#'

yyx\_is\_within\_range <- function(vec, range){

bool\_vec <- ( vec >= min(range) );

bool\_vec <- bool\_vec & ( vec <= max(range) );

bool\_vec;

}

#' Convert NA to FALSE

#'

#' \code{yyx\_NA2FALSE} just converts NA in input vector to FALSE

#'

#' @param vec A bool (or other type) vector.

#' @return A bool (or other type) vector, the same length as input \code{vec}.

#' NAs in vec will be replaced by FALSE.

#'

#' @seealso \code{\link{yyx\_convert\_NA}} and \code{\link{yyx\_convert\_NA\_vec}}

#' @family general functions

#'

#' @examples

#' yyx\_NA2FALSE(c(TRUE,TRUE,FALSE,NA,TRUE,FALSE,FALSE,TRUE,NA))

#' # TRUE TRUE FALSE FALSE TRUE FALSE FALSE TRUE FALSE

#' yyx\_NA2FALSE(c(1:5,NA,6:10))

#' # 1 2 3 4 5 0 6 7 8 9 10

#'

yyx\_NA2FALSE <- function(vec){

vec[is.na(vec)] <- FALSE

vec

}

#' Convert NA to default value

#'

#' \code{yyx\_convert\_NA} just converts NA in input vector to a specified default value

#'

#' @param vec A vector.

#' @param default The specified default value to replace NA.

#' @return A vector, the same length as input \code{vec}.

#' NAs in vec will be replaced by the specified default value.

#'

#' @seealso \code{\link{yyx\_NA2FALSE}} and \code{\link{yyx\_convert\_NA\_vec}}

#' @family general functions

#'

#' @examples

#' yyx\_convert\_NA(c(TRUE,TRUE,FALSE,NA,TRUE,FALSE,FALSE,TRUE,NA))

#' # TRUE TRUE FALSE FALSE TRUE FALSE FALSE TRUE FALSE

#' yyx\_convert\_NA(c(1:5,NA,6:10))

#' # 1 2 3 4 5 0 6 7 8 9 10

#' yyx\_convert\_NA(c(1:5,NA,6:10), -1)

#' # 1 2 3 4 5 -1 6 7 8 9 10

#'

yyx\_convert\_NA <- function(vec, default=FALSE){

vec[is.na(vec)] <- default

vec

}

#' Convert NA to default values in another vector

#'

#' \code{yyx\_convert\_NA\_vec} just converts NA in input vector to the value in another vector

#'

#' @param vec A vector.

#' @param default\_vec The specified vector, which should be length=1 or the same length of \code{vec}, so that NAs would be converted to the value in the same position in this vector.

#' @return A vector, the same length as input \code{vec}.

#' NAs in vec will be replaced by the value in the same position in \code{default\_vec}.

#'

#' @seealso \code{\link{yyx\_NA2FALSE}} and \code{\link{yyx\_convert\_NA}}

#' @family general functions

#'

#' @examples

#' yyx\_convert\_NA\_vec(c(TRUE,TRUE,FALSE,NA,TRUE,FALSE,FALSE,TRUE,NA), FALSE)

#' # TRUE TRUE FALSE FALSE TRUE FALSE FALSE TRUE FALSE

#' yyx\_convert\_NA\_vec(c(TRUE,TRUE,FALSE,NA,TRUE,FALSE,FALSE,TRUE,NA), 1:9)

#' # 1 1 0 4 1 0 0 1 9

#'

yyx\_convert\_NA\_vec <- function(vec, default\_vec=FALSE){

stopifnot(length(vec)==length(default\_vec) && length(default\_vec)!=1)

default\_vec <- rep(default\_vec, length.out=length(vec))

vec[is.na(vec)] <- default\_vec[is.na(vec)]

vec

}

### math functions

#' Compute log\_sum\_exp (ie. log(sum(exp(vec))) )

#'

#' \code{yyx\_log\_sum\_exp} computes log(sum(exp(vec)))

#'

#' This function is used to compute sum of probability in log-space (Note: not log10)

#'

#' @param vec A numeric vector, usually contains probabilities in log-space.

#' @return A value = log(sum(exp(vec))).

#'

#' @seealso \code{yyx\_log\_add\_exp\_\_c}, \code{yyx\_log\_sum\_exp\_\_c} for Rcpp version

#' @seealso \code{\link{yyx\_log10\_sum\_exp10}}

#' @family math functions

#'

#' @examples

#' yyx\_log\_sum\_exp(-(1:5))

#' # -0.5480856

#' yyx\_log\_sum\_exp(rep(log(0.1),10))

#' # 4.440892e-16

#' yyx\_log\_sum\_exp\_\_c(rep(log(0.1),10))

#' # 4.440892e-16

#'

yyx\_log\_sum\_exp <- function(vec){

vec\_max <- max(vec)

if(is.finite(vec\_max)){

log(sum(exp(vec - vec\_max))) + vec\_max

}else{

vec\_max

}

}

#' Compute log10\_sum\_exp10 (ie. log10(sum(10^(vec))) )

#'

#' \code{yyx\_log10\_sum\_exp10} computes log(sum(exp(vec)))

#'

#' This function is used to compute sum of probability in log10-space

#'

#' @param vec A numeric vector, usually contains probabilities in log10-space.

#' @return A value = log10(sum(10^(vec))).

#'

#' @seealso \code{\link{yyx\_log\_sum\_exp}}

#' @family math functions

#'

#' @examples

#' yyx\_log10\_sum\_exp10(-(1:5))

#' # -0.9542469

#' yyx\_log10\_sum\_exp10(rep(-1,10))

#' # 0

#'

yyx\_log10\_sum\_exp10 <- function(vec){

vec\_max <- max(vec)

if(is.finite(vec\_max)){

log10(sum(10^(vec - vec\_max))) + vec\_max

}else{

vec\_max

}

}

### mosaic model related functions

#' Compute log of P(o|r,q)

#'

#' \code{yyx\_log\_P\_o\_given\_r\_q} computes log of P(o|r,q), given o, q

#'

#' This function uses an iterative algorithm to compute the log-likelihood for each possible real altCount r to the observed altCount o, modelling each base's baseQ (see Reference: \url{???}).

#' I think it was a marvellous "dynamic programming" algorithm coming to my mind, although later we found that LoFreq used a similar algorithm (Reference: \url{???}).

#'

#' Time complexity = O(depth^2); space complexity = O(depth)

#'

#' @param ref\_baseQ\_vec A numeric vector, the phred baseQ vector for ref alleles. (no +33 or +64)

#' @param alt\_baseQ\_vec A numeric vector, the phred baseQ vector for alt alleles. (no +33 or +64)

#' @section Note:

#' Input phred baseQ should be exactly -10\*log(error\_rate), +33 or +64 not allowed

#' @return A vector with length = depth + 1,

#' as a function of discrete 'real' altCount r.

#'

#' @seealso \code{yyx\_log\_P\_o\_given\_r\_q\_\_c} for Rcpp version

#' @family mosaic model related functions

#'

#' @examples

#' yyx\_log\_P\_o\_given\_r\_q(rep(30,15), rep(20,12))

#' yyx\_log\_P\_o\_given\_r\_q\_\_c(rep(30,15), rep(20,12))

#' # [1] -55.2770497 -48.1970106 ... -94.1221201 -103.7369332 [28]

#'

yyx\_log\_P\_o\_given\_r\_q <- function(ref\_baseQ\_vec, alt\_baseQ\_vec){

ref\_count <- length(ref\_baseQ\_vec)

alt\_count <- length(alt\_baseQ\_vec)

depth <- ref\_count + alt\_count

ref\_error\_rate\_vec <- 10^(-ref\_baseQ\_vec/10)

alt\_error\_rate\_vec <- 10^(-alt\_baseQ\_vec/10)

tmpMat <- matrix(0, ncol=2, nrow=depth)

# 1st column: P(o\_i|r\_i=ref) ~ r+=0

# 2nd column: P(o\_i|r\_i=alt) ~ r+=1

# i = 1:ref\_count, o\_i=ref

# i = (ref\_count+1):depth, o\_i=alt

if(ref\_count > 0){

tmpMat[1:ref\_count, 1] <- 1 - ref\_error\_rate\_vec

tmpMat[1:ref\_count, 2] <- ref\_error\_rate\_vec

}

if(alt\_count > 0){

tmpMat[(ref\_count+1):depth, 2] <- 1 - alt\_error\_rate\_vec

tmpMat[(ref\_count+1):depth, 1] <- alt\_error\_rate\_vec

}

# convert to log space

tmpMat <- log(tmpMat)

ans <- rep(-Inf, depth+2)

# ans idx=1:(depth+2) for convenience, return(ans[-1])

# P(o|r,q) as function of r(real\_alt\_count)=0:depth (idx=1:(depth+1))

ans[2:3] <- tmpMat[1,]

for(i in 2:depth){

for(j in seq(i+2,2,by=-1)){

ans[j] <- yyx\_log\_sum\_exp( ans[c(j,j-1)] + tmpMat[i,] )

}

}

return(ans[-1])

}

#' Compute log of P(r|theta,depth)

#'

#' \code{yyx\_log\_P\_r\_given\_theta\_depth} computes log of P(r|theta,depth), given theta, depth

#'

#' This function computes log of P(r|theta,depth) as a function of discrete r.

#' In fact, P(r|theta,depth) = theta^r \* (1-theta)^(depth-r),

#' thus, log P(r|theta,depth) = r \* log(theta) + (depth-r) \* log(1-theta)

#'

#' Time complexity = O(depth); space complexity = O(depth)

#'

#' @param theta A value, the specified theoretical allele fraction.

#' @param depth A value, the specified depth.

#' @return A vector with length = depth + 1,

#' as a function of discrete 'real' altCount r.

#'

#' @seealso \code{yyx\_log\_P\_r\_given\_theta\_depth\_\_c} for Rcpp version

#' @family mosaic model related functions

#'

#' @examples

#' yyx\_log\_P\_r\_given\_theta\_depth(0.3, 20)

#' yyx\_log\_P\_r\_given\_theta\_depth\_\_c(0.3, 20)

#' # [1] -7.133499 -7.980797 ... -23.232158 -24.079456 [21]

#'

yyx\_log\_P\_r\_given\_theta\_depth <- function(theta, depth){

r\_vec <- 0:depth

log\_theta <- log(theta)

if(theta==0) log\_theta <- -1e30

log\_1\_theta <- log(1 - theta)

if(theta==1) log\_1\_theta <- -1e30

r\_vec \* log\_theta + (depth - r\_vec) \* log\_1\_theta

}

#' Compute log of P(o|theta,q)

#'

#' \code{yyx\_log\_P\_o\_given\_theta\_q} computes log of P(o|theta,q), given theta and baseQ vector

#'

#' This function wraps function \code{yyx\_log\_P\_o\_given\_r\_q} and \code{yyx\_log\_P\_r\_given\_theta\_depth}, to compute log of P(o|theta,q),

#' whereas \eqn{ P(o|\theta,q) = \sum\_r P(r|\theta,depth) \* P(o|r,q) }

#'

#' When input \code{log\_P\_o\_given\_r\_q\_vec} is provided, \code{ref\_baseQ\_vec} and \code{ref\_baseQ\_vec} will be ignored and thus can be skipped.

#' As the iterative step (function \code{yyx\_log\_P\_o\_given\_r\_q}) is slow (especially for high depth), it is better to pre-compute log\_P\_o\_given\_r\_q\_vec, when trying several theta for the same site, as shown in the code of funtion \code{yyx\_loglikelihood\_theta\_splinefun}.

#'

#' Time complexity = [ O(depth^2) ] + O(depth); space complexity = O(depth)

#'

#' @param theta A value, the specified theoretical allele fraction.

#' @param ref\_baseQ\_vec A numeric vector, the phred baseQ vector for ref alleles. (no +33 or +64)

#' @param alt\_baseQ\_vec A numeric vector, the phred baseQ vector for alt alleles. (no +33 or +64)

#' @param log\_P\_o\_given\_r\_q\_vec A numeric vector with length = depth + 1, the pre-computed result vector of function \code{yyx\_log\_P\_o\_given\_r\_q}.

#' @return A value = log of P(o|theta,q)

#'

#' @seealso \code{yyx\_log\_P\_o\_given\_theta\_q\_\_c} for Rcpp version

#' @family mosaic model related functions

#'

#' @examples

#' yyx\_log\_P\_o\_given\_theta\_q(0.3, rep(30,15), rep(20,12))

#' yyx\_log\_P\_o\_given\_theta\_q(0.3, log\_P\_o\_given\_r\_q\_vec = yyx\_log\_P\_o\_given\_r\_q(rep(30,15), rep(20,12)))

#' yyx\_log\_P\_o\_given\_theta\_q\_\_c(0.3, yyx\_log\_P\_o\_given\_r\_q\_\_c(rep(30,15), rep(20,12)))

#' # -19.64743

#'

yyx\_log\_P\_o\_given\_theta\_q <- function(theta, ref\_baseQ\_vec, alt\_baseQ\_vec, log\_P\_o\_given\_r\_q\_vec=NULL){

depth <- NA

if(is.null(log\_P\_o\_given\_r\_q\_vec)){

log\_P\_o\_given\_r\_q\_vec <- yyx\_log\_P\_o\_given\_r\_q(ref\_baseQ\_vec, alt\_baseQ\_vec)

depth <- length(ref\_baseQ\_vec) + length(alt\_baseQ\_vec)

}else{

depth <- length(log\_P\_o\_given\_r\_q\_vec) - 1

}

# print(length(log\_P\_o\_given\_r\_q\_vec))

log\_P\_r\_given\_theta\_depth\_vec <- yyx\_log\_P\_r\_given\_theta\_depth(theta, depth)

# print(length(log\_P\_r\_given\_theta\_depth\_vec))

stopifnot(length(log\_P\_o\_given\_r\_q\_vec)==length(log\_P\_r\_given\_theta\_depth\_vec))

yyx\_log\_sum\_exp( log\_P\_o\_given\_r\_q\_vec + log\_P\_r\_given\_theta\_depth\_vec )

}

#' Numerically compute log-likelihood for each theta

#'

#' \code{yyx\_loglikelihood\_theta\_vec} numerically compute log-likelihood for each theta

#'

#' This function wraps function \code{yyx\_log\_P\_o\_given\_r\_q} and \code{yyx\_log\_P\_o\_given\_theta\_q},

#' traverses every theta in \code{theta\_vec},

#' computes log-likelihood for each theta.

#'

#' The time complexity = O(depth^2) + O(#knots \* depth)

#'

#' @param ref\_baseQ\_vec A numeric vector, the phred baseQ vector for ref alleles. (no +33 or +64)

#' @param alt\_baseQ\_vec A numeric vector, the phred baseQ vector for alt alleles. (no +33 or +64)

#' @param theta\_vec A numeric vector, specifies the theta sample points.

#' @return A vector with length = length(theta\_vec),

#' one log-likelihood value for each theta in \code{theta\_vec}.

#'

#' @seealso \code{yyx\_loglikelihood\_theta\_vec\_\_c} for Rcpp version

#' @seealso \code{\link{yyx\_loglikelihood\_theta\_splinefun}} wraps this function and fitted splinefun.

#' @family mosaic model spline-fit functions

#'

#' @examples

#' yyx\_loglikelihood\_theta\_vec(rep(30,15), rep(20,12), seq(0,1,by=0.1))

#' # [1] -55.27705 -28.30124 ... -35.79072 -103.73693 [10]

#' yyx\_loglikelihood\_theta\_vec(rep(30,15), rep(20,12), c(0,0.2,0.5,1))

#' # -55.27705 -22.31696 -18.71497 -103.73693

#'

yyx\_loglikelihood\_theta\_vec <- function(ref\_baseQ\_vec, alt\_baseQ\_vec, theta\_vec){

log\_P\_o\_given\_r\_q\_vec <- yyx\_log\_P\_o\_given\_r\_q(ref\_baseQ\_vec, alt\_baseQ\_vec)

loglikelihood\_vec <- unlist(lapply(theta\_vec, function(theta){ yyx\_log\_P\_o\_given\_theta\_q(theta=theta, log\_P\_o\_given\_r\_q\_vec=log\_P\_o\_given\_r\_q\_vec) }))

stopifnot(length(loglikelihood\_vec)==length(theta\_vec))

loglikelihood\_vec

}

#' Generate spline fitted function of log-likelihood of theta

#'

#' \code{yyx\_loglikelihood\_theta\_splinefun} generates log-likelihood function of theta

#'

#' This function wraps function \code{yyx\_loglikelihood\_theta\_vec},

#' traverses every theta in \code{theta\_vec},

#' computes log-likelihood for each theta,

#' and finally fitted a spline function through these sample points.

#'

#' To avoid the log-likelihood numeric outlier at the boundary (0 and/or 1) when there is any baseQ = +Inf,

#' I first checked the input baseQ and then set the default theta\_vec:

#' \enumerate{

#' \item If no baseQ = +Inf, default theta\_vec = seq(0, 1, length.out=knots);

#' \item If existing ref baseQ = +Inf, the upper-bound was set to 1 - 1/2/knots;

#' \item If existing alt baseQ = +Inf, the lower-bound was set to 1/2/knots.

#' }

#'

#' To avoid underflow in following analysis,

#' I "normalize" the result to make the max log-likelihood = 0 (max likelihood = 1),

#' so only the relative value but not the absolute value is meaningful.

#'

#' The time complexity = O(depth^2) + O(#knots \* depth) + O(splinefun for #knots)

#'

#' @param ref\_baseQ\_vec A numeric vector, the phred baseQ vector for ref alleles. (no +33 or +64)

#' @param alt\_baseQ\_vec A numeric vector, the phred baseQ vector for alt alleles. (no +33 or +64)

#' @param knots An integer. How many theta sample points should be computed?

#' @param theta\_vec A numeric vector, specifies the theta sample points.

#' When \code{theta\_vec} is specified, \code{knots} is ignored

#' @param splinefun\_par A list, any paramters passed to function \code{splinefun}

#' @return An R function (result of splinefun) of theta, restricted to [0,1]

#'

#' @seealso \code{\link{yyx\_loglikelihood\_theta\_vec}} computes the log-likelihood of each theta in any specified theta\_vec.

#' @seealso \code{yyx\_loglikelihood\_theta\_splinefun\_\_call\_c} calls Rcpp version functions.

#' @family mosaic model spline-fit functions

#'

#' @examples

#' tmp\_fun <- yyx\_loglikelihood\_theta\_splinefun(rep(30,15), rep(20,12))

#' curve(tmp\_fun, 0, 1, n=1001L)

#' tmp\_fun <- yyx\_loglikelihood\_theta\_splinefun\_\_call\_c(rep(30,15), rep(20,12))

#' curve(tmp\_fun, 0, 1, n=1001L, add=T, col='red', lty=2)

#' # two exact-overlapping log-likelihood curve

#'

yyx\_loglikelihood\_theta\_splinefun <- function(ref\_baseQ\_vec, alt\_baseQ\_vec, knots = 1000, theta\_vec = NULL, splinefun\_par = list()){

splinefun <- do.call(pryr::partial, c(splinefun, splinefun\_par))

if(is.null(theta\_vec)){

left\_bound <- 0

right\_bound <- 1

warn\_str <- ""

if(any(is.infinite(alt\_baseQ\_vec))){

left\_bound <- 1/2/knots

warn\_str <- paste0(warn\_str, "As there are some alt baseQ = +Inf, I select theta\_vec sampling from theta = ", left\_bound, ".\n")

}

if(any(is.infinite(ref\_baseQ\_vec))){

right\_bound <- 1 - 1/2/knots

warn\_str <- paste0(warn\_str, "As there are some ref baseQ = +Inf, I select theta\_vec sampling to theta = ", right\_bound, ".\n")

}

if(warn\_str != ""){

warn\_str <- paste0(warn\_str, "Thus, outerpolating to theta = 0 and/or 1 might introduce inaccuracy, especially when called by \"yyx\_ref\_het\_alt\_mosaic\_posterior\" or \"yyx\_wrapped\_mosaic\_hunter\_for\_one\_site\".\nThis problem might be relieved by using more knots (eg. 10000).")

warning(warn\_str)

}

theta\_vec <- seq(left\_bound, right\_bound, length.out=knots)

}

if(any(theta\_vec < 0) || any(theta\_vec > 1)){

warning("input \"theta\_vec\" out of [0,1]")

}

# log\_P\_o\_given\_r\_q\_vec <- yyx\_log\_P\_o\_given\_r\_q(ref\_baseQ\_vec, alt\_baseQ\_vec)

# loglikelihood\_vec <- unlist(lapply(theta\_vec, function(theta){ yyx\_log\_P\_o\_given\_theta\_q(theta=theta, log\_P\_o\_given\_r\_q\_vec=log\_P\_o\_given\_r\_q\_vec) }))

loglikelihood\_vec <- yyx\_loglikelihood\_theta\_vec(ref\_baseQ\_vec, alt\_baseQ\_vec, theta\_vec)

loglikelihood\_vec <- loglikelihood\_vec - max(loglikelihood\_vec)

stopifnot(length(theta\_vec)==length(loglikelihood\_vec))

yyx\_restrict\_x\_interval\_function(splinefun(theta\_vec, loglikelihood\_vec), c(0,1), -Inf)

}

#' Generate spline fitted function of likelihood of theta

#'

#' \code{yyx\_likelihood\_theta\_splinefun} generates likelihood function of theta

#'

#' This function wraps function \code{yyx\_loglikelihood\_theta\_splinefun} and \code{yyx\_exp\_function},

#' to get fitted spline function for likelihood function of theta.

#'

#' Calling function \code{yyx\_loglikelihood\_theta\_splinefun} then exponentiate it,

#' to ensure the final value region >= 0.

#'

#' To avoid underflow in following analysis,

#' I "normalize" the result to make the max likelihood = 1,

#' so only the relative value but not the absolute value is meaningful.

#'

#' The time complexity = O(depth^2) + O(#knots \* depth) + O(splinefun for #knots)

#'

#' @param ref\_baseQ\_vec A numeric vector, the phred baseQ vector for ref alleles. (no +33 or +64)

#' @param alt\_baseQ\_vec A numeric vector, the phred baseQ vector for alt alleles. (no +33 or +64)

#' @param knots An integer. How many theta sample points should be computed?

#' @param theta\_vec A numeric vector, specifies the theta sample points.

#' When \code{theta\_vec} is specified, \code{knots} is ignored

#' @param splinefun\_par A list, any paramters passed to function \code{splinefun}

#' @return An R function (result of splinefun) of theta, restricted to [0,1]

#'

#' @seealso \code{\link{yyx\_loglikelihood\_theta\_splinefun}} generates log-likelihood function of theta

#' @seealso \code{yyx\_likelihood\_theta\_splinefun\_\_call\_c} calls Rcpp version functions.

#' @family mosaic model spline-fit functions

#'

#' @examples

#' tmp\_fun <- yyx\_likelihood\_theta\_splinefun(rep(30,15), rep(20,12))

#' curve(tmp\_fun, 0, 1, n=1001L)

#' tmp\_fun <- yyx\_likelihood\_theta\_splinefun\_\_call\_c(rep(30,15), rep(20,12))

#' curve(tmp\_fun, 0, 1, n=1001L, add=T, col='red', lty=2)

#' # two exact-overlapping likelihood curve

#'

yyx\_likelihood\_theta\_splinefun <- function(ref\_baseQ\_vec, alt\_baseQ\_vec, knots = 1000, theta\_vec = NULL, splinefun\_par = list()){

yyx\_exp\_function(yyx\_loglikelihood\_theta\_splinefun(ref\_baseQ\_vec, alt\_baseQ\_vec, knots = knots, theta\_vec = theta\_vec, splinefun\_par = splinefun\_par))

}

#sourceCpp("baseQ\_Rcpp.20140918.cpp")

#' Generate spline fitted function of log-likelihood of theta

#'

#' \code{yyx\_loglikelihood\_theta\_splinefun\_\_call\_c} generates log-likelihood function of theta

#'

#' This function wraps function \code{yyx\_loglikelihood\_theta\_vec\_\_c}

#' traverses every theta in \code{theta\_vec},

#' computes log-likelihood for each theta,

#' and finally fitted a spline function through these sample points.

#'

#' To avoid underflow in following analysis,

#' I "normalize" the result to make the max log-likelihood = 0 (max likelihood = 1),

#' so only the relative value but not the absolute value is meaningful.

#'

#' The time complexity = O(depth^2) + O(#knots \* depth) + O(splinefun for #knots)

#'

#' @section Note:

#' For ultra high depth, uniformly sampled theta might not be the best choice.

#' So user can manually write his/her own function to mostly sample around the peak,

#' call \code{yyx\_loglikelihood\_theta\_vec\_\_c}, and finally fitted by \code{splinefun}.

#'

#' @param ref\_baseQ\_vec A numeric vector, the phred baseQ vector for ref alleles. (no +33 or +64)

#' @param alt\_baseQ\_vec A numeric vector, the phred baseQ vector for alt alleles. (no +33 or +64)

#' @param knots An integer. How many theta sample points should be computed?

#' @param theta\_vec A numeric vector, specifies the theta sample points.

#' When \code{theta\_vec} is specified, \code{knots} is ignored

#' @param splinefun\_par A list, any paramters passed to function \code{splinefun}

#' @return An R function (result of splinefun) of theta, restricted to [0,1]

#'

#' @seealso \code{yyx\_loglikelihood\_theta\_vec\_\_c} for Rcpp version to compute the log-likelihood of each theta in any specified theta\_vec.

#' @seealso \code{\link{yyx\_loglikelihood\_theta\_splinefun}} calls pure R functions.

#' @family mosaic model spline-fit functions

#'

#' @examples

#' tmp\_fun <- yyx\_loglikelihood\_theta\_splinefun\_\_call\_c(rep(30,15), rep(20,12))

#' curve(tmp\_fun, 0, 1, n=1001L, col='red', lty=2)

#' # one log-likelihood curve

#'

yyx\_loglikelihood\_theta\_splinefun\_\_call\_c <- function(ref\_baseQ\_vec, alt\_baseQ\_vec, knots = 1000, theta\_vec = NULL, splinefun\_par = list()){

splinefun <- do.call(pryr::partial, c(splinefun, splinefun\_par))

if(is.null(theta\_vec)){

left\_bound <- 0

right\_bound <- 1

warn\_str <- ""

if(any(is.infinite(alt\_baseQ\_vec))){

left\_bound <- 1/2/knots

warn\_str <- paste0(warn\_str, "As there are some alt baseQ = +Inf, I select theta\_vec sampling from theta = ", left\_bound, ".\n")

}

if(any(is.infinite(ref\_baseQ\_vec))){

right\_bound <- 1 - 1/2/knots

warn\_str <- paste0(warn\_str, "As there are some ref baseQ = +Inf, I select theta\_vec sampling to theta = ", right\_bound, ".\n")

}

if(warn\_str != ""){

warn\_str <- paste0(warn\_str, "Thus, outerpolating to theta = 0 and/or 1 might introduce inaccuracy, especially when called by \"yyx\_ref\_het\_alt\_mosaic\_posterior\" or \"yyx\_wrapped\_mosaic\_hunter\_for\_one\_site\".\nThis problem might be relieved by using more knots (eg. 10000).")

warning(warn\_str)

}

theta\_vec <- seq(left\_bound, right\_bound, length.out=knots)

}

if(any(theta\_vec < 0) || any(theta\_vec > 1)){

warning("input \"theta\_vec\" out of [0,1]")

}

loglikelihood\_vec <- yyx\_loglikelihood\_theta\_vec\_\_c(ref\_baseQ\_vec, alt\_baseQ\_vec, theta\_vec)

loglikelihood\_vec <- loglikelihood\_vec - max(loglikelihood\_vec)

stopifnot(length(theta\_vec)==length(loglikelihood\_vec))

yyx\_restrict\_x\_interval\_function(splinefun(theta\_vec, loglikelihood\_vec), c(0,1), -Inf)

}

#' Generate spline fitted function of likelihood of theta

#'

#' \code{yyx\_likelihood\_theta\_splinefun\_\_call\_c} generates likelihood function of theta

#'

#' This function wraps function \code{yyx\_loglikelihood\_theta\_splinefun\_\_call\_c} and \code{yyx\_exp\_function},

#' to get fitted spline function for likelihood function of theta.

#'

#' Calling function \code{yyx\_loglikelihood\_theta\_splinefun\_\_call\_c} then exponentiate it,

#' to ensure the final value region >= 0

#'

#' To avoid underflow in following analysis,

#' I "normalize" the result to make the max likelihood = 1,

#' so only the relative value but not the absolute value is meaningful.

#'

#' The time complexity = O(depth^2) + O(#knots \* depth) + O(splinefun for #knots)

#'

#' @param ref\_baseQ\_vec A numeric vector, the phred baseQ vector for ref alleles. (no +33 or +64)

#' @param alt\_baseQ\_vec A numeric vector, the phred baseQ vector for alt alleles. (no +33 or +64)

#' @param knots An integer. How many theta sample points should be computed?

#' @param theta\_vec A numeric vector, specifies the theta sample points.

#' When \code{theta\_vec} is specified, \code{knots} is ignored

#' @param splinefun\_par A list, any paramters passed to function \code{splinefun}

#' @return An R function (result of splinefun) of theta, restricted to [0,1]

#'

#' @seealso \code{\link{yyx\_loglikelihood\_theta\_splinefun}} generates log-likelihood function of theta

#' @seealso \code{\link{yyx\_likelihood\_theta\_splinefun}} calls pure R functions.

#' @family mosaic model spline-fit functions

#'

#' @examples

#' tmp\_fun <- yyx\_likelihood\_theta\_splinefun\_\_call\_c(rep(30,15), rep(20,12))

#' curve(tmp\_fun, 0, 1, n=1001L, col='red', lty=2)

#' # one likelihood curve

#'

yyx\_likelihood\_theta\_splinefun\_\_call\_c <- function(ref\_baseQ\_vec, alt\_baseQ\_vec, knots = 1000, theta\_vec = NULL, splinefun\_par = list()){

yyx\_exp\_function(yyx\_loglikelihood\_theta\_splinefun\_\_call\_c(ref\_baseQ\_vec, alt\_baseQ\_vec, knots = knots, theta\_vec = theta\_vec, splinefun\_par =splinefun\_par))

}

### function parser

#' Sweep negative values to be zero

#'

#' \code{yyx\_keep\_positive\_function} wraps an R function and sweeps any negative values to be zero

#'

#' @param f An R function

#' @return An wrapped R function, with original negative results to be zero

#'

#' The parser result also contains two additional attributes: "function\_parser" and "original\_function".

#'

#' @seealso \code{\link{yyx\_restrict\_x\_interval\_function}}, \code{\link{yyx\_restrict\_y\_interval\_function}}

#' @family function parser functions

#'

#' @examples

#' curve(sin(x), 0, 4\*pi)

#' curve(yyx\_keep\_positive\_function(sin)(x), 0, 4\*pi, add=T, col='red', lty=2)

#' # black solid : sin, red dashed : yyx\_keep\_positive\_function(sin)

#'

yyx\_keep\_positive\_function <- function(f){

ans <- function(...){

tmp <- f(...)

tmp[tmp < 0] <- 0

tmp

}

attr(ans, "function\_parser") <- "yyx\_keep\_positive\_function"

attr(ans, "original\_function") <- f

ans

}

#' Exponentiate the value

#'

#' \code{yyx\_exp\_function} wraps an R function and exponentiates the result value

#'

#' If the input function \code{f} is a result from \code{yyx\_log\_function},

#' this function will retrieve and return the "original\_function" attribute.

#'

#' @param f An R function

#' @return An wrapped R function, exponentiates the original result value

#'

#' The parser result also contains two additional attributes: "function\_parser" and "original\_function".

#'

#' @seealso \code{\link{yyx\_log\_function}}

#' @family function parser functions

#'

#' @examples

#' curve(sin(x), 0, 4\*pi, ylim=c(-1,3))

#' curve(yyx\_exp\_function(sin)(x), 0, 4\*pi, add=T, col='red', lty=2)

#' # black solid : sin, red dashed : yyx\_exp\_function(sin)

#'

yyx\_exp\_function <- function(f){

if(!is.null(attr(f, "function\_parser")) && attr(f, "function\_parser")=="yyx\_log\_function"){

return(attr(f, "original\_function"))

}

ans <- function(...){

exp(f(...))

}

attr(ans, "function\_parser") <- "yyx\_exp\_function"

attr(ans, "original\_function") <- f

ans

}

#' Logarithmize the value

#'

#' \code{yyx\_log\_function} wraps an R function and logarithmize the result value

#'

#' If the input function \code{f} is a result from \code{yyx\_exp\_function},

#' this function will retrieve and return the "original\_function" attribute.

#'

#' @param f An R function

#' @return An wrapped R function, logarithmize the original result value

#'

#' The parser result also contains two additional attributes: "function\_parser" and "original\_function".

#'

#' @seealso \code{\link{yyx\_log\_function}}

#' @family function parser functions

#'

#' @examples

#' curve(exp(x), -1, 3, ylim=c(-1,10))

#' curve(yyx\_log\_function(exp)(x), -1, 3, add=T, col='red', lty=2)

#' # black solid : exp, red dashed : yyx\_log\_function(exp)

#'

yyx\_log\_function <- function(f){

if(!is.null(attr(f, "function\_parser")) && attr(f, "function\_parser")=="yyx\_exp\_function"){

return(attr(f, "original\_function"))

}

ans <- function(...){

log(f(...))

}

attr(ans, "function\_parser") <- "yyx\_log\_function"

attr(ans, "original\_function") <- f

ans

}

#' Restrict the x (definitional domain)

#'

#' \code{yyx\_restrict\_x\_interval\_function} wraps an R function and restricts its definitional domain

#'

#' @param f An R function, with only one x

#' @param x\_interval A numeric vector, the specified valid definitional domain

#' @param outside\_value A numeric value, the cast value when x is outside of \code{x\_interval}

#' @return The wrapped R function

#'

#' The parser result also contains two additional attributes: "function\_parser" and "original\_function".

#'

#' @seealso \code{\link{yyx\_restrict\_y\_interval\_function}}, \code{\link{yyx\_keep\_positive\_function}}

#' @family function parser functions

#'

#' @examples

#' curve(sin(x), 0, 4\*pi)

#' curve(yyx\_restrict\_x\_interval\_function(sin, c(1,7), -0.5)(x), 0, 4\*pi, add=T, col='red', lty=2)

#' # black solid : sin, red dashed : yyx\_restrict\_x\_interval\_function(sin, c(1,7), -0.5)

#'

yyx\_restrict\_x\_interval\_function <- function(f, x\_interval, outside\_value=NA){

ans <- function(x, ...){

ans <- rep(outside\_value, length(x))

bool\_idx <- yyx\_is\_within\_range(x, x\_interval)

ans[bool\_idx] <- f(x[bool\_idx], ...)

ans

}

attr(ans, "function\_parser") <- "yyx\_restrict\_x\_interval\_function"

attr(ans, "original\_function") <- f

ans

}

#' Restrict the y (value range)

#'

#' \code{yyx\_restrict\_y\_interval\_function} wraps an R function and restricts its value range

#'

#' @param f An R function

#' @param y\_interval A numeric vector, the specified valid value range

#' @param less\_value A numeric value, the cast value when original y is less than \code{y\_interval}

#' @param greater\_value A numeric value, the cast value when original y is greater than \code{y\_interval}

#' @return The wrapped R function

#'

#' The parser result also contains two additional attributes: "function\_parser" and "original\_function".

#'

#' @seealso \code{\link{yyx\_restrict\_x\_interval\_function}}, \code{\link{yyx\_keep\_positive\_function}}

#' @family function parser functions

#'

#' @examples

#' curve(sin(x), 0, 4\*pi)

#' curve(yyx\_restrict\_y\_interval\_function(sin, c(-0.5,0.5), -0.9)(x), 0, 4\*pi, n=1001L, add=T, col='red', lty=2)

#' # black solid : sin, red dashed : yyx\_restrict\_y\_interval\_function(sin, c(-0.5,0.5), -0.9)

#'

yyx\_restrict\_y\_interval\_function <- function(f, y\_interval, less\_value = min(y\_interval), greater\_value = max(y\_interval)){

y\_interval <- range(y\_interval)

ans <- function(...){

ans <- f(...)

ans[ans>max(y\_interval)] <- greater\_value

ans[ans<min(y\_interval)] <- less\_value

ans

}

attr(ans, "function\_parser") <- "yyx\_restrict\_y\_interval\_function"

attr(ans, "original\_function") <- f

ans

}

#' Multiply two functions

#'

#' \code{yyx\_multiply\_two\_functions} multiply the return value from input two functions

#'

#' @param funA An R function

#' @param funB An R function

#' @return An wrapped R function, with return value = funA(...) \* funB(...)

#'

#' The parser result also contains two additional attributes: "function\_parser" and "original\_function".

#'

#' @family function parser functions

#'

#' @examples

#' curve(sin(x), 0, 4\*pi)

#' curve(cos(x), 0, 4\*pi, add=T, col='blue')

#' curve(yyx\_multiply\_two\_functions(sin,cos)(x), 0, 4\*pi, add=T, col='red', lwd=2, lty=2)

#' curve(sin(x)\*cos(x), 0, 4\*pi, add=T, col='yellow2', lwd=2, lty=3)

#' # black solid : sin, blue solid : cos

#' # red dashed & yellow dashed: yyx\_multiply\_two\_functions(sin,cos) = sin(x) \* cos(x)

#'

yyx\_multiply\_two\_functions <- function(funA, funB){

ans <- function(...){

funA(...) \* funB(...)

}

attr(ans, "function\_parser") <- "yyx\_multiply\_two\_functions"

attr(ans, "original\_function") <- c(funA, funB)

ans

}

### distribution related functions

#' Compute the normalization factor (integral area)

#'

#' \code{yyx\_calculate\_normalization\_factor} computes the normalization factor (integral area)

#' using numeric integration (ie. function \code{integrate})

#'

#' @param f An R function, such as (unnormalized) distribution density function

#' @param integrate\_interval A numeric vector, the specified numeric integration interval of x

#' @param input\_log\_fun A bool value. Is input function in log-spece?

#' @param warn\_negative A bool value. Shouls I throw a warning, when the final result is negative?

#' @param integrate\_par A list, any paramters passed to function \code{integrate}

#' @return A numeric value = normalization factor (integral area)

#'

#' @seealso \code{\link{yyx\_normalize\_distribution\_function}}, \code{\link{yyx\_posterior\_mean}}

#' @family distribution related functions

#'

#' @examples

#' yyx\_calculate\_normalization\_factor(sin, c(0, 2\*pi))

#' # 2.032977e-16

#' yyx\_calculate\_normalization\_factor(dnorm, c(0, 100))

#' # 0.5

#'

yyx\_calculate\_normalization\_factor <- function(f, integrate\_interval, input\_log\_fun = FALSE, warn\_negative = TRUE, integrate\_par=list(rel.tol=.Machine$double.eps^0.5)){

integrate <- do.call(pryr::partial, c(integrate, integrate\_par))

normalization\_factor <- NA

if(input\_log\_fun){ # input\_log\_fun == TRUE

f\_exp <- yyx\_exp\_function(f)

normalization\_factor <- integrate(f\_exp, min(integrate\_interval), max(integrate\_interval), subdivisions = 1000L)$value

if(warn\_negative && normalization\_factor < 0){

warning("The calculated normalization\_factor is positive,\n which is wired for the input log-likelihood function;\n and you may need to check the input function")

}

}else{

normalization\_factor <- integrate(f, min(integrate\_interval), max(integrate\_interval), subdivisions = 1000L)$value

if(warn\_negative && normalization\_factor < 0){

warning("The calculated normalization\_factor is negative,\n which might indicate that you passed in a log-likelihood function\n but not a requested likelihood (density) function;\n and you may try \"input\_log\_fun = TRUE\"")

}

}

normalization\_factor

}

#' Normalize a distribution function

#'

#' \code{yyx\_normalize\_distribution\_function} wraps input distribution function

#' by dividing the normalization factor (calling function \code{yyx\_calculate\_normalization\_factor})

#'

#' @param f An R function, such as (unnormalized) distribution density function

#' @param integrate\_interval A numeric vector, the specified numeric integration interval of x

#' for computing the normalization factor

#' @param input\_log\_fun A bool value. Is input function in log-spece?

#' @param output\_log\_fun A bool value. Should output function be in log-spece?

#' @param warn\_negative A bool value. Shouls I throw a warning, when the final result is negative?

#' @param integrate\_par A list, any paramters passed to function \code{integrate}

#' @return A numeric value = normalization factor (integral area)

#'

#' @seealso \code{\link{yyx\_calculate\_normalization\_factor}}

#' @family distribution related functions

#'

#' @examples

#' curve(dnorm, -10, +10, ylim=c(0,0.9))

#' curve(yyx\_normalize\_distribution\_function(dnorm, c(0,100))(x), -10, +10, add=T, col='red', lty=2)

#' # black solid : dnorm, red dashed : yyx\_normalize\_distribution\_function(dnorm, c(0,100))

#'

yyx\_normalize\_distribution\_function <- function(f, integrate\_interval, input\_log\_fun = FALSE, output\_log\_fun = input\_log\_fun, warn\_negative = TRUE, integrate\_par=list(rel.tol=.Machine$double.eps^0.5)){

if(input\_log\_fun){ # input\_log\_fun == TRUE

normalization\_factor <- yyx\_calculate\_normalization\_factor(f, integrate\_interval, input\_log\_fun = input\_log\_fun, warn\_negative = warn\_negative, integrate\_par = integrate\_par)

if(output\_log\_fun){ # output\_log\_fun == TRUE

function(...){

f(...) - log(normalization\_factor)

}

}else{ # output\_log\_fun == FALSE

function(...){

yyx\_exp\_function(f)(...) / normalization\_factor

}

}

}else{ # input\_log\_fun == FALSE

normalization\_factor <- yyx\_calculate\_normalization\_factor(f, integrate\_interval, input\_log\_fun = input\_log\_fun, warn\_negative = warn\_negative, integrate\_par = integrate\_par)

if(output\_log\_fun){ # output\_log\_fun == TRUE

function(...){

yyx\_log\_function(f)(...) - log(normalization\_factor)

}

}else{ # output\_log\_fun == FALSE

function(...){

f(...) / normalization\_factor

}

}

}

}

#' Compute the mean of the (posterior) distribution

#'

#' \code{yyx\_posterior\_mean} computes the mean of given distribution,

#' by calling function \code{yyx\_calculate\_normalization\_factor}

#'

#' @param f An R function, such as (unnormalized) distribution density function

#' @param integrate\_interval A numeric vector, the specified numeric integration interval of x

#' @param input\_log\_fun A bool value. Is input function in log-spece?

#' @param integrate\_par A list, any paramters passed to function \code{integrate}

#' @return A numeric value = (posterior) mean

#'

#' @seealso \code{\link{yyx\_calculate\_normalization\_factor}}

#' @family distribution related functions

#'

#' @examples

#' yyx\_posterior\_mean(sin, c(0,pi)) \* 2

#' # 3.141593

#' yyx\_posterior\_mean(sin, c(0,2\*pi))

#' # -3.090633e+16

#' yyx\_posterior\_mean(dnorm, c(-100,100))

#' # 0

#' yyx\_posterior\_mean(function(x) dnorm(x, log=T), c(-100,100), input\_log\_fun = T)

#' # 0

#'

yyx\_posterior\_mean <- function(f, integrate\_interval, input\_log\_fun = FALSE, integrate\_par=list(rel.tol=.Machine$double.eps^0.5)){

integrate\_interval <- range(integrate\_interval)

integrate <- do.call(pryr::partial, c(integrate, integrate\_par))

normalization\_factor <- yyx\_calculate\_normalization\_factor(f, integrate\_interval, input\_log\_fun = input\_log\_fun, integrate\_par = integrate\_par)

f\_multiple\_x <- function(x, ...){

x \* f(x, ...)

}

if(input\_log\_fun==TRUE){

f\_multiple\_x <- function(x, ...){

x \* yyx\_exp\_function(f)(x, ...)

}

}

summation\_factor <- yyx\_calculate\_normalization\_factor(f\_multiple\_x, integrate\_interval, input\_log\_fun = FALSE, warn\_negative = FALSE, integrate\_par = integrate\_par)

summation\_factor / normalization\_factor

}

#' Compute the credible interval on distribution function

#'

#' \code{yyx\_get\_credible\_interval} numerically computes the credible interval

#' on a given (unnormalized) distribution function

#'

#' The credible interval was computed by

#' \enumerate{

#' \item normalizing (calling \code{yyx\_calculate\_normalization\_factor})

#' \item finding the maximum (calling \code{optimize}) on log distribution

#' \item finding the left and right bound of credible interval (calling \code{uniroot})

#' }

#'

#' @param f An R function, (unnormalized) distribution density function

#' @param integrate\_interval A numeric vector, the specified numeric integration interval of x

#' @param conf\_level A numeric value in (0,1), confidence level of the interval.

#' @param optimize\_uniroot\_interval\_min A numeric value, the left searching bound for \code{optimize} and \code{uniroot}

#' @param optimize\_uniroot\_interval\_max A numeric value, the right searching bound for \code{optimize} and \code{uniroot}

#' @param alternative A character string specifying the alternative hypothesis,

#' must be one of "two.sided" (default), "greater" or "less".

#' @param optimize\_interval A numeric vector, the specified interval of x for \code{optimize}

#' @param uniroot\_interval A numeric vector, the specified interval of x for \code{uniroot}

#' @param integrate\_par A list, any paramters passed to function \code{integrate}

#' @param optimize\_par A list, any paramters passed to function \code{optimize}

#' @param uniroot\_par A list, any paramters passed to function \code{uniroot}

#' @return A list, containing MLE(= MAP estimate), CI, conf\_level, alternative, posterior\_mean

#'

#' @seealso \code{\link{yyx\_get\_posterior\_p\_value}}, \code{\link{yyx\_calculate\_normalization\_factor}}

#' @family distribution related functions

#'

#' @examples

#' yyx\_get\_credible\_interval(dnorm, c(-100,100))

#' qnorm(c(0.025, 0.975))

#' # -1.959964 1.959964

#'

yyx\_get\_credible\_interval <- function(f, integrate\_interval, conf\_level=0.95, optimize\_uniroot\_interval\_min=-100, optimize\_uniroot\_interval\_max=100, alternative = c("two\_sided", "less", "greater"), optimize\_interval = integrate\_interval, uniroot\_interval = integrate\_interval, integrate\_par=list(rel.tol=.Machine$double.eps^0.5), optimize\_par=list(tol=.Machine$double.eps^0.5), uniroot\_par=list(tol=.Machine$double.eps^0.5)){

integrate\_interval <- range(integrate\_interval)

optimize\_interval <- range(optimize\_interval)

uniroot\_interval <- range(uniroot\_interval)

if(optimize\_interval[1] < optimize\_uniroot\_interval\_min) optimize\_interval[1] <- optimize\_uniroot\_interval\_min

if(optimize\_interval[2] > optimize\_uniroot\_interval\_max) optimize\_interval[2] <- optimize\_uniroot\_interval\_max

if(uniroot\_interval[1] < optimize\_uniroot\_interval\_min) uniroot\_interval[1] <- optimize\_uniroot\_interval\_min

if(uniroot\_interval[2] > optimize\_uniroot\_interval\_max) uniroot\_interval[2] <- optimize\_uniroot\_interval\_max

integrate <- do.call(pryr::partial, c(integrate, integrate\_par))

optimize <- do.call(pryr::partial, c(optimize, optimize\_par))

uniroot <- do.call(pryr::partial, c(uniroot, uniroot\_par))

alternative <- match.arg(alternative)

# normalization\_factor <- integrate(f, integrate\_interval[1], integrate\_interval[2], subdivisions = 1000L)$value

normalization\_factor <- yyx\_calculate\_normalization\_factor(f, integrate\_interval, input\_log\_fun = FALSE, integrate\_par = integrate\_par)

optimize\_max\_result <- optimize(yyx\_log\_function(f), optimize\_interval, maximum=TRUE)

# print(optimize\_max\_result)

mid\_max\_x <- optimize\_max\_result$maximum

if(yyx\_p\_distr(mid\_max\_x, f, integrate\_interval, integrate\_par) - 0.5 >= .Machine$double.eps^0.5){

warning(paste0("the normalization factor (numerical integral) calculated by integrate might be wrong (yyx\_p\_distr != 0.5), and you may try to specify a shorter integrate\_interval to fix it\n"))

}

if(any(abs(mid\_max\_x - optimize\_interval)<=.Machine$double.eps^0.5)){

warning(paste0("the MLE (",mid\_max\_x,") found by optimize might be wrong, and you may specify a shorter optimize\_interval to fix it\n"))

}

get\_right\_x\_given\_left\_x <- function(left\_x, warning\_on = TRUE){

if( f(left\_x) < f(max(uniroot\_interval)) ){

max(uniroot\_interval)

}else if( f(left\_x) > f(mid\_max\_x) ){

if(warning\_on){

warning(paste0("the MLE (",mid\_max\_x,") found by optimize might be wrong, and you may specify a shorter optimize\_interval to fix it\n because the function value of this left\_x (",left\_x,") is greater than the function value of the MLE\n"))

}

mid\_max\_x

}else{

uniroot(function(right\_x){ f(right\_x) - f(left\_x) }, c(mid\_max\_x, uniroot\_interval[2]))$root

}

}

left\_x\_estimate <- right\_x\_estimate <- NA

if(alternative == "two\_sided"){

left\_x\_estimate <- uniroot\_interval[1];

if(integrate(f, left\_x\_estimate, get\_right\_x\_given\_left\_x(left\_x\_estimate))$value / normalization\_factor > conf\_level){

left\_x\_estimate <- uniroot(function(left\_x){ integrate(f, left\_x, get\_right\_x\_given\_left\_x(left\_x, warning\_on=FALSE))$value / normalization\_factor - conf\_level}, c(uniroot\_interval[1], mid\_max\_x))$root

right\_x\_estimate <- get\_right\_x\_given\_left\_x(left\_x\_estimate)

}else{

right\_x\_estimate <- uniroot(function(right\_x){ integrate(f, left\_x\_estimate, right\_x)$value / normalization\_factor - conf\_level}, uniroot\_interval)$root

}

}else if(alternative == "greater"){

right\_x\_estimate <- integrate\_interval[2]

left\_x\_estimate <- uniroot(function(left\_x){ integrate(f, left\_x, right\_x\_estimate)$value / normalization\_factor - conf\_level}, uniroot\_interval)$root

}else if(alternative == "less"){

left\_x\_estimate <- integrate\_interval[1]

right\_x\_estimate <- uniroot(function(right\_x){ integrate(f, left\_x\_estimate, right\_x)$value / normalization\_factor - conf\_level}, uniroot\_interval)$root

}else{

stop(paste0("surprising to be here: unrecognized alternative = \"", alternative, "\""))

}

MLE <- mid\_max\_x

CI <- c(left\_x\_estimate, right\_x\_estimate)

posterior\_mean <- yyx\_posterior\_mean(f, integrate\_interval, input\_log\_fun = FALSE, integrate\_par = integrate\_par)

tmp <- list(MLE=MLE, CI=CI, conf\_level=conf\_level, alternative=alternative, posterior\_mean=posterior\_mean)

tmp

}

#' Compute the "posterior" p-value on a distribution

#'

#' \code{yyx\_get\_posterior\_p\_value} numerically computes the "posterior" p-value

#' on a given (unnormalized) distribution function

#'

#' The "posterior" p-value was computed by

#' \enumerate{

#' \item normalizing (calling \code{yyx\_calculate\_normalization\_factor})

#' \item finding the maximum (calling \code{optimize}) on log distribution

#' \item finding the reflecting position with equal density (calling \code{uniroot})

#' \item compute the two-tailed (or one-tailed) p-value (calling \code{integrate})

#' }

#'

#' @section Note:

#' If given \code{x} is out of \code{integrate\_interval}, strange value might be returned.

#'

#' @param x A numeric value (or vector), the "observed" x (quantile) for p-value computation.

#' @param f An R function, (unnormalized) distribution density function

#' @param integrate\_interval A numeric vector, the specified numeric integration interval of x

#' @param optimize\_uniroot\_interval\_min A numeric value, the left searching bound for \code{optimize} and \code{uniroot}

#' @param optimize\_uniroot\_interval\_max A numeric value, the right searching bound for \code{optimize} and \code{uniroot}

#' @param alternative A character string specifying the alternative hypothesis,

#' must be one of "two.sided" (default), "greater" or "less".

#' @param optimize\_interval A numeric vector, the specified interval of x for \code{optimize}

#' @param uniroot\_interval A numeric vector, the specified interval of x for \code{uniroot}

#' @param integrate\_par A list, any paramters passed to function \code{integrate}

#' @param optimize\_par A list, any paramters passed to function \code{optimize}

#' @param uniroot\_par A list, any paramters passed to function \code{uniroot}

#' @return A numeric value (or vector), the "posterior" p-value(s)

#'

#' @seealso \code{\link{yyx\_get\_credible\_interval}}, \code{\link{yyx\_calculate\_normalization\_factor}}

#' @family distribution related functions

#'

#' @examples

#' yyx\_get\_posterior\_p\_value(0, dnorm, c(-100,100))

#' pnorm(0) \* 2

#' # 1

#' yyx\_get\_posterior\_p\_value(-1, dnorm, c(-100,100))

#' pnorm(-1) \* 2

#' # 0.3173105

#'

yyx\_get\_posterior\_p\_value <- function(x, f, integrate\_interval, optimize\_uniroot\_interval\_min = -100, optimize\_uniroot\_interval\_max = +100, alternative = c("two\_sided", "less", "greater"), optimize\_interval = integrate\_interval, uniroot\_interval = integrate\_interval, integrate\_par=list(rel.tol=.Machine$double.eps^0.5), optimize\_par=list(tol=.Machine$double.eps^0.5), uniroot\_par=list(tol=.Machine$double.eps^0.5)){

integrate\_interval <- range(integrate\_interval)

optimize\_interval <- range(optimize\_interval)

uniroot\_interval <- range(uniroot\_interval)

if(optimize\_interval[1] < optimize\_uniroot\_interval\_min) optimize\_interval[1] <- optimize\_uniroot\_interval\_min

if(optimize\_interval[2] > optimize\_uniroot\_interval\_max) optimize\_interval[2] <- optimize\_uniroot\_interval\_max

if(uniroot\_interval[1] < optimize\_uniroot\_interval\_min) uniroot\_interval[1] <- optimize\_uniroot\_interval\_min

if(uniroot\_interval[2] > optimize\_uniroot\_interval\_max) uniroot\_interval[2] <- optimize\_uniroot\_interval\_max

integrate <- do.call(pryr::partial, c(integrate, integrate\_par))

optimize <- do.call(pryr::partial, c(optimize, optimize\_par))

uniroot <- do.call(pryr::partial, c(uniroot, uniroot\_par))

alternative <- match.arg(alternative)

if(any(!yyx\_is\_within\_range(x, integrate\_interval))){

warning("some given x is out of the integrate\_interval, strange value might be returned.\n")

}

# normalization\_factor <- integrate(f, min(integrate\_interval), max(integrate\_interval), subdivisions = 1000L)$value

normalization\_factor <- yyx\_calculate\_normalization\_factor(f, integrate\_interval, input\_log\_fun = FALSE, integrate\_par = integrate\_par)

optimize\_max\_result <- optimize(yyx\_log\_function(f), optimize\_interval, maximum=TRUE)

# print(optimize\_max\_result)

mid\_max\_x <- optimize\_max\_result$maximum

if(yyx\_p\_distr(mid\_max\_x, f, integrate\_interval, integrate\_par) - 0.5 >= .Machine$double.eps^0.5){

warning(paste0("the normalization factor (numerical integral) calculated by integrate might be wrong (yyx\_p\_distr != 0.5), and you may try to specify a shorter integrate\_interval to fix it\n"))

}

if(any(abs(mid\_max\_x - optimize\_interval)<=.Machine$double.eps^0.5)){

warning(paste0("the MLE (",mid\_max\_x,") found by optimize might be wrong, and you may specify a shorter optimize\_interval to fix it\n"))

}

get\_right\_x\_given\_left\_x <- function(left\_x, warning\_on = TRUE){

if( f(left\_x) < f(max(uniroot\_interval)) ){

max(uniroot\_interval)

}else if( f(left\_x) > f(mid\_max\_x) ){

if(warning\_on){

warning(paste0("the MLE (",mid\_max\_x,") found by optimize might be wrong, and you may specify a shorter optimize\_interval to fix it\n because the function value of this left\_x (",left\_x,") is greater than the function value of the MLE\n"))

}

mid\_max\_x

}else{

uniroot(function(right\_x){ f(right\_x) - f(left\_x) }, c(mid\_max\_x, uniroot\_interval[2]) )$root

}

}

get\_left\_x\_given\_right\_x <- function(right\_x, warning\_on = TRUE){

if( f(right\_x) < f(min(uniroot\_interval)) ){

min(uniroot\_interval)

}else if( f(right\_x) > f(mid\_max\_x) ){

if(warning\_on){

warning(paste0("the MLE (",mid\_max\_x,") found by optimize might be wrong, and you may specify a shorter optimize\_interval to fix it\n because the function value of this right\_x (",right\_x,") is greater than the function value of the MLE\n"))

}

mid\_max\_x

}else{

uniroot(function(left\_x){ f(right\_x) - f(left\_x) }, c(uniroot\_interval[1], mid\_max\_x) )$root

}

}

if(alternative == "two\_sided"){

tmp\_DF <- data.frame(left\_x = x, right\_x = NA)

tmp\_DF$right\_x[tmp\_DF$left\_x < mid\_max\_x] <- unlist(lapply(tmp\_DF$left\_x[tmp\_DF$left\_x < mid\_max\_x], get\_right\_x\_given\_left\_x))

tmp\_DF$right\_x[tmp\_DF$left\_x > mid\_max\_x] <- tmp\_DF$left\_x[tmp\_DF$left\_x > mid\_max\_x]

tmp\_DF$left\_x[tmp\_DF$left\_x > mid\_max\_x] <- unlist(lapply(tmp\_DF$right\_x[tmp\_DF$left\_x > mid\_max\_x], get\_left\_x\_given\_right\_x))

tmp\_DF$right\_x[tmp\_DF$left\_x == mid\_max\_x] <- tmp\_DF$left\_x[tmp\_DF$left\_x == mid\_max\_x]

apply(tmp\_DF, 1, function(tmp\_vec){

( integrate(f, min(integrate\_interval), tmp\_vec[1])$value + integrate(f, tmp\_vec[2], max(integrate\_interval))$value ) / normalization\_factor

})

}else if(alternative == "greater"){

tmp <- unlist(lapply(x, function(this\_x){

integrate(f, this\_x, max(integrate\_interval))$value / normalization\_factor

}))

stopifnot(length(tmp) == length(x))

tmp

}else if(alternative == "less"){

tmp <- unlist(lapply(x, function(this\_x){

integrate(f, min(integrate\_interval), this\_x)$value / normalization\_factor

}))

stopifnot(length(tmp) == length(x))

tmp

}else{

stop(paste0("surprising to be here: unrecognized alternative = \"", alternative, "\""))

}

}

#' Numerical(empirical) cumulative distribution function value(s)

#'

#' \code{yyx\_p\_distr} numerically computes the cumulative distribution function values

#' on a given (unnormalized) distribution function

#'

#' The cumulative distribution function value was computed by

#' \enumerate{

#' \item normalizing (calling \code{yyx\_calculate\_normalization\_factor})

#' \item compute the cumulative distribution function value (calling \code{integrate})

#' }

#'

#' @section Note:

#' If given \code{q} is out of \code{integrate\_interval}, 0 or 1 would be returned.

#'

#' @param q A numeric value (or vector), the quantiles.

#' @param f An R function, (unnormalized) distribution density function

#' @param integrate\_interval A numeric vector, the specified numeric integration interval of x

#' @param integrate\_par A list, any paramters passed to function \code{integrate}

#' @return A numeric value (or vector), the cumulative distribution function value(s)

#'

#' @seealso \code{\link{yyx\_q\_distr}}, \code{\link{yyx\_d\_distr}}

#' @seealso \code{\link{yyx\_get\_credible\_interval}}, \code{\link{yyx\_calculate\_normalization\_factor}}

#' @family distribution related functions

#'

#' @examples

#' yyx\_p\_distr(0, dnorm, c(-100,100))

#' pnorm(0)

#' # 0.5

#' yyx\_p\_distr(-1, dnorm, c(-100,100))

#' pnorm(-1)

#' # 0.1586553

#'

yyx\_p\_distr <- function(q, f, integrate\_interval, integrate\_par=list(rel.tol=.Machine$double.eps^0.5)){

integrate\_interval <- range(integrate\_interval)

integrate <- do.call(pryr::partial, c(integrate, integrate\_par))

normalization\_factor <- yyx\_calculate\_normalization\_factor(f, integrate\_interval, input\_log\_fun = FALSE, integrate\_par = integrate\_par)

tmp <- unlist(lapply(q, function(this\_q){

if(this\_q > integrate\_interval[1]){

if(this\_q < integrate\_interval[2]){

left\_area <- yyx\_calculate\_normalization\_factor(f, c(integrate\_interval[1], this\_q), input\_log\_fun = FALSE, integrate\_par = integrate\_par)

left\_area / normalization\_factor

}else{ # this\_q >= integrate\_interval[2]

1

}

}else{ # this\_q <= integrate\_interval[1]

0

}

}))

stopifnot(length(tmp) == length(q))

tmp

}

#' Numerical(empirical) quantile function value(s)

#'

#' \code{yyx\_q\_distr} numerically computes the quantile function values

#' on a given (unnormalized) distribution function

#'

#' The quantile function value was computed by

#' \enumerate{

#' \item normalizing (calling \code{yyx\_calculate\_normalization\_factor})

#' \item finding the position with the corresponding cumulative probability

#' (calling \code{uniroot} and \code{integrate})

#' \item compute the two-tailed (or one-tailed) p-value (calling \code{integrate})

#' }

#'

#' @section Note:

#' \code{uniroot\_interval} is suggested to be the same as \code{integrate\_interval}

#'

#' @param p A numeric value (or vector), the cumulative probability(ies).

#' @param f An R function, (unnormalized) distribution density function

#' @param integrate\_interval A numeric vector, the specified numeric integration interval of x

#' @param uniroot\_interval A numeric vector, the specified interval of x for \code{uniroot}

#' @param integrate\_par A list, any paramters passed to function \code{integrate}

#' @param uniroot\_par A list, any paramters passed to function \code{uniroot}

#' @return A numeric value (or vector), the quantile values

#'

#' @seealso \code{\link{yyx\_p\_distr}}, \code{\link{yyx\_d\_distr}}

#' @seealso \code{\link{yyx\_get\_credible\_interval}}, \code{\link{yyx\_calculate\_normalization\_factor}}

#' @family distribution related functions

#'

#' @examples

#' yyx\_q\_distr(0.5, dnorm, c(-100,100))

#' qnorm(0.5)

#' # 0

#' yyx\_q\_distr(0.975, dnorm, c(-100,100))

#' qnorm(0.975)

#' # 1.959964

#'

yyx\_q\_distr <- function(p, f, integrate\_interval, uniroot\_interval = integrate\_interval, integrate\_par=list(rel.tol=.Machine$double.eps^0.5), uniroot\_par=list(tol=.Machine$double.eps^0.5)){

integrate\_interval <- range(integrate\_interval)

uniroot\_interval <- range(uniroot\_interval)

integrate <- do.call(pryr::partial, c(integrate, integrate\_par))

uniroot <- do.call(pryr::partial, c(uniroot, uniroot\_par))

normalization\_factor <- yyx\_calculate\_normalization\_factor(f, integrate\_interval, input\_log\_fun = FALSE, integrate\_par = integrate\_par)

tmp <- unlist(lapply(p, function(this\_p){

uniroot(function(x){ integrate(f, min(integrate\_interval), x)$value - normalization\_factor \* this\_p }, uniroot\_interval )$root

}))

stopifnot(length(tmp) == length(p))

tmp

}

#' Numerical(empirical) density function value(s)

#'

#' \code{yyx\_d\_distr} numerically computes the density function values

#' on a given (unnormalized) distribution function

#'

#' The density function value was computed by

#' \enumerate{

#' \item normalizing (calling \code{yyx\_calculate\_normalization\_factor})

#' \item compute the density function value just by calling \code{f}

#' and divide the normalization factor

#' }

#'

#' @param x A numeric value (or vector), the quantiles.

#' @param f An R function, (unnormalized) distribution density function

#' @param integrate\_interval A numeric vector, the specified numeric integration interval of x

#' @param integrate\_par A list, any paramters passed to function \code{integrate}

#' @return A numeric value (or vector), the density function value(s)

#'

#' @seealso \code{\link{yyx\_p\_distr}}, \code{\link{yyx\_q\_distr}}

#' @seealso \code{\link{yyx\_get\_credible\_interval}}, \code{\link{yyx\_calculate\_normalization\_factor}}, \code{\link{yyx\_normalize\_distribution\_function}}

#' @family distribution related functions

#'

#' @examples

#' yyx\_d\_distr(0, dnorm, c(-100,100))

#' yyx\_d\_distr(0, function(x) dnorm(x)\*2, c(-100,100))

#' dnorm(0)

#' # 0.3989423

#' yyx\_d\_distr(1, dnorm, c(-100,100))

#' yyx\_d\_distr(1, function(x) dnorm(x)\*2, c(-100,100))

#' dnorm(1)

#' # 0.2419707

#'

yyx\_d\_distr <- function(x, f, integrate\_interval, integrate\_par=list(rel.tol=.Machine$double.eps^0.5)){

integrate\_interval <- range(integrate\_interval)

integrate <- do.call(pryr::partial, c(integrate, integrate\_par))

normalization\_factor <- yyx\_calculate\_normalization\_factor(f, integrate\_interval, input\_log\_fun = FALSE, integrate\_par = integrate\_par)

tmp <- unlist(lapply(x, function(this\_x){

f(this\_x) / normalization\_factor

}))

stopifnot(length(tmp) == length(x))

tmp

}

#' Generate spline fitted function for the minus random variable

#'

#' \code{yyx\_two\_distribution\_minus} numerically computes and generates spline fitted function for the minus random variable (difference of two independent random variable)

#'

#' The distribution of the difference variable was computed by numerically computing

#' the cross-correlation function of the two distribution function

#' in a straight-forward "brute-force" manner

#'

#' @param funA An R function, (unnormalized) distribution density function of random variable A

#' @param funB An R function, (unnormalized) distribution density function of random variable B

#' @param inputA\_interval A numeric vector, the specified valid definitional domain of random variable A

#' @param inputB\_interval A numeric vector, the specified valid definitional domain of random variable B

#' @param out\_interval A numeric vector, the specified valid definitional domain of output random variable A - B

#' @param knots An integer. How many theta sample points should be computed?

#' @param x\_vec An integer, specifies the output x sample points.

#' When \code{x\_vec} is specified, \code{knots} is ignored

#' @param integrate\_par A list, any paramters passed to function \code{integrate}

#' @param splinefun\_par A list, any paramters passed to function \code{splinefun}

#' @return An R function (result of splinefun) of A - B, restricted to \code{out\_interval}

#'

#' @seealso \code{\link{yyx\_get\_credible\_interval}}, \code{\link{yyx\_calculate\_normalization\_factor}}

#' @family distribution related functions

#'

#' @examples

#' tmp\_funA <- function(x) dnorm(x, mean=1, sd=1)

#' tmp\_funB <- function(x) dnorm(x, mean=0, sd=1.5)

#' curve(tmp\_funA, -10, 10)

#' curve(tmp\_funB, -10, 10, add=T, col='blue')

#' curve(yyx\_two\_distribution\_minus(tmp\_funA, tmp\_funB, c(-100,100), c(-100,100), c(-100,100))(x), -10, 10, add=T, col='red', lty=2, lwd=2)

#' curve(dnorm(x, mean=1, sd=sqrt(1^2+1.5^2)), -10, 10, add=T, col='yellow2', lty=3, lwd=2)

#' # black solid : dnorm(mean=1, sd=1), blue solid : dnorm(mean=0, sd=1.5)

#' # red & yellow dashed : yyx\_two\_distribution\_minus(tmp\_funA, tmp\_funB, c(-100,100), c(-100,100), c(-100,100)) = dnorm(mean=1, sd=sqrt(1^2+1.5^2))

yyx\_two\_distribution\_minus <- function(funA, funB, inputA\_interval=c(0,1), inputB\_interval=c(0,1), out\_interval=c(-1,1), knots=2001, x\_vec = seq(min(out\_interval), max(out\_interval), length.out=knots), integrate\_par=list(rel.tol=.Machine$double.eps^0.5), splinefun\_par=list()){

inputA\_interval <- range(inputA\_interval)

inputB\_interval <- range(inputB\_interval)

integrate <- do.call(pryr::partial, c(integrate, integrate\_par))

splinefun <- do.call(pryr::partial, c(splinefun, splinefun\_par))

tmp\_fun\_for\_one\_x <- function(x){

# valid t\_interval: [A1-x,A2-x] AND [B1,B2]

t\_interval\_min <- max(inputB\_interval[1], inputA\_interval[1]-x)

t\_interval\_max <- min(inputB\_interval[2], inputA\_interval[2]-x)

if(t\_interval\_min < t\_interval\_max){

integrate(

function(t){

funA(t+x)\*funB(t)

}, t\_interval\_min, t\_interval\_max)$value

}else{

0

}

}

y\_vec <- unlist(lapply(x\_vec, tmp\_fun\_for\_one\_x))

stopifnot(length(x\_vec)==length(y\_vec))

yyx\_restrict\_x\_interval\_function(splinefun(x\_vec, y\_vec), out\_interval, 0)

}

### baseQ string parse functions

## reference: http://www.r-bloggers.com/ascii-code-table-in-r/

#' Get ASCII code of a char

#'

#' \code{char\_to\_ascii} converts one char to its ASCII code (integer)

#'

#' @section Reference:

#' \url{http://www.r-bloggers.com/ascii-code-table-in-r/}

#'

#' @param ch A one-char character string

#' @return An integer, the ASCII code of the input \code{ch}

#'

#' @seealso \code{\link{yyx\_baseQ\_string\_to\_phred\_vec}}

#' @family baseQ string parse functions

#'

#' @examples

#' char\_to\_ascii("A")

#' # 65 # = 64 + 1

#' char\_to\_ascii("a")

#' # 97 # = 96 + 1

#'

char\_to\_ascii <- function(ch){

strtoi(charToRaw(ch),16L)

}

#' Convert phred baseQ string to the phred baseQ vector

#'

#' \code{yyx\_baseQ\_string\_to\_phred\_vec} converts phred baseQ string to the phred baseQ (integer) vector, by calling function \code{char\_to\_ascii}

#'

#' @param str A character string, the phred baseQ string (coded in +33 or +64)

#' @param phred\_shift An integer, the specified coding shift (+33 or +64)

#' @return A numeric vector containing integers, the phred baseQ vector

#'

#' @seealso \code{\link{char\_to\_ascii}}

#' @family baseQ string parse functions

#'

#' @examples

#' yyx\_baseQ\_string\_to\_phred\_vec("ABBCCDBA")

#' # 32 33 33 34 34 35 33 32

#'

yyx\_baseQ\_string\_to\_phred\_vec <- function(str, phred\_shift = 33){

if(length(str)>1) warning("input \"str\" has length > 1, only the first item will be converted")

if(!is.character(str[1])) stop("input \"str\" seems not to be a character string")

tmp <- unlist(lapply(strsplit(str[1], "")[[1]], char\_to\_ascii))

stopifnot(length(tmp)==nchar(str))

tmp - phred\_shift

}

### mosaic posterior (numeric) functions

#' Compute the posterior probabilities of 4 genotypes

#'

#' \code{yyx\_ref\_het\_alt\_mosaic\_posterior} computes the posterior probabilities of 4 genotypes

#' (ref-hom, het, alt-hom, mosaic) (not in log-space)

#'

#' @param likelihood\_fun An R function, the likelihood function of theoretical allele fraction theta

#' @param ref\_het\_alt\_mosaic\_prior A length=4 numeric vector, specifies the prior of 4 genotypes

#' (default = c(1,1,1,1), ie. posterior = "normalized" likelihood)

#' @param het\_beta\_prior\_param A length=2 numeric vector, specifies parameters for the het beta distribution

#' (recently only useful in exome sequencing)

#' @param output\_log10\_fun A bool value. Should output function be in log10-spece?

#' @param integrate\_par A list, any paramters passed to function \code{integrate}

#' @return A length=4 numeric vector, the posterior probabilities of 4 genotypes

#'

#' @seealso \code{\link{yyx\_wrapped\_mosaic\_hunter\_for\_one\_site}}, \code{\link{yyx\_likelihood\_theta\_splinefun\_\_call\_c}}, \code{\link{yyx\_calculate\_normalization\_factor}}

#' @family mosaic posterior functions

#'

#' @examples

#' tmp\_fun <- yyx\_likelihood\_theta\_splinefun\_\_call\_c(rep(20, 60), rep(20, 40))

#' yyx\_ref\_het\_alt\_mosaic\_posterior(tmp\_fun)

#' # 3.590668e-51 5.176832e-01 4.390069e-91 4.823168e-01

#' yyx\_ref\_het\_alt\_mosaic\_posterior(tmp\_fun, output\_log10 = TRUE)

#' # -50.4448247 -0.2859360 -90.3575286 -0.3166676

#' yyx\_ref\_het\_alt\_mosaic\_posterior(tmp\_fun, ref\_het\_alt\_mosaic\_prior = c(1,2e-4,1e-8,1e-7), het\_beta\_prior\_param = c(25,30))

#' # 9.585144e-48 9.998712e-01 1.171911e-95 1.287525e-04

#'

yyx\_ref\_het\_alt\_mosaic\_posterior <- function(likelihood\_fun, ref\_het\_alt\_mosaic\_prior = c(1,1,1,1), het\_beta\_prior\_param = NULL, output\_log10 = FALSE, integrate\_par = list(rel.tol=.Machine$double.eps^0.5)){

if(length(ref\_het\_alt\_mosaic\_prior)!=4){

stop("input \"ref\_het\_alt\_mosaic\_prior\" should have length=4, prior probability (not log10) for ref-hom, het, alt-hom, mosaic")

}

mosaic\_likelihood <- yyx\_calculate\_normalization\_factor(likelihood\_fun, c(0,1), integrate\_par = integrate\_par)

ref\_hom\_log10likelihood <- yyx\_log\_function(likelihood\_fun)(0) / log(10)

alt\_hom\_log10likelihood <- yyx\_log\_function(likelihood\_fun)(1) / log(10)

het\_log10likelihood <- yyx\_log\_function(likelihood\_fun)(0.5) / log(10)

mosaic\_log10likelihood <- log(mosaic\_likelihood) / log(10)

ref\_hom\_likelihood <- 10 ^ ref\_hom\_log10likelihood

alt\_hom\_likelihood <- 10 ^ alt\_hom\_log10likelihood

het\_likelihood <- 10 ^ het\_log10likelihood

if(!is.null(het\_beta\_prior\_param)){

if(length(het\_beta\_prior\_param) == 2){

tmp\_beta\_distr <- function(x) dbeta(x, shape1 = het\_beta\_prior\_param[1], shape2 = het\_beta\_prior\_param[2])

het\_distr <- yyx\_multiply\_two\_functions(likelihood\_fun, tmp\_beta\_distr)

het\_likelihood <- yyx\_calculate\_normalization\_factor(het\_distr, c(0,1), integrate\_par = integrate\_par)

het\_log10likelihood <- log(het\_likelihood) / log(10)

}else{

stop("input het\_beta\_prior\_param should be a length=2 vector, containing alpha and beta parameters for the beta distribution.")

}

}

if(output\_log10){

tmp\_vec <- c(ref\_hom\_log10likelihood, het\_log10likelihood, alt\_hom\_log10likelihood, mosaic\_log10likelihood)

tmp\_vec <- tmp\_vec + log10(ref\_het\_alt\_mosaic\_prior)

return( tmp\_vec - yyx\_log10\_sum\_exp10(tmp\_vec) )

}else{

tmp\_vec <- c(ref\_hom\_likelihood, het\_likelihood, alt\_hom\_likelihood, mosaic\_likelihood)

tmp\_vec <- tmp\_vec \* ref\_het\_alt\_mosaic\_prior

return( tmp\_vec / sum(tmp\_vec) )

}

}

#' Compute the likelihood function and posterior probabilities for one site

#'

#' \code{yyx\_wrapped\_mosaic\_hunter\_for\_one\_site} wraps function \code{yyx\_baseQ\_string\_to\_phred\_vec},

#' \code{yyx\_likelihood\_theta\_splinefun\_\_call\_c}, and \code{yyx\_ref\_het\_alt\_mosaic\_posterior},

#' as a pipeline function for one NGS site

#'

#' @param ref\_baseQ\_str The ref baseQ phred string (when input\_type = "string") or integer vector (when input\_type = "integer\_vector")

#' @param alt\_baseQ\_str The alt baseQ phred string (when input\_type = "string") or integer vector (when input\_type = "integer\_vector")

#' @param input\_type A character string specifying the input type of \code{ref\_baseQ\_str} and \code{alt\_baseQ\_str},

#' must be one of "string" (default) or "integer\_vector".

#' @param phred\_shift An integer, the specified coding shift (+33 or +64)

#' @param knots An integer. How many theta sample points should be computed?

#' @param theta\_vec A numeric vector, specifies the theta sample points.

#' When \code{theta\_vec} is specified, \code{knots} is ignored.

#' (default see \code{\link{yyx\_loglikelihood\_theta\_splinefun}})

#' @param ref\_het\_alt\_mosaic\_prior A length=4 numeric vector, specifies the prior of 4 genotypes

#' (default = c(1,1,1,1), ie. posterior = "normalized" likelihood)

#' @param het\_beta\_prior\_param A length=2 numeric vector, specifies parameters for the het beta distribution

#' (recently only useful in exome sequencing)

#' @param output\_log10\_fun A bool value. Should output function be in log10-spece?

#' @param integrate\_par A list, any paramters passed to function \code{integrate}

#' @param splinefun\_par A list, any paramters passed to function \code{splinefun}

#' @return A list, contains \code{ref\_het\_alt\_mosaic\_posterior} and \code{likelihood\_fun}.

#' \describe{

#' \item{\code{ref\_het\_alt\_mosaic\_posterior}}{ is a length=4 numeric vector, the posterior probabilities of 4 genotypes }

#' \item{\code{likelihood\_fun}}{ is the R function of the spline fitted likelihood function of theta }

#' }

#'

#' @seealso \code{\link{yyx\_wrapped\_mosaic\_hunter\_for\_one\_site}}, \code{yyx\_likelihood\_theta\_splinefun\_\_call\_c}, \code{\link{yyx\_calculate\_normalization\_factor}}

#' @family mosaic posterior functions

#'

#' @examples

#' yyx\_wrapped\_mosaic\_hunter\_for\_one\_site("AAAAAAAAAA", "DDDDDDDD")

#' # $ref\_het\_alt\_mosaic\_posterior 1.980340e-23 7.602227e-01 1.987842e-27 2.397773e-01

#' yyx\_wrapped\_mosaic\_hunter\_for\_one\_site(rep(20, 60), rep(20, 40), input\_type = "integer\_vector")

#' # $ref\_het\_alt\_mosaic\_posterior 3.590668e-51 5.176832e-01 4.390069e-91 4.823168e-01

#' yyx\_wrapped\_mosaic\_hunter\_for\_one\_site(rep(20, 60), rep(20, 40), input\_type = "integer\_vector", het\_beta\_prior\_param=c(25,30))

#' # 1.524626e-51 7.952045e-01 1.864058e-91 2.047955e-01

#' yyx\_wrapped\_mosaic\_hunter\_for\_one\_site(rep(20, 60), rep(20, 40), input\_type = "integer\_vector", ref\_het\_alt\_mosaic\_prior=c(1,0,1e-4,1e-7), output\_log10 = T)

#' # -43.12816 -Inf -87.04086 0.00000

#'

yyx\_wrapped\_mosaic\_hunter\_for\_one\_site <- function(ref\_baseQ\_str, alt\_baseQ\_str, input\_type = "string", phred\_shift = 33, knots = 1000, theta\_vec = NULL, ref\_het\_alt\_mosaic\_prior = c(1,1,1,1), het\_beta\_prior\_param = NULL, output\_log10 = FALSE, integrate\_par = list(rel.tol=.Machine$double.eps^0.5), splinefun\_par = list()){

ref\_baseQ\_vec <- ref\_baseQ\_str

alt\_baseQ\_vec <- alt\_baseQ\_str

if(input\_type == "string"){

ref\_baseQ\_vec <- yyx\_baseQ\_string\_to\_phred\_vec(ref\_baseQ\_str, phred\_shift = phred\_shift)

alt\_baseQ\_vec <- yyx\_baseQ\_string\_to\_phred\_vec(alt\_baseQ\_str, phred\_shift = phred\_shift)

}else if(input\_type == "integer\_vector"){

# good, do nothing

}else{

stop("Unknown input\_type, which should be \"string\" or \"integer\_vector\"")

}

likelihood\_fun <- yyx\_likelihood\_theta\_splinefun\_\_call\_c(ref\_baseQ\_vec, alt\_baseQ\_vec, knots = knots, theta\_vec = theta\_vec, splinefun\_par = splinefun\_par)

ref\_het\_alt\_mosaic\_posterior <- yyx\_ref\_het\_alt\_mosaic\_posterior(likelihood\_fun, ref\_het\_alt\_mosaic\_prior=ref\_het\_alt\_mosaic\_prior, het\_beta\_prior\_param = het\_beta\_prior\_param, output\_log10 = output\_log10, integrate\_par = integrate\_par)

list(ref\_het\_alt\_mosaic\_posterior = ref\_het\_alt\_mosaic\_posterior, likelihood\_fun = likelihood\_fun)

}

3. 混样产生已知“真”嵌合位点的模拟数据

3-1. 将sam文件（bam的非压缩文本格式）两两按conf\_filename文件中指定比例进行二项抽样置换读段（sample1\_sam\_filename中的读段置换为sample2\_sam\_filename中的读段）的混样Perl脚本

#!/usr/bin/perl

use strict;

use warnings;

my $temp\_dirname = 'temp';

my $tool\_dirname = 'tools';

my $version = '

Version: 0.1.5 (2017-04-08)

Author: Adam Yongxin Ye @ CBI

';

my $usage = "

Usage: $0

<sam\_root\_dir> <conf\_filename> <output\_merge\_sam\_filename>

<output\_replaced\_read\_name\_filename> <output\_log\_filename>

[seed\_for\_R] [should\_log\_normal]

Input:

The conf\_file should have 3 columns:

sample1\_sam\_filename, sample2\_sam\_filename (from sam\_root\_dir)

mixture\_proportion\_for\_sample2

seed\_for\_R should be an integer for set.seed() in R [default = 3]

should\_log\_normal: should I log seed\_for\_R and successful lines

in output\_log\_filename? [default = 0]

Output:

Output output\_merge\_sam\_filename, output\_log\_filename

for unsuccessful lines, you can just grep

'No enough sample2 read to retrieve on this site'

in output\_log\_filename

".$version;

my($sam\_root\_dirname, $conf\_filename, $output\_merge\_sam\_filename, $output\_replaced\_read\_name\_filename, $output\_log\_filename, $seed\_for\_R, $should\_log\_normal);

if(@ARGV<5){

die $usage;

}else{

($sam\_root\_dirname, $conf\_filename, $output\_merge\_sam\_filename, $output\_replaced\_read\_name\_filename, $output\_log\_filename, $seed\_for\_R, $should\_log\_normal) = @ARGV;

}

if(!defined($seed\_for\_R)){

$seed\_for\_R = 3;

}

if(!defined($should\_log\_normal)){

$should\_log\_normal = 0;

}

my $start\_time = time();

print STDERR "[PERL-START] ".scalar(localtime())."\n";

my $step\_start\_time = time();

# step1. read in conf file

print STDERR "[PROCESS] Step 1. read in conf file ...\n";

open(CONFIG, $conf\_filename) or die "Error: cannot open conf\_file $conf\_filename for input\n";

my (@sample1\_sam\_filename, @sample2\_sam\_filename, @mixture\_proportion\_for\_sample2, @original\_sample1\_sam\_filename, @original\_sample2\_sam\_filename) = ((),(),(),(),());

my @F;

my $i = 0;

while(<CONFIG>){

chomp;

if(/^$/){ next; }

@F = split/\t/;

$sample1\_sam\_filename[$i] = $sam\_root\_dirname.'/'.$F[0];

$sample2\_sam\_filename[$i] = $sam\_root\_dirname.'/'.$F[1];

$mixture\_proportion\_for\_sample2[$i] = $F[2];

$original\_sample1\_sam\_filename[$i] = $F[0];

$original\_sample2\_sam\_filename[$i] = $F[1];

$i++;

}

close(CONFIG);

my $input\_sam\_file\_count = $i;

check\_elapsed\_time($step\_start\_time, "[STEP-ELAPSED-TIME]");

$step\_start\_time = time();

# generate a temp random string for R input, and output filename

print STDERR "[PROCESS] Step 2.1. generate temp R input tsv file ...\n";

my @ch = (0..9, 'a'..'z', 'A'..'Z');

my $rand\_str = join '', map { $ch[int rand @ch] } 0..24;

my $R\_script\_filename = $tool\_dirname.'/Yyx\_two\_sam\_binomial\_mixer.r';

my $R\_input\_filename = $temp\_dirname.'/Yyx\_proportion\_merge\_two\_bam.'.$rand\_str.'.input.tsv';

my $R\_output\_filename = $temp\_dirname.'/Yyx\_proportion\_merge\_two\_bam.'.$rand\_str.'.output.tsv';

my (@depth1, @depth2);

open(R\_INPUT, '>'.$R\_input\_filename) or die "Error: cannot open $R\_input\_filename for output\n";

#my ($depth1, $depth2);

my $command;

my (%read\_name\_to\_line\_0, %read\_name\_to\_line\_1, %read\_name\_to\_line\_2);

my ($read\_name, $flag);

my (@read\_names\_0, @read\_names\_1, @read\_names\_2, @read\_names);

for($i=0; $i<$input\_sam\_file\_count; $i++){

$depth1[$i] = 0; $depth2[$i] = 0;

open(INPUT, $sample1\_sam\_filename[$i]) or die "Error: cannot open ".$sample1\_sam\_filename[$i]." for input\n";

(%read\_name\_to\_line\_0, %read\_name\_to\_line\_1, %read\_name\_to\_line\_2) = ((),(),());

while(<INPUT>){

if(/^$/ || /^@/){ next; }

chomp;

@F = split/\t/;

($read\_name, $flag) = @F[0,1];

if($flag & 0x40){ # PE 1

$read\_name\_to\_line\_1{$read\_name} = $\_;

}elsif($flag & 0x80){ # PE 2

$read\_name\_to\_line\_2{$read\_name} = $\_;

}else{ # SE

$read\_name\_to\_line\_0{$read\_name} = $\_;

}

}

close(INPUT);

@read\_names\_0 = sort keys %read\_name\_to\_line\_0;

@read\_names\_1 = sort keys %read\_name\_to\_line\_1;

@read\_names\_2 = sort keys %read\_name\_to\_line\_2;

@read\_names = (@read\_names\_0, unique(@read\_names\_1, @read\_names\_2));

$depth1[$i] = scalar(@read\_names);

open(INPUT, $sample2\_sam\_filename[$i]) or die "Error: cannot open $sample2\_sam\_filename[$i] for input\n";

(%read\_name\_to\_line\_0, %read\_name\_to\_line\_1, %read\_name\_to\_line\_2) = ((),(),());

while(<INPUT>){

if(/^$/ || /^@/){ next; }

chomp;

@F = split/\t/;

($read\_name, $flag) = @F[0,1];

if($flag & 0x40){ # PE 1

$read\_name\_to\_line\_1{$read\_name} = $\_;

}elsif($flag & 0x80){ # PE 2

$read\_name\_to\_line\_2{$read\_name} = $\_;

}else{ # SE

$read\_name\_to\_line\_0{$read\_name} = $\_;

}

}

close(INPUT);

@read\_names\_0 = sort keys %read\_name\_to\_line\_0;

@read\_names\_1 = sort keys %read\_name\_to\_line\_1;

@read\_names\_2 = sort keys %read\_name\_to\_line\_2;

@read\_names = (@read\_names\_0, unique(@read\_names\_1, @read\_names\_2));

$depth2[$i] = scalar(@read\_names);

print R\_INPUT join("\t", $depth1[$i], $depth2[$i], $mixture\_proportion\_for\_sample2[$i])."\n";

}

close(R\_INPUT);

check\_elapsed\_time($step\_start\_time, "[STEP-ELAPSED-TIME]");

$step\_start\_time = time();

# execute R script to get binomial sampling for mixing

print STDERR "[PROCESS] Step 2.2. execute R script $R\_script\_filename ...\n";

$command = 'Rscript '.$R\_script\_filename.' '.$R\_input\_filename.' '.$R\_output\_filename.' '.$seed\_for\_R;

if(system("/bin/bash", "-c", $command)!=0){

die "Error: when exec $command, return value = $?\n";

}

check\_elapsed\_time($step\_start\_time, "[STEP-ELAPSED-TIME]");

$step\_start\_time = time();

# read in the mixing sampling result

print STDERR "[PROCESS] Step 2.3. read in temp R output tsv file ...\n";

open(R\_OUTPUT, $R\_output\_filename) or die "Error: cannot open $R\_output\_filename for input\n";

my (@sample1\_line\_idxs, @sample2\_line\_idxs) = ((),());

$i = 0;

while(<R\_OUTPUT>){

chomp;

@F = split/\t/;

$sample1\_line\_idxs[$i] = $F[0];

$sample2\_line\_idxs[$i] = $F[1];

$i++;

}

close(R\_OUTPUT);

unlink($R\_input\_filename);

unlink($R\_output\_filename);

check\_elapsed\_time($step\_start\_time, "[STEP-ELAPSED-TIME]");

$step\_start\_time = time();

# clear output files: output\_merge\_sam\_filename output\_replaced\_read\_name\_filename (for appending initialization)

$command = 'rm -f '.$output\_replaced\_read\_name\_filename.' '.$output\_merge\_sam\_filename;

if(system("/bin/bash", "-c", $command)!=0){

warn "Warning: when exec $command, return value = $?\n";

}

# really mix those two sam file

print STDERR "[PROCESS] Step 3. actually mix sam using Yyx\_mix\_two\_sam\_by\_read\_name\_idxs() ...\n";

my $output\_merge\_sam\_fileHandle;

open($output\_merge\_sam\_fileHandle, ">", $output\_merge\_sam\_filename) or die "Error: cannot open $output\_merge\_sam\_filename for output\n";

my $mix\_by\_line\_pl\_filename = "Yyx\_mix\_two\_sam\_by\_read\_name\_idxs.pl";

my $log\_fileHandle;

open($log\_fileHandle, '>'.$output\_log\_filename) or warn "Warning: cannot open $output\_log\_filename for log output\n";

if($should\_log\_normal && defined($log\_fileHandle)){

print {$log\_fileHandle} "seed\_for\_R = ".$seed\_for\_R."\n";

}

my ($retrieve\_depth1, $retrieve\_depth2);

for($i=0; $i<$input\_sam\_file\_count; $i++){

if($sample1\_line\_idxs[$i] eq '-' && $sample2\_line\_idxs[$i] eq '-'){ # sample2 does not have enough depth, so skip this site...

if(defined($log\_fileHandle)){

print {$log\_fileHandle} "line ".($i+1)." :\t".$original\_sample1\_sam\_filename[$i]."\t".$original\_sample2\_sam\_filename[$i]."\t".$mixture\_proportion\_for\_sample2[$i]."\tNo enough sample2 read to retrieve on this site\n";

}

print STDERR "line ".($i+1)." :\t".$original\_sample1\_sam\_filename[$i]."\t".$original\_sample2\_sam\_filename[$i]."\t".$mixture\_proportion\_for\_sample2[$i]."\tNo enough sample2 read to retrieve on this site\n";

}else{

print $output\_merge\_sam\_fileHandle Yyx\_mix\_two\_sam\_by\_read\_name\_idxs($sample1\_sam\_filename[$i], $sample2\_sam\_filename[$i], $sample1\_line\_idxs[$i], $sample2\_line\_idxs[$i], 0, $output\_replaced\_read\_name\_filename);

# $command = 'perl '.$mix\_by\_line\_pl\_filename.' '.$sample1\_sam\_filename[$i].' '.$sample2\_sam\_filename[$i].' '.$sample1\_line\_idxs[$i].' '.$sample2\_line\_idxs[$i].' 0 '.$output\_replaced\_read\_name\_filename.' >>'.$output\_merge\_sam\_filename;

# if(system("/bin/bash", "-c", $command)!=0){

# warn "Warning: when exec $command, return value = $?\n";

# }

if($should\_log\_normal && defined($log\_fileHandle)){

if($sample1\_line\_idxs[$i] eq '-'){

$retrieve\_depth1 = 0;

}else{

$retrieve\_depth1 = scalar(split(/,/,$sample1\_line\_idxs[$i]));

}

if($sample2\_line\_idxs[$i] eq '-'){

$retrieve\_depth2 = 0;

}else{

$retrieve\_depth2 = scalar(split(/,/,$sample2\_line\_idxs[$i]));

}

print {$log\_fileHandle} "line ".($i+1)." :\t".$original\_sample1\_sam\_filename[$i]."\t".$original\_sample2\_sam\_filename[$i]."\t".$mixture\_proportion\_for\_sample2[$i]."\t".$retrieve\_depth1." / ".$depth1[$i]."\t".$retrieve\_depth2." / ".$depth2[$i]."\n";

}

}

}

close($log\_fileHandle);

close($output\_merge\_sam\_fileHandle);

check\_elapsed\_time($step\_start\_time, "[STEP-ELAPSED-TIME]");

print STDERR "[PERL-END] ".scalar(localtime())."\n";

check\_elapsed\_time($start\_time, "[TOTAL-ELAPSED-TIME]");

0;

sub unique{

my @input = @\_;

my %tmp\_hash = ();

foreach (@input){

$tmp\_hash{$\_} = 1;

}

return sort keys %tmp\_hash;

}

sub check\_elapsed\_time{

my ($start\_time, $end\_time, $elapsed\_time);

my $prompt\_str = "[PERL-SYSTEM-TIME]";

my ($hour, $min, $sec, $day);

$start\_time = shift(@\_);

if(scalar(@\_)>0){ $prompt\_str = shift(@\_); }

$end\_time = time();

$elapsed\_time = $end\_time - $start\_time;

$day = int($elapsed\_time / (3600\*24));

$hour = int($elapsed\_time % (3600\*24) / 3600);

$min = int($elapsed\_time % 3600 / 60);

$sec = $elapsed\_time % 60;

$elapsed\_time = "";

if($day>0){ $elapsed\_time .= $day."day "; }

if($hour>0){ $elapsed\_time .= $hour."h"; }

if($min>0){ $elapsed\_time .= $min."min"; }

if($sec>0 || $elapsed\_time eq ""){ $elapsed\_time .= $sec."s"; }

print STDERR $prompt\_str." ".$elapsed\_time."\n";

}

## copy from Yyx\_mix\_two\_sam\_by\_read\_name\_idxs.pl

sub Yyx\_mix\_two\_sam\_by\_read\_name\_idxs{

my $version = '

Version: 0.1.0 (2013-06-20)

Author: Adam\_YeYongxin, CBI

';

my $usage = "

Usage: $0 <sample1\_sam> <sample2\_sam>

<sample1\_read\_name\_idxs> <sample2\_read\_name\_idxs> [need\_sam\_head] [sample1\_read\_name\_filename]

Input:

sample1\_sam and sample2\_sam could be unsorted by read\_name, could contain SE and PE reads

sample1\_line\_idxs and sample2\_line\_idxs should be separated by ','

need\_sam\_head: should I output sam head of sample1\_bam? [default = 0]

sample1\_read\_name\_filename: appended, will contain all read\_name in sample1 [default = NULL]

Output:

STDOUT (sam format): mixed sam

sample1\_read\_name\_filename (append): all read\_name in sample1\_sam

".$version;

my ($sample1\_sam\_filename, $sample2\_sam\_filename, $sample1\_line\_idxs, $sample2\_line\_idxs, $need\_sam\_head, $log\_filename);

if(@\_<4){

die $usage;

}else{

($sample1\_sam\_filename, $sample2\_sam\_filename, $sample1\_line\_idxs, $sample2\_line\_idxs, $need\_sam\_head, $log\_filename) = @\_;

}

if(!defined($need\_sam\_head)){

$need\_sam\_head = 1;

}

my (@sample1\_line\_idx\_vec, @sample2\_line\_idx\_vec) = ((),());

if($sample1\_line\_idxs ne '-'){

@sample1\_line\_idx\_vec = split(/,/, $sample1\_line\_idxs);

}

if($sample2\_line\_idxs ne '-'){

@sample2\_line\_idx\_vec = split(/,/, $sample2\_line\_idxs);

}

my $ans;

open(MIX\_INPUT\_1, $sample1\_sam\_filename) or die "Error: cannot open $sample1\_sam\_filename for input\n";

my (%read\_name\_to\_line\_0, %read\_name\_to\_line\_1, %read\_name\_to\_line\_2) = ((),(),());

my @F;

my ($read\_name, $flag);

while(<MIX\_INPUT\_1>){

if(/^$/){ next; }

if(/^@/){

if($need\_sam\_head){

# print STDOUT $\_;

$ans .= $\_;

}

next;

}

chomp;

@F = split/\t/;

($read\_name, $flag) = @F[0,1];

if($flag & 0x40){ # PE 1

$read\_name\_to\_line\_1{$read\_name} = $\_;

}elsif($flag & 0x80){ # PE 2

$read\_name\_to\_line\_2{$read\_name} = $\_;

}else{ # SE

$read\_name\_to\_line\_0{$read\_name} = $\_;

}

}

close(MIX\_INPUT\_1);

my @read\_names\_0 = sort keys %read\_name\_to\_line\_0;

my @read\_names\_1 = sort keys %read\_name\_to\_line\_1;

my @read\_names\_2 = sort keys %read\_name\_to\_line\_2;

my @read\_names = (@read\_names\_0, unique(@read\_names\_1, @read\_names\_2));

my $depth = scalar(@read\_names);

my $depth0 = scalar(@read\_names\_0);

my $read\_name\_idx;

foreach $read\_name\_idx (@sample1\_line\_idx\_vec){

$read\_name = $read\_names[$read\_name\_idx-1];

if($read\_name\_idx > $depth0){ # in @read\_names\_1 and/or @read\_names\_2

if(exists($read\_name\_to\_line\_1{$read\_name})){

# print STDOUT $read\_name\_to\_line\_1{$read\_name}."\n";

$ans .= $read\_name\_to\_line\_1{$read\_name}."\n";

}

if(exists($read\_name\_to\_line\_2{$read\_name})){

# print STDOUT $read\_name\_to\_line\_2{$read\_name}."\n";

$ans .= $read\_name\_to\_line\_2{$read\_name}."\n";

}

}else{

if(exists($read\_name\_to\_line\_0{$read\_name})){

# print STDOUT $read\_name\_to\_line\_0{$read\_name}."\n";

$ans .= $read\_name\_to\_line\_0{$read\_name}."\n";

}

}

}

#print STDERR join("\n", @read\_names)."\n"; # for debug

if(defined($log\_filename)){

if(open(LOG, '>>'.$log\_filename)){

print LOG join("\n", @read\_names)."\n";

close(LOG);

}else{

warn "Warning: cannot open $log\_filename for read\_name appending\n";

}

}

open(MIX\_INPUT\_2, $sample2\_sam\_filename) or die "Error: cannot open $sample2\_sam\_filename for input\n";

(%read\_name\_to\_line\_0, %read\_name\_to\_line\_1, %read\_name\_to\_line\_2) = ((),(),());

while(<MIX\_INPUT\_2>){

if(/^$/ || /^@/){ next; }

chomp;

@F = split/\t/;

($read\_name, $flag) = @F[0,1];

if($flag & 0x40){ # PE 1

$read\_name\_to\_line\_1{$read\_name} = $\_;

}elsif($flag & 0x80){ # PE 2

$read\_name\_to\_line\_2{$read\_name} = $\_;

}else{ # SE

$read\_name\_to\_line\_0{$read\_name} = $\_;

}

}

close(MIX\_INPUT\_2);

@read\_names\_0 = sort keys %read\_name\_to\_line\_0;

@read\_names\_1 = sort keys %read\_name\_to\_line\_1;

@read\_names\_2 = sort keys %read\_name\_to\_line\_2;

@read\_names = (@read\_names\_0, unique(@read\_names\_1, @read\_names\_2));

$depth = scalar(@read\_names);

$depth0 = scalar(@read\_names\_0);

foreach $read\_name\_idx (@sample2\_line\_idx\_vec){

$read\_name = $read\_names[$read\_name\_idx-1];

# print STDERR "\@read\_names = ".join(",",@read\_names)."\n";

# print STDERR $read\_name\_idx."\t".$read\_name."\n";

if($read\_name\_idx > $depth0){ # in @read\_names\_1 and/or @read\_names\_2

if(exists($read\_name\_to\_line\_1{$read\_name})){

# print STDOUT $read\_name\_to\_line\_1{$read\_name}."\n";

$ans .= $read\_name\_to\_line\_1{$read\_name}."\n";

}

if(exists($read\_name\_to\_line\_2{$read\_name})){

# print STDOUT $read\_name\_to\_line\_2{$read\_name}."\n";

$ans .= $read\_name\_to\_line\_2{$read\_name}."\n";

}

}else{

if(exists($read\_name\_to\_line\_0{$read\_name})){

# print STDOUT $read\_name\_to\_line\_0{$read\_name}."\n";

$ans .= $read\_name\_to\_line\_0{$read\_name}."\n";

}

}

}

} # end of sub Yyx\_mix\_two\_sam\_by\_read\_name\_idxs()

2-2. Yyx\_two\_sam\_binomial\_mixer.r 即2-1的Perl脚本所调用二项抽样混样R脚本

#!/usr/bin/env Rscript

# command line args should be: input\_tsv\_filename, output\_filename, [seed]

args <- commandArgs(TRUE);

if(length(args)<2){

stop("No enough command line parameters\n\nCommand line args for this R script should be:\n\tinput\_tsv\_filename, output\_filename, [seed]\n\n");

}

input\_tsv\_filename <- args[1];

output\_filename <- args[2];

if(length(args)>2){

set.seed(args[3]);

}

input\_table <- read.delim(input\_tsv\_filename, header=F);

# input\_tsv\_filename should have 3 columns:

# sample1\_depth, sample2\_depth, mixture\_proportion\_of\_sample2

mix.sample2.depth <- rbinom(dim(input\_table)[1], input\_table[,1], input\_table[,3]);

sink(output\_filename);

for(i in 1:length(mix.sample2.depth)){

if(mix.sample2.depth[i] <= input\_table[i,2]){

if(input\_table[i,1]-mix.sample2.depth[i]>0){

cat(paste0(collapse=',', sort(sample.int(input\_table[i,1], input\_table[i,1]-mix.sample2.depth[i]))));

}else{

cat('-');

}

cat('\t');

if(mix.sample2.depth[i]>0){

cat(paste0(collapse=',', sort(sample.int(input\_table[i,2], mix.sample2.depth[i]))));

}else{

cat('-');

}

cat('\n');

}else{

cat('-\t-\n');

}

}

sink();

3. 卵裂Galton-Watson分支过程模拟

进行卵裂Galton-Watson分支过程模拟的R脚本：

if(!interactive()){

args = commandArgs(TRUE)

}

# Usage: Rscript Yyx\_cleavage\_simulation.step\_0\_03.20171230.r

# <initial\_size> <initial\_mut\_prop> <part\_idx> <simu\_times> <death\_ratio> <division\_ratio>

# [output\_flag(=7)] [random\_seed(=1234567)]

#

# Output:

# Yyx\_cleavage\_simulation.step\_0\_03.20171230.output/RData/\*.RData if bitwAnd(output\_flag, 0x01) > 0

# Yyx\_cleavage\_simulation.step\_0\_03.20171230.output/png/\*.png if bitwAnd(output\_flag, 0x02) > 0

# Yyx\_cleavage\_simulation.step\_0\_03.20171230.output/png/\*.RData if bitwAnd(output\_flag, 0x04) > 0

step\_size = 0.03

output\_dir\_root = "Yyx\_cleavage\_simulation.step\_0\_03.20171230.output"

library(ggplot2)

library(grid)

library(gridExtra)

`%.%` <- function(x,y) paste0(x,y)

join = function(sep, vec, ...) paste(collapse=sep, c(vec, ...))

cat0 = function(...) cat(sep="", ...)

echo\_str <- function(x, sep=" =\t", collapse=", ") deparse(substitute(x)) %.% sep %.% join(collapse, x)

echo <- function(x, sep=" =\t", collapse=", ") cat0(deparse(substitute(x)), sep, join(collapse, x), "\n");

yyx\_convert\_NA <- function(vec, default=FALSE){

vec[is.na(vec)] <- default

vec

}

table\_join = function(sep, vec){

tmp = table(vec)

join(sep, names(tmp) %.% "\*" %.% tmp)

}

is = function(x, target\_class){

target\_class %in% class(x)

}

get\_legend = function(p){

g <- ggplotGrob(p)$grobs

g[[which(sapply(g, function(x) x$name) == "guide-box")]]

}

Yyx\_aligned\_ggplot\_arrange = function(plotlist, ignore\_idx=c(), return\_grob=FALSE, nrow=nrow, ncol=ncol, layout\_matrix=matrix(1:(nrow\*ncol), nrow=nrow, ncol=ncol, byrow=TRUE), ...){

if(length(plotlist) < 2){

return(grid.arrange(plotlist, ...))

}

groblist = lapply(plotlist, function(x){

if(is.null(x)){

NULL

}else if(is(x, "ggplot")){

ggplotGrob(x)

}else if(is(x, "grob")){

x

}else{

warning("Warning: unrecognized object in plotlist, with class" %.% join(", ", class(x)))

x

}

})

plot\_len = length(groblist)

for(j in 1:ncol){

maxWidth = rep(unit(0, "mm"), 2)

for(i in 1:nrow){

k = layout\_matrix[i,j]

if(k %in% ignore\_idx){

next

}

maxWidth = unit.pmax(maxWidth, groblist[[ k ]]$widths[2:3])

}

for(i in 1:nrow){

k = layout\_matrix[i,j]

if(k %in% ignore\_idx){

next

}

groblist[[ k ]]$widths[2:3] = maxWidth[1:2]

}

}

# print(lapply(groblist, function(x) x$widths))

if(return\_grob){

arrangeGrob(grobs=groblist, nrow=nrow, ncol=ncol, layout\_matrix=layout\_matrix, ...)

}else{

grid.arrange(grobs=groblist, nrow=nrow, ncol=ncol, layout\_matrix=layout\_matrix, ...)

}

}

one\_set\_one\_step\_simu = function(cell\_set, death\_rates, division\_rates){

# \_rates[1] for ref(0), \_rates[2] for alt(1)

rand = runif(length(cell\_set))

rep\_num = rep(1, length(cell\_set))

death\_rate\_vec = death\_rates[cell\_set+1]

division\_rate\_vec = division\_rates[cell\_set+1]

rep\_num[rand <= death\_rate\_vec] = 0

rep\_num[rand >= 1-division\_rate\_vec] = 2

rep(cell\_set, rep\_num)

}

one\_set\_simu = function(n\_s, n\_f, death\_rates, division\_rates, initial\_cell\_set=c(0)){

cell\_set = initial\_cell\_set

if(n\_s >= 1){

for(t in 1:n\_s){

cell\_set = one\_set\_one\_step\_simu(cell\_set, 0, 1)

}

}

if(n\_f >= n\_s+1){

for(t in (n\_s+1):n\_f){

cell\_set = one\_set\_one\_step\_simu(cell\_set, death\_rates, division\_rates)

}

}

cell\_set

}

one\_step\_simu = function(cell\_set\_list, death\_rates, division\_rates){

lapply(cell\_set\_list, function(cell\_set) one\_set\_one\_step\_simu(cell\_set, death\_rates, division\_rates))

}

summary\_cell\_set\_list = function(cell\_set\_list){

cell\_num = sapply(cell\_set\_list, length)

mut\_num = sapply(cell\_set\_list, sum)

mut\_prop = mut\_num / cell\_num

data.frame(cell\_num=cell\_num, mut\_num=mut\_num, mut\_prop=mut\_prop)

}

Yyx\_heat\_colors = c(hsv(0.167, seq(0.5,1,length=30), 1), hsv(seq(0.167,0,length=40)[-1], 1, 1), hsv(seq(1,0.7,length=40)[-1], 1, 1), hsv(0.7, seq(1,0.25,length=40)[-1], seq(1,0.25,length=40)[-1]))

empty\_ggplot = ggplot() + geom\_point(aes(1,1), colour="white") + theme(axis.ticks=element\_blank(), panel.background=element\_blank(), axis.text.x=element\_blank(), axis.text.y=element\_blank(), axis.title.x=element\_blank(), axis.title.y=element\_blank(), axis.line=element\_blank(), panel.grid=element\_blank())

ggplot\_cell\_set\_list = function(cell\_set\_list, bin\_widths = c(2, 0.01), main="simu\_times = " %.% length(cell\_set\_list), return\_grob=FALSE){

summary\_DF = summary\_cell\_set\_list(cell\_set\_list)

alive\_num = sum(summary\_DF$cell\_num > 0)

death\_num = sum(summary\_DF$cell\_num <= 0)

no\_death\_summary\_DF = summary\_DF[summary\_DF$cell\_num > 0,]

cell\_num\_range = range(no\_death\_summary\_DF$cell\_num)

cell\_num\_range = cell\_num\_range + c(-1, 1)

cell\_num\_range = (cell\_num\_range - mean(cell\_num\_range)) \* 1.05 + mean(cell\_num\_range)

mut\_prop\_range = range(no\_death\_summary\_DF$mut\_prop)

mut\_prop\_range = mut\_prop\_range + c(-0.05, 0.05)

mut\_prop\_range = (mut\_prop\_range - mean(mut\_prop\_range)) \* 1.05 + mean(mut\_prop\_range)

cell\_num\_summary\_vec = with(no\_death\_summary\_DF, c(mean(cell\_num), sd(cell\_num), quantile(cell\_num, c(.05,.25,.5,.75,.95))))

mut\_prop\_summary\_vec = with(no\_death\_summary\_DF, c(mean(mut\_prop), sd(mut\_prop), quantile(mut\_prop, c(.05,.25,.5,.75,.95))))

quantile\_DF = data.frame(cell\_num=cell\_num\_summary\_vec[-(1:2)], mut\_prop=mut\_prop\_summary\_vec[-(1:2)])

mean\_sd\_DF = data.frame(cell\_num\_mean=cell\_num\_summary\_vec[1], cell\_num\_sd=cell\_num\_summary\_vec[2], mut\_prop\_mean=mut\_prop\_summary\_vec[1], mut\_prop\_sd=mut\_prop\_summary\_vec[2])

used\_theme = theme\_bw()

bin2d\_ggplot = ggplot(no\_death\_summary\_DF, aes(x=cell\_num, y=mut\_prop)) + geom\_bin2d(binwidth=bin\_widths) + scale\_fill\_gradientn(colours=Yyx\_heat\_colors) + coord\_cartesian(xlim=cell\_num\_range, ylim=mut\_prop\_range) + used\_theme

x\_hist\_ggplot = ggplot(no\_death\_summary\_DF, aes(x=cell\_num)) + geom\_histogram(binwidth=bin\_widths[1])

x\_hist\_ymax = max(ggplot\_build(x\_hist\_ggplot)$data[[1]]$count)

x\_hist\_label = "death%=" %.% death\_num %.% "/" %.% (death\_num+alive\_num) %.% "=" %.% signif(death\_num/(death\_num+alive\_num),3)

x\_hist\_label = x\_hist\_label %.% "\n" %.% "mean=" %.% signif(cell\_num\_summary\_vec[1],3) %.% ", sd=" %.% signif(cell\_num\_summary\_vec[2],3)

x\_hist\_ggplot = x\_hist\_ggplot + geom\_vline(aes(xintercept=cell\_num), data=quantile\_DF, colour="green") + geom\_text(aes(x=cell\_num, label=cell\_num), y=0, data=quantile\_DF, colour="green", size=4.5, hjust=0, vjust=0) + geom\_vline(aes(xintercept=cell\_num\_mean), data=mean\_sd\_DF, colour="red") + geom\_text(x=cell\_num\_range[1], y=x\_hist\_ymax, label=x\_hist\_label, colour="red", size=5, hjust=0, vjust=1)

x\_hist\_ggplot = x\_hist\_ggplot + coord\_cartesian(xlim=cell\_num\_range) + used\_theme + theme(axis.title.x=element\_blank(), axis.text.x=element\_blank())

y\_hist\_ggplot = ggplot(no\_death\_summary\_DF, aes(x=mut\_prop)) + geom\_histogram(binwidth=bin\_widths[2])

y\_hist\_ymax = max(ggplot\_build(y\_hist\_ggplot)$data[[1]]$count)

y\_hist\_label = "mean=" %.% signif(mut\_prop\_summary\_vec[1],3) %.% "\nsd=" %.% signif(mut\_prop\_summary\_vec[2],3)

y\_hist\_label = y\_hist\_label %.% "\n" %.% ">=0.4(AF>=0.2)% = " %.% signif(sum(no\_death\_summary\_DF$mut\_prop >= 0.4) / nrow(no\_death\_summary\_DF), 3) %.% "\n" %.% "<0.1(AF<0.05)% = " %.% signif(sum(no\_death\_summary\_DF$mut\_prop < 0.1) / nrow(no\_death\_summary\_DF), 3)

y\_hist\_ggplot = y\_hist\_ggplot + geom\_vline(aes(xintercept=mut\_prop), data=quantile\_DF, colour="green") + geom\_text(aes(x=mut\_prop, label=signif(mut\_prop,3)), y=0, data=quantile\_DF, colour="green", size=4.5, hjust=0, vjust=0) + geom\_vline(aes(xintercept=mut\_prop\_mean), data=mean\_sd\_DF, colour="red") + geom\_text(x=mut\_prop\_range[2], y=y\_hist\_ymax, label=y\_hist\_label, colour="red", size=5, hjust=1, vjust=1)

y\_hist\_ggplot = y\_hist\_ggplot + coord\_flip(xlim=mut\_prop\_range) + used\_theme + theme(axis.title.y=element\_blank(), axis.text.y=element\_blank())

bin2d\_no\_legend\_ggplot = bin2d\_ggplot + theme(legend.position="none")

bin2d\_only\_legend\_grob <- get\_legend(bin2d\_ggplot)

Yyx\_aligned\_ggplot\_arrange(plotlist=list(empty\_ggplot + ggtitle(main), x\_hist\_ggplot, bin2d\_only\_legend\_grob, bin2d\_no\_legend\_ggplot, y\_hist\_ggplot), ignore\_idx=c(1,3), ncol=2, nrow=3, widths=c(3,1), heights=c(0.3,1,3), layout\_matrix=matrix(c(1,1,2,3,4,5),ncol=2,nrow=3,byrow=TRUE), return\_grob=return\_grob)

}

summary\_DF\_for\_cell\_set\_list = function(cell\_set\_list){

summary\_DF = summary\_cell\_set\_list(cell\_set\_list)

alive\_num = sum(summary\_DF$cell\_num > 0)

death\_num = sum(summary\_DF$cell\_num <= 0)

no\_death\_summary\_DF = summary\_DF[summary\_DF$cell\_num > 0,]

cell\_num\_summary\_vec = with(no\_death\_summary\_DF, c(mean(cell\_num), sd(cell\_num), quantile(cell\_num, c(.05,.25,.5,.75,.95))))

mut\_prop\_summary\_vec = with(no\_death\_summary\_DF, c(mean(mut\_prop), sd(mut\_prop), quantile(mut\_prop, c(.05,.25,.5,.75,.95))))

ans\_DF = data.frame(death\_num=death\_num, alive\_num=alive\_num, all\_cell\_num=alive\_num+death\_num, death\_percentage=death\_num/(death\_num+alive\_num), cell\_num.mean=cell\_num\_summary\_vec[1], cell\_num.sd=cell\_num\_summary\_vec[2], cell\_num.Q05=cell\_num\_summary\_vec[3], cell\_num.Q25=cell\_num\_summary\_vec[4], cell\_num.Q50=cell\_num\_summary\_vec[5], cell\_num.Q75=cell\_num\_summary\_vec[6], cell\_num.Q95=cell\_num\_summary\_vec[7], mut\_prop.mean=mut\_prop\_summary\_vec[1], mut\_prop.sd=mut\_prop\_summary\_vec[2], mut\_prop.Q05=mut\_prop\_summary\_vec[3], mut\_prop.Q25=mut\_prop\_summary\_vec[4], mut\_prop.Q50=mut\_prop\_summary\_vec[5], mut\_prop.Q75=mut\_prop\_summary\_vec[6], mut\_prop.Q95=mut\_prop\_summary\_vec[7], mut\_prop.lt0\_1=sum(no\_death\_summary\_DF$mut\_prop < 0.1) / nrow(no\_death\_summary\_DF))

for(i in 1:8){

ans\_DF[["mut\_prop.ge0\_" %.% i %.% ".lt0\_" %.% (i+1) %.% ""]] = sum(no\_death\_summary\_DF$mut\_prop >= i/10 & no\_death\_summary\_DF$mut\_prop < (i+1)/10) / nrow(no\_death\_summary\_DF)

}

ans\_DF$mut\_prop.ge0\_9 = sum(no\_death\_summary\_DF$mut\_prop >= 0.9) / nrow(no\_death\_summary\_DF)

ans\_DF$mut\_prop.reach\_0 = sum(no\_death\_summary\_DF$mut\_prop == 0) / nrow(no\_death\_summary\_DF)

ans\_DF$mut\_prop.reach\_1 = sum(no\_death\_summary\_DF$mut\_prop == 1) / nrow(no\_death\_summary\_DF)

ans\_DF

}

initialize\_cell\_set\_list = function(n\_s, n\_f, death\_rates, division\_rates, initial\_cell\_set=c(0)){

ans = list()

attr(ans, "n\_s") = n\_s

attr(ans, "n\_f") = n\_f

attr(ans, "death\_rates") = death\_rates

attr(ans, "division\_rates") = division\_rates

attr(ans, "initial\_cell\_set") = initial\_cell\_set

ans

}

add\_one\_set\_to\_cell\_set\_list = function(cell\_set\_list){

ans = cell\_set\_list

n\_s = attr(ans, "n\_s")

n\_f = attr(ans, "n\_f")

death\_rates = attr(ans, "death\_rates")

division\_rates = attr(ans, "division\_rates")

initial\_cell\_set = attr(ans, "initial\_cell\_set")

already\_simu\_times = length(ans)

ans[[already\_simu\_times+1]] = one\_set\_simu(n\_s, n\_f, death\_rates, division\_rates, initial\_cell\_set=initial\_cell\_set)

ans

}

one\_series\_simu = function(n\_s, n\_f, death\_rates, division\_rates, initial\_cell\_set=c(0)){

cell\_series = list()

cell\_set = initial\_cell\_set

if(n\_s >= 1){

for(t in 1:n\_s){

cell\_set = one\_set\_one\_step\_simu(cell\_set, c(0,0), c(1,1))

cell\_series[[t]] = cell\_set

}

}

if(n\_f >= n\_s+1){

for(t in (n\_s+1):n\_f){

cell\_set = one\_set\_one\_step\_simu(cell\_set, death\_rates, division\_rates)

cell\_series[[t]] = cell\_set

}

}

cell\_series

}

initialize\_cell\_series\_list = function(n\_s, n\_f, death\_rates, division\_rates, initial\_cell\_set=c(0)){

ans = list()

attr(ans, "n\_s") = n\_s

attr(ans, "n\_f") = n\_f

attr(ans, "death\_rates") = death\_rates

attr(ans, "division\_rates") = division\_rates

attr(ans, "initial\_cell\_set") = initial\_cell\_set

if(n\_f >= 1){

for(t in 1:n\_f){

ans[[t]] = list() # empty cell\_set\_list

}

}

ans

}

add\_one\_series\_to\_cell\_series\_list = function(cell\_series\_list){

ans = cell\_series\_list

n\_s = attr(ans, "n\_s")

n\_f = attr(ans, "n\_f")

death\_rates = attr(ans, "death\_rates")

division\_rates = attr(ans, "division\_rates")

initial\_cell\_set = attr(ans, "initial\_cell\_set")

already\_simu\_times = length(ans)

one\_cell\_series = one\_series\_simu(n\_s, n\_f, death\_rates, division\_rates, initial\_cell\_set=initial\_cell\_set)

if(n\_f >= 1){

for(t in 1:n\_f){

ans[[t]] = c(ans[[t]], list(one\_cell\_series[[t]]))

}

}

ans

}

ggplot\_cell\_series\_list = function(cell\_series\_list, bin\_widths = c(2, 0.01), main=NULL){

n\_s = attr(cell\_series\_list, "n\_s")

n\_f = attr(cell\_series\_list, "n\_f")

death\_rates = attr(cell\_series\_list, "death\_rates")

division\_rates = attr(cell\_series\_list, "division\_rates")

initial\_cell\_set = attr(cell\_series\_list, "initial\_cell\_set")

already\_simu\_times = length(cell\_series\_list[[1]])

groblist = list()

if(n\_f >= 1){

for(t in 1:n\_f){

echo(t)

groblist[[t]] = ggplot\_cell\_set\_list(cell\_series\_list[[t]], bin\_widths=bin\_widths, main="t = "%.%t, return\_grob=TRUE)

}

}

tmp = list(ggplotGrob(empty\_ggplot + ggtitle("n\_s=" %.% n\_s %.% "\n" %.% "n\_f=" %.% n\_f %.% "\n" %.% "death\_rates=(" %.% join(", ", death\_rates) %.% "), ratio=" %.% signif(death\_rates[2]/death\_rates[1], 3) %.% "\n" %.% "division\_rates=(" %.% join(", ", division\_rates) %.% "), ratio=" %.% signif(division\_rates[2]/division\_rates[1], 3) %.% "\n" %.% "relative\_fitness=" %.% signif( (1+division\_rates[2]-death\_rates[2]) / (1+division\_rates[1]-death\_rates[1]), 3) %.% "\n\n" %.% "initial\_cell\_set=(" %.% table\_join(",",initial\_cell\_set) %.% ")\n" %.% "initial\_mut\_prop=" %.% signif(mean(initial\_cell\_set),3) %.% "\n\n" %.% "simu\_times=" %.% already\_simu\_times)))

while(length(groblist) < 8){

groblist = c(tmp, groblist)

if(length(groblist) < 8){

groblist = c(groblist, tmp)

}

}

grid.arrange(grobs=groblist, nrow=2, ncol=4)

}

summary\_DF\_for\_cell\_series\_list = function(cell\_series\_list){

n\_s = attr(cell\_series\_list, "n\_s")

n\_f = attr(cell\_series\_list, "n\_f")

death\_rates = attr(cell\_series\_list, "death\_rates")

division\_rates = attr(cell\_series\_list, "division\_rates")

initial\_cell\_set = attr(cell\_series\_list, "initial\_cell\_set")

already\_simu\_times = length(cell\_series\_list[[1]])

ans\_DF = character(0)

for(t in 1:n\_f){

echo(t)

now\_cell\_set\_list\_summary\_DF = summary\_DF\_for\_cell\_set\_list(cell\_series\_list[[t]])

NR = nrow(now\_cell\_set\_list\_summary\_DF)

left\_info\_DF = data.frame(death\_rate.1 = rep(death\_rates[1], NR), death\_rate.2 = rep(death\_rates[2], NR), division\_rate.1 = rep(division\_rates[1], NR), division\_rate.2 = rep(division\_rates[2], NR), initial\_n = rep(length(initial\_cell\_set), NR), initial\_mut\_n = rep(sum(initial\_cell\_set), NR), initial\_mut\_prop = rep(mean(initial\_cell\_set), NR), simu\_times=rep(already\_simu\_times, NR), t=rep(t, NR))

ans\_DF = rbind(ans\_DF, cbind(left\_info\_DF, now\_cell\_set\_list\_summary\_DF))

}

ans\_DF

}

initial\_size = as.integer(args[1])

initial\_mut\_prop = as.numeric(args[2])

part\_idx = as.integer(args[3])

simu\_times = as.integer(args[4])

death\_ratio = as.numeric(args[5])

division\_ratio = as.numeric(args[6])

echo(initial\_size)

echo(initial\_mut\_prop)

echo(part\_idx)

echo(simu\_times)

echo(death\_ratio)

echo(division\_ratio)

output\_flag = 7

if(length(args)>=7){

output\_flag = as.integer(args[7])

}

random\_seed = 1234567

if(length(args)>=8){

random\_seed = as.integer(args[8])

}

echo(output\_flag)

echo(random\_seed)

if(death\_ratio != 1 && division\_ratio != 1){

warning("Warning: death\_ratio will be at priority than division\_ratio, when alpha+gamma>1")

}

set.seed(random\_seed)

simu\_results\_list = list()

n\_s = 0

n\_f = 7

death\_rate = 0

division\_rate = 1

initial\_mut\_count = round(initial\_size \* initial\_mut\_prop)

initial\_cell\_set = c(rep(0, (initial\_size - initial\_mut\_count)), rep(1, initial\_mut\_count))

echo(initial\_mut\_count)

echo(initial\_cell\_set)

death\_rate\_vec\_list = list(

seq(0,0.14,by=step\_size),

seq(0.15,0.29,by=step\_size),

seq(0.3,0.44,by=step\_size),

seq(0.45,0.6,by=step\_size))

#death\_rate\_vec = seq(0,0.6,by=0.03)

death\_rate\_vec = death\_rate\_vec\_list[[part\_idx]]

print(system.time({

for(death\_rate in death\_rate\_vec){

#death\_rate = death\_rate\_vec[part\_idx]

echo(death\_rate)

for(division\_rate in rev(seq(0.4,1-death\_rate,by=step\_size))){

echo(division\_rate)

death\_rates = death\_rate \* c(1, death\_ratio)

division\_rates = division\_rate \* c(1, division\_ratio)

if(death\_rates[2] + division\_rates[2] > 1){

if(death\_ratio != 1){

division\_rates[2] = 1 - death\_rates[2]

}else{

death\_rates[2] = 1 - division\_rates[2]

}

}

echo(death\_rates)

echo(division\_rates)

key = n\_s %.% "\_" %.% n\_f %.% "\_" %.% death\_rates[1] %.% "-" %.% death\_rates[2] %.% "\_" %.% division\_rates[1] %.% "-" %.% division\_rates[2] %.% "\_0-" %.% sum(initial\_cell\_set==0) %.% "\_1-" %.% sum(initial\_cell\_set==1) %.% ""

echo(key)

if(file.exists(output\_dir\_root %.% "/RData/" %.% key %.% ".RData")){

cat("[Notice] File " %.% output\_dir\_root %.% "/RData/" %.% key %.% ".RData already exists, so I skip it\n")

}else if(file.exists(output\_dir\_root %.% "/png/" %.% key %.% ".RData")){

cat("[Notice] File " %.% output\_dir\_root %.% "/png/" %.% key %.% ".RData already exists, so I skip it\n")

}else{

print(system.time({

cat("Now doing simulation ...\n")

now\_cell\_series\_list = initialize\_cell\_series\_list(n\_s, n\_f, death\_rates, division\_rates, initial\_cell\_set)

for(i in 1:simu\_times){

now\_cell\_series\_list = add\_one\_series\_to\_cell\_series\_list(now\_cell\_series\_list)

}

})) # ~0.5min

#ggplot\_cell\_set\_list(now\_cell\_series\_list[[8]])

if(bitwAnd(output\_flag, 0x01) > 0){

print(system.time({

cat("Now saving RData ...\n")

simu\_results\_list[[key]] = now\_cell\_series\_list

save(simu\_results\_list, file=output\_dir\_root %.% "/RData/" %.% key %.% ".RData")

})) # ~5s

}

if(bitwAnd(output\_flag, 0x02) > 0){

print(system.time({

cat("Now plotting png ...\n")

png(output\_dir\_root %.% "/png/" %.% key %.% ".png", width=2000, heigh=1000)

ggplot\_cell\_series\_list(now\_cell\_series\_list)

dev.off()

})) # ~1min

}

if(bitwAnd(output\_flag, 0x04) > 0){

print(system.time({

cat("Now saving png/ .tsv (rearranged summay\_DF) ...\n")

now\_summary\_DF = summary\_DF\_for\_cell\_series\_list(now\_cell\_series\_list)

#write.table(now\_summary\_DF, file=output\_dir\_root %.% "/png/" %.% key %.% ".tsv", sep="\t", row.names=FALSE, quote=FALSE)

## Yyx Note: use save to RData rather than write.table to preserve more numeric precision

save(now\_summary\_DF, file=output\_dir\_root %.% "/png/" %.% key %.% ".RData")

})) # ~1min

}

} # if file.exists

} # for division\_rate

} # for death\_rate

}))