Program code and software commands

Msms commands Haplotypes were simulated under the basic coalescent or under the coalescent with population growth and/or structure using msms. In the following, exemplary msms commands are presented for the simulation of 1,000 haplotypes under the coalescent with growth α=5% starting 5,000 years ago (corresponding to 200 generations/(4)N_e =0.005) and 5 subpopulations of equal size that splitted 50,000 years ago (corresponding to 2,000 generations/(4)N_e =0.05).

```
ms 1000 1 -t 10 -s 100 -I 5 200 200 200 200 200 -g 1 0.05 -g 2 0.05 -g 3 0.05 -g
4 0.05 -g 5 0.05 -ej 0.05 2 1 -ej 0.05 3 1 -ej 0.05 4 1 -ej 0.05 5 1 -eg 200 1
0.0 -T -L -seed 19830717
```

SAS program 1 This exemplary SAS program processes the haplotype data simulated under the basic coalescent (random pairing of haplotypes to simulate genotypes of diploid individuals, allocation of haplotypes to the study and reference population, random selection of measured SNP sites) and creates input files for BATWING. Haplotype data simulated under the coalescent with population growth and/or structure was processed accordingly. In case of standard imputation the program was modified to create input files for IMPUTE2 including recombination_map.map, reference_hap.haps, reference_hap.legend and known_haps.impute.haps. We used SAS due to its good data management capabilities and the simplicity of generated programs, which can easily be translated into open source code (R, Phyton,…).

```
/*********************************************************************
* Program: 01_import_fromMSMS_export_toBATWING.sas*
* Author: Maria Kabisch*
* Date: 07/04/2014*
*********************************************************************
* Purpose of program: processes simulated haplotypes,*
* exports input files for BATWING*
*********************************************************************
* Input files: sim_hap.txt*
* Output files: reference.sas, study_tobe_imputed.sas, study_true.sas,*
* combined_study_ref_hap.sas, s_r_hap.txt, infile.txt, commands.txt*
*********************************************************************/

/*set length of simulated haplotypes and number of 'measured' SNPs********/

%let hap_len=100;
%let tag_num=10;
%let study_num=800;
%let ref_num=200;

options noquotelenmax;

/*********************************************************************/
/*import and modify simulated haplotype data*/
/*********************************************************************/

/*import simulated haplotypes*************************************************/

libname simdata "Path:\";

data work.sim1_hap;
```

```
 infile 'Path:\sim_hap.txt'
 delimiter =' ' missover dsd lrecl=32767 firstobs=2;
 informat hap $&hap_len..;
 format hap $&hap_len..;
 input hap $;
run;

data sim1_hap_index;
 set sim1_hap;
 index=_n_;
run;

/*draw study and reference haplotypes***********************************/

proc surveyselect data=sim1_hap_index seed=7952 method=srs n=&study_num
out=random_study_selected;
run;

data study_hap;
 set random_study_selected;
 flag='s';
run;

data random_ref_selected;
 merge sim1_hap_index study_hap;
 by index;
run;

data ref_hap_hilfe;
 set random_ref_selected;
 if flag eq ' ' then flag='r';
run;

data ref_hap;
 set ref_hap_hilfe;
 where flag='r';
run;

/*random pairing of haplotypes*****************************************/

data random_order_study_hap (keep=hap);
 call streaminit(9284);
 set study_hap;
 u=rand("Uniform");
  proc sort;
   by u;
  run;
run;

data random_order_ref_hap (keep=hap);
 call streaminit(3792);
 set ref_hap;
 u=rand("Uniform");
  proc sort;
   by u;
  run;
run;

/*assign identifier for study individuals*******************************/
```

```
data h_ident_study;
 set random_order_study_hap;
 index=_n_;
 h_id_hilfe='s_h_'||index;
 h_id=compress(h_id_hilfe,'');
 drop h_id_hilfe;
run;

data d_hilfe_study;
 set h_ident_study;
 if mod(index,2)=1 then d_pair='a';
 if mod(index,2)=0 then d_pair='b';
 drop index;
run;

data d_a_study;
 set d_hilfe_study;
 where d_pair eq 'a';
 index=_n_;
  proc sort;
   by index;
  run;
run;

data d_b_study;
 set d_hilfe_study;
 where d_pair eq 'b';
 index=_n_;
  proc sort;
   by index;
  run;
run;

data d_ab_study;
 set d_a_study d_b_study;
  proc sort;
   by index;
  run;
run;

data d_ident_study;
 set d_ab_study;
 d_id_hilfe='s_d_'||index||d_pair;
 d_id=compress(d_id_hilfe,'');
 drop index d_pair d_id_hilfe;
run;

/*assign identifier for reference individuals*****************************/

data h_ident_ref;
 set random_order_ref_hap;
 index=_n_;
 h_id_hilfe='r_h_'||index;
 h_id=compress(h_id_hilfe,'');
 drop h_id_hilfe;
run;

data d_hilfe_ref;
```

```
 set h_ident_ref;
 if mod(index,2)=1 then d_pair='a';
 if mod(index,2)=0 then d_pair='b';
 drop index;
run;

data d_a_ref;
 set d_hilfe_ref;
 where d_pair eq 'a';
 index=_n_;
  proc sort;
   by index;
  run;
run;

data d_b_ref;
 set d_hilfe_ref;
 where d_pair eq 'b';
 index=_n_;
  proc sort;
   by index;
  run;
run;

data d_ab_ref;
 set d_a_ref d_b_ref;
  proc sort;
   by index;
  run;
run;

data d_ident_ref;
 set d_ab_ref;
 d_id_hilfe='r_d_'||index||d_pair;
 d_id=compress(d_id_hilfe,'');
 drop index d_pair d_id_hilfe;
run;

/*simulated study/reference haplotypes together**************************/

data sim_hap_study_ref;
 set d_ident_study d_ident_ref;
run;

/*************************************************************************/
/*generate haplotype states and select randomly 'measured' SNPs*/
/*************************************************************************/

/*generate one variable for each haplotype state*************************/

%macro states();

data study_ref_states;
 set sim_hap_study_ref;
  %do i=1 %to &hap_len.;
   length state_&i $1;
   state_&i=substr(hap,&i,1);
  %end;
 output;
```

```
run;

%mend states;
%states();

/*select randomly &tag_num 'measured' SNPs from all %hap_len SNPs**/

data random_tags;
 do snp=1 to &hap_len.;
  output;
 end;
run;

proc surveyselect data=random_tags seed=3836 method=srs n=&tag_num
out=random_tags_selected;
run;

data random_tags_selected;
 set random_tags_selected;
 tag_snp_hilfe="tag_state_"||snp;
 tag_snp=compress(tag_snp_hilfe,' ');
 drop tag_snp_hilfe;
run;

data _null_;
 length allvars $100;
 retain allvars ' ';
 set random_tags_selected end=eof;
 allvars=trim(left(allvars))||' '||left(snp);
 if eof then call symput('tags',allvars);
run;

data _null_;
 length allvars $500;
 retain allvars ' ';
 set random_tags_selected end=eof;
 allvars=trim(left(allvars))||' '||left(tag_snp);
 if eof then call symput('tag_names',allvars);
run;

/*all study/reference haplotypes with marked 'measured' SNPs**************/

%macro tags();

data study_ref_states_tagged;
 %do i=1 %to &tag_num.;
  set study_ref_states;
  tag_state_%scan(&tags,&i)=state_%scan(&tags,&i);
  drop state_%scan(&tags,&i);
 %end;
run;

%mend tags;
%tags();

data study_ref_states_tagged;
 retain h_id d_id &tag_names. state_1-state_100;
 set study_ref_states_tagged;
run;
```

```
/*build haplotypes for 'measured' SNPs***********************************/

data complete;
 length tag_hap $ &tag_num;
 set study_ref_states_tagged;
 tag_hap=cats(of tag_:);
 r_s=substr(h_id,1,1);
run;

data complete2;
 length tag_hap $ &tag_num;
 set study_ref_states_tagged;
 tag_hap=cats(of tag_:);
 r_s=substr(h_id,1,1);
 rename tag_hap=imp_temp_ref;
run;

/*save modified haplotype data*******************************************/

libname imp "Path:\";

data imp.reference;
 set complete2;
 if r_s eq 'r';
 drop &tag_names. r_s hap;
run;

data imp.study_true;
 set complete2;
 if r_s eq 's';
 drop &tag_names. r_s hap;
run;

/*include only if genotypes are really imputed***************************/

/*data imp.study_tobe_imputed;
 set complete2;
 if r_s eq 's';
 keep tag_hap h_id d_id;
run;*/

/************************************************************************/
/*identify unique study/reference haplotypes*/
/************************************************************************/

/*separate study/reference haplotypes************************************/

data flag;
 set complete;
 if r_s eq 's' then flag='study';
 if r_s eq 'r' then flag='ref';
 keep tag_hap h_id d_id r_s flag;
run;

data study;
 set flag;
 where flag eq 'study';
run;
```

```
data ref;
 set flag;
 where flag eq 'ref';
run;

/*unique study/reference haplotypes*********************************/

ods rtf file="Path:\freq_study_haplotypes.rtf" STYLE=minimal;
ods output OneWayFreqs=freq_study;
 proc freq data=study;
  tables tag_hap;
 run;
ods rtf close;

data unique_study_hap;
 set freq_study;
 tag_hap_s='s';
 keep tag_hap tag_hap_s;
  proc sort;
   by tag_hap;
  run;
run;

ods rtf file="Path:\freq_reference_haplotypes.rtf" STYLE=minimal;
ods output OneWayFreqs=freq_ref;
 proc freq data=ref;
  tables tag_hap;
 run;
ods rtf close;

data unique_ref_hap;
 set freq_ref;
 tag_hap_r='r';
 keep tag_hap tag_hap_r;
  proc sort;
   by tag_hap;
  run;
run;

/*unique study/reference haplotypes together*****************************/

data unique_study_ref_hap;
 merge unique_study_hap unique_ref_hap;
 by tag_hap;
run;

data unique_study_ref_hap;
 length tag_hap_flag $25;
 set unique_study_ref_hap;
 if tag_hap_s eq 's' and tag_hap_r eq ' ' then
 tag_hap_flag='only study hap';
 if tag_hap_s eq ' ' and tag_hap_r eq 'r' then
 tag_hap_flag='only ref hap';
 if tag_hap_s eq 's' and tag_hap_r eq 'r' then
 tag_hap_flag='study and ref hap';
 keep tag_hap tag_hap_flag;
run;
```

```
/*one variable for each 'measured' SNP haplotype state*******************/

%macro tag_states();

data unique_hap_states;
 set unique_study_ref_hap;
  %do i=1 %to &tag_num.;
   length %scan(&tag_names,&i) $1;
   %scan(&tag_names,&i)=substr(tag_hap,&i,1);
  %end;
 output;
run;

%mend tag_states;
%tag_states();

data uninform;
 set unique_hap_states;
 drop tag_hap_flag tag_hap;
run;

/*remove uninformative sites*********************************************/

%let lib=WORK;
%let mem=uninform;

proc sql noprint;
 select name, put(count(name),5.-L) into :clist separated by ' ' , :charct
 from dictionary.columns
 where libname=upcase("&lib") and memname=upcase("&mem") and type='char';
quit;

data _null_;
 array char(*) $ &clist;
  array c_allmiss (&charct) $ (&charct*'true');
   set &mem end=done;
    do i=1 to dim(c_allmiss);
     if char(i) ne '0' then c_allmiss(i)='false';
    end;
   if done then do;
   cnt=0;
    do i= 1 to dim(c_allmiss);
     if c_allmiss(i) ='true' then do;
     cnt+1;
     call symput('var'||put(cnt,3.-l),vname(char(i)));
    end;
   end;
  call symput('cnt',put(cnt,3.-l));
 end;
run;
%macro dropem;

%do i = 1 %to &cnt;
 &&var&i.
%end;

%mend;

data unique_hap_states_inform;
```

```
 set unique_hap_states;
 drop %dropem ;
run;

/*save unique haplotype information*************************************/

libname interim "Path:\";

data interim.combined_study_ref_hap;
 set unique_hap_states_inform;
run;

/***********************************************************************/
/*provide input files for BATWING*/
/***********************************************************************/

data inform_sites;
 set unique_hap_states_inform;
 drop tag_hap_flag tag_hap;
run;

proc contents data=inform_sites out=inform_sites_hilfe;
run;

%macro batwing_ready();

data _null_;
 length allvars $500;
 retain allvars ' ';
 set inform_sites_hilfe end=eof;
 allvars = trim(left(allvars))||' '||left(name);
 if eof then call symput('inform_sites', allvars);
run;

%let is_hilfe="&inform_sites";


data _null_;
 call symput ("is_num",compress(countw(&is_hilfe," ")," "));
run;

/*export BATWING input files*********************************************/

proc export
 data=inform_sites
 dbms=tab
 outfile="Path:\s_r_hap.txt"
 replace;
 putnames=no;
run;

data hilfe1;
 do var=1 to 14;
  output;
 end;
run;
data infile;
 length text $250;
 set hilfe1;
```

```
 if var eq 1 then text="datafile: ./batwing/s_r_hap";
 if var eq 2 then text="infsites: &is_num.";
 if var eq 3 then text="UEPtimes: 1";
 if var eq 4 then text="inftype: 1";
 if var eq 5 then text="sizemodel: 0";
 if var eq 6 then text="migmodel: 0";
 if var eq 7 then text="samples: 1000";
 if var eq 8 then text="warmup: 100";
 if var eq 9 then text="treebetN: 10";
 if var eq 10then text="Nbetsamp: 10";
 if var eq 11 then text="seed: 1983";
 if var eq 12 then text="outroot: 1";
 if var eq 13 then text="tree_concensus: 1";
 if var eq 14 then text="picgap: 1";
 drop var;
run;

proc export
 data=infile
 dbms=tab
 outfile="Path:\infile.txt"
 replace;
 putnames=no;
run;

data command;
 length text1 text2 text3 $250;
 text1="./batwing";
 text2="./batwing/infile";
 text3="./batwing/out";
run;
%mend batwing_ready;
%batwing_ready();

/*export commands to be executed in BATWING*****************************/

proc export
 data=command
 dbms=tab
 outfile="Path:\command.txt"
 replace;
 putnames=no;
run;

/************************************************************************/
/*End*/
/************************************************************************/
```

BATWING commands BATWING generates 1000 gene trees based on simulated haplotype data and an infile containing information about the population model applied and corresponding priors.

```
batwing ./Path/to/infile ./Path/to/output
```

When haplotypes were simulated under the coalescent with population growth and/or structure, the BATWING infile as given in SAS program 1 was modified. In the following, an examplary BATWING infile is

presented for a population comprising 1,000 haplotypes with growth α=5% starting 5,000 years ago (corresponding to 200 generations/(4)N_e =0.005) and 5 subpopulations of equal size that splitted 50,000 years ago (corresponding to 2,000 generations/(4)N_e =0.05).

```
datafile: ./Path/to/haplotypes/s_r_hap
locationfile: ./Path/to/subpopulations/pop_hap
infsites: 10
UEPtimes: 1
inftype: 1
sizemodel: 2
migmodel: 1
samples: 1000
warmup: 100
treebetN: 10
Nbetsamp: 10
seed: 1983
outroot: 1
tree_concensus: 1
picgap: 1
thetaprior: uniform(0,100)
alphaprior: gamma(2,40)
omegaprior: gamma(2000,1)
betaprior: gamma(2,400)
splitprior: gamma(2,40)
propprior: dirichlet(5,2)
```

SAS program 2 BATWING generated 1100 gene trees. This SAS program imports all gene trees, discards the first 100 (burn-in) and creates an input file for SumTrees including the topologies of 1000 gene trees.

```
/*************************************************************************
* Program: 02_import_fromBATWING_export_toSumTrees.sas*
* Author: Maria Kabisch*
* Date: 31/04/2014*
*************************************************************************
* Purpose of program: reads and appends topologies of 1000 gene
* trees generated with BATWING,*
*************************************************************************
*************************************************************************/
* Input files:out.1.txt to out.1100.txt*
* Output files: alltrees.txt*
*************************************************************************/

%let tree_len=1100;

options noquotelenmax;

/*list BATWING output files*************************************************/

%let dirname = Path:\;
filename dirlist pipe "dir /B &dirname\*.txt";

data batwing_out;
 length filename $15;
 infile dirlist length=reclen ;
 input filename $varying15. reclen ;
 index=strip(scan(filename,2,"."));
 index_n=index*1;
  proc sort;
   by index_n;
  run;
run;

/*import BATWING output files***********************************************/

%macro trees();

 %do k=1 %to 1100;
  data _null_;
   set batwing_out;
   if index_n=&k;
   call symput ('filename',filename);
  run;

  proc import
   datafile="Path:\&filename."
   dbms=tab
   out=out_&k
   replace;
   delimiter=";";
   getnames=no;
   guessrows=250;
  run;
  data mod_&k;
   set out_&k;
```

```
   var2=substr(var1,length(var1));
   if var2 in (')','~') then flag=1;
   else flag=0;
  run;

  data mod2_&k;
   set mod_&k end=eof;
   where flag=1;
   drop var2 flag;
  run;

  data mod3_&k;
   length topolist $1500.;
   set mod2_&k end=eof;
   retain topolist;
   if _n_=1 then topolist=var1;
   else topolist=cats(topolist,"",var1);
   if eof then output;
  run;

  data mod4_&k;
   set mod3_&k;
   topolist=compress(topolist," ");
   topolist2=compress(topolist,"<>");
   drop var1 topolist;
  run;

/*append all tree topologies*********************************************/

  proc append base=alltopologies data=mod4_&k force;
  run;
  quit;

 %end;

%mend trees;
%trees();

data redtopologies;
 set alltopologies;
 if _n_ gt 100;
run;

data redtopologies2 (keep=topolist3);
 set redtopologies;
 syn=';';
 topolist3=compress(topolist2||syn, " ");
run;

/*export input file for SumTrees*****************************************/

proc export
 data=redtopologies2
 dbms=tab
 outfile='Path:\alltrees.txt'
 replace;
 putnames=no;
run;
```

```
/*************************************************************************/
/*End*/
/*************************************************************************/
```

SumTrees commands The following SumTrees commands were applied to generate a rooted consensus tree according to the majority rule. The branch lengths of the consensus tree (posterior coalescence times) are set to the mean of the corresponding branch lengths of the input gene trees.

```
sumtrees.py --output=consensus.tre --min-clade-freq=0.001 --rooted alltrees.tre
```

IMPUTE2 commands Standard imputation was performed with following IMPUTE2 commands.

```
impute2 -use_prephased_g -m recombination_map.map -h reference_hap.haps -l
reference_hap.legend -known_haps_g known_haps.impute_haps -int 1 100 -Ne 10000 -
seed 7622 -o impute_out
```

SAS program 3 Based on the coalescence times in the consensus gene tree, imputation templates were identified for coalescent-based genotype imputation. This exemplary SAS program reads the reference haplotypes that serve as imputation templates, estimates the distribution of the missing study haplotype states, compares derived genotypes with true genotypes and calculates imputation accuracy (genotype concordance rates, imputation quality scores) across all individuals and SNPs stratified by minorallele frequency. In case of standard imputation this program was modified to read output files from IMPUTE2.

```
/*************************************************************************
* Program: 03_imputation_accuracy.sas*
* Author: Maria Kabisch*
* Date:02/05/2014*
*************************************************************************
* Purpose of program: calculates imputation accuracy*
*************************************************************************
* Input files: temp_master.sas, reference.sas, study_true.sas*
* Output files: accuracy.rtf, accuracy_bymaf.rtf*
*************************************************************************/
%let study_num_half=400;

options noquotelenmax;

/*************************************************************************/
/*import imputation templates for each study haplotype*/
/*************************************************************************/

libname imp "Path:\";

data import;
 length tag_hap_flag $25 unique_study_hap $10 imp_temp_ref $10;
 input tag_hap_flag $ unique_study_hap $ imp_temp_ref $;
 datalines;
 sr   0000000000  0000000000
 sr   0000000010  0000000010
 sr   0010001000  0010001000
 sr   0110101000  0110101000
```

```
 sr   0110101100  0110101100
 sr   0110101101  0110101101
 sr   1010001000  1010001000
 os   0001000010  0000000010
 os   0010101000  0110101000
 os   0010101000  0110101100
 os   0010101000  0110101101
 os   0010111000  0110101000
 os   0010111000  0110101100
 os   0010111000  0110101101;
run;

data temp_master;
 set import;
 if tag_hap_flag eq 'sr' then tag_hap_flag='study and ref hap';
 if tag_hap_flag eq 'os' then tag_hap_flag='only study hap';
run;

data temp_master_flag_ref;
 set temp_master;
 keep flag imp_temp_ref;
 flag=1;
  proc sort nodupkey;
   by imp_temp_ref;
  run;
run;

data temp_master_flag_study;
 set temp_master;
 keep flag unique_study_hap;
 flag=1;
  proc sort nodupkey;
   by unique_study_hap;
  run;
run;

/*read imputation templates from reference*******************************/
data reference;
 set imp.reference;
  proc sort;
   by imp_temp_ref;
  run;
run;

data imp_ident;
 merge reference temp_master_flag_ref;
 by imp_temp_ref;
run;

data imp_ident2;
 set imp_ident;
 if flag=1;
run;

/************************************************************************/
/*infer distribution of missing study haplotype states*/
/************************************************************************/

data states_list;
```

```
 set imp_ident2;
 drop imp_temp_ref h_id d_id flag;
run;
proc contents data=states_list out=name_states;
run;
data _null_;
 length allvars $1000;
 retain allvars ' ';
 set name_states end=eof;
 allvars = trim(left(allvars))||' '||left(name);
 if eof then call symput('states_list', allvars);
run;

%let states="&states_list";

data _null_;
 call symput ("states_num",countw(&states," "));
run;

%macro distr_states();

 %do i=1 %to &states_num.;

  ods output CrossTabFreqs=freq_&i;
   proc freq data=imp_ident2;
    tables %scan(&states_list,&i)*imp_temp_ref;
   run;

  data distr_&i;
   set freq_&i;
   keep %scan(&states_list,&i) imp_temp_ref ColPercent;
   where ColPercent ne .;
    proc sort;
     by imp_temp_ref;
    run;
  run;
  proc transpose data=distr_&i out=distr_trans_&i prefix=distr;
   var ColPercent;
   by imp_temp_ref;
   id %scan(&states_list,&i);
  run;

  data distr_state0_&i;
   length state_name $25 hilfe_0 $10;
   set distr_trans_&i;
   state_name="%scan(&states_list,&i)";
   if distr0 eq 100 then hilfe_0='mono_0';
   if distr0 ne 100 then hilfe_0='not_mono_0';
   if hilfe_0 not in ('not_mono_0') then distr1=0;
   keep imp_temp_ref distr0 hilfe_0 distr1 state_name;
  run;

  data distr_state1_&i;
   length hilfe_1 $10;
   set distr_state0_&i;
   if distr1 eq 100 then hilfe_1='mono_1';
   if distr1 ne 100 then hilfe_1='not_mono_1';
   if hilfe_1 not in ('not_mono_1') then distr0=0;
  keep imp_temp_ref distr0 distr1 state_name;
```

```
  proc append base=alldistr data=distr_state1_&i force;
  run;
  quit;
 %end;

%mend distr_states;
%distr_states();

/*modify haplotype state distribution data*******************************/

data alldistr_mod;
 set alldistr;
 if distr0 eq . then distr0=0;
 if distr1 eq . then distr1=0;
 distr_0=round(distr0/100, 0.01);
 distr_1=round(distr1/100, 0.01);
 drop distr0 distr1;
  proc sort;
   by imp_temp_ref;
  run;
run;

proc transpose data=alldistr_mod out=alldistr_trans0  prefix=distr_0_;
 var distr_0;
 by imp_temp_ref;
 id state_name;
run;

proc transpose data=alldistr_mod out=alldistr_trans1 prefix=distr_1_;
 var distr_1;
 by imp_temp_ref;
 id state_name;
run;

data hap_distr;
 merge alldistr_trans0 alldistr_trans1;
 by imp_temp_ref;
 drop _NAME_;
run;

/*join distribution information with master file*************************/

data temp_master_sort;
 set temp_master;
  proc sort;
   by imp_temp_ref;
  run;
run;

data hap_distr_master;
 merge temp_master_sort hap_distr;
 by imp_temp_ref;
  proc sort;
   by unique_study_hap;
  run;
run;

data distr_liste;
 set hap_distr_master;
```

```
 drop unique_study_hap imp_temp_ref tag_hap_flag;
run;
proc contents data=distr_liste out=name_distr;
run;

data _null_;
 length allvars $5000;
 retain allvars ' ';
 set name_distr end=eof;
 allvars = trim(left(allvars))||' '||left(name);
 if eof then call symput('distr_list', allvars);
run;

%let distr="&distr_list";

data _null_;
 call symput ("distr_num",countw(&distr," "));
run;

data _null_;
 call symput ("distrhalf_num",&distr_num/2);
run;

data _null_;
 length allvars $5000;
 retain allvars ' ';
 set temp_master_flag_study end=eof;
 allvars = trim(left(allvars))||' '||left(unique_study_hap);
 if eof then call symput('allstudy_hap', allvars);
run;

%let hap="&allstudy_hap";

data _null_;
 call symput ("as_hap_num",countw(&hap," "));
run;

%macro geno_acc();

 %do i=1 %to &distr_num.;

/*account for several possible imputation templates per study
haplotype*************************************************************/

  proc means data=hap_distr_master nway mean;
   class unique_study_hap;
   var %scan(&distr_list,&i);
   output out=mean_&i;
  run;
  data mean_mod_&i;
   set mean_&i;
   where _stat_ eq 'MEAN';
   drop _type_ _freq_ _stat_;
   %scan(&distr_list,&i)=round(%scan(&distr_list,&i),0.01);
    proc sort;
     by unique_study_hap;
    run;
  run;
```

```
 %end;

 %do j=1 %to &distrhalf_num.;

  %let x=%sysevalf(&j+&distrhalf_num);

  data together_&j;
   merge mean_mod_&j mean_mod_&x;
   by unique_study_hap;
  run;

 %end;

 %do i=1 %to &distrhalf_num.;

  proc contents data=together_&i out=name_&i;
  run;

  data _null_;
   length allvars $100;
   retain allvars ' ';
   set name_&i end=eof;
   where varnum ne 1;
   allvars = trim(left(allvars))||' '||left(name);
   if eof then call symput('rename', allvars);
  run;

  %let vars="&rename";

  data _null_;
   call symput ("rename_num",countw(&vars," "));
  run;

  data rename_&i;
   set together_&i;
    %do j=1 %to &rename_num.;
     %let y=%sysevalf(&j-1);
     distr_&y=%scan(&rename,&j);
    %end;
  run;
  data imp_&i;
   set rename_&i;
   study_hap=unique_study_hap;
   drop unique_study_hap;
    proc sort;
     by study_hap;
    run;
  run;

/*************************************************************************/
/*unmasking true genotypes*/
/*************************************************************************/

/*import true haplotypes*************************************************/

  data true;
   length study_hap $100;
   set imp.study_true;
   study_hap=imp_temp_ref;
```

```
    proc sort;
     by study_hap;
  run;

  data true_&i;
   set true;
   keep study_hap h_id d_id %scan(&states_list,&i);
  run;

/*combine true and estimated haplotypes**********************************/

  data true_imp_&i;
   merge true_&i imp_&i ;
   by study_hap;
  run;

  data mod_&i;
   set true_imp_&i;
   hilfe=scan(d_id,3,'_');
   hilfe2=compress(hilfe,'ab');
   d_index=input(hilfe2,5.0);
   d_ab=substr(hilfe,length(hilfe),1);
   drop h_id study_hap hilfe hilfe2;
    proc sort;
     by d_index d_ab;
    run;
  run;

  data part_a_&i;
   set mod_&i;
   where d_ab eq 'a';
   a_true_%scan(&states_list,&i)=%scan(&states_list,&i);
   a_true=%scan(&states_list,&i);
   a_distr_0_%scan(&states_list,&i)=distr_0_%scan(&states_list,&i);
   a_distr_1_%scan(&states_list,&i)=distr_1_%scan(&states_list,&i);
   a_distr_0=distr_0;
   a_distr_1=distr_1;
   drop d_id d_ab %scan(&states_list,&i)
   distr_1_%scan(&states_list,&i)
   distr_0_%scan(&states_list,&i) distr_0 distr_1;
  run;

  data part_b_&i;
   set mod_&i;
   where d_ab eq 'b';
   b_true_%scan(&states_list,&i)=%scan(&states_list,&i);
   b_true=%scan(&states_list,&i);
   b_distr_0_%scan(&states_list,&i)=distr_0_%scan(&states_list,&i);
   b_distr_1_%scan(&states_list,&i)=distr_1_%scan(&states_list,&i);
   b_distr_0=distr_0;
   b_distr_1=distr_1;
   drop d_id d_ab %scan(&states_list,&i)
   distr_0_%scan(&states_list,&i) distr_1_%scan(&states_list,&i)
   distr_0 distr_1;
  run;

  data ab_row_&i;
   merge part_a_&i part_b_&i;
   by d_index;
```

```
  run;

/*generate true and estimated genotypes**********************************/

  data geno_&i;
   set ab_row_&i;
   if a_true eq '0' and b_true eq '0' then true_geno_0='1';
   if a_true eq '0' and b_true eq '0' then true_geno_1='0';
   if a_true eq '0' and b_true eq '0' then true_geno_2='0';
   if (a_true eq '1' and b_true eq '0') or
   (a_true eq '0' and b_true eq '1') then true_geno_0='0';
   if (a_true eq '1' and b_true eq '0') or
   (a_true eq '0' and b_true eq '1') then true_geno_1='1';
   if (a_true eq '1' and b_true eq '0') or
   (a_true eq '0' and b_true eq '1') then true_geno_2='0';
   if a_true eq '1' and b_true eq '1' then true_geno_0='0';
   if a_true eq '1' and b_true eq '1' then true_geno_1='0';
   if a_true eq '1' and b_true eq '1' then true_geno_2='1';
   imp_geno_0=a_distr_0*b_distr_0;
   imp_geno_1=(a_distr_0*b_distr_1)+(a_distr_1*b_distr_0);
   imp_geno_2=a_distr_1*b_distr_1;
  run;
/*************************************************************************/
/*assessing imputation accuracy*/
/*************************************************************************/

/*calculate genotype concordance, rate of false positives and rate of false
negatives*************************************************************************/

  data acc_&i;
   set geno_&i;

/*concordance*/
   if true_geno_0 eq '1' and true_geno_1 eq '0' and
   true_geno_2 eq '0' then conc00=imp_geno_0;
   if true_geno_0 eq '1' and true_geno_1 eq '0' and
   true_geno_2 eq '0' then conc11=0;
   if true_geno_0 eq '1' and true_geno_1 eq '0' and
   true_geno_2 eq '0' then conc22=0;
   if true_geno_0 eq '0' and true_geno_1 eq '1' and
   true_geno_2 eq '0' then conc00=0;
   if true_geno_0 eq '0' and true_geno_1 eq '1' and
   true_geno_2 eq '0' then conc11=imp_geno_1;
   if true_geno_0 eq '0' and true_geno_1 eq '1' and
   true_geno_2 eq '0' then conc22=0;
   if true_geno_0 eq '0' and true_geno_1 eq '0' and
   true_geno_2 eq '1' then conc00=0;
   if true_geno_0 eq '0' and true_geno_1 eq '0' and
   true_geno_2 eq '1' then conc11=0;
   if true_geno_0 eq '0' and true_geno_1 eq '0' and
   true_geno_2 eq '1' then conc22=imp_geno_2;

/*false positives*/
   if true_geno_0 eq '1' and true_geno_1 eq '0' and
   true_geno_2 eq '0' then fp10=imp_geno_1;
   if true_geno_0 eq '1' and true_geno_1 eq '0' and
   true_geno_2 eq '0' then fp20=imp_geno_2;
   if true_geno_0 eq '1' and true_geno_1 eq '0' and
   true_geno_2 eq '0' then fp21=0;
```

```
   if true_geno_0 eq '0' and true_geno_1 eq '1' and
   true_geno_2 eq '0' then fp10=0;
   if true_geno_0 eq '0' and true_geno_1 eq '1' and
   true_geno_2 eq '0' then fp20=0;
   if true_geno_0 eq '0' and true_geno_1 eq '1' and
   true_geno_2 eq '0' then fp21=imp_geno_2;
   if true_geno_0 eq '0' and true_geno_1 eq '0' and
   true_geno_2 eq '1' then fp10=0;
   if true_geno_0 eq '0' and true_geno_1 eq '0' and
   true_geno_2 eq '1' then fp20=0;
   if true_geno_0 eq '0' and true_geno_1 eq '0' and
   true_geno_2 eq '1' then fp21=0;

/*false negatives*/
   if true_geno_0 eq '1' and true_geno_1 eq '0' and
   true_geno_2 eq '0' then fn01=0;
   if true_geno_0 eq '1' and true_geno_1 eq '0' and
   true_geno_2 eq '0' then fn02=0;
   if true_geno_0 eq '1' and true_geno_1 eq '0' and
   true_geno_2 eq '0' then fn12=0;
   if true_geno_0 eq '0' and true_geno_1 eq '1' and
   true_geno_2 eq '0' then fn01=imp_geno_0;
   if true_geno_0 eq '0' and true_geno_1 eq '1' and
   true_geno_2 eq '0' then fn02=0;
   if true_geno_0 eq '0' and true_geno_1 eq '1' and
   true_geno_2 eq '0' then fn12=0;
   if true_geno_0 eq '0' and true_geno_1 eq '0' and
   true_geno_2 eq '1' then fn01=0;
   if true_geno_0 eq '0' and true_geno_1 eq '0' and
   true_geno_2 eq '1' then fn02=imp_geno_0;
   if true_geno_0 eq '0' and true_geno_1 eq '0' and
   true_geno_2 eq '1' then fn12=imp_geno_1;
  run;

  data measure_&i;
   set acc_&i;
   conc=conc00+conc11+conc22;
   fp=fp10+fp20+fp21;
   fn=fn01+fn02+fn12;
   disconc=fp10+fp20+fp21+fn01+fn02+fn12;
   snp_name="%scan(&states_list,&i)";
   keep d_index snp_name conc conc00 conc11 conc22 fp fp10 fp20 fp21
   fn fn01 fn02 fn12 disconc;
  run;

  data maf_&i;
   length snp_name $50;
   set acc_&i;
   snp_name="%scan(&states_list,&i)";
   if true_geno_0 eq '1' then rare=0;
   if true_geno_1 eq '1' then rare=1;
   if true_geno_2 eq '1' then rare=2;
   keep d_index snp_name true_geno_0 true_geno_1 true_geno_2 rare;
  run;

  data all_rare_&i;
   set maf_&i;
   retain sum_rare 0;
   sum_rare=sum_rare+rare;
```

```
  run;
  data anz_rare_&i;
   set all_rare_&i;
   if d_index eq &study_num_half.;
   keep d_index snp_name sum_rare
  run;

/*averaging accuracy over all individuals*********************************/

  proc means data=measure_&i mean std min max;
   var conc conc00 conc11 conc22 fp fp10 fp20 fp21 fn fn01 fn02 fn12
   disconc;
   output out=all_ind_&i;
  run;

  data avg_ind_&i;
   length snp_name $50;
   set all_ind_&i;
   where _STAT_ eq 'MEAN';
   snp_name="%scan(&states_list,&i)";
   keep snp_name conc conc00 conc11 conc22 fp fp10 fp20 fp21 fn fn01
   fn02 fn12 disconc;
  run;

/*calculating IQS*********************************************************/

  data true_rand_&i;
   length snp_name $50;
   set acc_&i;
   snp_name="%scan(&states_list,&i)";
   true_geno_0_num=true_geno_0*1;
   true_geno_1_num=true_geno_1*1;
   true_geno_2_num=true_geno_2*1;
   keep d_index snp_name true_geno_0_num true_geno_1_num
   true_geno_2_num;
  run;

  proc means data=true_rand_&i mean std min max;
   var true_geno_0_num true_geno_1_num true_geno_2_num;
   output out=all_ind_true_&i;
  run;

  data avg_ind_true_&i;
   length snp_name $50;
   set all_ind_true_&i;
   where _STAT_ eq 'MEAN';
   snp_name="%scan(&states_list,&i)";
   keep snp_name true_geno_0_num true_geno_1_num true_geno_2_num;
  run;
  proc append base=all_snp data=avg_ind_&i force;
  run;
  quit;

  proc append base=all_snp_maf data=anz_rare_&i force;
  run;
  quit;

  proc append base=all_snp_true data=avg_ind_true_&i force;
  run;
```

```
  quit;
 %end;

%mend geno_acc;
%geno_acc();

/*calculate MAF and account for MAFs ge 50%*******************************/

proc sort data=all_snp out=all_snp_sort;
 by snp_name;
run;

proc sort data=all_snp_maf out=all_snp_maf_sort;
 by snp_name;
run;

proc sort data=all_snp_true out=all_snp_true_sort;
 by snp_name;
run;

data all;
 merge all_snp_sort all_snp_maf_sort all_snp_true_sort;
 by snp_name;
run;

data all_bymaf;
 set all;
 maf=sum_rare/(d_index*2);
 maf_corr=maf;
 if maf gt 0.5 then maf_corr=1-maf;
 if maf gt 0.5 then flag=1;
 else flag=0;
run;

data part1;
 set all_bymaf;
 if flag=0;
run;

data part2;
 set all_bymaf;
 if flag=1;
 conc00_corr=conc22;
 conc22_corr=conc00;
 fp10_corr=fn12;
 fp20_corr=fn02;
 fp21_corr=fn01;
 fn01_corr=fp21;
 fn02_corr=fp20;
 fn12_corr=fp10;
 drop conc22 conc00 fn12 fn02 fn01 fp21 fp20 fp10;
run;

data part2_corr;
 set part2;
 rename conc00_corr=conc00;
 rename conc22_corr=conc22;
 rename fp10_corr=fp10;
 rename fp20_corr=fp20;
```

```
 rename fp21_corr=fp21;
 rename fn01_corr=fn01;
 rename fn02_corr=fn02;
 rename fn12_corr=fn12;
run;

data all_corr;
 set part1 part2_corr;
 drop d_index sum_rare maf flag;
  proc sort;
   by snp_name;
  run;
run;

/*stratify by true genotype************************************************/

data all_num;
 set all_corr;
 conc00_num=conc00*400;
 fp10_num=fp10*400;
 fp20_num=fp20*400;
 conc11_num=conc11*400;
 fn01_num=fn01*400;
 fp21_num=fp21*400;
 conc22_num=conc22*400;
 fn02_num=fn02*400;
 fn12_num=fn12*400;
 num00=conc00_num+fp10_num+fp20_num;
 num01=conc11_num+fn01_num+fp21_num;
 num11=conc22_num+fn02_num+fn12_num;
 conc_true00=conc00_num/num00;
 conc_true01=conc11_num/num01;
 conc_true11=conc22_num/num11;
run;

data all_num_mod;
 set all_num;
 if num00 eq 0 then conc_true00=0;
 if num01 eq 0 then conc_true01=0;
 if num11 eq 0 then conc_true11=0;
 drop conc00_num fp10_num fp20_num conc11_num fn01_num fp21_num
 conc22_num fn02_num fn12_num num00 num01 num11;
run;

data bychance;
 set all_num_mod;
 true_geno_0_freq=true_geno_0_num*400;
 true_geno_1_freq=true_geno_1_num*400;
 true_geno_2_freq=true_geno_2_num*400;
 ew_00=(true_geno_0_freq*true_geno_0_freq)/400;
 ew_11=(true_geno_1_freq*true_geno_1_freq)/400;
 ew_22=(true_geno_2_freq*true_geno_2_freq)/400;
 pc=(ew_00+ew_11+ew_22)/400;
 iqs=(conc-pc)/(1-pc);
 drop true_geno_0_num true_geno_1_num true_geno_2_num
 true_geno_0_freq true_geno_1_freq true_geno_2_freq ew_00
 ew_11 ew_22;
run;
```

```
/*save imputation accuracy*************************************************/

data imp.accuracy;
 set bychance;
run;

ods rtf file="Path:\accuracy.rtf" STYLE=minimal;
 proc means data=bychance n nmiss mean clm std min max median p25
 p75 qrange;
  var conc conc00 conc11 conc22 fp fp10 fp20 fp21 fn fn01 fn02 fn12
  disconc iqs;
  output out=acc;
 run;
ods rtf close;

data bat_maf;
 length maf_cat $20;
 set bychance;
 if maf_corr lt 0.01 then maf_cat='rare';
 if maf_corr ge 0.01 and maf_corr le 0.05 then maf_cat='low';
 if maf_corr gt 0.05 then maf_cat='common';
  proc sort;
   by maf_cat;
  run;
run;

ods rtf file="Path:\accuracy_bymaf.rtf" STYLE=minimal;
 proc means data=bat_maf n nmiss mean clm std min max median p25
 p75 qrange;
  var conc fp fn disconc iqs;
  by maf_cat;
 run;
ods rtf close;

/*************************************************************************/
/*End*/
/*************************************************************************/
```

SAS program 4 This exemplary SAS program reads real genotypes of the CEU population for the low recombination region as downloaded from the 1000 Genomes Project and converts the genotype data from vcf format to input files for SHAPEIT2. Genotypes of the AFR and AMR populations were converted accordingly.

```
/*************************************************************************
* Program: 04_import_fromVCF_export_toSHAPEIT.sas*
* Author: Maria Kabisch*
* Date: 09/09/2014*
*************************************************************************
* Purpose of program: converts real genotypes from vcf to gen/sample,*
* exports input files for SHAPEIT2*
*************************************************************************
* Input files: vcf.csv*
* Output files: CEU_low_gen.txt,CEU_low_sample.txt,
*************************************************************************/

/*import real genotypes for low recombination region*********************/
```

```
proc import
 datafile="Path:\vcf.csv"
 dbms=csv
 out=import
 replace;
 delimiter=';';
run;

data import2;
 set import;
 if QUAL eq '100';
run;

/*convert CEU genotypes**************************************************/

data Varumben;
 set import2;
  array T{*} HG00096 -- NA20828;
   do i = 1 to dim(T);
    if scan(T(i),1,":") = "0|0" THEN T(i) = "1 0 0";
    else if (scan(T(i),1,":") = "1|0") OR (Scan(T(i),1,":") = "0|1") THEN
    T(i) = "0 1 0";
    else if Scan(T(i),1,":") = "1|1" THEN T(i) = "0 0 1";
    else T(i) = " ";
   end;
run;

data gen;
 retain chrom id pos ref alt;
 set varumben;
 drop i qual filter info format;
run;

/*export real genotype data*****************************************/

proc export
 data=gen
 dbms=tab
 outfile='Path:\CEU_low_gen.txt'
 replace;
 putnames=no;
run;

data head_sample;
 length ID_1 ID_2 missing $10;
 ID_1='0';
 ID_2='0';
 missing='0';
run;

proc transpose data=gen out=gen_trans;
 var HG00096 -- NA20828;
 id id;
run;

data var;
 retain ID_1 ID_2 missing;
 length missing $10;
 set gen_trans (rename=(_NAME_=ID_1));
```

```
 ID_2=ID_1;
 missing='0';
 keep ID_1 ID_2 missing;
run;

data sample;
 set head_sample var;
run;

proc export
 data=sample
 dbms=tab
 outfile='Path:\CEU_low_sample.txt'
 replace;
 putnames=yes;
run;

/*************************************************************************/
/*End*/
/*************************************************************************/
```

SHAPEIT2 commands In the following, the commands are presented that were applied to phase real genotypes into haplotypes using SHAPEIT2.

```
--input -gen CEU_low.gen CEU_low.sample -M genetic_map_chr22_combined_b37.txt -O
CEU_low.phased
```

SAS program 5 This SAS program processes phased haplotypes and creates input files for Genetree.

```
/*************************************************************************
* Program: 05_import_SHAPEIT_export_toGENETREE.sas*
* Author: Maria Kabisch*
* Date: 26/09/2014*
*************************************************************************
* Purpose of program: processes phased haplotypes,*
* exports input files for Genetree*
*************************************************************************
* Input files: CEU_low_phased.csv,ALL_pop.txt*
* Output files: id_snps.txt,ALL_haps.txt,ALL_check.txt
*************************************************************************/

%let snp_num=205;

libname real "Path:\";

/*import estimated haplotypes*********************************************/

proc import datafile="Path:\CEU_low_phased.csv"
 dbms=csv
 out=haps
 replace;
 delimiter=';';
 getnames=no;
run;
```

```
data red;
 retain states VAR2;
 set haps;
 drop VAR1 VAR3 VAR4 VAR5;
run;

proc transpose data=red out=haps_trans;
 var VAR6 -- VAR2189;
 id VAR2;
run;

data h_ident;
 set haps_trans;
 index=_n_;
 h_id_hilfe='h_'||index;
 h_id=compress(h_id_hilfe,'');
 drop h_id_hilfe;
run;

data d_ident;
 set h_ident;
 if mod(index,2)=1 then d_pair='a';
 if mod(index,2)=0 then d_pair='b';
 drop index;
run;

data d_a;
 set d_ident;
 where d_pair eq 'a';
 index=_n_;
  proc sort;
   by index;
  run;
run;

data d_b;
 set d_ident;
 where d_pair eq 'b';
 index=_n_;
  proc sort;
   by index;
  run;
run;

data d_ab;
 set d_a d_b;
  proc sort;
   by index;
  run;
run;

data d_ident2;
 retain h_id d_id;
 set d_ab;
 d_id_hilfe='d_'||index||d_pair;
 d_id=compress(d_id_hilfe,'');
 drop index d_pair d_id_hilfe _NAME_;
run;
```

```
proc import datafile="Path:\ALL_pop.txt"
 dbms=tab
 out=pop
 replace;
 getnames=yes;
run;

/*merge and save population identifier***********************************/

data pop_index;
 set pop;
 index=_n_;
  proc sort;
   by index;
  run;
run;

data id_index;
 set d_ident2;
 hilfe=compress(d_id,'d_ab');
 index=hilfe*1;
 drop hilfe;
  proc sort;
   by index;
  run;
run;

data pop_id;
 merge pop_index id_index;
 by index;
run;

data real.id_pop;
 set pop_id;
run;

/*remove uninformative sites*********************************************/

data uninform;
 set d_ident2;
 drop h_id d_id;
run;

data _null_;
 length allvars $5000;
 retain allvars ' ';
 set red end=eof;
 allvars=trim(left(allvars))||' '||left(VAR2);
 if eof then call symput('snp_names',allvars);
run;

%macro snps();

data snp_char;
 set uninform;
  %do i=1 %to &snp_num.;
   char_%scan(&snp_names,&i)=put(%scan(&snp_names,&i),1.);
   drop %scan(&snp_names,&i);
```

```
  %end;
run;

data snp_char2;
 set snp_char;
  %do i=1 %to &snp_num.;
   %scan(&snp_names,&i)=char_%scan(&snp_names,&i);
   drop char_%scan(&snp_names,&i);
  %end;
run;

%mend snps;
%snps();

data check;
 set d_ident2;
run;

%let lib=WORK;
%let mem=snp_char2;

proc sql noprint;
 select name, put(count(name),32.-L) into :clist separated by ' ' , :charct
 from dictionary.columns
 where libname=upcase("&lib") and memname=upcase("&mem") and type='char';
quit;

data _null_;
 array char(*) $ &clist;
 array c_allmiss (&charct) $ (&charct*'true');
  set &mem end=done;
   do i=1 to dim(c_allmiss);
    if char(i) ne '0' then c_allmiss(i)='false';
   end;
  if done then do;
   cnt=0;
    do i= 1 to dim(c_allmiss);
      if c_allmiss(i) ='true' then do;
      cnt+1;
      call symput('var'||put(cnt,3.-l),vname(char(i)));
    end;
   end;
  call symput('cnt',put(cnt,3.-l));
 end;
run;

%macro dropem;
 %do i = 1 %to &cnt;
  &&var&i.
 %end;
%mend;

data check_inform;
 set check;
 drop %dropem ;
run;

data real;
 set check_inform;
```

```
 drop h_id d_id;
run;

proc transpose data=real out=real_trans;
 var rs116333571 -- rs483926;
run;
data real2;
 retain states;
 set real_trans;
 hilfe1='state_';
 hilfe2=_n_;
 states=compress(hilfe1)||compress(hilfe2);
 *drop hilfe1 hilfe2 _NAME_;
 keep states _NAME_;
run;

proc export
 data=real2
 dbms=tab
 outfile="Path:\id_snps.txt"
 replace;
 putnames=no;
run;

proc transpose data=real2 out=real2_trans;
 var col1 -- col2184;
 id states;
run;

data real3;
 set real2_trans;
 drop _NAME_;
run;

/*save real haplotypes*****************************************************/

data real.all_real_haps_ALL;
 set real3;
run;

proc export
 data=real3
 dbms=tab
 outfile="Path:\ALL_haps.txt"
 replace;
 putnames=no;
run;

/*prepare for genetree*****************************************************/

data together;
 length all_states $32767.;
 set real3;
 hilfe=state_1||state_6||state_14||state_18||state_25||state_48||state_55
 ||state_62||state_65||state_75||state_78||state_90||state_94||state_114
 ||state_133||state_148||state_159||state_188||state_197||state_203;
 all_states=compress(hilfe,' ');
run;
```

```
data together_red;
 set together;
 keep all_states;
  proc sort nodupkey;
   by all_states;
  run;
run;

data genetree;
 retain freq trenn all_states;
 set together_red;
 freq=1;
 trenn=':';
  do i = length(all_states)-1 to 1 by -1;
   all_states = substr(all_states,1,i) || ' ' || substr(all_states,i+1);
  end;
 drop i;
run;

proc export
 data=genetree
 dbms=tab
 outfile="Path:\ALL_check.txt"
 replace;
 putnames=no;
run;

/*************************************************************************/
/*End*/
/*************************************************************************/
```

Genetree commands Based on all real genotypes Genetree exports a *nxn* matrix indicating compatible and incompatible SNPs.

```
seq2tr ALL_check.dat CEU_low_comp.txt
```

Subsequently a subset of ten percent compatible SNPs was selected, which served as measured SNP sites in the study population for genotype imputation in real data scenarios. Imputation accuracy was then calculated as shown above in SAS program 3.