`mlRho`, v. 2.7: Maximum Likelihood Estimation of Recombination Rate from Resequencing Data

Bernhard Haubold[1], Peter Pfaffelhuber[2] & Michael Lynch[3]
[1]Max-Planck-Institute for Evolutionary Biology, Plön
[2]Mathematical Institute, Albert-Ludwigs University, Freiburg, Germany
[3]Department of Biology, Indiana University, Bloomington

October 17, 2013

1 Introduction

Given shotgun data from a single diploid individual, `mlRho` computes maximum likelihood estimators of the following four quantities:

1. mutation rate, $\theta = 4N_{\rm e}\mu$,
2. error rate, ε,
3. zygosity correlation, Δ, and
4. recombination rate, $\rho = 4N_{\rm e}c$.

The background to these computations is explained in [6, 3]. In Sections 2 and 3 we describe the input data for `mlRho` and how to get started with the program. Section 4 is a tutorial on the main features of `mlRho`. For those wishing to understand `mlRho` in detail, Section 6 lists its C sources.

2 Input Data

2.1 Profile Format

The raw input data consists of multiple contigs each comprising multiple positions that have been sequenced to some depth. Here is an example:

```
>Contig1
1 0  0  0 5
2 1 10  0 0
>Contig2
5  2  0 7 0
6 12  0 0 0
8  0 10 1 0
```

Every contig starts with a header line, which must begin with $>$. Contig1 in our example consists of two sequenced positions. Each sequenced nucleotide is represented by one row of tab-delimited numbers giving its position in the contig, and the counts of `A`, `C`, `G`, and `T` (in this order) observed at a given alignment position. We call such a quartet of nucleotide counts a *profile*. The *coverage* refers to the number of times a certain position has been sequenced. Hence the coverage is equal to the quartet sum at a given position. In our example, position 1 in Contig1 has coverage 5 and position 2 has coverage 11. Positions need to be ordered but, as shown in Contig2, they need not start at 1 and may contain gaps.

Table 1: Simulation programs

Program	Author	Description	Reference	URL
`ms`	Richard R. Hudson	Coalescent simulation	[4]	`home.uchicago.edu/ ~rhudson1/source/mksamples.html`
`ms2dna`	Bernhard Haubold & Peter Pfaffelhuber	Convert `ms` output to DNA sequences	unpublished	`guanine.evolbio.mpg.de/bioBox`
`sequencer`	Bernhard Haubold	Simulate shotgun sequencing	unpublished	`guanine.evolbio.mpg.de/bioBox`
`formatPro`	Format profiles	Bernhard Haubold	unpublished	`guanine.evolbio.mpg.de/mlRho`

3 Getting Started

`mlRho` is written in C on a computer running Linux; it is intended to run on any UNIX system with a C compiler and the GNU Scientific Library installed [2]. However, please contact BH at `haubold@evolbio.mpg.de` if you have problems with the program.

- Make sure the GNU Scientific Library is installed

  ```
  gsl-config
  ```

  `mlRho` does not compile without this library.

- Unpack `mlRho`

  ```
  tar -xvzf mlRho_XXX.tgz
  ```

  where `XXX` indicates the version.

- Change into the newly created directory

  ```
  cd MlRho_XXX
  ```

  and generate `mlRho`

  ```
  make
  ```

- List its options

  ```
  ./mlRho -h
  ```

4 Tutorial

First we look at simulated data to assess the accuracy of `mlRho`, before applying it to two real data sets: *Ciona intestinalis* and human.

4.1 Simulated Data

We simulate data under the model that underlies `mlRho`, neutral evolution with constant population size and reciprocal recombination. Hence you need to install the three programs listed in Table 1 to run the following examples.

- Here is the central simulation code:

```
for i in $(seq 1000)
do
    ms 2 1 -t 1000 -r 500 100000    |
    ms2dna                          |
    sequencer -c 4 -p
done                                |
formatPro
mlRho -M 0 -I
mlRho -m 1000 -M 1005
```

The third line means: generate one gene sample of size 2, mutate with rate $\theta = -\texttt{t} = 1000$ and recombine with rate $\rho = -\texttt{r} = 500$ among 100,000 nucleotides. The output from `ms` is then converted to DNA sequence, shotgun sequenced to a coverage of 4, formatted, and finally analyzed by `mlRho`. This code is contained in

```
Scripts/sim.sh
```

and you can run it

```
sh Scripts/sim.sh
```

This takes approximately 2 minutes. The results should be similar to

```
#Positions written to profileDb.pos
#Profile summary written to profileDb.sum
#Contig lengths written to profileDb.con
d n theta epsilon -log(L)
0 95739917 9.30e-03<9.32e-03<9.34e-03 1.07e-03<1.07e-03<1.07e-03 1.46e+08
#Likelihoods written to profileDb.lik
d n theta epsilon -log(L) delta rho
0 95739917 9.30e-03<9.32e-03<9.34e-03 1.07e-03<1.07e-03<1.07e-03 1.46e+08
1000 17077344 5.21e+07 1.45e-03<2.00e-03<2.56e-03 7.08e-03<4.93e-03<3.62e-03
1001 17047829 5.20e+07 1.28e-03<1.82e-03<2.38e-03 8.12e-03<5.49e-03<3.97e-03
1002 17018361 5.19e+07 9.60e-04<1.49e-03<2.05e-03 1.09e-02<6.85e-03<4.78e-03
1003 16988937 5.18e+07 1.56e-03<2.10e-03<2.67e-03 6.55e-03<4.62e-03<3.41e-03
1004 16959567 5.17e+07 1.28e-03<1.82e-03<2.38e-03 8.09e-03<5.47e-03<3.95e-03
1005 16930252 5.16e+07 1.03e-03<1.57e-03<2.13e-03 1.01e-02<6.47e-03<4.55e-03
```

We get $\theta = 0.00932$ when it should be $1000/100,000 = 0.01$ and $\epsilon = 0.00107$, when it should be 0.001. ρ fluctuates between 0.00462 and 0.00647 when it should be $500/100,000 = 0.005$. Run the script again with double the coverage (`-c 8`) to achieve more accurate results:

```
sh Scripts/sim.sh
#Positions written to profileDb.pos
#Profile summary written to profileDb.sum
#Contig lengths written to profileDb.con
d n theta epsilon -log(L)
0 99966470 9.90e-03<9.92e-03<9.94e-03 1.00e-03<1.00e-03<1.00e-03 1.57e+08
#Likelihoods written to profileDb.lik
d n theta epsilon -log(L) delta rho
0 99966470 9.90e-03<9.92e-03<9.94e-03 1.00e-03<1.00e-03<1.00e-03 1.57e+08
1000 98056994 3.08e+08 1.96e-03<2.18e-03<2.40e-03 5.42e-03<4.78e-03<4.24e-03
1001 98055097 3.08e+08 1.77e-03<1.99e-03<2.21e-03 6.10e-03<5.33e-03<4.71e-03
1002 98053201 3.08e+08 1.98e-03<2.20e-03<2.43e-03 5.33e-03<4.71e-03<4.18e-03
1003 98051305 3.08e+08 1.81e-03<2.03e-03<2.25e-03 5.92e-03<5.19e-03<4.58e-03
1004 98049409 3.08e+08 1.96e-03<2.18e-03<2.40e-03 5.41e-03<4.77e-03<4.24e-03
1005 98047514 3.08e+08 2.01e-03<2.24e-03<2.46e-03 5.22e-03<4.61e-03<4.11e-03
```

Now $\theta = 0.00992$, $\epsilon = 0.001$, and $0.00461 \leq \rho \leq 0.00533$.

4.2 Real Data

4.2.1 *Ciona intestinalis*

- Download the *Ciona intestinalis* profiles [1] from the `mlRho` web site:

  ```
  guanine.evolbio.mpg.de/mlRho/
  ```

- Take a look at the options of `formatPro`, which you installed in Section 4.1

  ```
  formatPro -h
  Usage: formatPro [options] [inputFiles]
  Format profiles for subsequent analysis with proStats and/or mlRho
  formatPro myProfiles1.pro myProfiles2.pro
  Options:
  [-n <FileName>; default: profileDb]
  [-b <NUM> buffer size; default: 1024]
  [-c <NUM> minimum coverage; default: 4]
  [-r create only profiles file; default: profiles, contigs, and positions]
  [-h print this help message and exit]
  [-v print program information and exit]
  ```

- Format the *Ciona* profiles that have at least coverage 4:

  ```
  formatPro ci.pro
  #Positions written to profileDb.pos
  #Profile summary written to profileDb.sum
  #Contig lengths written to profileDb.con
  ```

- To speed up `mlRho`, the three `profileDb.*` files are binary, which means that you cannot just inspect them in an editor. But you can do this using the program `inspectPro`, which is also posted on the `mlRho` web site:

  ```
  http://guanine/evolbio.mpg.de/mlRho/
  ```

  Take a look at the profile summary file

  ```
  inspectPro profileDb.sum | head
  #ID Count A C G T
  0 1498864 10 0 0 0
  1 1498389 0 0 0 10
  2 855309 0 0 10 0
  3 853797 0 10 0 0
  4 3031 6 0 4 0
  5 1303132 0 0 0 11
  6 750275 0 0 11 0
  7 747983 0 11 0 0
  8 1303368 11 0 0 0
  ```

  This means that profile 0 was found 1,498,864 times and consists of 10 `A`s, and so on.

- If you'd like to know which profile was found most frequently, type

  ```
  inspectPro profileDb.sum | tail -n +2 | sort -n -k 2 -r | head
  33 1768945 0 0 0 7
  30 1766461 7 0 0 0
  ```

```
18 1753782 8 0 0 0
15 1751975 0 0 0 8
24 1710000 0 0 0 6
20 1709480 6 0 0 0
11 1650977 9 0 0 0
12 1650260 0 0 0 9
27 1568026 0 0 0 5
26 1560829 5 0 0 0
```

- Take a look at the contig file

  ```
  inspectPro profileDb.pos | head
  #Pos Pro
  1 0
  2 1
  3 2
  4 2
  5 0
  6 0
  7 2
  8 2
  9 1
  ```

  This means that position 1 is occupied by profile 0, position 2 by profile 1, and so on.

- The contigs are listed in `profileDb.con`

  ```
  inspectPro profileDb.con | head
  #ID Length
  0 8201
  1 8927
  2 8202
  3 7983
  4 8379
  5 7129
  6 6604
  7 8810
  8 9428
  ```

  Again, we can write

  ```
  inspectPro profileDb.con | tail -n +2 | sort -k 2 -n -r | head
  1033 489307
  1534 452905
  1659 440643
  1770 375642
  1992 361548
  1881 350620
  2214 341381
  2103 339997
  2325 338882
  221 329268
  ```

  to find that the longest contig consists of 489,307 positions with coverage ≥ 4.

- Compute the genome-wide population mutation rate:

```
mlRho -M 0 -I
d n theta epsilon -log(L)
0 54423190 1.17e-02<1.17e-02<1.18e-02 2.44e-03<2.45e-03<2.45e-03 8.57e+07
#Likelihoods written to profileDb.lik
```

At zero distance (`d`), i.e. all positions are looked at individually, 54,423,190 (`n`) profiles were analyzed to determine a mean $\theta = 0.0117$ with 95% confidence interval $0.0117 \leq \theta \leq 0.0118$. The sequencing error is $\epsilon = 0.00245$ and the negative log-likelihood of the data given these values of θ and ϵ is 8.57×10^7.

- The `-I` switch in the last command instructed `mlRho` to write the likelihoods to the binary file `profileDb.lik`:

```
inspectPro profileDb.lik  | head
#id l_1 l_2
0 3.089e-01 2.838e-04
1 3.086e-01 2.837e-04
2 1.795e-01 1.969e-04
3 1.788e-01 1.964e-04
4 2.911e-11 1.604e-02
5 3.078e-01 1.416e-04
6 1.790e-01 9.828e-05
7 1.784e-01 9.802e-05
8 3.082e-01 1.417e-04
```

where `#id` refers to the profile, and l_1 and l_2 to its likelihood given that the site is homozygous or heterozygous, respectively.

- To compute Δ and ρ as a function of Δ, use

```
mlRho -m 100 -M 200 | head
d n theta epsilon -log(L) delta rho
0   54423190 1.17e-02<1.17e-02<1.18e-02 2.44e-03<2.45e-03<2.45e-03 8.57e+07
100 30253330 9.51e+07 8.23e-03<8.74e-03<9.25e-03 5.90e-03<4.70e-03<3.61e-03
101 30174102 9.49e+07 8.11e-03<8.62e-03<9.13e-03 6.14e-03<4.92e-03<3.82e-03
102 30123188 9.47e+07 9.24e-03<9.76e-03<1.03e-02 3.56e-03<2.58e-03<1.69e-03
103 30075459 9.46e+07 8.21e-03<8.72e-03<9.23e-03 5.77e-03<4.60e-03<3.54e-03
104 30029489 9.44e+07 8.04e-03<8.55e-03<9.06e-03 6.14e-03<4.93e-03<3.84e-03
105 29984169 9.43e+07 8.54e-03<9.05e-03<9.57e-03 4.90e-03<3.82e-03<2.84e-03
106 29939659 9.41e+07 7.70e-03<8.20e-03<8.71e-03 6.91e-03<5.64e-03<4.49e-03
107 29897070 9.40e+07 7.70e-03<8.21e-03<8.72e-03 6.83e-03<5.57e-03<4.43e-03
```

4.2.2 Human

The alignments generated by the 1000 Genomes Project [7] are accessible over the internet as `bam` files. This is currently the de facto standard for distributing genome-scale alignments. To access `bam` files, you need to have `samtools` [5] installed on your computer.

- The file `Scripts/bam.sh` contains an example of how to access positions 1–20,000,000:

```
samtools view -b ftp://ftp.1000genomes.ebi.ac.uk/vol1/ftp/data/HG00096/
    alignment/HG00096.mapped.ILLUMINA.bwa.GBR.low_coverage.20120522.bam
    1:1-20000000 |
samtools mpileup - |
cut -f 2,5 |
awk -f ./Scripts/bam2pro.awk |
formatPro
mlRho -M 0 -I
mlRho -m 1000 -M 1005
```

- It calls the AWK-script `Scripts/bam2pro.awk`:

```
BEGIN{print ">Contig";
}{
  if($2 !~ /[^acgtACGT]/){
    s["A"]=0;
    s["C"]=0;
    s["G"]=0;
    s["T"]=0;
    s["a"]=0;
    s["c"]=0;
    s["g"]=0;
    s["t"]=0;
    for(i=1;i<=length($2);i++)
      s[substr($2,i,1)]++;
    print $1 "\t" s["A"]+s["a"] "\t" s["C"]+s["c"] "\t" s["G"]+s["g"] "
       \t" s["T"]+s["t"];
  }
}
```

- Run `bam.sh`

```
sh Scripts/bam.sh
```

5 Frequently Asked Question

- Why does `mlRho` produce negative ρ-values? `mlRho` estimates the population recombination rate, ρ, from the disequilibrium measure Δ using the following equation [3]:

$$\Delta = \frac{\theta(18+\rho+18\theta+\rho\theta+4\theta^2)}{18+13\rho+\rho^2+54\theta+40\theta^2+8\theta^3+\rho(\rho\theta+19\theta+6\theta^2)}. \tag{1}$$

  By solving this for ρ we get

$$\begin{aligned}\rho &= \frac{1}{2(\Delta+\Delta\theta)}\\ &\quad\times\Bigg(\theta-13\Delta-19\Delta\theta+\theta^2-6\Delta\theta^2+\sqrt{1+\theta}\\ &\quad\times\sqrt{97\Delta^2+46\Delta\theta+109\Delta^2\theta+\theta^2+34\Delta\theta^2+32\Delta^2\theta^2+\theta^3+4\Delta\theta^3+4\Delta^2\theta^3}\Bigg).\end{aligned}$$

  If you plug in, for example, $\theta = 0.219$ and $\Delta = 0.484$, then $\rho = -1.44$. This makes no sense, but is a legitimate result under equation (1).

6 Listings

The following listings document central parts of the code for `mlRho`.

6.1 The Driver Program: `mlRho.c`

```
/***** mlRho.c ************************************
 * Description: Maximum-likelihood estimation of
 *   mutation and recombination rates from re-
```

```c
 *   sequencing data.
 * Author: Bernhard Haubold, haubold@evolbio.mpg.de
 * Date: Wed Feb 18 16:35:16 2009.
 ***************************************************/
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <unistd.h>
#include <assert.h>
#include "eprintf.h"
#include "interface.h"
#include "ld.h"
#include "profile.h"
#include "profileTree.h"
#include "mlComp.h"

void runAnalysis(Args *args);
void freeMem(Node **profilePairs, int numProfiles);

int main(int argc, char *argv[]){
  Args *args;
  char *version;

  version = "2.7";
  setprogname2("mlRho");
  args = getArgs(argc, argv);
  if(args->v)
    printSplash(version);
  if(args->h || args->e)
    printUsage(version);
  runAnalysis(args);
  free(args);
  free(progname());
  return 0;
}

void runAnalysis(Args *args){
  Result *r;
  int i;
  int numProfiles;
  char *headerPi, *headerDeltaRho, *outStrPi, *outStrDeltaRho;
  char *inclPro; /* indicates whether or not to include a profile */
  double numPos, c, n;
  Profile *profiles;
  ContigDescr *contigDescr;
  FILE *fp;
  Node **profilePairs;

  headerPi = "d\tn\ttheta\t\t\t\tepsilon\t\t\t\t-log(L)\n";
  headerDeltaRho = "d\tn\ttheta\t\t\t\tepsilon\t\t\t\t-log(L)\t\tdelta\t\t\
     t\trho\n";
  outStrPi = "%d\t%.0f\t%8.2e<%8.2e<%8.2e\t%8.2e<%8.2e<%8.2e\t%8.2e\n";
  outStrDeltaRho = "%d\t%.0f\t\t\t\t\t\t\t\t\t%8.2e\t%8.2e<%8.2e<%8.2e\t
     %8.2e<%8.2e<%8.2e\n";
  r = newResult();
  /* heterozygosity analysis */
  readProfiles(args->n);
  numProfiles = getNumProfiles();
  profiles = getProfiles();
  if(args->M == 0 && numProfiles)
    printf("%s", headerPi);
  else
    printf("%s", headerDeltaRho);
  if(numProfiles){
    r = estimatePi(profiles,numProfiles,args,r);
    numPos = piComp_getNumPos(profiles, numProfiles);
    if(args->c){
      /* including the correction from Lynch (2008), p. 2412 */
      n = getCoverage();
      c = n*pow(0.5,n-1);
      c = 1-c;
      r->pLo /= c;
      r->pi /= c;
      r->pUp /= c;
    }
    printf(outStrPi,0,numPos,r->pLo,r->pi,r->pUp,r->eLo,r->ee,r->eUp,r->l);
  }
  fflush(NULL);
  /* linkage analysis */
  fp = iniLdAna(args);
  inclPro = filterPro(r->pi,args->f);
  contigDescr = getContigDescr();
  profilePairs = NULL;
  if(args->o)
    readPositions(fp, contigDescr);
  for(i=args->m;i<=args->M;i+=args->S){
    profilePairs = getProfilePairs(numProfiles, inclPro, contigDescr, fp,
        args, i);
    r = estimateDelta(profilePairs,numProfiles,args,r,i);
    printf(outStrDeltaRho,i,getNumPos(),r->l,r->dLo,r->de,r->dUp,r->rLo,r->
        rh,r->rUp);
    fflush(NULL);
  }
  fclose(fp);
  if(args->I){
    if(args->c){
      /* save uncorrected values */
      n = getCoverage();
      c = n*pow(0.5,n-1);
      c = 1-c;
      r->pLo *= c;
      r->pi *= c;
      r->pUp *= c;
    }
    writeLik(args->n,r);
  }
  free(r);
  freeMem(profilePairs, numProfiles);
```

```
}

void freeMem(Node **profilePairs, int numProfiles){
  int i;
  double *lOnes, *lTwos;
  Profile *profiles;
  ContigDescr *cd;

  if(profilePairs){
    for(i=0;i<numProfiles;i++)
      freeTree(profilePairs[i]);
    free(profilePairs);
  }
  cd = getContigDescr();
  if(cd){
    if(cd->pos){
      for(i=0;i<cd->n;i++)
        free(cd->pos[i]);
      free(cd->pos);
    }
    free(cd->posBuf);
    free(cd->len);
    free(cd);
  }
  lOnes = getLones();
  lTwos = getLtwos();
  profiles = getProfiles();
  free(lOnes);
  free(lTwos);
  free(profiles);
  freeMlComp();
}
```

6.2 Count Nucleotide Profiles: `profileTree.c`

```
/***** profileTree.c ****************************
 * Description: Tree for counting allele profiles.
 * Author: Bernhard Haubold, haubold@evolbio.mpg.de
 * Date: Wed Feb 18 17:06:35 2009.
 **********************************************/
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <string.h>
#include <assert.h>
#include <fcntl.h>
#include <unistd.h>
#include <math.h>
#include "eprintf.h"
#include "stringUtil.h"
#include "interface.h"
#include "profileTree.h"
#include "ld.h"

double numPos;
```

```
Node **resetProfilePairs(Node **profilePairs, int numProfiles);
Node **countPairs(Node **pairs, int numProfiles, char *inclPro, ContigDescr
     *contigDescr, FILE *fp, int dist, int memory);

Node **getProfilePairs(int numProfiles, char *inclPro, ContigDescr *
    contigDescr, FILE *fp, Args *args, int d){
  static Node **profilePairs = NULL;
  int i;

  numPos = 0;
  profilePairs = resetProfilePairs(profilePairs,numProfiles);
  if(args->L){
    for(i=0;i<args->S;i++)
      profilePairs = countPairs(profilePairs, numProfiles, inclPro,
          contigDescr, fp, d+i, args->o);
  }else
    profilePairs = countPairs(profilePairs, numProfiles, inclPro,
        contigDescr, fp, d, args->o);

  return profilePairs;
}

Node **resetProfilePairs(Node **profilePairs, int numProfiles){
  int i;

  if(profilePairs == NULL){
    profilePairs = (Node **)emalloc(numProfiles * sizeof(Node *));
    for(i=0;i<numProfiles;i++)
      profilePairs[i] = NULL;
  }else{
    for(i=0;i<numProfiles;i++){
      freeTree(profilePairs[i]);
      profilePairs[i] = NULL;
    }
  }
  return profilePairs;
}

/* readPositions: read position information into contig descriptor
 *   assuming that fp points to the position file
 */
void readPositions(FILE *fp, ContigDescr *cd){
  int i, numRead;

  fseek(fp,3,SEEK_SET);
  cd->pos = (Position **)emalloc(cd->n*sizeof(Position *));
  for(i=0;i<cd->n;i++){
    cd->pos[i] = (Position *)emalloc(cd->len[i]*sizeof(Position));
    numRead = fread(cd->pos[i],sizeof(Position),cd->len[i],fp);
    assert(numRead == cd->len[i]);
  }

}
```

```
Node **countPairs(Node **pairs, int numProfiles, char *inclPro, ContigDescr
     *contigDescr, FILE *fp, int dist, int memory){
  int a, b, i, l, r, tmp, numRead, bound;
  Position *pb;

  pb = NULL;
  if(!memory){
    fseek(fp,3,SEEK_SET);
    pb = contigDescr->posBuf;
  }
  for(i=0;i<contigDescr->n;i++){
    if(memory)
      pb = contigDescr->pos[i];
    else{
      numRead = fread(pb,sizeof(Position),contigDescr->len[i],fp);
      assert(numRead == contigDescr->len[i]);
    }
    l = 0;
    r = 0;
    bound = contigDescr->len[i]-1;
    while(r<contigDescr->len[i]){
      while(pb[r].pos - pb[l].pos < dist && r < bound)
        r++;
      while(pb[r].pos - pb[l].pos > dist && l<=r)
        l++;
      if(pb[r].pos-pb[l].pos == dist){
        a = pb[l].pro;
        b = pb[r].pro;
        if(inclPro[a] && inclPro[b]){
          if(a > b){ /* sort indexes to halve the number of index pairs */
            tmp = b;
            b = a;
            a = tmp;
          }
          pairs[b] = addTree(pairs[b],a);
          numPos++;
        }
      }
      r++;
      l++;
    }
  }
  return pairs;
}


/* addTree: add key to tree */
Node *addTree(Node *node, int key){

  if(node == NULL)   /* new key has arrived */
    node = newNode(key);
  else if(node->key == key)
    node->n++;       /* repeated key */
  else if(node->key < key) /* descend into left subtree */
    node->left = addTree(node->left, key);
  else              /* descend into right subtree */
    node->right = addTree(node->right, key);

  return node;
}

/*newNode: generate and initialize new node */
Node *newNode(int key){
  Node *node;

  node = (Node *)emalloc(sizeof(Node));
  node->key = key;
  node->n = 1;
  node->left = NULL;
  node->right = NULL;

  return node;
}

/* freeTree: traverse tree and free the memory for each node */
void freeTree(Node *n){
  if(n != NULL){
    freeTree(n->left);
    freeTree(n->right);
    free(n);
  }
}

double getNumPos(){
  return numPos;
}
```

6.3 Maximum Likelihood Computation Part I: `mlComp.c`

```
/***** mlComp.c **********************************
 * Description: ML computations.
 * Author: Bernhard Haubold, haubold@evolbio.mpg.de
 * Date: Thu Feb 19 09:56:43 2009
 ************************************************/
#include <stdio.h>
#include <assert.h>
#include "eprintf.h"
#include "interface.h"
#include "profile.h"
#include "profileTree.h"
#include "mlComp.h"

double *freqNuc;
double totalNuc;
double S, logS;
double likelihood;
double lo2;
double *lngamma;
```

```
gsl_sf_result *result;
int maxCov;
int *coverages;

void compS();
inline double powInt(double x, int y);

int *getCoverages(){
  return coverages;
}

void iniMlComp(Profile *profiles, int numProfile){
  int i, j;

  freqNuc = (double *)emalloc(4*sizeof(double));
  result = (gsl_sf_result *)emalloc(sizeof(gsl_sf_result));
  for(i=0;i<4;i++)
    freqNuc[i] = 0.0;
  totalNuc = 0;
  maxCov = 0;
  coverages = (int *)emalloc(numProfile*sizeof(int));
  for(i=0;i<numProfile;i++){
    coverages[i] = 0;
    for(j=0;j<4;j++){
      freqNuc[j] += (profiles[i].profile[j] * profiles[i].n);
      coverages[i] += profiles[i].profile[j];
    }
    if(maxCov < coverages[i])
      maxCov = coverages[i];
  }
  lngamma = (double *)emalloc((maxCov+2)*sizeof(double));
  for(i=1;i<=maxCov+1;i++)
    lngamma[i] = gsl_sf_lngamma(i);
  for(i=0;i<4;i++)
    totalNuc += freqNuc[i];
  for(i=0;i<4;i++)
    freqNuc[i] /= totalNuc;

  compS();
}

/* lOne: equation (4a) of Lynch (2008) as revised by Stephen Bates on June
    24, 2012 */
inline double lOne(int cov, int *profile, double ee){
  int i;
  double s, compEe, eeThird, g;

  s = 0.0;
  compEe = 1.0 - ee;
  eeThird = ee / 3.0;
  g = 0.0;
  for(i=0;i<4;i++){
    s += freqNuc[i] * powInt(compEe, profile[i]) * powInt(eeThird,cov-
        profile[i]);
    g += lngamma[profile[i]+1];
  }
  g = exp(lngamma[cov+1]-g);
  s *= g;
  return s;
}

/* lTwo: equation (4b) of Lynch (2008) as revised by Stephen Bates on June
   24, 2012 */
inline double lTwo(int cov, int *profile, double ee){
  int i, j;
  double s, g, x, eeThird;

  /* compute multinomial coefficient */
  g = 0.0;
  for(i=0;i<4;i++)
    g += lngamma[profile[i]+1];
  g = exp(lngamma[cov+1]-g);
  /* compute likelihood */
  s = 0.0;
  eeThird = ee/3.0;
  x = (1.0-2.0*eeThird)/2.0;
  for(i=0;i<4;i++)
    for(j=i+1;j<4;j++){
      s += freqNuc[i]*freqNuc[j]/S * powInt(x,profile[i]+profile[j]) *
         powInt(eeThird,cov-profile[i]-profile[j]);
    }
  s *= g;
  return s;
}

inline double powInt(double x, int y){
  int i;
  double p;

  p = 1.;
  for(i=0;i<y;i++)
    p *= x;

  return p;
}

/* compS: compute global variable S */
void compS(){
  int i;

  S = 0.0;
  for(i=0;i<4;i++)
    S += freqNuc[i]*freqNuc[i];
  S = 1.0 - S;
  logS = log(S);
}

/* setFreqNuc: set nucleotide frequencies for testing purposes */
```

```
void setFreqNuc(double *f)
{
  freqNuc = f;
}

void freeMlComp()
{
  /* int i; */

  free(freqNuc);
  free(result);
  free(lngamma);
  free(coverages);
}

Result *newResult(){
  Result *r;
  r = (Result *)calloc(1,sizeof(Result));
  assert(r);
  return r;
}
```

6.4 Maximum Likelihood Computation Part II: `piComp.c`

```
/***** piComp.c ***********************************
 * Description: Compute heterozygosity, pi, for
 *   single diploid individual.
 * Reference: Lynch, M. (2008). Estimation of nuc-
 *   leotide diversity, disequilibrium coefficients,
 *   and mutation rates from high-coverage genome-
 *   sequencing projects. Mol. Biol. Evol. 25:
 *   2409-2419.
 * Author: Bernhard Haubold, haubold@evolbio.mpg.de
 * Date: Tue Mar 17 21:20:57 2009
 **************************************************/
#include <float.h>
#include <gsl/gsl_errno.h>
#include <gsl/gsl_math.h>
#include <gsl/gsl_roots.h>
#include <assert.h>
#include <string.h>
#include <sys/stat.h>
#include "interface.h"
#include "profile.h"
#include "mlComp.h"
#include "eprintf.h"

double *lOnes = NULL;
double *lTwos = NULL;
double numPos;

double likP(Profile *profiles, int numProfiles, double pi, double ee);
double myP(const gsl_vector *v, void *params);
double myPconf(double x, void *params);
double myEconf(double x, void *params);
```

```
void confP(Args *args, Result *result);
void confE(Args *args, Result *result);
void compSiteLik(Profile *profiles, int numProfiles, double ee);
FILE *openLikFile(char *baseName);
Result *readLik(FILE *fp, int numProfiles, Result *result);

/* estimatePi: estimate pi and epsilon using the Nelder-Mead
 * Simplex algorithm; code adapted from Galassi, M., Davies,
 * J., Theiler, J., Gough, B., Jungman, G., Booth, M.,
 * Rossi, F. (2005). GNU Scientific Library Reference Manual.
 * Edition 1.6, for GSL Version 1.6, 17 March 2005, p 472f.
 */
Result *estimatePi(Profile *profiles, int numProfiles, Args *args, Result *
    result){
  size_t np;
  const gsl_multimin_fminimizer_type *T;
  gsl_multimin_fminimizer *s;
  gsl_vector *ss, *x;
  gsl_multimin_function minex_func;
  size_t iter;
  int status;
  double size;
  FILE *fp;

  if((fp = openLikFile(args->n)) != NULL && !args->I){
    result = readLik(fp,numProfiles,result);
    fclose(fp);
    return result;
  }

  np = 2;
  T = gsl_multimin_fminimizer_nmsimplex;
  s = NULL;
  iter = 0;

  /*set up likelihood computation */
  iniMlComp(profiles, numProfiles);
  /* initialize vertex size vector */
  ss = gsl_vector_alloc(np);
  /* set all step sizes */
  gsl_vector_set_all(ss, args->s);
  /* starting point */
  x = gsl_vector_alloc(np);
  gsl_vector_set(x, 0, args->P);
  gsl_vector_set(x, 1, args->E);
  /* initialize method and iterate */
  minex_func.f = &myP;
  minex_func.n = np;
  minex_func.params = (void *)NULL;
  s = gsl_multimin_fminimizer_alloc(T, np);
  gsl_multimin_fminimizer_set(s, &minex_func, x, ss);
  do{
    iter++;
    status = gsl_multimin_fminimizer_iterate(s);
```

```
      if(status)
        break;
      size = gsl_multimin_fminimizer_size(s);
      status = gsl_multimin_test_size(size, args->t);
    }while(status == GSL_CONTINUE && iter < args->i);
    if(GSL_CONTINUE)
      assert("ERROR[mlRho]:␣increase␣the␣number␣of␣iterations␣using␣option␣-i
         .\n");

    result->pi = gsl_vector_get(s->x, 0);
    result->ee = gsl_vector_get(s->x, 1);
    result->l = s->fval;
    result->i = iter;
    gsl_set_error_handler_off();
    confP(args, result);
    confE(args, result);
    gsl_vector_free(x);
    gsl_vector_free(ss);
    gsl_multimin_fminimizer_free(s);
    compSiteLik(getProfiles(),getNumProfiles(),result->ee);
    return result;
  }
  /* myF: function called during the minimization procedure */
  double myP(const gsl_vector* v, void *params){
    double pi, ee;
    double likelihood;

    pi = gsl_vector_get(v, 0);
    ee = gsl_vector_get(v, 1);
    likelihood = 0;
    if(pi < 0 || ee < 0 || pi > 1 || ee > 1){
      return DBL_MAX;
    }
    likelihood = likP(getProfiles(), getNumProfiles(), pi, ee);
    return -likelihood;
  }

  double likP(Profile *profiles, int numProfiles, double pi, double ee){
    int i;
    int *coverages;
    double l, likelihood, lOneLoc, lTwoLoc;

    coverages = getCoverages();
    likelihood = 0.;
    for(i=0;i<numProfiles;i++){
      lOneLoc = lOne(coverages[i],profiles[i].profile,ee);
      lTwoLoc = lTwo(coverages[i],profiles[i].profile,ee);
      l = lOneLoc * (1.0 - pi) + lTwoLoc * pi;
      if(l>0)
        likelihood += log(l) * profiles[i].n;
    }
    return likelihood;
  }
```

```
void compSiteLik(Profile *profiles, int numProfiles, double ee){
  int i;
  int *coverages;

  lOnes = (double *)emalloc(numProfiles*sizeof(double));
  lTwos = (double *)emalloc(numProfiles*sizeof(double));
  coverages = getCoverages();
  for(i=0;i<numProfiles;i++){
    lOnes[i] = lOne(coverages[i],profiles[i].profile,ee);
    lTwos[i] = lTwo(coverages[i],profiles[i].profile,ee);
  }
}

double piComp_getNumPos(Profile *profiles, int numProfiles){
  int i;

  if(profiles){
    numPos = 0;
    for(i=0;i<numProfiles;i++)
      numPos += profiles[i].n;
  }
  return numPos;
}

/* myPconf: called for confidence interval estimation of pi */
double myPconf(double x, void *params){
  Result *res;
  double likelihood;

  res = (Result *)params;

  likelihood = likP(getProfiles(), getNumProfiles(), x, res->ee);

  return likelihood + res->l + 2;
}

/* myEconf: called for confidence interval estimation of epsilon */
double myEconf(double x, void *params){
  Result *res;
  double l;
  double likelihood;

  res = (Result *)params;
  likelihood = likP(getProfiles(), getNumProfiles(), res->pi, x);

  l =  likelihood + res->l + 2;
  return l;
}

void confE(Args *args, Result *result){
  int status;
  int iter = 0;
  int max_iter = args->i;
  const gsl_root_fsolver_type *solverType;
  gsl_root_fsolver *s;
  double r = 0;
  gsl_function fun;
  double xLo, xHi;

  fun.function = &myEconf;
  fun.params = result;

  solverType = gsl_root_fsolver_brent;

  s = gsl_root_fsolver_alloc(solverType);
  /* search for upper bound */
  xLo = result->ee;
  xHi = 0.5;
  status = gsl_root_fsolver_set(s,&fun,xLo,xHi);
  if(status)
    result->eUp = 0.5;
  else{
    do{
      iter++;
      status = gsl_root_fsolver_iterate(s);
      r =gsl_root_fsolver_root(s);
      xLo = gsl_root_fsolver_x_lower(s);
      xHi = gsl_root_fsolver_x_upper(s);
      status = gsl_root_test_interval(xLo, xHi, 0, args->t);
    }while(status == GSL_CONTINUE && iter < max_iter);
    result->eUp = r;
  }
  /* search for lower bound */
  xLo = 0.0;
  xHi = result->ee;
  status = gsl_root_fsolver_set(s,&fun,xLo,xHi);
  if(status)
    result->eLo = 0;
  else{
    do{
      iter++;
      status = gsl_root_fsolver_iterate(s);
      r =gsl_root_fsolver_root(s);
      xLo = gsl_root_fsolver_x_lower(s);
      xHi = gsl_root_fsolver_x_upper(s);
      status = gsl_root_test_interval(xLo, xHi, 0, args->t);
    }while(status == GSL_CONTINUE && iter < max_iter);
    result->eLo = r;
  }
  gsl_root_fsolver_free(s);
}


void confP(Args *args, Result *result){
  int status;
  int iter = 0;
  int max_iter = args->i;
  const gsl_root_fsolver_type *solverType;
  gsl_root_fsolver *s;
  double r = 0;
  gsl_function fun;
  double xLo, xHi;

  fun.function = &myPconf;
  fun.params = result;

  solverType = gsl_root_fsolver_brent;

  s = gsl_root_fsolver_alloc(solverType);
  /* search for upper bound */
  xLo = result->pi;
  xHi = 0.75;
  status = gsl_root_fsolver_set(s,&fun,xLo,xHi);
  if(status)
    result->pUp = 0.75;
  else{
    do{
      iter++;
      status = gsl_root_fsolver_iterate(s);
      r =gsl_root_fsolver_root(s);
      xLo = gsl_root_fsolver_x_lower(s);
      xHi = gsl_root_fsolver_x_upper(s);
      status = gsl_root_test_interval(xLo, xHi, 0, 0.001);
    }while(status == GSL_CONTINUE && iter < max_iter);
    result->pUp = r;
  }
  /* search for lower bound */
  xLo = 0.0;
  xHi = result->pi;
  status = gsl_root_fsolver_set(s,&fun,xLo,xHi);
  if(status)
    result->pLo = 0;
  else{
    do{
      iter++;
      status = gsl_root_fsolver_iterate(s);
      r =gsl_root_fsolver_root(s);
      xLo = gsl_root_fsolver_x_lower(s);
      xHi = gsl_root_fsolver_x_upper(s);
      status = gsl_root_test_interval(xLo, xHi, 0, 0.001);
    }while(status == GSL_CONTINUE && iter < max_iter);
    result->pLo = r;
  }
  gsl_root_fsolver_free(s);
}

double *getLones(){
  return lOnes;
}

double *getLtwos(){
  return lTwos;
```

```
}

void writeLik(char *baseName, Result *result){
  char *fileName;
  int n;
  double np;
  FILE *fp;

  fileName = (char *)emalloc(256*sizeof(char));
  fileName = strcpy(fileName,baseName);
  fileName = strcat(fileName,".lik");

  fp = efopen(fileName,"wb");
  n = fwrite("lik",sizeof(char),3,fp);
  assert(n == 3);
  n = fwrite(result,sizeof(Result),1,fp);
  assert(n == 1);
  np = getNumProfiles();
  n = fwrite(&np,sizeof(double),1,fp);
  assert(n == 1);
  n = fwrite(getLones(),sizeof(double),getNumProfiles(),fp);
  assert(n == getNumProfiles());
  n = fwrite(getLtwos(),sizeof(double),getNumProfiles(),fp);
  assert(n == getNumProfiles());
  printf("#Likelihoods␣written␣to␣%s\n",fileName);
  free(fileName);
  fclose(fp);
}

FILE *openLikFile(char *baseName){
  char *fileName;
  FILE *fp;
  struct stat stbuf;
  int n;
  char *tag;

  fileName = (char *)emalloc(256*sizeof(char));
  tag = (char *)emalloc(4*sizeof(char));
  fileName = strcpy(fileName,baseName);
  fileName = strcat(fileName,".lik");
  if(stat(fileName,&stbuf) != -1){
    fp = efopen(fileName,"rb");
    n = fread(tag,sizeof(char),3,fp);
    tag[3] = '\0';
    assert(n == 3);
    if(strcmp(tag,"lik") != 0){
      assert(0);
    }
  }else
    fp = NULL;
  free(tag);
  free(fileName);
  return fp;
```

```
}

Result *readLik(FILE *fp, int numProfiles, Result *result){
  int n, i, sod;
  double np;

  assert(lOnes == NULL);
  assert(lTwos == NULL);
  fseek(fp,3,SEEK_SET);
  n = fread(result,sizeof(Result),1,fp);
  assert(n == 1);
  sod = sizeof(double);
  n = fread(&np,sod,1,fp);
  assert(n == 1 && np == numProfiles);
  lOnes = (double *)emalloc(numProfiles*sod);
  lTwos = (double *)emalloc(numProfiles*sod);
  for(i=0;i<numProfiles;i++){
    n = fread(&lOnes[i],sod,1,fp);
    assert(n == 1);
  }
  for(i=0;i<numProfiles;i++){
    n = fread(&lTwos[i],sod,1,fp);
    assert(n == 1);
  }
  return result;
}
```

6.5 Prepare for LD Analysis

```
/***** ld.c ***************************************
 * Description:
 * Author: Bernhard Haubold, haubold@evolbio.mpg.de
 * Date: Thu Oct 25 08:50:39 2012
 **************************************************/
#include <stdio.h>
#include <assert.h>
#include <string.h>
#include <stdlib.h>
#include "eprintf.h"
#include "interface.h"
#include "ld.h"

ContigDescr *thisContigDescr = NULL;

void setContigDescr(ContigDescr *contigDescr){
  thisContigDescr = contigDescr;
}


FILE *iniLdAna(Args *args){
  char *fileName;
  char *tag;
  FILE *fp;
  ContigDescr *cp;
  int i, max, n, numRead;
  /* get contig lengths from file */
  tag = (char *)emalloc(4*sizeof(char));
  fileName = (char *)emalloc(256*sizeof(char));
  fileName = strcpy(fileName,args->n);
  fileName = strcat(fileName,".con");
  fp = efopen(fileName,"rb");
  numRead = fread(tag,sizeof(char),3,fp);
  assert(numRead == 3);
  tag[3] = '\0';
  if(strcmp(tag,"con") != 0)
    assert(0);
  cp = (ContigDescr *)emalloc(sizeof(ContigDescr));
  numRead = fread(&n,sizeof(int),1,fp);
  assert(numRead == 1);
  cp->pos = NULL;
  cp->n = n;
  cp->len = (int *)emalloc(n*sizeof(int));
  for(i=0;i<cp->n;i++){
    numRead = fread(&cp->len[i],sizeof(int),1,fp);
    assert(numRead == 1);
  }
  fclose(fp);
  max = 0;
  for(i=0;i<n;i++){
    if(cp->len[i] > max)
      max = cp->len[i];
  }
  cp->posBuf = (Position *)emalloc(max*sizeof(Position));
  setContigDescr(cp);

  /* get file pointer for position file */
  fileName = strcpy(fileName,args->n);
  fileName = strcat(fileName,".pos");
  fp = efopen(fileName,"rb");
  numRead = fread(tag,sizeof(char),3,fp);
  if(strcmp(tag,"pos") != 0)
    assert(0);
  free(fileName);
  free(tag);
  return fp;
}

ContigDescr *getContigDescr(){
  return thisContigDescr;
}

/***** filterPro.c ********************************
 * Description:
 * Author: Bernhard Haubold, haubold@evolbio.mpg.de
 * Date: Sun Jul  7 17:18:48 2013
 **************************************************/
#include <stdio.h>
#include <stdlib.h>
#include "profile.h"
```

```
#include "mlComp.h"
#include "eprintf.h"

int compDouble(const void *a, const void *b);

char *filterPro(double pi, double inclFract){
  Profile *profiles;
  int i, numProfiles;
  double *lOnes, *lTwos;
  double *lik, *originalLik, piComp, sum;
  double minLik, minSum;
  char *incl;

  numProfiles = getNumProfiles();

  incl = (char *)emalloc(numProfiles*sizeof(char));
  if(inclFract == 1.0){
    for(i=0;i<numProfiles;i++)
      incl[i] = 1;
    return incl;
  }
  profiles = getProfiles();
  lOnes = getLones();
  lTwos = getLtwos();

  lik = (double *)emalloc(numProfiles*sizeof(double));
  originalLik = (double *)emalloc(numProfiles*sizeof(double));
  piComp = 1. - pi;
  sum = 0.0;
  for(i=0;i<numProfiles;i++){
    lik[i] = -log(lOnes[i]*piComp + lTwos[i]*pi) * profiles[i].n;
    originalLik[i] = lik[i];
    sum += lik[i];
  }
  qsort(lik,numProfiles,sizeof(double),compDouble);
  minSum = sum * inclFract;
  sum = 0;
  i = 0;
  while(sum < minSum)
    sum += lik[i++];
  minLik = lik[i-1];
  for(i=0;i<numProfiles;i++)
    if(originalLik[i] >= minLik)
      incl[i] = 1;
    else
      incl[i] = 0;
  return incl;
}

int compDouble(const void *a, const void *b){
  double x;

  x = *((double *) b) - *((double *) a);
  if(x > 0)
    return 1;
  else if(x < 0)
    return -1;
  else
    return 0;
}
```

6.6 Maximum Likelihood Computation Part III: `deltaComp.c`

```
/***** deltaComp.c ******************************
 * Description: Compute diesequilibrium coefficient,
 *   delta, for single diploid individual.
 * Reference: Lynch, M. (2008). Estimation of nuc-
 *   leotide diversity, disequilibrium coefficients,
 *   and mutation rates from high-coverage genome-
 *   sequencing projects. Mol. Biol. Evol. 25:
 *   2409-2419.
 * Author: Bernhard Haubold, haubold@evolbio.mpg.de
 * Date: Tue Mar 17 21:21:02 2009
 ************************************************/
#include <float.h>
#include <gsl/gsl_errno.h>
#include <gsl/gsl_math.h>
#include <gsl/gsl_roots.h>
#include <assert.h>
#include "interface.h"
#include "ld.h"
#include "mlComp.h"
#include "eprintf.h"
#include "profile.h"

double globalPi, globalEpsilon;
int globalDist;
Node **globalProfilePairs;
int globalNumProfiles;
double *lOnes, *lTwos;
double likelihood;

double rhoFromDelta(double t, double d);
void lik(double de);
double myF(const gsl_vector *v, void *params);
double confFun(double x, void *params);
void conf(Args *args, Result *result);
double iterate(Args *args, gsl_root_fsolver *s, double xLo, double xHi);
void traverse(int a, Node *np, double h0, double h2, double complementHalf)
   ;

/* estimateDelta: estimate pi, delta, and epsilon using
 * the Nelder-Mead Simplex algorithm; code adapted
 * from Galassi, M., Davies, J., Theiler, J., Gough, B.,
 * Jungman, G., Booth, M., Rossi, F. (2005). GNU
 * Scientific Library Reference Manual. Edition 1.6,
 * for GSL Version 1.6, 17 March 2005, p 472f.
 */
```

```
Result *estimateDelta(Node **profilePairs, int numProfiles, Args *args,
  Result *result, int dist){
 const gsl_multimin_fminimizer_type *T;
 gsl_multimin_fminimizer *s;
 gsl_vector *ss, *x;
 gsl_multimin_function minex_func;
 size_t iter;
 int status, numPara;
 double size;

 globalProfilePairs = profilePairs;
 globalNumProfiles = numProfiles;
 T =  gsl_multimin_fminimizer_nmsimplex;
 s = NULL;
 iter = 0;
 numPara = 1; /* one parameter estimation */
 globalPi = result->pi;
 globalEpsilon = result->ee;
 globalDist = dist;
 /* initialize vertex size vector */
 ss = gsl_vector_alloc(numPara);
 /* set all step sizes */
 gsl_vector_set_all(ss, args->s);
 /* starting point */
 x = gsl_vector_alloc(numPara);
 gsl_vector_set(x, 0, args->D);
 /* initialize method and iterate */
 minex_func.f = &myF;
 minex_func.n = numPara;
 minex_func.params = (void *)NULL;
 s = gsl_multimin_fminimizer_alloc(T, numPara);
 gsl_multimin_fminimizer_set(s, &minex_func, x, ss);
 do{
  iter++;
  status = gsl_multimin_fminimizer_iterate(s);
  if(status)
   break;
  size = gsl_multimin_fminimizer_size(s);
  status = gsl_multimin_test_size(size, args->t);
 }while(status == GSL_CONTINUE && iter < args->i);
 if(status != GSL_SUCCESS)
  printf("WARNING:␣Estimation␣of␣\\Delta␣failed:␣%d\n",status);
 result->de = gsl_vector_get(s->x, 0);
 result->l = s->fval;
 result->i = iter;
 result->rh = rhoFromDelta(result->pi,result->de)/dist;
 conf(args, result);
 result->rLo = rhoFromDelta(result->pi,result->dUp)/dist;
 result->rUp = rhoFromDelta(result->pi,result->dLo)/dist;
 gsl_vector_free(x);
 gsl_vector_free(ss);
 gsl_multimin_fminimizer_free(s);
 /* freeMlComp(); */
 return result;
}


double myF(const gsl_vector* v, void *params){
  double de;

  /* get delta */
  de = gsl_vector_get(v, 0);
  if(de < -1 || de > 1)
    return DBL_MAX;
  else
    lik(de);

  return -likelihood;
}

void lik(double de){
  int i;
  double pi, h0, h2;
  double complementHalf;

  pi = globalPi;
  h0 = 1./(1.+pi)/(1.+pi) + de*pi/(1.+pi)/(1.+pi);
  h2 = pi*pi/(1.+pi)/(1.+pi) + de*pi/(1.+pi)/(1.+pi);
  complementHalf = (1.-h0-h2)/2.;
  likelihood = 0.;
  lOnes = getLones();
  lTwos = getLtwos();
  for(i=0;i<globalNumProfiles;i++)
    traverse(i,globalProfilePairs[i],h0,h2,complementHalf);

}


void traverse(int a, Node *np, double h0, double h2, double complementHalf)
    {
  double li;
  int b;

  if(np != NULL){
    traverse(a,np->left,h0,h2,complementHalf);

    b = np->key;
    li = h0*lOnes[a]*lOnes[b]
       + h2*lTwos[a]*lTwos[b]
       + complementHalf*(lOnes[a]*lTwos[b]+lTwos[a]*lOnes[b]);
    if(li>0)
      likelihood += log(li) * np->n;
    else
      likelihood += log(DBL_MIN) * np->n;

    traverse(a,np->right,h0,h2,complementHalf);
  }
}
```

```

/* conFun: called for confidence interval estimation of pi */
double confFun(double x, void *params){
  Result *res;
  double l;

  res = (Result *)params;
  lik(x);
  l = likelihood + res->l + 2.;

  return l;
}

void conf(Args *args, Result *result){
  const gsl_root_fsolver_type *solverType;
  gsl_root_fsolver *s;
  gsl_function fun;
  double xLo, xHi;
  int status;

  /* preliminaries */
  gsl_set_error_handler_off();
  fun.function = &confFun;
  fun.params = result;
  solverType = gsl_root_fsolver_brent;
  s = gsl_root_fsolver_alloc(solverType);
  /* search for lower bound of delta */
  result->type = 2;
  if(result->de < 0)
    xLo = -1;
  else
    xLo = 0;
  xHi = result->de;
  status = gsl_root_fsolver_set(s,&fun,xLo,xHi);
  if(status){
    printf("WARNING:␣Lower␣confidence␣limit␣of␣\\Delta␣cannot␣be␣estimated;
        ␣setting␣it␣to␣-1.\n");
    result->dLo = -1.0;
  }else
    result->dLo = iterate(args, s, xLo, xHi);
  /* search for upper bound of delta */
  xLo = result->de;
  xHi = 1.0;
  status = gsl_root_fsolver_set(s,&fun,xLo,xHi);
  if(status){
    printf("WARNING:␣Upper␣confidence␣limit␣of␣\\Delta␣cannot␣be␣estimated;
        ␣setting␣it␣to␣1.\n");
    result->dUp = 1;
  }else
    result->dUp = iterate(args, s, xLo, xHi);
  gsl_root_fsolver_free(s);
}

double iterate(Args *args, gsl_root_fsolver *s, double xLo, double xHi){
  int iter;
  double r;
  int status;

  iter = 0;
  do{
    iter++;
    status = gsl_root_fsolver_iterate(s);
    r = gsl_root_fsolver_root(s);
    xLo = gsl_root_fsolver_x_lower(s);
    xHi = gsl_root_fsolver_x_upper(s);
    status = gsl_root_test_interval(xLo, xHi, 0, args->t);
  }while(status == GSL_CONTINUE && iter < args->i);

  return r;
}

void setPi(double pi){
  globalPi = pi;
}

double rhoFromDelta(double t, double d){
  double r;

  r = (t + pow(t,2.) -
       d*(13. + 19.*t +
          6.*pow(t,2.)) +
       sqrt(1. + t)*
       sqrt(pow(t,2.)*(1. + t) +
            2.*d*t*
            (23. + 17*t +
             2.*pow(t,2.)) +
            pow(d,2.)*
            (97. + 109.*t +
             32.*pow(t,2.) +
             4*pow(t,3.)))))/
    (2.*d*(1. + t));
  return r;
}
```

7 Change Log

- v. 0.2 (February 25, 2009)
  - First running version distributed for review.
- v. 0.3 (March 11, 2009)
  - Fixed nucleotide frequency computation.
- v. 0.5 (March 18, 2009)
  - Implemented linkage disequilibrium computation.
  - Implemented confidence interval computation.
- v. 0.6 (March 23, 2009)

  - Implemented linkage disequilibrium computation across all possible distances.
  - Implemented 1-parameter estimation of disequilibrium coefficients. User can switch between 3-parameter estimation and 1-parameter estimation.
  - Implemented test mode to allow comparison with testMlDelta.
  - Realized that equations (5) and (11) are not equivalent for $\Delta = 0$; hence the separate computation of π and Δ in `piComp.c` and `deltaComp.c`.

- v. 0.7 (March 28, 2009)
  - Adapted `profileTree.readProfiles` to five-column input format generated by `ace2pro`. Positions with no coverage in a contig are included as zero-coverage entries in the profile collection to facilitate subsequent disequilibrium computation (c.f. `profileTree.readProfiles` and `profileTree.getProfileTree`).

- v. 0.8 (March 30, 2009)
  - Fixed error allocation bugs in `profileTree.readProfiles`.
  - Ignore likelihood values of zero in maximum likelihood estimation.
  - Included switch for minimum coverage.

- v. 0.9 (April 11, 2009)
  - Fixed overflow in binomial probability computation by using `gsl_sf_lnchoose(n,k)` instead of `gsl_sf_choose(n,k).`

- v. 0.10 (April 16, 2009)
  - Added code to check format of input lines.

- v. 0.11 (April 18, 2009)
  - Improved output formatting.

- v. 0.12 (April 19, 2009)
  - Added computation of the fraction of homozygous pairs, H_0, and of the fraction of heterozygous pairs H_2.
  - Added `-d` switch to print H_0 and H_2 as debugging information.
  - Added `-m` switch for setting minimum distance in Δ computation.
  - Added `-S` switch for setting step size in Δ computation.

- v. 0.14 (June 1, 2009)
  - Fixed $0 \leq \Delta \leq 1 \rightarrow -1 \leq \Delta \leq 1$
  - Set initial value of confidence computation for Δ as follows

    ```
    xLo = result->de - result->de*0.5;
    ```

    The magic factor of 0.5 might need revision in the future. However, setting `xLo` to the obvious value of -1 does not work.

- v. 0.15 (June 26, 2009)
  - Included Lynch's correction term [6, p. 2412] in the computation of π
  - Implemented new ρ computation

- v. 0.16 (June 27, 2009)
  - Implemented Peter Pfaffelhuber's new ρ computation (from Δ)

- v. 0.17 (June 28, 2009)
  - Implemented Peter Pfaffelhuber's new ρ computation (full maximum likelihood version)
- v. 0.18 (June 30, 2009)
  - Allow negative $\hat{\rho}$ in likelihood maximization
  - Set starting value of $\hat{\rho} = 1$ rather than 10^{-3}
  - Improved formatting of output
- v. 1.0 (July 3, 2009)
  - Changed name from `mlDiv` to `mlRho`
  - Prepared software for release
- v. 1.1 (Nov. 12, 2009)
  - Changed `char c;` in `int c;` in `getArgs` (this seemed to confuse some compilers).
  - Eliminated padding on right hand side of a given array of profiles in `readProfiles`.
- v. 1.2 (Nov. 21–24, 2009)
  - Major rewrite to minimize the memory requirement: In the previous versions all primary data was loaded into memory and the analyzed. This was simple to program and resulted in fast disequilibrium analyses. In this version the primary data stays on disk and only the distinct profiles and their counts are kept in memory. The resulting program is appreciably slower than before but much more memory efficient.
  - Use log-based probability computation in `lOne` and `lTwo` of `mlComp.c` to enable analysis of high coverage data.
- v. 1.3 (Nov. 25, 2009)
  - Fixed a few minor memory leaks.
  - Removed storage of binomial coefficients as this created a memory overhead $O(\max(\text{coverage}) \times \max(\text{coverage}))$, which is severe for high-coverage data.
  - Changed the type of variable `numPos` in `profileTree.c` from `long` to `double`.
- v. 1.4 (Dec. 7, 2009)
  - Removed `-F` switch—currently input can only be read from file, not from `stdin`.
  - Fixed `make test`.
  - Removed "generating extra queue item" message in `profileTree.getPairwProfiles`.
- v. 1.5 (Dec. 14, 2009)
  - Fixed interface.
- v. 1.6 (Sept. 13, 2010) Realized that the likelihood function can become numerically unstable as Δ goes to ward zero and from there toward -1. This should be explored further, but here is the current status:
  - Previously, if `likelihood` ≤ 0 in `deltaComp.lik`, the likelihood was simply discarded, making it possible that *less* correct values of Δ were assigned a greater likelihood than more correct values. Now `likelihood` $=$ `DBL_MIN` in such cases. This improves matters, though the estimation of Δ remains problematic for small values.
  - To give the user feedback on the quality of the Δ estimate, errors either in the ML estimation or in the confidence estimation are now printed as warnings. They were previously simply ignored.
  - Also included diagnostic messages in the ρ estimation procedure.

  - Improved description of the `-f` option in the interface.

- v. 1.7 (Sept. 17, 2010)
  - Removed superfluous printing of message that program can only read from file.
  - Ensured that no results are printed for empty input files.

- v. 1.8 (May 25, 2012)
  - Reverted to keeping all input data in memory to speed up program.
  - Removed memory leaks diagnosed with valgrind.
  - If ρ estimation fails the program now reports "n/a" for "not available" as the value of ρ.
  - If the upper confidence limit of ρ cannot be estimated, it is set to 1, and this value is now actually printed (before the output was $1/\mathrm{dist}$).
  - Set the default maximum number of iterations in ML-estimation to 100. The previous value of 1000 resulted in time wasted on hopeless cases.
  - In one of the previous versions I lost the full likelihood computation (I think inadvertently). I reinstalled it in this version.

- v. 1.9 (May 31, 2012)
  - Sped program up by using precomputed binomial coefficients for $n \leq 100, k \leq 100$.
  - Reverted to more memory-efficient mode (the increase in memory consumption was greater than 70-fold for an example data set for little speed gained).

- v. 1.10 (June 1, 2012)
  - Implemented option for lumping the distance classes specified by the "step" option. For example,

    ```
    mlRho -m 100 -M 110 -S 2 -L test.pro
    ```

    means that in the disequilibrium computation distances 100 and 101 are lumped, followed by 102 and 103, and so on.

- v. 1.11 (June 26, 2012)
  - Revised `lOne` and `lTwo` in `mlComp` to implement the revision of the original likelihood functions by Stephen Bates.

- v. 1.12 (September 11, 2012)
  - Incorporated samtools for BAM file input—NOT WORKING YET!

- v. 1.13 (September 12, 2012)
  - Implemented input in summary format (`-u`).
  - Added `pro2sum.awk` to convert profiles to summary format.

- v. 1.14 (October 3, 2012)
  - In response to *Initial Performance Analysis of mlRho using Vampir* by Thota, Henschel, and Michael (October 2, 2012): Removed the string-¿int conversion in the likelihood computations. Instead, I now store the profiles as strings and int-arrays. I also store the corresponding coverages to avoid their repeated computation.

- v. 1.15 (October 9, 2012)
  - Replaced calls to `pow` library function by `powInt`, which reduced the run time on `test.pro` from 11.3s to 7.2s (36%).

  - Removed samtools; users can filter BAM files via `samtools view` and, if they like, the summary format (`-u`).
- v. 1.16 (October 10, 2012)
  - Speed up `lOne` and `lTwo` through pre-computation of lngamma-values.
- v. 1.17 (October 11, 2012)
  - Reorganized reading of profiles from disk to speed up disequilibrium estimation.
- v. 1.18 (October 12, 2012)
  - Changed the logic of data storage: Keep tree of individual profiles between repeated disequilibrium computations. Write to file position and node-pointer per profile. Reconstruct from this the pairwise profiles.
- v. 1.19 (October 12, 2012)
  - Improved memory management in `profileTree.c`.
  - Fixed error in extraction of pairs of given distance.
- v. 1.20 (October 18, 2012)
  - Integrated `mlRho` with `formatPro` v. 0.3.
- v. 1.21 (October 21, 2012)
  - Integrated `mlRho` with the all-binary `formatPro` v. 0.4.
  - Removed `testMlPi` as its function is duplicated by `testMlDelta`.
  - Revised documentation.
- v. 1.22 (October 25, 2012)
  - During Δ computation profile probabilities are looked up rather than recomputed. Profile pairs not stored any more;
- v. 1.23 (October 29, 2012)
  - Major revision of disequilibrium analysis:
    * Reverted to storing of profile pairs; however, the memory requirement per node in profile tree is now $4 \times 4 = 16$ bytes, while previously it was approximately 100 bytes.
    * Halved the number of nodes in the profile tree by ignoring order of profiles.
    * Use profile indexes rather than character arrays as search keys in profile tree.
    * Used the pair of profile indexes, i, j as follows: (i) i is the index into an array of profile trees, and j is the search key in the i-th profile tree.
    * ρ is now computed solely as a function of Δ.
    * The full likelihood estimation of Δ and ρ is abolished. Instead, the profile likelihoods established during the estimation of π are carried over to the disequilibrium analysis.
  - Abolished option to set minimum coverage: set this via `formatPro`.
- v. 1.24 (November 9, 2012)
  - Writing and reading of likelihood file (`.lik`) to cut out repeated computation of single-profile likelihoods for disequilibrium computations.
  - Number of pairs of profiles encoded as double.
  - Flushing after every step in the delta computation.

- v. 1.25 (November 14, 2012)
  - The following chain of commands would lead to a discrepancy between new `*.sum`, `*.pos`, and `*.con` files, and the old `*.lik` file:

    ```
    ms 2 1 -t 10 -r 0 250001 -c 100 20 |
    ms2dna                             |
    sequencer -c 10 -p                 |
    formatPro;
    mlRho -M 0 -I;
    mlRho -m 1000 -M 1005
    ```

    Fixed this now insisting that new profiles are computed if `-I` is used; hence the first line of `estimatePi` in `piComp.c` now reads

    ```
      if((fp = openLikFile(args->n)) != NULL && !args->I){
    ```

    instead of previously

    ```
      if((fp = openLikFile(args->n)) != NULL){
    ```
  - Changed format of `*.lik` file:
    - tag: "`lik`"
    - result: `Result`
    - number of profiles: `double`
    - l_1: `double *`
    - l_2: `double *`
- v. 2.0 (December 12, 2012)
  - Revised documentation.
  - Released this major revision together with `formatPro` and `inspectPro`.
- v. 2.1 (January 25, 2013)
  - Converted all calls where `fread` reads multiple objects to do-loops with single-object reads. This has sped up the program and the hope is that it also improves concurrent runs on large data sets.
- v. 2.2 (February 26, 2013)
  - Noticed that the confidence interval for ρ computation had been switched in `deltaComp.c`. Fixed.
- v. 2.3 (July 6, 2013)
  - Implemented the `-o` switch for keeping the position information in memory between linkage analyses with different distances. The hope is that by thus reducing I/O concurrent runs are sped up.
  - Now prevent variable `r` to go out of bounds in function `countPairs` in `profileTree.c`. This slightly changes the number of positions analyzed by the disequilibrium analysis. The end result is usually not affected.
  - Pulled

    ```
    lOnes = getLones();
    lTwos = getLtwos();
    ```

    out of the traversal loop.
- v. 2.4 (July 7, 2013)
  - Included the `-f` option for filtering the profiles according to the contribution they are making to the total likelihood in the single-site estimation. This is implemented in `filterPro.c`

- v. 2.5 (Oct. 9, 2013)
  - Implemented correction of diversity computation suggested on p. 2412 of [6]. This can be switched off using `-u` for "uncorrected diversity".
- v. 2.6 (Oct 10, 2013)
  - Cleaned up user interface.
- v. 2.7 (Oct. 17, 2013)
  - Fixed inconsistency between corrected diversity measures with and without likelihoods read from disk.
  - Default behavior is that the correction is off; it can be switched on with `-c`.

References

[1] P. Dehal, Y. Satou, R.K. Campbell, J. Chapman, B. Degnan, A. De Tomaso, B. Davidson, A. Di Gregorio, M. Gelpke, D.M. Goodstein, N. Harafuji, K.E. Hastings, I. Ho, K. Hotta, W. Huang, T. Kawashima, P. Lemaire, D. Martinez, I.A. Meinertzhagen, S. Necula, M. Nonaka, N. Putnam, S. Rash, H. Saiga, M. Satake, A. Terry, L. Yamada, H.G. Wang, S. Awazu, K. Azumi, J. Boore, M. Branno, S. Chin-Bow, R. DeSantis, S. Doyle, P. Francino, D.N. Keys, S. Haga, H. Hayashi, K. Hino, K.S. Imai, K. Inaba, S. Kano, K. Kobayashi, M. Kobayashi, B.I. Lee, K.W. Makabe, C. Manohar, G. Matassi, M. Medina, Y. Mochizuki, S. Mount, T. Morishita, S. Miura, A. Nakayama, S. Nishizaka, H. Nomoto, F. Ohta, K. Oishi, I. Rigoutsos, M. Sano, A. Sasaki, Y. Sasakura, E. Shoguchi, T. Shin-i, A. Spagnuolo, D. Stainier, M.M. Suzuki, O. Tassy, N. Takatori, M. Tokuoka, K. Yagi, F. Yoshizaki, S. Wada, C. Zhang, P.D. Hyatt, F. Larimer, C. Detter, N. Doggett, T. Glavina, T. Hawkins, P. Richardson, S. Lucas, Y. Kohara, M. Levine, N. Satoh, and D.S. Rokhsar. The draft genome of *Ciona intestinalis*: insights into chordate and vertebrate origins. *Science*, 298(5601):2157–2167, Dec 2002.

[2] M. Galassi, J. Davies, J. Theiler, B. Gough, G. Jungman, M. Booth, and F. Rossi. *GNU Scientific Library Reference Manual*. Network Theory Ltd, 1.6, for gsl version 1.6, 17 march 2005 edition, 2005.

[3] B. Haubold, P. Pfaffelhuber, and M. Lynch. `mlRho`: A program for estimating the population mutation and recombination rates from shotgun-sequenced diploid genomes. *Molecular Ecology*, 19:277–284, 2010.

[4] R. R. Hudson. Generating samples under a Wright-Fisher neutral model of genetic variation. *Bioinformatics*, 18:337–338, 2002.

[5] H. Li, B. Handsaker, A. Wysoker, T. Fennell, J. Ruan, N. Homer, G. Marth, G. Abecasis, R. Durbin, and 1000 Genome Project Data Processing Subgroup. The sequence alignment/map format and SAMtools. *Bioinformatics*, 25:2078–2079, 2009.

[6] M. Lynch. Estimation of nucleotide diversity, disequilibrium coefficients, and mutation rates from high-coverage genomic-sequencing projects. *Molecular Biology and Evolution*, 25:2409–2419, 2008.

[7] The 1000 Genomes Project Consortium. A map of human genome variation from population-scale sequencing. *Nature*, 467:1061–1073, 2010.