`gd`, v. 0.12: Estimating Genetic Diversity and Other Population Genetic Parameters from Aligned DNA Sequence Data

Bernhard Haubold

October 30, 2013

1 Introduction

`gd` is a computer program that implements routine population genetic analyses. Given a set of aligned sequences, it can compute globally or for sliding windows

1. the number of segregating sites, S;
2. the number of average pairwise differences, π;
3. a statistic for testing the neutral model of evolution based on S and π known as Tajima's D, D_{T} [3].

There are a number of good programs already available to carry out these and many related tasks [2]. `gd` is intended to combine simplicity with efficiency to take some of the tedium out of genome-scale population genetic analysis.

2 Getting Started

`gd` was written in C on a computer running Mac OS X; it is intended to run on any UNIX system with a C compiler. However, please contact me at `haubold@evolbio.mpg.de` if you have problems with the program.

- Unpack the program

  ```
  tar -xvzf gd_XXX.tgz
  ```

  where `XXX` indicates the version.
- Change into the newly created directory

  ```
  cd Gd_XXX
  ```

  and list its contents

  ```
  ls
  ```
- Generate `gd`

  ```
  make
  ```
- List its options

  ```
  ./gd -h
  ```
- Test the program

  ```
  make test
  ```

3 Tutorial

This tutorial is intended to demonstrate the usage of `gd` by applying it to simulated data sets. In order to carry out the simulations, you will need to have installed on your computer Richard Hudson's program `ms` [1] and the auxiliary program `ms2dna` by Peter Pfaffelhuber and myself, which is available from

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

1. Generate one sample of 10 haplotypes with mutation rate $\theta = 4N_{\mathrm{e}}\mu = 100$ and no recombination among 10,000 potentially recombining sites:

   ```
   ms 10 1 -t 100 -r 0 10000 > sample.ms
   ```

   The `-r` option is included here to allow later conversion to DNA sequences 10,000 bp long. `ms` by itself would give the same result without it.

2. Compute standard statistics for this sample using the program `sample_stats`, which is part of the `ms` package:

   ```
   sample_stats < sample.ms
   ```

3. Convert the haplotypes in `sample.ms` to DNA sequences in FASTA format:

   ```
   ms2dna sample.ms > sample.fasta
   ```

4. Compute π for the sequences in `sample.fasta` using `gd` and compare the result to that obtained with `sample_stats`:

   ```
   ./gd sample.fasta
   ```

5. Repeat this computation for the number of segregating sites, S

   ```
   ./gd -s s sample.fasta
   ```

   and for D_{T}:

   ```
   ./gd -s t sample.fasta
   ```

6. For each statistic the program can be switched into sliding window mode by using the `-w` option, for example:

   ```
   gd -w 1000 sample.fasta
   ```

   Notice that the distance between windows is 100, that is, one tenth of the window length. This default step length can be changed using the `-S` option. So for printing every window, type

   ```
   gd -w 1000 -S 1 sample.fasta
   ```

7. The rather verbose output from the last command is usually summarized in a graph and a simple way to draw one is to use the program `graph`, which is part of the GNU `plotutils` package:

   ```
   gd -w 1000 sample.fasta | graph -T X
   ```

4 Change Log

1. v. 0.4 (April 20, 2010)
    - First version distributed.
2. v. 0.5
    - Fixed polymorphism positions printed using `-P`.
    - Fixed output of sliding window analysis.
3. v. 0.6 (May 3, 2010)
    - Removed printing of error message `no_complete_column_in_window`; now windows without a single complete column are simply not reported.
    - Fixed serious bug in sliding window analysis.
4. v. 0.7 (May 11, 2010)
    - Introduced `-W` option for setting the minimum number of nucleotides in sliding window.
5. v. 0.8 (October 25, 2010)
    - Fixed comparison of `-W` option from $>$ to $\geq$.
    - Fixed positioning of window.
6. v. 0.9 (November 20, 2012)
    - Fixed handling of data with zero polymorphisms.
7. v. 0.10 (December 20, 2012)
    - Removed colon in output with `-s s` (segregating sites).
8. v. 0.11 (April 16, 2013)
    - Fixed `-p` option.
9. v. 0.12 (October 30, 2013)
    - Fixed treatment of sliding windows in case `-w` is greater than the length of the alignment.
    - Improved program interface.

5 Listings

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

5.1 The Driver Program: `gd.c`

```
/***** gd.c ***************************************
 * Description: Program to quantify genetic
 *   diversity.
 * Author: Bernhard Haubold, haubold@evolbio.mpg.de
 * Date: Wed Feb 17 13:24:02 2010
 **************************************************/
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
```

```
#include <unistd.h>
#include "eprintf.h"
#include "interface.h"
#include "genDiv.h"

void runAnalysis(Args *args, int fd);

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

  version = "0.12";
  setprogname2("gd");
  args = getArgs(argc, argv);
  if(args->v)
    printSplash(version);
  if(args->h || args->e)
    printUsage(version);
  if(args->numInputFiles){
    for(i=0;i<args->numInputFiles;i++){
      fd = open(args->inputFiles[i],O_RDONLY,0);
      runAnalysis(args,fd);
      close(fd);
    }
  }else{
    fd = 0;
    runAnalysis(args,fd);
  }
  free(args);
  free(progname());
  return 0;
}

void runAnalysis(Args *args, int fd){
  Sequence *seq;
  Alignment *al;
  long i;
  double s, sum, p, winPos;
  double *arr;

  seq = readFasta(fd);
  al = seq2al(seq);
  if(args->P){
    printPoly(al);
    return;
  }
  if(args->w){
    args->w = args->w < al->n ? args->w : al->n;
    if(args->W == -1)
      args->W = args->w / 2 + 1;
    if(args->s == ’s’)
      arr = winSs(args,al);
    else if(args->s == 't')
      arr = winTajima(args,al);
    else
      arr = winPi(args,al);
    winPos = (args->w-1.)/2. + 1;
    for(i=0;i<al->numWin;i++){
      if(args->s != 't'){
        if(al->winNumNuc[i] >= args->W)
          printf("%g\t%.6f\n",winPos,arr[i]/al->winNumNuc[i]);
      }else{
        printf("%g\t%.6f\n",winPos,arr[i]);
      }
      winPos += args->S;
    }
  }else{
    if(args->s == 's'){
      sum = 0;
      for(i=1;i<al->m;i++)
        sum += 1./i;
      s = ss(al);
      printf("S:\t%g\tnumsites:\t%ld\tS/site:\t%.6f\ttheta_W/site:\t%.6f\n"
         , s,al->numNuc,s/al->numNuc,s/al->numNuc/sum);
    }else if(args->s == 't'){
      printf("D_T:\t%.6f\n",tajima(al));
    }else{
      p = pi(al);
      printf("pi:\t%.6f\tnumsites:\t%ld\tpi/site:\t%.6f\ttheta_pi/site:\t
         %.6f\n",p,al->numNuc,p/al->numNuc,p/al->numNuc);
    }
  }
}
```

5.2 Diversity Estimation

5.3 `genDiv.h`

```
/***** genDiv.h ***********************************
 * Description:
 * Author: Bernhard Haubold, haubold@evolbio.mpg.de
 * Date: Tue Feb 16 21:58:07 2010
 *************************************************/
#include "sequenceData.h"

typedef struct alignment{
  char **al;         /* nucleotide alignment */
  long m;            /* number of sequences in alignment */
  long n;            /* number of nucleotides per sequence in alignment */
  char **headers;    /* headers of sequences */
  long *poly;        /* polymorphic positions in alignment */
  long *nuc;         /* positions consisting solely of canonical nucleotides
     */
  char *polInd;      /* is position polymorphic? */
  char *nucInd;      /* does position consist of nucleotides only? */
  int *winNumNuc;    /* number of canonical nucleotides per window */
  int numWin;        /* the number of windows */
  float *polyFreq; /* frequency of minor allele */
  long numPoly;    /* number of polymorphic positions in alignment */
  long numNuc;     /* number of positions consisting solely of canonical
     nucleotides */
}Alignment;

Alignment *seq2al(Sequence *seq);
double pi(Alignment *al);
double ss(Alignment *al);
double *winPi(Args *args, Alignment *al);
double *winSs(Args *args, Alignment *al);
double *winTajima(Args *args, Alignment *al);
void printPoly(Alignment *al);
double tajima(Alignment *al);
```

5.4 genDiv.c

```
/***** genDiv.c *********************************
 * Description: Routines for measuring genetic
 *   diversity.
 * Author: Bernhard Haubold, haubold@evolbio.mpg.de
 * Date: Tue Feb 16 21:33:24 2010
 **************************************************/
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
#include <math.h>
#include <limits.h>
#include "interface.h"
#include "genDiv.h"
#include "queue.h"
#include "eprintf.h"

void findPoly(Alignment *al);
void prepareWin(Alignment *al, int winLen, int stepLen);

double td(double a1, double a2, double s, double p, int n);

void printPoly(Alignment *al){
  int i, j, c;

  if(al->poly == NULL)
    findPoly(al);
  /* Print positions of polymorphisms */
  printf(">Positions_frequencies:%ld\n",al->numPoly);
  for(i=0;i<al->numPoly;i++)
    printf("%ld\t%.3f\n",al->poly[i]+1,al->polyFreq[i]);
  /* Print alleles at polymorphic positions */
  for(i=0;i<al->m;i++){
    printf("%s\n",al->headers[i]);
    c=0;
    for(j=0;j<al->numPoly;j++){
      printf("%c",al->al[i][al->poly[j]]);
      c++;
      if(c==60){
        printf("\n");
        c = 0;
      }
    }
    if(c)
      printf("\n");
  }
}

/* pi: number of average pairwise differences per site */
double pi(Alignment *al){
  int i, j, k;
  double  s1, s2;

  if(al->poly == NULL){
    findPoly(al);
  }
  s2 = 0;
  for(i=0;i<al->m-1;i++){
    for(j=i+1;j<al->m;j++){
      s1 = 0;
      for(k=0;k<al->numPoly;k++){
        if(al->al[i][al->poly[k]] != al->al[j][al->poly[k]])
          s1++;
      }
      s2 += s1;
    }
  }

  s2 /= al->m * (al->m - 1) / 2;

  return s2;
}
/* tajima: Tajima's D */
double tajima(Alignment *al){
  double s, p, a1, a2, n;
  int i;

  s = ss(al);
  p = pi(al);
  n = al->m;

  a1 = 0;
  a2 = 0;
  for(i=1;i<n;i++){
    a1 += 1.0/i;
    a2 += (1.0/i/i);
  }

  return td(a1, a2, s, p, n);
}

/* ss: number of segregating sites per site */
double ss(Alignment *al){
  if(al->poly == NULL)
    findPoly(al);

  return (double)al->numPoly;
}

double *winPi(Args *args, Alignment *al){
  int i, j, k, numWin;
  int lb, rb;
  double *arr, *pol, factor;
  double sum;

  arr = (double *)emalloc(al->n*sizeof(double));
  pol = (double *)emalloc(al->n*sizeof(double));
  if(al->poly == NULL)
    findPoly(al);
  if(al->polInd == NULL)
    prepareWin(al,args->w,args->S);
  /* compute average number of mismatches for all positions */
  factor = al->m*(al->m-1)/2;
  for(i=0;i<al->n;i++){
    pol[i] = 0;
    if(al->polInd[i]){
      for(j=0;j<al->m-1;j++)
        for(k=j+1;k<al->m;k++)
          if(al->al[j][i] != al->al[k][i])
            pol[i]++;
      pol[i] /= factor;
    }
  }
  numWin = 0;
  /* take care of first window */
  rb = 0;
  sum = 0;
  while(rb < args->w && rb < al->n)
    sum += pol[rb++];
  arr[numWin++] = sum;
  /* scan remaining windows */
  lb = 0;
  while(rb < al->n-args->S+1){
    for(i=0;i<args->S;i++){
      sum += pol[rb+i];
      sum -= pol[lb+i];
    }
    rb += args->S;
    lb += args->S;
    arr[numWin++] = sum;
  }
  free(pol);
  return arr;
}

/* winSs: sliding window analysis of segregating sites */
```

```c
double *winSs(Args *args, Alignment *al){
  int i, numWin, numPol, rb, lb;
  double *arr;

  arr = (double *)emalloc(al->n*sizeof(double));
  if(al->poly == NULL)
    findPoly(al);
  if(al->polInd == NULL)
    prepareWin(al,args->w,args->S);
  numWin = 0;
  /* take care of first window */
  rb = 0;
  numPol = 0;
  while(rb < args->w && rb < al->n)
    numPol += al->polInd[rb++];
  arr[numWin++] = numPol;
  /* scan remaining windows */
  lb = 0;
  while(rb < al->n-args->S+1){
    for(i=0;i<args->S;i++){
      if(al->polInd[rb+i])
         numPol++;
      if(al->polInd[lb+i])
        numPol--;
    }
    rb += args->S;
    lb += args->S;
    arr[numWin++] = numPol;
  }
  return arr;
}

/* td: compute Tajima's D */
double td(double a1, double a2, double s, double p, int n){
  double b1, b2, c1, c2, e1, e2;
  double d;

  b1=(n+1)/3./(n-1);
  b2=2.*(n*n+n+3.)/(9.*n*(n-1));
  c1=b1-1./a1;
  c2=b2-(n+2)/(a1*n)+a2/a1/a1;
  e1=c1/a1;
  e2=c2/(a1*a1+a2);

  if(s>0)
    d=(p-s/a1)/sqrt(e1*s+e2*s*(s-1));
  else
    d=0.;

  return d;
}

double *winTajima(Args *args, Alignment *al){
  long i;
  double *pArr, *sArr, *dArr;
  double a1, a2;

  a1 = 0;
  a2 = 0;
  for(i=1;i<al->m;i++){
    a1 += 1.0/i;
    a2 += (1.0/i/i);
  }

  pArr = winPi(args, al);
  sArr = winSs(args, al);
  dArr = (double *)emalloc((al->n-args->w+1)*sizeof(double));

  for(i=0;i<al->n-args->w+1;i++){
    dArr[i] = td(a1, a2, sArr[i], pArr[i], al->m);
  }
  return dArr;
}


Alignment *seq2al(Sequence *seq){
  int i;
  Alignment *al;

  al = (Alignment *)emalloc(sizeof(Alignment));
  al->m = seq->numSeq;
  al->headers = seq->headers;
  al->n = seq->borders[0];
  al->al = (char **)emalloc(al->m*sizeof(char *));
  al->al[0] = seq->seq;
  for(i=1;i<seq->numSeq;i++)
    al->al[i] = seq->seq + seq->borders[i-1] + 1;

  al->numPoly = 0;
  al->poly = NULL;
  al->polInd = NULL;
  al->nucInd = NULL;
  return al;
}

/* findPoly: fills an array of positions that are polymorphic
 *     and consist solely of the four canonical nucleotides.
 */
void findPoly(Alignment *al){
  int i, j, c, p;
  int *dic;

  dic = getRestrictedDnaDictionary(NULL);
  al->poly = (long *)emalloc(al->n*sizeof(long));
  al->nuc = (long*)emalloc(al->n*sizeof(long));
  al->polyFreq = (float *)emalloc(al->n*sizeof(float));

  al->numPoly = 0;
  al->numNuc = 0;
  for(i=0;i<al->n;i++){
    p = 0;
    if(dic[(int)al->al[0][i]])
      c = 1;
    else{
      c = 0;
      continue;
    }
    for(j=1;j<al->m;j++){
      if(!dic[(int)al->al[j][i]]){
        c = 0;
        break;
      }
      if(al->al[0][i] != al->al[j][i])
        p = 1;
    }
    if(c){
      al->nuc[al->numNuc++] = i;
      if(p){
        al->poly[al->numPoly] = i;
        al->polyFreq[al->numPoly] = 1.;
        for(j=1;j<al->m;j++)
          if(al->al[j][i] == al->al[0][i])
            al->polyFreq[al->numPoly]++;
        al->polyFreq[al->numPoly] /= (float)al->m;
        if(al->polyFreq[al->numPoly] > 0.5)
          al->polyFreq[al->numPoly] = 1. - al->polyFreq[al->numPoly];
        al->numPoly++;
      }
    }
  }
  if(al->numPoly)
    al->poly = (long *)erealloc(al->poly,al->numPoly*sizeof(long));
  if(al->numNuc)
    al->nuc = (long *)erealloc(al->nuc,al->numNuc*sizeof(long));
  if(al->numPoly)
    al->polyFreq = (float *)erealloc(al->polyFreq,al->numPoly*sizeof(float)
       );
  free(dic);
}

/* prepareWin: prepare sliding window analysis
 *   should be preceded by findPoly, though this
 *   is checked for
 */
void prepareWin(Alignment *al, int winLen, int stepLen){
  int i, j, rb, lb, numNuc;

  if(al->poly == NULL)
    findPoly(al);
  al->polInd = (char *)emalloc(al->n*sizeof(char));
  al->nucInd = (char *)emalloc(al->n*sizeof(char));
  /* mark canonical nucleotides */
  j = 0;
  for(i=0;i<al->numNuc;i++){
    while(j<al->nuc[i])
      al->nucInd[j++] = 0;
    al->nucInd[j++] = 1;
  }
  for(i=j;i<al->n;i++)
    al->nucInd[i] = 0;

  /* mark polymorphisms */
  j = 0;
  for(i=0;i<al->numPoly;i++){
    while(j<al->poly[i])
      al->polInd[j++] = 0;
    al->polInd[j++] = 1;
  }
  for(i=j;i<al->n;i++)
    al->polInd[i] = 0;
  /* count canonical nucleotides per window */
  al->winNumNuc = (int *)emalloc(al->n*sizeof(int));
  /* take care of first window */
  rb = 0;
  numNuc = 0;
  al->numWin = 0;
  while(rb < winLen && rb < al->n)
    numNuc += al->nucInd[rb++];
  al->winNumNuc[al->numWin++] = numNuc;
  /* scan remaining windows */
  lb = 0;
  while(rb < al->n-stepLen+1){
    for(i=0;i<stepLen;i++){
      if(al->nucInd[rb+i])
        numNuc++;
      if(al->nucInd[lb+i])
        numNuc--;
    }
    rb += stepLen;
    lb += stepLen;
    al->winNumNuc[al->numWin++] = numNuc;
  }
}
```

References

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

[2] J. Rozas, J. C. Sánchez-DelBarrio, X. Messeguer, and R. Rozas. DnaSP, DNA polymorphism analyses by the coalescent and other methods. *Bioinformatics*, 19:2496–2497, 2003.

[3] F. Tajima. Statistical method for testing the neutral mutation hypothesis by DNA polymorphism. *Genetics*, 123:585–595, 1989.