We are IntechOpen, the world's leading publisher of Open Access books Built by scientists, for scientists

6,900
Open access books available

185,000
International authors and editors

200M
Downloads

Our authors are among the

154
Countries delivered to

TOP 1%
most cited scientists

12.2%
Contributors from top 500 universities



WEB OF SCIENCE™

Selection of our books indexed in the Book Citation Index in Web of Science™ Core Collection (BKCI)

Interested in publishing with us?
Contact book.department@intechopen.com

Numbers displayed above are based on latest data collected.
For more information visit www.intechopen.com



3

Evolutionary Bioinformatics with a Scientific Computing Environment

James J. Cai
Texas A&M University,
College Station, Texas
USA

1. Introduction

Modern scientific research depends on computer technology to organize and analyze large data sets. This is more true for evolutionary bioinformatics—a relatively new discipline that has been developing rapidly as a sub-discipline of bioinformatics. Evolutionary bioinformatics devotes to leveraging the power of nature's experiment of evolution to extract key findings from sequence and experimental data. Recent advances in high-throughput genotyping and sequencing technologies have changed the landscape of data collection. Acquisition of genomic data at the population scale has become increasingly cost-efficient. Genomic data sets are accumulating at an exponential rate and new types of genetic data are emerging. These come with the inherent challenges of new methods of statistical analysis and modeling. Indeed new technologies are producing data at a rate that outpaces our ability to analyze its biological meanings.

Researchers are addressing this challenge by adopting mathematical and statistical software, computer modeling, and other computational and engineering methods. As a result, bioinformatics has become the latest engineering discipline. As computers provide the ability to process the complex models, high-performance computer languages have become a necessity for implementing state-of-the-art algorithms and methods.

This chapter introduces one of such emerging programming languages—Matlab. Examples are provided to demonstrate Matlab-based solutions for preliminary and advanced analyses that are commonly used in molecular evolution and population genetics. The examples relating to molecular evolution focus on the mathematical modeling of sequence evolution; the examples relating to population genetics focus on summary statistics and neutrality tests. Several examples use functions in toolboxes specifically developed for molecular evolution and population genetics—MBEToolbox (Cai, Smith et al. 2005; Cai, Smith et al. 2006) and PGEToolbox (Cai 2008). The source code of some examples is simplified for the publication purpose.

2. Starting Matlab

Matlab is a high-level language and computing environment for high-performance numerical computation and visualization. Matlab integrates matrix computation, numerical analysis, signal processing, and graphics in an easy-to-use environment and simplifies the process of

solving technical problems in a variety of disciplines. With Matlab, users access very extensive libraries (i.e., toolboxes) of predefined functions to perform computationally intensive tasks faster than with traditional programming languages such as C, C++, and Fortran. Over the years, Matlab has evolved as a premier program in industrial and educational settings for solving practical engineering and mathematical problems. Researchers in bioinformatics are increasingly relying on Matlab to accelerate scientific discovery and reduce development time.

2.1 Creating & manipulating vectors and matrices

Matlab was designed in the first instance for the purposes of numerical linear algebra. Since its conception, it has acquired many advanced features for the manipulation of vectors and matrices. These features make Matlab an ideal computing language for manipulating genomic data. The basic data type of Matlab is the matrix. Many commonly used genomic data, such as sequences, genotypes, and haplotypes, can be naturally represented as numeric matrices in the computer memory. Therefore, highly efficient basic functions of Matlab can be applied directly to handling many kinds of genomic data. Here is an example of aligned DNA sequences:

```
Seq1   ATCAGGCATCGATGAATCGT
Seq2   ATCGGGCATCGATCAAGCGT
Seq3   ATCGGTCATCTATGAAGGCT
Seq4   ATCGGTCATCGAAGAAGGCG
Seq5   ATCGGTCATCGATCAAGGCG
```

As these sequences are in the same length and are aligned, the alignment can be represented by a Matlab matrix of integers:

```
seq=[1 4 2 1 3 3 2 1 4 2 3 1 4 3 1 1 4 2 3 4
     1 4 2 3 3 3 2 1 4 2 3 1 4 2 1 1 3 2 3 4
     1 4 2 3 3 4 2 1 4 2 4 1 4 3 1 1 3 3 2 4
     1 4 2 3 3 4 2 1 4 2 3 1 1 3 1 1 3 3 2 3
     1 4 2 3 3 4 2 1 4 2 3 1 4 2 1 1 3 3 2 3];
```

The simple mapping converts nucleotide sequences from letter representations (A, C, G, and T) to integer representations (1, 2, 3, and 4). Similarly, genotypic data can be converted into a matrix of integers. The genotypic data below contains nine markers (SNPs) sampled from eight diploid individuals.

```
Idv1   CT GT AG AT AG AG CT AG AG
Idv2   CT GT AG AT AG AG CT AG AG
Idv3   CC GG GG AA AA AA TT GG GG
Idv4   TT TT AA TT GG GG CC AA AA
Idv5   CT GT AG AT AG AG CT AG AG
Idv6   CT GT AG AT AG AG CT AG AG
Idv7   CC GG GG AA AA AA TT GG GG
Idv8   CT GT AG AT AG AG CT AG AG
```

This genotypic data can be converted into the following matrix of integers:

```
geno=[2 4 3 4 1 3 1 4 1 3 1 3 2 4 1 3 1 3
      2 4 3 4 1 3 1 4 1 3 1 3 2 4 1 3 1 3
      2 2 3 3 3 3 1 1 1 1 1 1 4 4 3 3 3 3
      4 4 4 4 1 1 4 4 3 3 3 3 2 2 1 1 1 1
      2 4 3 4 1 3 1 4 1 3 1 3 2 4 1 3 1 3
      2 4 3 4 1 3 1 4 1 3 1 3 2 4 1 3 1 3
      2 2 3 3 3 3 1 1 1 1 1 1 4 4 3 3 3 3
      2 4 3 4 1 3 1 4 1 3 1 3 2 4 1 3 1 3];
```

Structures and cell arrays in Matlab provide a way to store dissimilar types of data in the same array. In this example, information about markers, such as the chromosomal position and SNP identification, can be represented in a structure called `mark`:

```
mark.pos=[38449934,38450800,38455228,38456851,38457117,38457903,...
          38465179,38467522,38469351];

mark.rsid={'rs12516','rs8176318','rs3092988','rs8176297',...
           'rs8176296','rs4793191','rs8176273','rs8176265',...
           'rs3092994'};
```

In the same way, you can represent haplotypes with an integer matrix, `hap`, and represent makers' information of the haplotype with a `mark` structure. The difference between sequences of `hap` and `seq` is that `hap` usually contains only sites that are polymorphic and chromosomal positions of these sites are likely to be discontinuous; whereas, `seq` includes both monoallelic and polymorphic sites, which are continuous in their chromosomal position.

Matlab supports many different data types, including integer and floating-point data, characters and strings, and logical true and false states. By default, all numeric values are stored as double-precision floating point. You can choose to build numeric matrices and arrays as integers or as single-precision. Integer and single-precision arrays offer more memory-efficient storage than double-precision. You can convert any number, or array of numbers, from one numeric data type to another. For example, a double-precision matrix `geno` can be converted into an unsigned 8-bit integer matrix by using command `uint8(geno)` without losing any information. The output matrix takes only one-eighth the memory of its double-precision version. The signed or unsigned 8-bit integer, like logical value, requires only 1 byte. They are the smallest data types. Sparse matrices with mostly zero-valued elements, such as adjacency matrices of most biological networks, occupy a fraction of the storage space required for an equivalent full matrix.

2.2 Numerical analysis

Matlab has many functions for numerical data analysis, which makes it a well suited language for numerical computations. Typical uses include problem solving with matrix formulations, general purpose numeric computation, and algorithm prototyping. Using Matlab in numerical computations, users can express the problems and solutions just as they are written mathematically—without traditional programming. As a high-level language, Matlab liberates users from implementing many complicated algorithms and commonly used numerical solutions, and allows users to focus on the "real" problems they want to

solve, without understanding the details of routine computational tasks. This section introduces three numerical routines: optimization, interpolation, and integration.

2.2.1 Optimization

Matlab built-in functions and specific toolboxes provide widely used algorithms for standard and large-scale optimization, solving constrained and unconstrained continuous and discrete problems. Users can use these algorithms to find optimal solutions, perform tradeoff analyses, balance multiple design alternatives, and incorporate optimization methods into algorithms and models. Here I use two functions `fminbnd` and `fminsearch` to illustrate the general solutions to the problems of constrained linear optimization and unconstrained nonlinear optimization, respectively.

Function `fminbnd` finds the minimum of a single-variable function, $\min_x f(x)$, within a fixed interval $x_1 < x < x_2$. In Matlab,

```
[x,fval]=fminbnd(@fun,x1,x2);
```

returns scalar `x` a local minimizer of the scalar valued function, which is described by a function handle `@fun`, in the interval between `x1` and `x2`. The second returning variable `fval` is the value of the objective function computed in `@fun` at the solution `x`. Function `fminsearch` finds minimum of unconstrained multivariable function using a derivative-free method. As above, the minimum of a problem is specified by $\min_{\vec{x}} f(\vec{x})$, where $\vec{x}$ is a vector instead of a scalar, and f is a function of several variables. In Matlab this is written as:

```
[x,fval]=fminsearch(@fun,x0);
```

where `x0` is a vector of initial values of `x`. Note that `fminsearch` can often handle discontinuities particularly if they do not occur near the solution.

Depending on the nature of the problem, you can choose to use `fminbnd`, `fminsearch`, or other optimization functions to perform likelihood inference. When doing this, you first need to write a likelihood function that accepts initial parameters as inputs. The likelihood function typically returns a value of the negative log likelihood. Input parameters that produce the minimum of the function are those that give the maximum likelihood for the model. Here is an example showing how to use function `fminbnd`.

```
options=optimset('fminbnd');
[x,fval]=fminbnd(@likefun,eps,20,options,tree,site,model);
```

where `@likefun` is a function handle of the following negative log-likelihood function:

```
function [L]=likefun(x,tree,site,model)
rate=x(1);
L=-log(treelike(tree,site,model,rate));
```

This function takes four parameters: the evolutionary rate, `rate`, (which is what we are going to optimize), a phylogenetic tree, a site of alignment of sequences, and a substitution

model. `fminbnd` returns estimate `x(1)`, which is the optimized `rate` that gives maximum likelihood `fval`. The function `treelike` computes the likelihood of a tree for a given site under the substitution model (see Section 3.4 for details).

2.2.2 Interpolation

Interpolation is one of the classical problems in numerical analysis. Here I show how a one dimensional interpolation problem is formulated and how to use the interpolation technique to determine the recombination fraction between two chromosomal positions. The relationships between physical distance (Mb) and genetic distance (cM) vary considerably at different positions on the chromosome due to the heterogeneity in recombination rate. A recombination map correlates the increment of genetic distance with that of physical distance. The distances between two points in a recombination map are defined in terms of recombination fractions. The incremental step of the physical distance is fixed by the distance between each pair of consecutive makers. Given a set of n makers $[x_k, y_k]$, $1 \le k \le n$, with $x_1 < x_2 < \ldots < x_n$, the goal of interpolation is to find a function $f(x)$ whose graph interpolates the data points, i.e., $f(x_k) = y_k$, for $k = 1, 2, \ldots, n$. The general form of Matlab function `interp1` is as follows:

```
yi=interp1(x,y,xi,method)
```

where `x` and `y` are the vectors holding x-coordinates (i.e., the chromosomal positions) and y-coordinates (i.e., the cumulative recombination rate) of points to be interpolated, respectively. `xi` is a vector holding points of evaluation, i.e., `yi=f(xi)` and method is an optional string specifying an interpolation method. Default interpolation method `'linear'` produces a piecewise linear interpolant. If `xi` contains two positions on the chromosome, `xi=[pos1,pos2]`, `yi` computed will contain two values `[rec1,rec2]`. The local recombination rate (cM/Mb) can then be calculated as `(rec2-rec1)/(pos2-pos1)`.

2.2.3 Integration

The basic problem considered by numerical integration is to compute an approximate solution to a definite integral $\int_a^b f(x)dx$. If $f(x)$ is a smooth well-behaved function, integrated over a small number of dimensions and the limits of integration are bounded, there are many methods of approximating the integral with arbitrary precision. `quad(@fun,a,b)` approximates the integral of function `@fun` from `a` to `b` to within an error of 1e-6 using recursive adaptive Simpson quadrature. You can use function `trapz` to compute an approximation of the integral of `Y` via the trapezoidal method. To compute the integral with unit spacing, you can use `Z=trapz(Y);` for spacing other than one, multiply Z by the spacing increment. You can also use `Z=trapz(X,Y)` to compute the integral of `Y` with respect to `X` using trapezoidal integration.

2.3 Data visualization & graphical user interfaces

Matlab adopts powerful visualization techniques to provide excellent means for data visualization. The graphics system of Matlab includes high-level commands for two-dimensional and three-dimensional data visualization, image processing, animation, and presentation graphics. The graphic system of Matlab is also highly flexible as it includes

low-level commands that allow users to fully customize the appearance of graphics. Fig. 1 gives some examples of graphic outputs from data analyses in evolutionary bioinformatics.

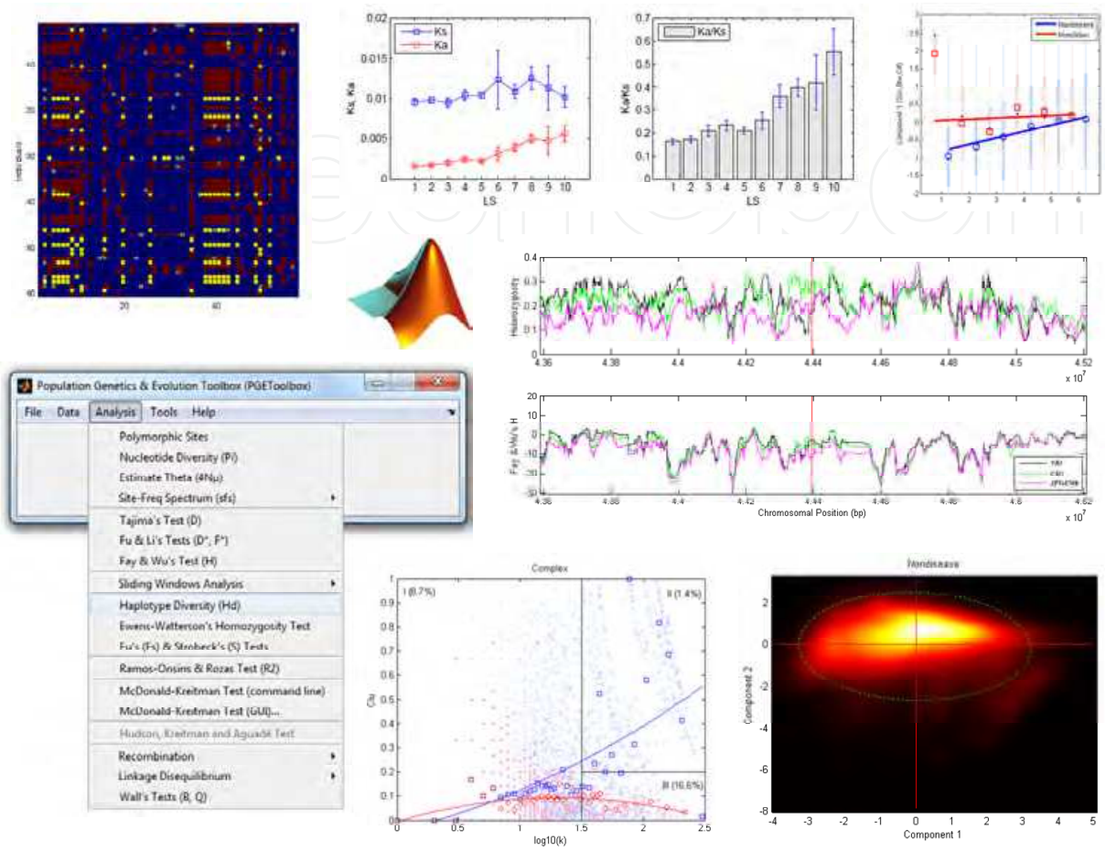


Fig. 1. Examples of graphic outputs and GUIs.

Matlab is also a convenient environment for building graphical user interfaces (GUI). A good GUI can make programs easier to use by providing them with a consistent appearance and intuitive controls. Matlab provides many programmable controls including push buttons, toggle buttons, lists, menus, text boxes, and so forth. A tool called `guide`, the GUI Development Environment, allows a programmer to select, layout and align the GUI components, edit properties, and implement the behavior of the components. Together with `guide`, many GUI-related tools make Matlab suitable for application development. `PGEGUI` and `MBEGUI` are two menu-driven GUI applications in PGEToolbox and MBEToolbox, respectively.

2.4 Extensibility & scalability

Matlab has an open, component-based, and platform-independent architecture. Scientific applications are difficult to develop from scratch. Through a variety of toolboxes, Matlab offers infrastructure for data analyses, statistical tests, modeling and visualization, and other services. A richer set of general functions for statistics and mathematics allows scientists to manipulate and view data sets with significantly less coding effort. Many special-purpose

tool boxes that address specific areas are provided and developers choose only the tools and extensions needed. Thus extensibility is one of the most important features of the Matlab environment. Matlab functions have a high degree of portability, which stems from a complete lack of coupling with the underlying operating system and platform. Matlab application deployment tools enable automatic generation and royalty-free distribution of applications and components. You can distribute your code directly to others to use in their own Matlab sessions, or to people who do not have Matlab.

You can run a Matlab program in parallel. The parallel computing toolbox allows users to offload work from one Matlab session (the client) to other Matlab sessions (the workers). It is possible to use multiple workers to take advantage of the parallel processing on a remote cluster of computers. This is called "remote" parallel processing. It is also possible to do parallel computing with Matlab on a single multicore or multiprocessor machine. This is called "local" parallel computing. Fig. 2 is a screenshot of the task manager showing the CPU usage on a single 8-core PC in local parallel computing with Matlab.

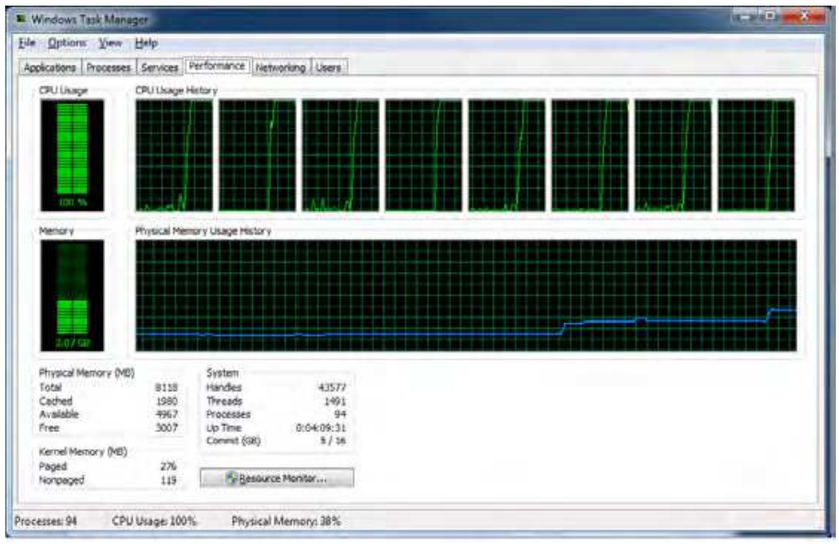


Fig. 2. CPU usage of a single 8-core PC in local parallel computing with Matlab.

With a copy of Matlab that has the parallel computing features, the simplest way of parallelizing a Matlab program is to use the `for` loops in the program. If a `for` loop is suitable for parallel execution, this can be indicated simply by replacing the word `for` by the word `parfor`. When the Matlab program is run, and if workers have been made available by the `matlabpool` command, then the work in each `parfor` loop will be distributed among the workers. Another way of parallelizing a Matlab program is to use a `spmd` (single program, multiple data) statement. Matlab executes the `spmd` body denoted by statements on several Matlab workers simultaneously. Inside the body of the `spmd` statement, each Matlab worker has a unique value of `labindex`, while `numlabs` denotes the total number of workers executing the block in parallel. Within the body of the `spmd` statement, communication functions for parallel jobs (such as `labSend` and `labReceive`) can transfer data between the workers. In addition, Matlab is developing new capabilities for the graphics processing unit (GPU) computing with CUDA-enabled NVIDIA devices.

3. Using Matlab in molecular evolution

Molecular evolution focuses on the study of the process of evolution at the scale of DNA, RNA, and proteins. The process is reasonably well modeled by using finite state continuous time Markov chains. In Matlab we obtain compact and elegant solutions in modeling this process.

3.1 Evolutionary distance by counting

Before explicitly modeling the evolution of sequences, let's start with simple counting methods for estimating evolutionary distance between DNA or protein sequences. If two DNA or protein sequences were derived from a common ancestral sequence, then the evolutionary distance refers to the cumulative amount of difference between the two sequences. The simplest measure of the distance between two DNA sequences is the number of nucleotide differences (N) between the two sequences, or the portion of nucleotide differences ($p = N/L$) between the two sequences. In Matlab, the p-distance between two aligned sequences can be computed like this:

```
p=sum(seq(1,:)~=seq(2,:))/size(seq,1);
```

To correct for the hidden changes that have occurred but cannot be directly observed from the comparison of two sequences, the formula for correcting multiple hits of nucleotide substitutions can be applied. The formulae used in these functions are analytical solutions of a variety of Markov substitution models. The simplest model is the JC model (Jukes and Cantor 1969). Analytic solution of the JC model corrects p-distance when $p < 0.75$:

```
d=-(3/4)*log(1-4*p/3);
```

Other commonly used models include Kimura-two-parameter (K2P)(Kimura 1980), Felsenstein (F84)(Felsenstein 1984), and Hasegawa-Kishono-Yano (HKY85)(Hasegawa, Kishino et al. 1985). When the numbers of parameters used to define a model increase with the complicity of the model, we reach a limit where there is no analytical solution for the expression of evolutionary distance. In these cases, we can use the maximum likelihood method, as described in Section 3.3, to estimate the evolutionary distance.

For protein sequences, the simplest measure is the p-distance between two sequences. Assume that the number of amino acid substitutions at a site follows the Poisson distribution; a simple approximate formula for the number of substitutions per site is given by:

```
d=-log(1-p);
```

This is called Poisson correction distance. Given that different amino acid residues of a protein have different levels of functional constraints and the substitution rate varies among the sites, it is suggested that the rate variation can be fitted by the gamma distribution (Nei and Kumar 2000). The gamma distance between two sequences can be computed by:

```
d=a*((1-p)^(-1/a)-1);
```

where `a` is the shape parameter of the gamma distribution. Several methods have been proposed to estimate `a` (Yang 1994; Gu and Zhang 1997). The gamma distance with `a=2.4`

is an approximate of the JTT distance based on the 20×20 amino acid substitution matrix developed by Jones, Taylor et al. (1992). The maximum likelihood estimation of JTT distance is described in Section 3.3.2.

Protein sequences are encoded by strings of codons, each of which is a triplet of nucleotides and specifies an amino acid according to the genetic code. Codon-based distance can be estimated by using the heuristic method developed by Nei and Gojobori (1986). The method has been implemented with an MBEToolbox function called `dc_ng86`. The function counts the numbers of synonymous and nonsynonymous sites (L_S and L_A) and the numbers of synonymous and nonsynonymous differences (S_S and S_A) by considering all possible evolutionary pathways. The codon-based distance is measured as $K_S = S_S/L_S$ and $K_A = S_A/L_A$ for synonymous and nonsynonymous sites, respectively. Comparison of K_S and K_A provide useful information about natural selection on protein-coding genes: $K_A/K_S = 1$ indicates neutral evolution, $K_A/K_S < 1$ negative selection, and $K_A/K_S > 1$ positive selection.

3.2 Markov models of sequence evolution

Markov models of sequence evolution have been widely used in molecular evolution. A Markov model defines a continuous-time Markov process to describe the change between nucleotides, amino acids, or codons over evolutionary time. Markov models are flexible and parametrically succinct. A typical Markov model is characterized by an instantaneous rate matrix $\boldsymbol{R}$, which defines the instantaneous relative rates of interchange between sequence states.

$\boldsymbol{R}$ has off-diagonal entries R_{ij} equal to the rates of replacement of i by j: $R_{ij} = r(i \to j)$, $i \neq j$. The diagonal entries, R_{ii}, are defined by a mathematical requirement that the row sums are all zero, that is, $R_{ii} = \sum_{j \neq i}(-R_{ij})$. The dimension of $\boldsymbol{R}$ depends on the number of statuses of the substitution: 4×4 for nucleotides, 20×20 for amino acids, and 61×61 for codons. We denote $\boldsymbol{\Pi}$ the vector that contains equilibrium frequencies for 4 nucleotides, 20 amino acids, or 61 sense codons, depending on the model. By multiplying the diagonal matrix of $\boldsymbol{\Pi}$, $\boldsymbol{R}$ is transformed into a "frequency-scaled" rate matrix $\boldsymbol{Q}$=diag($\boldsymbol{\Pi}$)*$\boldsymbol{R}$. Subsequently, we can compute the substitution probability matrix $\boldsymbol{P}$ according to the matrix exponential $P(t) = e^{Qt}$, where $\boldsymbol{P}(t)$ is the matrix of substitution probabilities over an arbitrary time (or branch length) t.

3.3 Model-based evolutionary distance

3.3.1 Nucleic acid substitutions

For a nucleotide substitution probability matrix $\boldsymbol{P}(t)$, $P_{i \to j}(t)$ is the probability that nucleotide i becomes nucleotide j after time t. An example of divergence of two sequences (each contains only 1 base pair) from a common ancestral sequence is shown in Fig. 3.

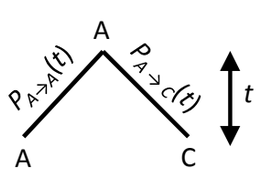


Fig. 3. Divergence of two sequences. Sequences 1 (left) and 2 (right) were derived from a common ancestral sequence t years ago. $P_{A \to C}(t)$ is the probability that nucleotide A becomes C after time t. $P_{A \to A}(t)$ is the probability that no substitution occurs at the site during time t.

In order to construct the substitution probability matrix `P` in Matlab, let's first define an instantaneous rate matrix `R`:

```
>> R=[0,.3,.4,.3;.3,0,.3,.4;.4,.3,0,.3;.3,.4,.3,0]

R =

         0    0.3000    0.4000    0.3000
    0.3000         0    0.3000    0.4000
    0.4000    0.3000         0    0.3000
    0.3000    0.4000    0.3000         0
```

We can use the following command to normalize the rate matrix so that the sum of each column is one:

```
x=sum(R,2); for k=1:4, R(k,:)=R(k,:)./x(k); end
```

This step is unnecessary in this particular example, as original `R` meets this requirement. Let's assume the equilibrium frequencies of four nucleotides are known (that is, π_A=0.1, π_C=0.2, π_G=0.3, and π_T=0.4).

```
freq=[.1 .2 .3 .4];
```

Here is how to compute and normalize matrix `Q`:

```
function [Q]=composeQ(R,freq)
PI=diag(freq);
Q=R*PI;
Q=Q+diag(-1*sum(Q,2));
Q=(Q./abs(trace(Q)))*size(Q,1);
```

In Matlab, function `EXPM` computes the matrix exponential using the Padé approximation. Using this function we can compute substitution probability matrix `P` for a given time `t`.

```
P=expm(Q*t);
```

For one site in two aligned sequences, without knowing the ancestral status of the site, we assume one of them is in the ancestral state and the other is in the derived state. If two nucleotides are C and T, and we pick C as the ancestral state, that is, the substitution from C to T, then the probability of substitution $P_{C\to T}(t)$ = `P(2,4)`. In fact, `P(2,4)` equals to `P(4,2)`, which means the process is reversible. So it does not matter which nucleotide we picked as ancestral one, the result is the same. The total likelihood of the substitution model for the two given sequences is simply the multiplication of substitution probabilities for all sites between the two sequences. In order to estimate the evolutionary distance between two sequences, we try different `t`-s and compute the likelihood each time until we find the `t` that gives the maximum value of the total likelihood. This process can be done with optimization functions in Matlab (see Section 2.2.1). The optimized value of `t` is a surrogate of evolutionary distance between two sequences.

The model of substitution can be specified with two variables `R` and `freq`. So we can define the model in a structure:

```
model.R=R;
model.freq=freq;
```

The general time reversible (GTR) model has 8 parameters (5 for rate matrix and 3 for stationary frequency vector). There is no analytical formula to calculate the GTR distance directly. We can invoke the optimization machinery of Matlab to estimate the evolutionary distance and obtain the best-fit values of parameters that define the substitution model.
A convenient method that does not depend on the optimization to compute GTR distance also exists (Rodriguez, Oliver et al. 1990). The first step of this method is to form a matrix $\boldsymbol{F}$, where F_{ij} denotes the number of sites for which sequence 1 has an i and sequence 2 has a j. The GTR distance between the two sequences is then given by the following formula:

$$d = -tr(\Pi \log(\Pi^{-1} F)),$$

where Π is the diagonal matrix with values of nucleotide equilibrium frequencies on the diagonal, and $tr(X)$ is the trace of matrix X. Here is an example:

```
seq1=[2 3 4 2 3 3 1 4 3 3 3 4 1 3 3 2 4 2 3 2 2 2 1 3 1 3 1 3 3 3];
seq2=[4 2 2 2 3 3 2 4 3 3 2 4 1 2 3 2 4 4 1 4 2 2 1 3 1 2 4 3 1 3];
X=countntchange(seq1,seq2)

X =

     3     0     2     0
     1     4     4     1
     0     0     8     0
     1     3     0     3
```

The formula for computing GTR distance is expressed in Matlab as:

```
F=((sum(sum(X))-trace(X))*R)./4;
F=eye(4)*trace(X)./4+F;
PI=diag(freq);
d=-trace(PI*logm(inv(PI)*F));
```

3.3.2 Amino acid substitutions

For an amino acid substitution probability matrix $\boldsymbol{P}(t)$, $P_{i\to j}(t)$ is the probability that amino acid i becomes amino acid j after time t. In order to compute P, we need to specify the substitution model. As in the case of nucleotides, we need an instantaneous rate matrix `model.R` and equilibrium frequency `model.freq` for amino acids. Commonly used `R` and `freq` are given by empirical models including Dayhoff, JTT (Jones, Taylor et al. 1992)(Fig. 4), and WAG (Whelan 2008).

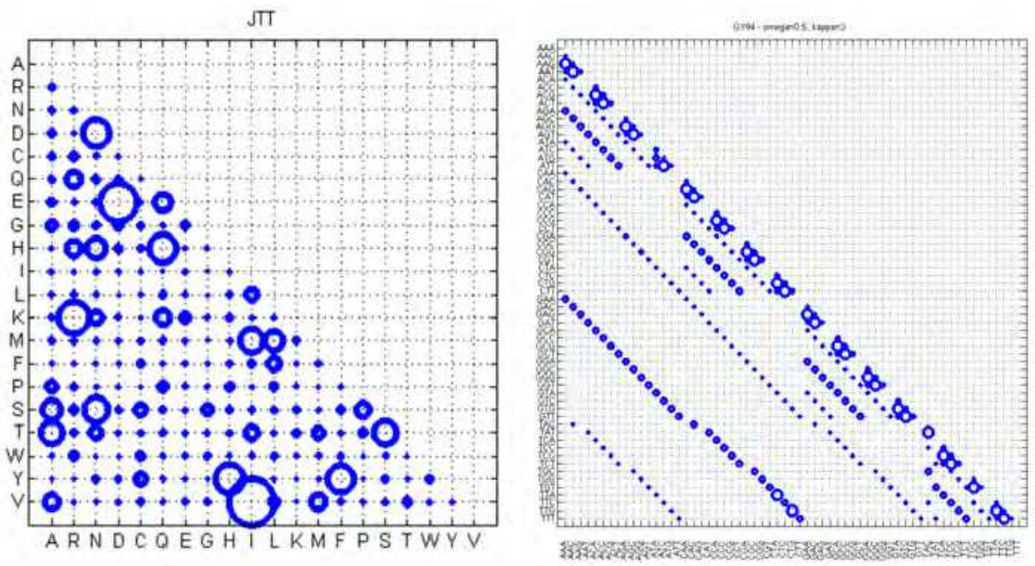


Fig. 4. Visual representations for instantaneous rate matrix ***R***. The JTT model of amino acid substitutions (Jones, Taylor et al. 1992) is shown on the left, and the GY94 model of codon substitutions (Goldman and Yang 1994) on the right. The circle size is proportional to the value of the relative rate between pairs of substitutions.

Here I present a function called `seqpairlike` that computes the log likelihood of distance `t` (i.e., branch length, or time) between two protein sequences `seq1` and `seq2` using the model defined with `R` and `freq`. The function `countaachange` is a `countntchange` counterpart for amino acid substitutions.

```
function [lnL]=seqpairlike(t,model,seq1,seq2)
Q=composeQ(model.R,model.freq);
P=expm(Q*t);
X=countaachange(seq1,seq2);
lnL=sum(sum(log(P.^X)));
```

Using the likelihood function, you can adopt an optimization technique to find the optimized `t` as the evolutionary distance between the two sequences.

3.3.3 Codon substitutions

Codon substitutions can be modeled using a Markov process similar to those that are used to describe nucleotide substitutions and amino acid substitutions. The difference is that there are 61 states in the Markov process for codon substitutions as the universal genetic code contains 61 sense codons or nonstop codons. Here I describe a simplified model of Goldman and Yang (1994)(gy94 model). The rate matrix of the model accounts for the transition-transversion rate difference by incorporating the factor κ if the nucleotide change between two codons is a transition, and for unequal synonymous and nonsynonymous substitution rates by incorporating ω if the change is a nonsynonymous substitution. Thus, the rate of relative substitution from codon i to codon j ($i \neq j$) is:

$$q_{ij} = \begin{cases} 0, & \text{if } i \text{ and } j \text{ differ at two or three codon positions,} \\ \pi_j, & \text{if } i \text{ and } j \text{ differ by a synonymous transversion,} \\ \kappa\pi_j, & \text{if } i \text{ and } j \text{ differ by a synonymous transition,} \\ \varpi\pi_j, & \text{if } i \text{ and } j \text{ differ by a nonsynonymous transversion,} \\ \varpi\kappa\pi_j, & \text{if } i \text{ and } j \text{ differ by a nonsynonymous transition,} \end{cases}$$

A schematic diagram representing the codon-based rate matrix $\boldsymbol{R}$ with $\omega = 0.5$ and $\kappa = 3.0$ is given in Fig. 4. The function `modelgy94` in MBEToolbox generates the matrix R from given ω and κ:

```
model=modelgy94(omega,kappa);
```

Now let π_j indicate the equilibrium frequency of the codon j. In the GY94 model, $\pi_j = 1/61$, $j = 1, 2, \ldots, 61$.
Here is how we can use GY94 model to estimate dN and dS for two protein-coding sequences. Two sequences are encoded with 61 integers—each represents a sense codon. For example, the following two protein-coding sequences:

```
Seq1   AAA AAC AAG AAT ACA ACC
Seq2   AAT AAC AAG TTA TCA CCC
```

are represented in Matlab with `seq1` and `seq2` like this:

```
seq1=[1 2 3 4 5 6];
seq2=[4 2 3 58 51 22];
```

The codons in original sequences are converted into corresponding indexes in the 61 sense codon list (when the universal codon table is used). This conversion can be done with the function `codonise61` in MBEToolbox: `seq1=codonise61('AATAACAAGTTATCACCC');`
You also need a 61×61 mask matrix that contains 1 for every synonymous substitution between codons, and 0 otherwise.

```
% Making a mask matrix, M
T='KNKNTTTTRSRSIIMIQHQHPPPPRRRRLLLLEDEDAAAAGGGGVVVVYYSSSSCWCLFLF';
M=zeros(61);
for i=1:61
for j=i:61
       if i~=j
       if T(i)==T(j)% synonymous change
              M(i,j)=1;
       end
       end
end
end
M=M+M';
```

In the above code, `T` is the universal code translation table for 61 codons and the corresponding amino acids. Below is the likelihood function that will be used to obtain the three parameters (`t`, `kappa` and `omega`) for the given sequences `seq1` and `seq2`. The input variable `x` is a vector of `[t,kappa,omega]`.

```
function [lnL]=codonpairlike(x,seq1,seq2)
lnL=inf;
if (any(x<eps)||any(x>999)), return; end
t=x(1); kappa=x(2); omega=x(3);
if (t<eps||t>5), return; end
if (kappa<eps||kappa>999), return; end
if (omega<eps||omega>10), return; end
md=modelgy94(omega,kappa);
R=md.R; freq=md.freq;
Q=composeQ(R,freq);
P=expm(Q*t);
lnL=0;
for k=1:length(seq1)
       s1=seq1(k); s2=seq2(k);
       p=P(s1,s2);
       lnL=lnL+log(p*freq(s1));
end
lnL=-lnL;
```

Given all these, you can now compute the synonymous and nonsynonymous substitution rates per site, d_S and d_N, using maximum likelihood approach:

```
et=0.5; ek=1.5; eo=0.8;   % initial values for t, kappa and omega
options=optimset('fminsearch');
[para,fval]=fminsearch(@codonpairlike,[et,ek,eo],options,seq1,seq2);
lnL=-fval;
t=para(1);
kappa=para(2);
omega=para(3);

% build model using optimized values
md=modelgy94(omega,kappa);
Q=composeQ(md.R,md.freq)./61;

% Calculate pS and pN, assuming omega=optimized omega
pS=sum(sum(Q.*M));
pN=1-pS;

% Calculate pS and pN, assuming omega=1
md0=modelgy94(1,kappa);
Q0=composeQ(md0.R,md0.freq)./61;
pS0=sum(sum(Q0.*M));
pN0=1-pS0;

% Calculate dS and dN
dS=t*pS/(pS0*3);
dN=t*pN/(pN0*3);
```

3.4 Likelihood of a tree

You have learned how to compute the likelihood of substitutions between pairs of sequences. Here I show how to calculate the likelihood of a phylogenic tree given nucleotide sequences. Same technique applies to protein and codon sequences. Imagine you have a tree like the one in Fig. 5. In this example, the four sequences are extremely short, each containing only one nucleotide (i.e., G, A, T, and T). For longer sequences, you can first compute the likelihood for each site independently, and then multiply them together to get the full likelihood for the sequences. The tree describes the evolutionary relationship of the four sequences.

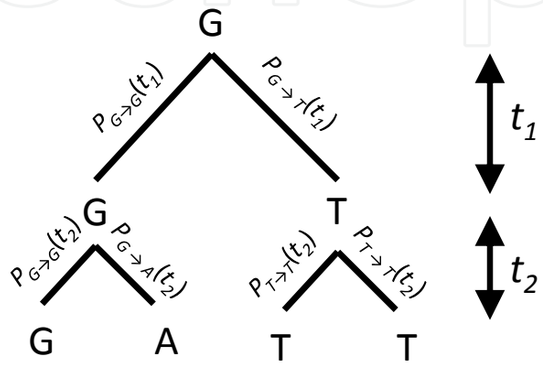


Fig. 5. One path of a tree with 4 external nodes and 3 internal nodes with known states.

Suppose that all internal nodes of the tree are known, which means the ancestral or intermediate states of the site are known. In this case, the likelihood of the tree is:

$$L = P_{G\to G}(t_1)\cdot P_{G\to T}(t_1)\cdot P_{G\to G}(t_2)\cdot P_{G\to A}(t_2)\cdot P_{T\to T}(t_2)\cdot P_{T\to T}(t_2)$$

Thus the likelihood of a phylogenetic tree with known internal nodes at one site can be calculated once the transition probability matrix $\boldsymbol{P}$ is computed as described in Section 3.3.1. In reality, the internal nodes of a tree are unlikely to be known, and the internal nodes can be any of nucleotides. In this case, we need to let every internal node be one of four possible nucleotides each time and compute the likelihood for all possible combinations of nodes. Each distinct combination of nucleotides on all nodes is called a path. Fig. 5 is an instance of one possible path. To get the likelihood of the tree, we multiply all likelihood values (or sum over log likelihood values) that are computed from all possible paths.
Here I use an example to illustrate how to do it using Matlab. Suppose the tree is given in the Newick format:

```
tree='((seq1:0.0586,seq2:0.0586):0.0264,(seq3:0.0586,seq4:0.0586):0.
0264):0.043;';
```

The function `parsetree` in MBEToolbox reads through the input tree and extracts the essential information including the topology of the tree, `treetop`, the total number of external nodes, `numnode`, and the branch lengths, `brchlen`.

```
[treetop,numnode,brchlen]=parsetree(tree);
```

The outputs of `parsetree` are equivalent to the following direct assignment for the three variables:

```
treetop='((1,2),(3,4))';
numnode=4;
brchlen=[0.0586 0.0586 0.0586 0.0586 0.0264 0.0264 0]';
```

Then we prepare an array of transition matrices `P`. Each transition matrix stacked in `P` is for one branch. The total number of branches, including both external and internal branches, is `2*numnode-2`.

```
n=4;          % number of possible nucleotides
numbrch=2*numnode-2;
P=zeros(numbrch*n,n);
for j=1:numbrch
   P((j-1)*n+1:j*n,:)=expm(Q*brchlen(j));
end
```

In the next step, we use a function called `mbelfcreator`, which is adapted from Phyllab (Morozov, Sitnikova et al. 2000), to construct an inline function `LF`. The function `mbelfcreator` takes two inputs, `treetop` and `numnod`, and "synthesizes" the function body of `LF`. The major operation encoded in the function body is the multiplication of all sub-matrices of the master `P` matrix. Each sub-matrix is 4×4 in dimension and is pre-computed for the corresponding branch of the tree. The order of serial multiplications is determined by the topology of tree.

```
>>LF=inline(mbelfcreator(treetop,numnode),'P','f','s','n')

LF =

     Inline function:
     LF(P,f,s,n) =
(f*(eye(n)*((P((4*n+1):(5*n),:)*(P((0*n+1):(1*n),s(1)).*P((1*n+1):(2
*n),s(2)))).*(P((5*n+1):(6*n),:)*(P((2*n+1):(3*n),s(3)).*P((3*n+1):(
4*n),s(4)))))))
```

The constructed inline function `LF` takes four parameters as inputs: `P` is the stacked matrix, `f` is the stationary frequency, `s` is a site of the sequence alignment, and `n` equals 4 for nucleotide data. With the inline function, we can compute the log likelihood of a site as follows:

```
siteL=log(LF(P,freq,site,n));
```

Finally, we sum over `siteL` for all sites in the alignment to get the total log likelihood of the tree for the given alignment.

Computing the likelihood of a tree is an essential step from which many further analyses can be derived. These analyses may include branch length optimization, search for best tree, branch- or site-specific evolutionary rate estimation, tests between different substitution models, and so on.

4. Using Matlab in population genetics

Population genetics studies allele frequency distribution and change under the influence of evolutionary processes, such as natural selection, genetic drift, mutation and gene flow. Traditionally, population genetics has been a theory-driven field with little empirical data. Today it has evolved into a data-driven discipline, in which large-scale genomic data sets test the limits of theoretical models and computational analysis methods. Analyses of whole-genome sequence polymorphism data from humans and many model organisms are yielding new insights concerning population history and the genomic prevalence of natural selection.

4.1 Descriptive statistics

Assessing genetic diversity within populations is vital for understanding the nature of evolutionary processes at the molecular level. In aligned sequences, a site that is polymorphic is called a "segregating site". The *number of segregating sites* is usually denoted by S. The expected number of segregating sites E(S) in a sample of size n can be used to estimate population scaled mutation rate $\theta = 4N_e\mu$, where N_e is the diploid effective population size and μ is the mutation rate per site:

$$\theta_W = S \Big/ \sum_{i=1}^{n-1} (1/i) \cdot$$

In Matlab, this can be written as:

```
[n,L]=size(seq);
S=countsegregatingsites(seq);
theta_w=S/sum(1./[1:n-1]);
```

In the above code, `countsegregatingsites` is a function in PGEToolbox.

Nucleotide diversity, π, is the average number of pairwise nucleotide differences between sequences:

$$\pi = \frac{1}{[n(n-1)/2]} \sum_{i}^{N} \sum_{j<i}^{N} d_{ij} \ ,$$

where d_{ij} is the number of nucleotide differences between the ith and jth DNA sequences and n is the sample size. The expected value of π is another estimator of θ, i.e., $\theta_\pi = \pi$.

```
n=size(seq,1);
x=0;
for i=1:n-1
       for j=i+1:n
              d=sum(seq(i,:)~=seq(j,:));
              x=x+d;
       end
end
theta_pi=x/(n*(n-1)/2);
```

Note that, instead of using the straightforward approach that examines all pairs of sequences and counts the nucleotide differences, it is often faster to start by counting the

number of copies of each type in the sequence data. Let n_i denote the number of copies of type i, and let $n_{hap} = \sum n_i$. To count the number of copies of the type i, we use the function `counthaplotype` in PGEToolbox. The general form of the function call is like this:

```
[numHap,sizHap,seqHap]=counthaplotype(hap);
```

where `numHap` is the total number of distinct sequences or haplotypes, and `sizHap` is a vector of numbers of each haplotypes. Apparently, `sum(sizHap)` equals `numHap`. `seqHap` is a matrix that contains the distinct haplotype sequences. Using this function, we can calculate nucleotide diversity faster in some circumstances.

```
[nh,ni,sh]=counthaplotype(seq);
x=0;
for i=1:nh-1
for j=i+1:nh
       d=sum(sh(i,:)~=sh(j,:));
       x=x+ni(i)*ni(j)*d;
end
end
theta_pi=x/(n*(n-1)/2);
```

If the sequences are L bases long, it is often useful to normalize θ_S and θ_π by diving them by L. If the genotypic data (`geno`) is given, the corresponding θ_S and θ_π can be calculated as follows:

```
n=2*size(geno,1);   % n is the sample size (number of chromosomes).
p=snp_maf(geno);    % p is a vector containing MAF of SNPs.
S=numel(p);
theta_w=S/sum(1./(1:n-1));
theta_pi=(n/(n-1))*sum(2.*p.*(1-p));
```

Haplotype diversity (or heterozygosity), H, is the probability that two random haplotypes are different. The straightforward approach to calculate H is to examine all pairs and count the fraction of the pairs in which the two haplotypes differ from each other. The faster approach starts by counting the number of copies of each haplotype, n_i. Then the haplotype diversity is estimated by

$$H = \frac{1 - \sum_i \left(\frac{n_i}{n_{hap}} \right)^2}{1 - 1/n_{hap}}.$$

Using the function `counthaplotype`, we can get the number of copies of each haplotype and then compute H as follows:

```
[nh,ni]=counthaplotype(hap);
h=(1-sum((ni./nh).^2))./(1-1./nh);
```

Site frequency spectrum (SFS) is a histogram whose *i*th entry is the number of polymorphic sites at which the mutant allele is present in *i* copies within the sample. Here, *i* ranges from 1 to *n*-1. When it is impossible to tell which allele is the mutant and which is the ancestral one, we combine the entries for *i* and *n-i* to make a folded SFS. *Mismatch distribution* is a histogram whose *i*th entry is the number of pairs of sequences that differ by *i* sites. Here, *i* ranges from 0 through the maximal difference between pairs in the sample. Two functions in PGEToolbox, `sfs` and `mismch`, can be used to calculate SFS and mismatch distribution, respectively.

4.2 Neutrality tests

The standard models of population genetics, such as the Wright–Fisher model and related ones, constitute null models. Population geneticists have used these models to develop theory, and then applied the theory to test the goodness-of-fit of the standard model on a given data set. Using summary statistics, they can reject the standard model and take into account other factors, such as selection or demographic history, to build alternative hypotheses. These tests that compute the goodness-of-fit of the standard model have been referred to as "neutrality tests", and have been widely used to detect genes, or genomic regions targeted by natural selection. An important family of neutrality tests is based on summary statistics derived from the SFS. The classical tests in this family include Tajima's *D* test (Tajima 1989), Fu and Li's tests (Fu and Li 1993), and Fay and Wu's *H* test (Fay and Wu 2000), which have been widely used to detect signatures of positive selection on genetic variation in a population.

Under evolution by genetic drift (i.e., neutral evolution), different estimators of θ, such as, θ_W and θ_π, are unbiased estimators of the true value of θ: $E(\hat{\theta}_W) = E(\hat{\theta}_\pi) = \theta$. Therefore, the difference between θ_W and θ_π can be used to infer non-neutral evolution. Using this assumption, Tajima's *D* test examines the deviation from neutral expectation (Tajima 1989). The statistic *D* is defined by the equation:

$$D = (\theta_\pi - \theta_W) \Big/ \sqrt{V(\theta_\pi - \theta_W)} \,,$$

where $V(d)$ is an estimator of the variance of d. The value of *D* is 0 for selectively neutral mutations in a constant population infinite sites model. A negative value of *D* indicates either purifying selection or population expansion (Tajima 1989).

```
% n is the sample size; S is the number of segregating sites
% theta_w and theta_pi have been calculated

nx=1:(n-1);
a1=sum(1./nx);
a2=sum(1./nx.^2);
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/(a1^2);
e1=c1/a1;
e2=c2/(a1^2+a2);
tajima_d=(theta_pi-theta_w)/sqrt(e1*S+e2*S*(S-1));
```

The other SFS-based neutrality tests, like Fu and Li's tests (Fu and Li 1993) and Fay and Wu's *H* test (Fay and Wu 2000), share a common structure with Tajima's *D* test. Many other neutrality tests exhibit important diversity. For example, R2 tests try to capture specific tree deformations (Ramos-Onsins and Rozas 2002), and the haplotype tests use the distribution of haplotypes (Fu 1997; Depaulis and Veuille 1998).

4.3 Long-range haplotype tests

When a beneficial mutation arises and rapidly increases in frequency in the process leading to fixation, chromosomes harbouring the beneficial mutation experience less recombination events. This results in conservation of the original haplotype. Several so called long-range haplotype (LRH) tests have been developed to detect long haplotypes at unusually high frequencies in genomic regions, which have undergone recent positive selection.

The test based on the extended haplotype homozygosity (EHH) developed by Sabeti et al. (2002) is one of the earliest LRH tests. EHH is defined as the probability that two randomly chosen chromosomes carrying an allele (or a haplotype) at the core marker (or region) are identical at all the markers in the extended region. EHH between two markers, *s* and *t*, is defined as the probability that two randomly chosen chromosomes are homozygous at all markers between *s* and *t*, inclusively. Explicitly, if *N* chromosomes in a sample form *G* homozygous groups, with each group *i* having n_i elements, EHH is defined as:

$$EHH = \frac{\sum_{i=1}^{G} \binom{n_i}{2}}{\binom{N}{2}} \quad .$$

Equivalently, EHH can be calculated in a convenient form as the statistic *haplotype homozygosity*:

$$HH = (\sum p_i^2 - 1/n) \big/ (1 - 1/n) \, ,$$

where p_i is the frequency of haplotype *i* and *n* is the sample size. For a core marker, EHH is calculated as HH in a stepwise manner. The EHH is computed with respect to a distinct allele of a core maker or a distinct formation of a core region. In Fig. 6, for example, we focus on allele A of the core maker (a diallelic SNP) at the position `x`. Variable hap contains A-carrying haplotypes of size `n×m`.

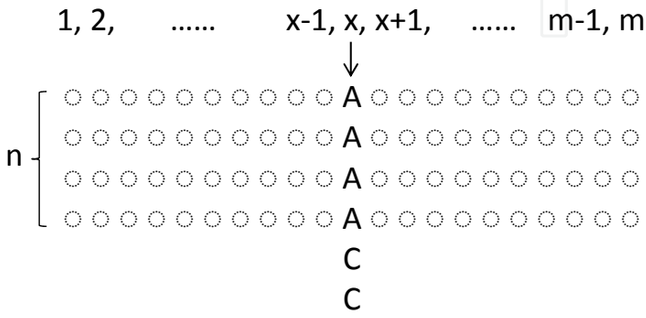


Fig. 6. Calculation of EHH for *n* haplotypes carrying allele A at the focal position *x*.

The EHH values, `ehh1`, around *x* in respect to the allele A, can be computed as follows:

```
ehh1=ones(1,m);
for i=1:x-1
    [n,ni]=counthaplotype(hap(:, i:x-1));
    p=ni./n;
    ehh1(i)=(sum(p.^2)-1/n)/(1-1/n);
end
for j=x+1:m
    [n,ni]=counthaplotype(hap(:, x+1:end));
    p=ni./n;
    ehh1(j)=(sum(p.^2)-1/n)/(1-1/n);
end
```

Similarly, the EHH around `x` with respect to the allele C, `ehh2`, can be computed using the same machinery. Both `ehh1` and `ehh2` are calculated for all markers around the core maker. Fig. 7 shows the EHH curves for two alleles C and T in the core SNP. The EHH values for the markers decrease as the distance from the core marker increases.

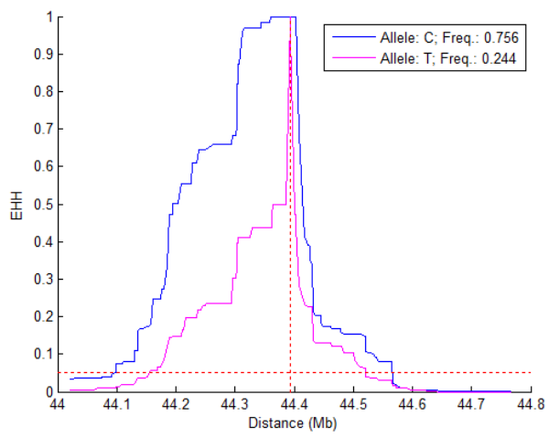


Fig. 7. EHH decay as a function of the distance between a test marker and the core marker. Vertical dash line indicates the location of the core marker. Horizontal dash line indicates the cut-off=0.05 for computing EHH integral.

The integrated EHH (iHH) is the integral of the observed decay of EHH away from the core marker. iHH is obtained by integrating the area under the EHH decay curve until EHH reaches a small value (such as 0.05). Once we obtain `ehh1` and `ehh2` values for the two alleles, we can integrate EHH values with respect to the genetic or physical distance between the core marker and other markers, with the result defined as iHH_1 and iHH_2. The statistic $ln(iHH_1/iHH_2)$ is called the integrated haplotype score (iHS), which is a measure of the amount of EHH at a given maker along one allele relative to the other allele. The iHS can be standardized (mean 0, variance 1) empirically to the distribution of the observed iHS scores over a range of SNPs with similar allele frequencies. The measure has been used to detect partial selective sweeps in human populations (Voight, Kudaravalli et al. 2006).

In Matlab, we invoke the function `trapz(pos,ehh)` to compute the integral of EHH, `ehh`, with respect to markers' position, `pos`, using trapezoidal integration. The position is in units of either physical distance (Mb) or genetic distance (cM). The unstandardized integrated haplotype score (iHS) can be computed as the log ratio between the two iHHs:

```
ihh1=trapz(pos,ehh1);
ihh2=trapz(pos,ehh2);
ihs=log(ihh1/ihh2);
```

The cross population EHH (XP-EHH) has been used to detect selected alleles that have risen to near fixation in one but not all populations (Sabeti, Varilly et al. 2007). The statistic XP-EHH uses the same formula as iHS, that is, $ln(iHH_1/iHH_2)$. The difference is that iHH_1 and iHH_2 are computed for the same allele in two different populations. An unusually positive value suggests positive selection in population 1, while a negative value suggests the positive selection in population 2.

4.4 Population differentiation

Genomic regions that show extraordinary levels of genetic population differentiation may be driven by selection (Lewontin 1974). When a genomic region shows unusually high or low levels of genetic population differentiation compared with other regions, this may then be interpreted as evidence for positive selection (Lewontin and Krakauer 1973; Akey, Zhang et al. 2002). The level of genetic differentiation is quantified with F_{ST}, which was introduced by Wright (Wright 1931) measuring the effect of structure on the genetics of a population. There are several definitions of F_{ST} in the literature; the simple concept is $F_{ST} = (H_T - H_S)/H_T$, where H_T is the heterozygosity of the total population and H_S is the average heterozygosity across subpopulations.

Suppose you know the frequencies, `p1` and `p2`, of an allele in two populations. The sample sizes in two populations are `n1` and `n2`. Wright's F_{ST} can be computed as follows:

```
pv=[p1 p2];
nv=[n1 n2];
x=(nv.*(nv-1)/2);
Hs=sum(x.*2.*(nv./(nv-1)).*pv.*(1-pv))./sum(x);
Ht=sum(2.*(n./(n-1)).*p_hat.*(1-p_hat));
Fst=1-Hs./Ht;
```

Below is a function that calculates an unbiased estimator of F_{ST}, which corrects for the error associated with incomplete sampling of a population (Weir and Cockerham 1984; Weir 1996).

```
function [f]=fst_weir(n1,n2,p1,p2)
n=n1+n2;
nc=(1/(s-1))*((n1+n2)-(n1.^2+n2.^2)./(n1+n2));
p_hat=(n1./n).*p1+(n2./n).*p2;
s=2; % number of subpopulations
MSP=(1/(s-1))*((n1.*(p1-p_hat).^2 + n2.*(p2-p_hat).^2));
MSG=(1./sum([n1-1, n2-1])).*(n1.*p1.*(1-p1)+n2.*p2.*(1-p2));
Fst=(MSP-MSG)./(MSP+(nc-1).*MSG);
```

`NC` is the variance-corrected average sample size, `p_hat` is the weighted average allele frequency across subpopulations, `MSG` is the mean square error within populations, and `MSP` is the mean square error between populations.

5. Conclusion

Matlab, as a powerful scientific computing environment, should have many potential applications in evolutionary bioinformatics. An important goal of evolutionary bioinformatics is to understand how natural selection shapes patterns of genetic variation within and between species. Recent technology advances have transformed molecular evolution and population genetics into more data-driven disciplines. While the biological data sets are becoming increasingly large and complex, we hope that the programming undertakings that are necessary to deal with these data sets remain manageable. A high-level programming language like Matlab guarantees that the code complexity only increases linearly with the complexity of the problem that is being solved.

Matlab is an ideal language to develop novel software packages that are of immediate interest to quantitative researchers in evolutionary bioinformatics. Such a software system is needed to provide accurate and efficient statistical analyses with a higher degree of usability, which is more difficult to achieve using traditional programming languages. Limited functionality and inflexible architecture of existing software packages and applications often hinder their usability and extendibility. Matlab can facilitate the design and implementation of novel software systems, capable of conquering many limitations of the conventional ones, supporting new data types and large volumes of data from population-scale sequencing studies in the genomic era.

6. Acknowledgment

The work was partially supported by a grant from the Gray Lady Foundation. I thank Tomasz Koralewski and Amanda Hulse for their help in the manuscript preparation. MBEToolbox and PGEToolbox are available at http://www.bioinformatics.org/mbetoolbox/ and http://www.bioinformatics.org/pgetoolbox/, respectively.

7. References

Akey, J. M., G. Zhang, et al. (2002). Interrogating a high-density SNP map for signatures of natural selection. *Genome Res* 12(12): 1805-1814.

Cai, J. J. (2008). PGEToolbox: A Matlab toolbox for population genetics and evolution. *J Hered* 99(4): 438-440.

Cai, J. J., D. K. Smith, et al. (2005). MBEToolbox: a MATLAB toolbox for sequence data analysis in molecular biology and evolution. *BMC Bioinformatics* 6: 64.

Cai, J. J., D. K. Smith, et al. (2006). MBEToolbox 2.0: an enhanced version of a MATLAB toolbox for molecular biology and evolution. *Evol Bioinform Online* 2: 179-182.

Depaulis, F. & M. Veuille (1998). Neutrality tests based on the distribution of haplotypes under an infinite-site model. *Mol Biol Evol* 15(12): 1788-1790.

Fay, J. C. & C. I. Wu (2000). Hitchhiking under positive Darwinian selection. *Genetics* 155(3): 1405-1413.

Felsenstein, J. (1984). Distance Methods for Inferring Phylogenies: A Justification. *Evolution* 38(1): 16-24.

Fu, Y. X. (1997). Statistical tests of neutrality of mutations against population growth, hitchhiking and background selection. *Genetics* 147(2): 915-925.

Fu, Y. X. & W. H. Li (1993). Statistical tests of neutrality of mutations. *Genetics* 133(3): 693-709.

Goldman, N. & Z. Yang (1994). A codon-based model of nucleotide substitution for protein-coding DNA sequences. *Mol Biol Evol* 11(5): 725-736.

Gu, X. & J. Zhang (1997). A simple method for estimating the parameter of substitution rate variation among sites. *Mol Biol Evol* 14(11): 1106-1113.

Hasegawa, M., H. Kishino, et al. (1985). Dating of the human-ape splitting by a molecular clock of mitochondrial DNA. *J Mol Evol* 22(2): 160-174.

Jones, D. T., W. R. Taylor, et al. (1992). The rapid generation of mutation data matrices from protein sequences. *Comput Appl Biosci* 8(3): 275-282.

Jukes, T. H. & C. Cantor (1969). Evolution of protein molecules. *Mammalian Protein Metabolism*. H. N. Munro. New York, Academic Press: 21-132.

Kimura, M. (1980). A simple method for estimating evolutionary rates of base substitutions through comparative studies of nucleotide sequences. *J Mol Evol* 16(2): 111-120.

Lewontin, R. C. (1974). *The genetic basis of evolutionary change*. New York,, Columbia University Press.

Lewontin, R. C. & J. Krakauer (1973). Distribution of gene frequency as a test of the theory of the selective neutrality of polymorphisms. *Genetics* 74(1): 175-195.

Morozov, P., T. Sitnikova, et al. (2000). A new method for characterizing replacement rate variation in molecular sequences. Application of the Fourier and wavelet models to Drosophila and mammalian proteins. *Genetics* 154(1): 381-395.

Nei, M. & T. Gojobori (1986). Simple methods for estimating the numbers of synonymous and nonsynonymous nucleotide substitutions. *Mol Biol Evol* 3(5): 418-426.

Nei, M. & S. Kumar (2000). *Molecular evolution and phylogenetics*. Oxford ; New York, Oxford University Press.

Ramos-Onsins, S. E. & J. Rozas (2002). Statistical properties of new neutrality tests against population growth. *Mol Biol Evol* 19(12): 2092-2100.

Rodriguez, F., J. L. Oliver, et al. (1990). The general stochastic model of nucleotide substitution. *J Theor Biol* 142(4): 485-501.

Sabeti, P. C., D. E. Reich, et al. (2002). Detecting recent positive selection in the human genome from haplotype structure. *Nature* 419(6909): 832-837.

Sabeti, P. C., P. Varilly, et al. (2007). Genome-wide detection and characterization of positive selection in human populations. *Nature* 449(7164): 913-918.

Tajima, F. (1989). Statistical method for testing the neutral mutation hypothesis by DNA polymorphism. *Genetics* 123(3): 585-595.

Voight, B. F., S. Kudaravalli, et al. (2006). A map of recent positive selection in the human genome. *PLoS Biol* 4(3): e72.

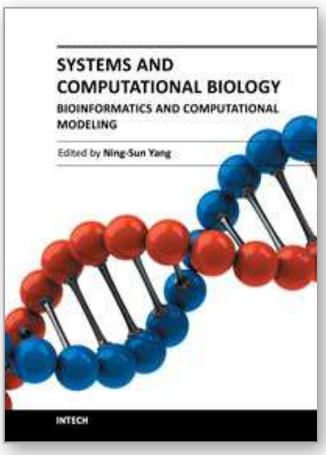
Weir, B. S. (1996). *Genetic data analysis II : methods for discrete population genetic data*. Sunderland, Mass., Sinauer Associates.

Weir, B. S. & C. C. Cockerham (1984). Estimating F-statistics for the analysis of population structure. *Evolution* 38: 1358-1370.

Whelan, S. (2008). Spatial and temporal heterogeneity in nucleotide sequence evolution. *Mol Biol Evol* 25(8): 1683-1694.

Wright, S. (1931). The genetical structure of populations. *ann Eugenics* 15: 323-354.

Yang, Z. (1994). Maximum likelihood phylogenetic estimation from DNA sequences with variable rates over sites: approximate methods. *J Mol Evol* 39(3): 306-314.



Systems and Computational Biology - Bioinformatics and Computational Modeling

Edited by Prof. Ning-Sun Yang

ISBN 978-953-307-875-5
Hard cover, 334 pages
Publisher InTech
Published online 12, September, 2011
Published in print edition September, 2011

Whereas some â€œmicroarrayâ€ or â€œbioinformaticsâ€ scientists among us may have been criticized as doing â€œcataloging researchâ€, the majority of us believe that we are sincerely exploring new scientific and technological systems to benefit human health, human food and animal feed production, and environmental protections. Indeed, we are humbled by the complexity, extent and beauty of cross-talks in various biological systems; on the other hand, we are becoming more educated and are able to start addressing honestly and skillfully the various important issues concerning translational medicine, global agriculture, and the environment. The two volumes of this book present a series of high-quality research or review articles in a timely fashion to this emerging research field of our scientific community.

How to reference

In order to correctly reference this scholarly work, feel free to copy and paste the following:

James J. Cai (2011). Evolutionary Bioinformatics with a Scientific Computing Environment, Systems and Computational Biology - Bioinformatics and Computational Modeling, Prof. Ning-Sun Yang (Ed.), ISBN: 978-953-307-875-5, InTech, Available from: http://www.intechopen.com/books/systems-and-computational-biology-bioinformatics-and-computational-modeling/evolutionary-bioinformatics-with-a-scientific-computing-environment

INTECH
open science | open minds

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821

© 2011 The Author(s). Licensee IntechOpen. This chapter is distributed under the terms of the Creative Commons Attribution-NonCommercial-ShareAlike-3.0 License, which permits use, distribution and reproduction for non-commercial purposes, provided the original is properly cited and derivative works building on this content are distributed under the same license.

IntechOpen

IntechOpen