

Code for hidden Markov model and simulated phenotypes

Hidden Markov Model for pA RILs

This program is written in perl. It utilizes the SNP tables for the founders and the RILs available for download at: <http://flyrils.org/Data>.

The results are also available at: <http://flyrils.org/Data>.

```

use warnings;
use strict;
use Math::Cephes qw (:gammas);
use Math::Cephes qw (:betas);
use DBI;

my $dsn = "DBI:mysql:database";
my $user_name = "username";
my $password = "password";

my $dbh = DBI->connect ($dsn, $user_name, $password,
                        { RaiseError => 1, PrintError => 0 } );

#####
#####
#####

# Get array of founder ids
my @founders = ('A1', 'A2', 'A3', 'A4', 'A5', 'A6', 'A7', 'A8');
my %foundkey = (
    1 => 'A1',
    3 => 'A2',
    4 => 'A3',
    6 => 'A4',
    9 => 'A5',
    10 => 'A6',
    14 => 'A7',
    15 => 'A8',
);

# Get number of founders
my $n_founders = scalar(@founders);

# Make states hash with homozygous and heterozygous states and founders array (0,1)
my %statecodes=();
for (my $i=0; $i<$n_founders; ++$i){
    $statecodes{$founders[$i].$founders[$i]}=[ $founders[$i], $founders[$i] ];
    for (my $j=0; $j < $n_founders; ++$j){
        if ($i < $j){
            $statecodes{$founders[$i].$founders[$j]}=[ $founders[$i], $founders[$j] ];
        }
    }
}
}

```

```

my @statecodes = sort keys(%statecodes);

#####
#####
#####

# Set up initiation probability hash
my %initiation = ();

# Number of heterozygous states
my $n_hets=0;
for (my $i=$n_founders-1; $i > 0; --$i){
    $n_hets = $n_hets+$i;
}

# Set parameters needed
# Frequency of homozygous states
my $freq_homo = 0.95;

# Frequency of heterozygous states
my $freq_hets = 1 - $freq_homo;

# Fill initiation hash with initiation probabilities
foreach my $state (@statecodes){
    if ($statecodes{$state}->[0] eq $statecodes{$state}->[1]){
        $initiation{$state} = eln((1/$n_founders) * $freq_homo);
    }else{
        $initiation{$state} = eln((1/$n_hets) * $freq_hets);
    }
}

#####
#####
#####

# TRANSITION PROBABILITIES

open (RF, './RF.txt') || die ("Can't open RF.txt: $!");

# Put arm, position (in 10*MB), and RF (in M/bp/50gens)
my %recombdatt = ();

# Populate %recombdatt using reads from the table
foreach my $line (<RF>){
    my($blank1,$blank2,$chr,$blank3,$cm,$blank4,$RF) = split /"/,$line;
    if ($chr eq 'X') {
        $recombdatt{$chr}{$cm*10} = 1E-8 * $RF * 33;
    } else {
        $recombdatt{$chr}{$cm*10} = 1E-8 * $RF * 25;
    }
}

close RF;

# Set parameters for transition probabilities
my $prob_homo = 0.95;
my $prob_transhomo = 0.85;

```

```

# Genotyping error rate
my $err = 0.005;

#####
#####
#####

my %positions = ();
my %founders = ();
my %RILS = ();
my %prob_state = ();
my %positioncode = ();
my $totalcount = 0;
my $alpha = 0.5;
my $beta = 0.5;

# Choose RIL
my @RILnames = ();
my $list = $dbh->prepare ("SELECT DISTINCT A.genotype FROM RILcountNEW A, RILID B
WHERE B.pop='A' AND B.library='sjmRIL05' AND B.ril=A.genotype");
$list->execute();
while (my @ary = $list->fetchrow_array ()) {
    push(@RILnames, $ary[0]);
}
$list->finish ();

my $trans_scale = 0.1;

my @chromosomes = (shift @ARGV);
my $chrprint = $chromosomes[0];

foreach my $arm (@chromosomes){
    #####
    open(PROBFILE, ">A_run10LIB5_state_probabilities_{$chrprint}.csv");

    #print header
    print PROBFILE "rilID", ",", "index", ",", "chr", ",", "pos", ",", " ";
    foreach my $state (@statecodes){
        print PROBFILE "{$state}", ",", " ";
    }
    foreach my $founder (@founders){
        print PROBFILE "A{$founder}", ",", " ";
    }
    print PROBFILE "\n";

    # Get founder info
    # Put in a hash with founder id, physical position (in 10*MB), allele, count

    %founders = ();

    if ($arm eq '3R'){
        my $fdat = $dbh->prepare ("SELECT A.pos, A.foundernum, A.minor_count,
A.major_count FROM SNPcount A, RILposNEW B WHERE
        A.chr=B.chr AND A.pos=B.pos AND A.chr = '{$arm}' AND A.foundernum
        IN(1,18,4,6,9,10,14,15)");
    }
}

```

```

$fdat->execute();
while (my @ary = $fdat->fetchrow_array ()) {
    if ($ary[1] == 18) {
        $founders{$foundkey{3}}{$ary[0]/100000}{'A'}=$ary[2];
        $founders{$foundkey{3}}{$ary[0]/100000}{'B'}=$ary[3];
    } else {
        $founders{$foundkey{$ary[1]}}{$ary[0]/100000}{'A'}=$ary[2];
        $founders{$foundkey{$ary[1]}}{$ary[0]/100000}{'B'}=$ary[3];
    }
}
$fdat->finish ();

} else {
    my $fdat = $dbh->prepare ("SELECT A.pos, A.foundernum, A.minor_count,
A.major_count FROM SNPcount A, RILposNEW B WHERE
    A.chr=B.chr AND A.pos=B.pos AND A.chr = '$arm' AND A.foundernum
IN(1,3,4,6,9,10,14,15)");
    $fdat->execute();
    while (my @ary = $fdat->fetchrow_array ()) {
        $founders{$foundkey{$ary[1]}}{$ary[0]/100000}{'A'}=$ary[2];
        $founders{$foundkey{$ary[1]}}{$ary[0]/100000}{'B'}=$ary[3];
    }
    $fdat->finish ();
}

#####
# Make index and position hash for arm
# Positions are stored as 10 x MB

%positions = ();

my $poscount = 0;

my $sth = $dbh->prepare ("SELECT pos FROM RILposNEW WHERE chr= '$arm'");
$sth->execute();
while (my @ary = $sth->fetchrow_array ()) {
    $positions{$poscount}=$ary[0]/100000;
    $poscount = $poscount+1;

    my @set = ($arm, $ary[0]);
    $positioncode{$totalcount}=\@set;
    $totalcount = $totalcount+1
}
$sth->finish ();

my $seq_length = $poscount;

#####
#####
# Get RILS
# HASH WITH POSITION (IN 10*MB), ALLELE, COUNT

#####
# Loop through RILS
foreach my $test_ril (@RILnames) {
    %RILS = ();
    my $rdat = $dbh->prepare ("SELECT A.pos, A.minor_allele_count,
A.major_allele_count FROM RILcountNEW A, RILposNEW B WHERE

```

```

A.chr=B.chr AND A.pos=B.pos AND A.chr = '$arm' AND A.genotype = '$test_ril');
$rdat->execute();
while (my @ary = $rdat->fetchrow_array ()) {
    $RILS{$ary[0]/100000}{'A'}=$ary[1];
    $RILS{$ary[0]/100000}{'B'}=$ary[2];
}
$rdat->finish ();

#####
# HMM

#####
#forward equations (alphas)
my %alphas = ();

# For each position ($t) get alpha: count forward
for (my $t=0; $t<=$seq_length-1; ++$t){
    # for first position
    if ($t == 0){
        my %emission = emission($t);
        foreach my $state (@statecodes){
            $alphas{$t}{$state} = elnproduct($initiation{$state}, $emission{$state});
        }
    }else{
        # Get transition probabilities for position and previous position
        my %transitions = trans($t-1,$t,$arm);

        # Get emission probabilities for position
        my %emission = emission($t);

        foreach my $state1 (@statecodes){
            my $sum = 'logzero';

            # Sum over founders
            foreach my $state2 (@statecodes){
                # The alpha is already eln from above etc.
                my $a1_trans = elnproduct($alphas{$t-1}{$state2},
$transitions{$state1}{$state2});
                $sum = elnsum($sum, $a1_trans);
            }
            $alphas{$t}{$state1}=elnproduct($sum, $emission{$state1});
        }
    }
}

#####
# Backward equations (betas)
my %betas = ();

# For each position ($t) get beta: count backwards
for (my $t=$seq_length-1; $t>=0; --$t){
    #for last position
    if ($t == $seq_length-1){
        foreach my $state (@statecodes){
            $betas{$t}{$state} = eln(1);
        }
    }else{

```

```

# Get transition for position and position ahead
my %transitions = trans($t,$t+1,$arm);
# Get emission for position ahead
my %emission = emission($t+1);
foreach my $stater1 (@statecodes){
  my $sum = 'logzero';
  # Sum over founders
  foreach my $state2 (@statecodes){
    my $b_tm1 = elnproduct(elnproduct($betas{$t+1}{$state2},
$transitions{$stater1}{$state2}), $emission{$state2});
    $sum = elnsum($sum, $b_tm1);
  }
  $betas{$t}{$stater1}=$sum;
}
}

# Hash to store probability of state for each position
# Get Y for each position
for (my $t=0; $t<=$seq_length-1; ++$t){
  foreach my $stater1 (@statecodes){
    my $sum = 'logzero';
    foreach my $state2 (@statecodes){
      my $value = elnproduct($alphas{$t}{$state2},$betas{$t}{$state2});
      $sum = elnsum($sum, $value);
    }
    # Get Y out of log space
    $prob_state{$ttest_ril}{$arm}{$positions{$t}*100000}{$stater1} =
eexp(elnproduct(elnproduct($alphas{$t}{$stater1},$betas{$t}{$stater1}), -$sum));
  }
}
for (my $j=0; $j<$totalcount; ++$j){
  my @dat = @{$positioncode{$j}};
  print PROBFIL " $ttest_ril", ",", "$j", ",", "$dat[0]", ",", "$dat[1]", ",", " ";
  foreach my $state (@statecodes){
    print PROBFIL $prob_state{$ttest_ril}{$dat[0]}{$dat[1]}{$state}, ",";
  }
  foreach my $founder (@founders){
    my $addprob = additive($dat[1],$founder,$ttest_ril,$dat[0]);
    print PROBFIL "$addprob", ",";
  }
  print PROBFIL "\n";
}
} #close foreach for rils
} #close foreach for arms

close(PROBFIL);

```

```

#####
#####
#####
# Subroutines

```

```

#####
# Get additive model
sub additive {
  my ($pos, $founder, $ril, $arm) = @_;

```

```

my $addsum=0;
foreach my $state (@statecodes){
  my $found1 = $statecodes{$state}->[0];
  my $found2 = $statecodes{$state}->[1];
  if ($found1 eq $founder && $found2 eq $founder){
    $addsum=$addsum+$prob_state{$ril}{$arm}{$pos}{$state};
  }
  elsif (($found1 eq $founder) || ($found2 eq $founder)){
    $addsum=$addsum+(0.5*$prob_state{$ril}{$arm}{$pos}{$state});
  }
}
return $addsum;
}

# Transition probabilities
sub trans {
  my ($pos1, $pos2, $arm) = @_ ;
  my %transitions = ();
  my $trans_prob = recomb_prob($pos1,$pos2,$arm);
  foreach my $state (@statecodes){
    foreach my $statal (@statecodes){
      if ($state eq $statal){
        # Probability of staying in state
        if ($statecodes{$state}->[0] eq $statecodes{$state}->[1]){
          $transitions{$state}{$statal} = eln((1 - $trans_prob) +
(($prob_homo*$trans_prob)/$n_founders));
        }else{
          $transitions{$state}{$statal} = eln((1 - $trans_prob) + 2 * (1-
$prob_transhomo) * $trans_prob * (1/(2*($n_founders-1))));
        }
      }else{
        # If original is homozygous:
        if ($statecodes{$state}->[0] eq $statecodes{$state}->[1]){
          # And if new is homozygous
          if ($statecodes{$statal}->[0] eq $statecodes{$statal}->[1]){
            # Probability of moving to another homozygous state
            $transitions{$state}{$statal} =
eln($trans_prob*$prob_homo*(1/($n_founders)));
          }else{
            # Original homozygous, new is heterozygous with one copy same as original
            if (($statecodes{$state}->[0] eq $statecodes{$statal}->[0]) or
($statecodes{$state}->[0] eq $statecodes{$statal}->[1])){
              # Probability of comment above
              $transitions{$state}{$statal} = eln($trans_prob*(1-
$prob_homo)*(1/($n_founders-1)));
            }else{
              # All other moves from homozygous state involve 2 events, prob near 0
              $transitions{$state}{$statal} = eln(0);
            }
          }
        }else{
          # If original heterozygous, new is one of homo states (only btw two
          represented in het)
          if (($statal eq $statecodes{$state}->[0].$statecodes{$state}->[0]) or
($statal eq $statecodes{$state}->[1].$statecodes{$state}->[1])){
            $transitions{$state}{$statal} = eln($trans_prob * $prob_transhomo * 0.5);
          }
        }
      }
    }
  }
}

```

```

    }else{
        # Original heterozygous, new is het with one founder same as original het
        if (($statecodes{$state1}->[0] eq $statecodes{$state}->[0]) or
($statecodes{$state1}->[0] eq $statecodes{$state}->[1]) or ($statecodes{$state1}-
>[1] eq $statecodes{$state}->[0]) or ($statecodes{$state1}->[1] eq
$statecodes{$state}->[1])){
            $transitions{$state}{$state1} = eln($trans_prob * (1-$prob_transhomo) *
(1/(2*($n_founders-1))));
        }else{
            # All other moves from het involve two events, prob near zero
            $transitions{$state}{$state1} = eln(0);
        }
    }
}
}
}
}
return %transitions;
}
}

```

```

sub recomb_prob {
    #pass positions in Mb and arm
    my ($pos1, $pos2, $arm) = @_ ;

    #get distance between markers in bp
    my $dist = $positions{$pos2}*1E5 - $positions{$pos1}*1E5;

    #get midpoint between markers in 10*MB for lookup
    my $midpt = $positions{$pos1} + (($positions{$pos2} - $positions{$pos1})/2);

    #find positions for table lookup--uses RF TABLE
    my $LB = int($midpt);
    my $UB = $LB + 1;

    #check to see if midpt hits exact point
    #below is reg expression to test if midpoint is an integer
    if ($midpt =~ /\^d+$/) {
        #if is at exact integer, just get from table
        my $rf = $recombdatt{$arm}{$midpt};
        #and multiply by distance
        my $trans_prob = (1-exp -($rf*$dist*$trans_scale));
        return $trans_prob;
    }else{
        #if not, calculate linear function slope and intercept and extrapolate
        my $slope = ($recombdatt{$arm}{$UB} - $recombdatt{$arm}{$LB})/($UB-$LB);
        my $intercept = -$slope*$UB+$recombdatt{$arm}{$UB};
        #get value for midpoint
        my $rf = $slope*$midpt + $intercept;
        #and multiply by distance
        my $trans_prob = (1-exp -($rf*$dist*$trans_scale));
        return $trans_prob;
    }
}
}

```

```

#####
# Emission probabilities
sub emission {
    #pass index

```



```

my ($pos) = @_;
my %emission;

if (exists $RILS{$positions{$pos}}){
  #get observed counts (k,N)
  my $O_n1 = $RILS{$positions{$pos}}{'A'}; #k
  my $O_n2 = $RILS{$positions{$pos}}{'B'};
  my $O_N = $O_n1 + $O_n2; #N
  #put in if for 0,0
  if ($O_N>0){
    foreach my $state (@statecodes){

      #get founders for each state (homo or het)
      my $found1 = $statecodes{$state}->[0];
      my $found2 = $statecodes{$state}->[1];
      my $found1_n =
      $founders{$found1}{$positions{$pos}}{'A'}+$founders{$found1}{$positions{$pos}}{'B'};
      my $found2_n =
      $founders{$found2}{$positions{$pos}}{'A'}+$founders{$found2}{$positions{$pos}}{'B'};

      #get probability of obs "A" for each founder
      my $p1 =
      $founders{$found1}{$positions{$pos}}{'A'}/($founders{$found1}{$positions{$pos}}{'A'}
      +$founders{$found1}{$positions{$pos}}{'B'});
      my $p2 =
      $founders{$found2}{$positions{$pos}}{'A'}/($founders{$found2}{$positions{$pos}}{'A'}
      +$founders{$found2}{$positions{$pos}}{'B'});

      if ($p1 >0.995) {$p1=0.995;}
      if ($p1 <0.005) {$p1=0.005;}
      if ($p2 >0.995) {$p2=0.995;}
      if ($p2 <0.005) {$p2=0.005;}

      #get probability of obs k "A"'s in N observations given state.
      #sum over possible genotypes of ril
      my $prob_sum = eln(($p1 * $p2 * (1-$err)**$O_n1 * $err**($O_N-$O_n1))+
      ((1-$p1) * $p2 * (Math::Cephes::beta($alpha+$O_n1, $O_N+$beta-
      $O_n1)/Math::Cephes::beta($alpha, $beta)))+
      ($p1 * (1-$p2) * (Math::Cephes::beta($alpha+$O_n1, $O_N+$beta-
      $O_n1)/Math::Cephes::beta($alpha, $beta)))+
      ((1-$p1) * (1-$p2) * $err**$O_n1 * (1-$err)**($O_N-$O_n1)));
      #get binomial coefficient
      my $binom_coef = Math::Cephes::lgam($O_N + 1) - Math::Cephes::lgam($O_n1+1) -
      Math::Cephes::lgam($O_N-$O_n1+1);
      #calculate probability
      if ($prob_sum eq 'logzero'){
        $emission{$state} = 'logzero';
      }else{
        $emission{$state} = $binom_coef + $prob_sum;
      }
    }#foreach close
  }else{
    foreach my $state (@statecodes){
      $emission{$state} = eln(1);
    }
  }
}
}
}

```

```

    foreach my $state (@statecodes){
        $emission{$state} = eln(1);
    }
}#ifndef close
return %emission;
} #sub close

#####
#FUNCTIONS FOR LOG SPACE#

# Extended exp
sub eexp{
    # $X is a ln prob
    my ($X) = @_ ;
    if ($X eq 'logzero'){
        return 0;
    }else{
        return exp $X;
    }
}

# Extended ln
sub eln{
    # $X is raw prob
    my ($X) = @_ ;
    if ($X eq 0){
        return 'logzero';
    }else{
        return log $X;
    }
}

# Extended ln sum
sub elnsum{
    # $X, $Y are ln probs
    my ($X, $Y) = @_ ;
    if (($X eq 'logzero') or ($Y eq 'logzero')){
        if ($X eq 'logzero'){
            return $Y;
        }else{
            return $X;
        }
    }else{
        if ($X > $Y){
            return $X + eln(1 + exp($Y-$X));
        }else{
            return $Y + eln(1 + exp($X-$Y));
        }
    }
}

# Extended product
sub elnproduct{
    # $X, $Y are ln probs
    my ($X, $Y) = @_ ;
    if (($X eq 'logzero') or ($Y eq 'logzero')){
        return 'logzero';
    }else{

```

```
    return $X + $Y;  
  }  
}
```

Hidden Markov Model for pB RILs

This program is written in perl. It utilizes the SNP tables for the founders and the RILs available for download at: <http://flyrils.org/Data>.

The results are also available at: <http://flyrils.org/Data>.

```
use warnings;
use strict;
use Math::Cephes qw (:gammas);
use Math::Cephes qw (:betas);
use DBI;

my $dsn = "DBI:mysql:database";
my $user_name = "username";
my $password = "password";

my $dbh = DBI->connect ($dsn, $user_name, $password,
                        { RaiseError => 1, PrintError => 0 } );

#####
#####
#####

#get array of founder ids
my @founders = ('B1', 'B2', 'B3', 'B4', 'B5', 'B6', 'B7', 'B8');
my %foundkey = (
    2 => 'B1',
    5 => 'B2',
    7 => 'B3',
    8 => 'B4',
    11 => 'B5',
    12 => 'B6',
    13 => 'B7',
    15 => 'B8',
);

# Get number of founders
my $n_founders = scalar(@founders);

# Make states hash with homozygous and heterozygous states and founders array (0,1)
my %statecodes=();
for (my $i=0; $i<$n_founders; ++$i){
    $statecodes{$founders[$i].$founders[$i]}=[ $founders[$i], $founders[$i] ];
    for (my $j=0; $j < $n_founders; ++$j){
        if ($i < $j){
            $statecodes{$founders[$i].$founders[$j]}=[ $founders[$i], $founders[$j] ];
        }
    }
}

my @statecodes = sort keys(%statecodes);

#####
#####
#####
```

```

# Initiation probabilities

# Set up initiation probability hash
my %initiation = ();

# Number of heterozygous states
my $n_hets=0;
for (my $i=$n_founders-1; $i > 0; --$i){
    $n_hets = $n_hets+$i;
}

# Set parameters needed
# Frequency of homozygous states
my $freq_homo = 0.95;

# Frequency of heterozygous states
my $freq_hets = 1 - $freq_homo;

# Fill initiation hash with initiation probabilities
foreach my $state (@statecodes){
    if ($statecodes{$state}->[0] eq $statecodes{$state}->[1]){
        $initiation{$state} = eln((1/$n_founders) * $freq_homo);
    }else{
        $initiation{$state} = eln((1/$n_hets) * $freq_hets);
    }
}

##### transition probabilities

open (RF, './RF.txt') || die ("Can't open RF.txt: $!");

# Put arm, position (in 10*MB), and RF (in M/bp/50gens)
my %recombdat = ();

# Populate %recombdat using reads from the table
foreach my $line (<RF>){
    my($blank1,$blank2,$chr,$blank3,$cm,$blank4,$RF) = split /"/,$line;
    if ($chr eq 'X') {
        $recombdat{$chr}{$cm*10} = 1E-8 * $RF * 33;
    }else{
        $recombdat{$chr}{$cm*10} = 1E-8 * $RF * 25;
    }
}

close RF;

#Set parameters for transition probabilities
my $prob_homo = 0.95;
my $prob_transhomo = 0.85;

#####
#####
#####
# Emission probabilities

#Genotyping error rate

```

```

my $err = 0.005;

my %positions = ();
my %founders = ();
my %RILS = ();
my %prob_state = ();
my %positioncode = ();
my $totalcount = 0;
my $alpha = 0.5;
my $beta = 0.5;

#Choose RIL
my @RILnames = ();
my $list = $dbh->prepare ("SELECT DISTINCT A.genotype FROM RILcountNEW A, RILID B
WHERE B.pop='B' AND B.ril=A.genotype");
$list->execute();
while (my @ary = $list->fetchrow_array ()) {
    push(@RILnames, $ary[0]);
}
$list->finish ();

my $trans_scale = 0.1;

my @chromosomes = (shift @ARGV);
my $chrprint = $chromosomes[0];

foreach my $arm (@chromosomes){

    open(PROBFILE, ">/B_run9_state_probabilities_{$chrprint}.csv");
    print PROBFILE "rilID", ",", "index", ",", "chr", ",", "pos", ",", " ";
    foreach my $state (@statecodes){
        print PROBFILE "{$state}", ",", " ";
    }
    foreach my $founder (@founders){
        print PROBFILE "B{$founder}", ",", " ";
    }
    print PROBFILE "\n";

    # Get founder info
    # Put in a hash with founder id, physical position (in 10*MB), allele, count
    %founders = ();

    if ($arm eq '2L'){
        my $fdat = $dbh->prepare ("SELECT A.pos, A.foundernum, A.minor_count,
A.major_count FROM SNPcount A, RILposNEW B WHERE
A.chr=B.chr AND A.pos=B.pos AND A.chr = '$arm' AND A.foundernum
IN(2,5,7,8,12,13,15)");
        $fdat->execute();
        while (my @ary = $fdat->fetchrow_array ()) {
            $founders{$foundkey{$ary[1]}}{$ary[0]/100000}{'A'}=$ary[2];
            $founders{$foundkey{$ary[1]}}{$ary[0]/100000}{'B'}=$ary[3];
        }
        $fdat->finish ();

        my $f1ldat = $dbh->prepare ("SELECT A.pos, A.foundernum, A.minor_count,
A.major_count FROM T0_2L A, RILposNEW B WHERE
A.chr=B.chr AND A.pos=B.pos AND A.chr = '$arm'");
        $f1ldat->execute();
    }
}

```

```

while (my @ary = $f11dat->fetchrow_array ()) {
    $founders{$foundkey{$ary[1]}}{$ary[0]/100000}{'A'}=$ary[2];
    $founders{$foundkey{$ary[1]}}{$ary[0]/100000}{'B'}=$ary[3];
}
$f11dat->finish ();
} else {
    my $fdat = $dbh->prepare ("SELECT A.pos, A.foundernum, A.minor_count,
A.major_count FROM SNPcount A, RILposNEW B WHERE
    A.chr=B.chr AND A.pos=B.pos AND A.chr = '$arm' AND A.foundernum
IN(2,5,7,8,11,12,13,15)");
    $fdat->execute();
    while (my @ary = $fdat->fetchrow_array ()) {
        $founders{$foundkey{$ary[1]}}{$ary[0]/100000}{'A'}=$ary[2];
        $founders{$foundkey{$ary[1]}}{$ary[0]/100000}{'B'}=$ary[3];
    }
    $fdat->finish ();
}

#####
# Make index and position hash for arm
# Positions are stored as 10 x MB

%positions = ();

my $poscount = 0;

my $sth = $dbh->prepare ("SELECT pos FROM RILposNEW WHERE chr= '$arm'");
$sth->execute();
while (my @ary = $sth->fetchrow_array ()) {
    $positions{$poscount}=$ary[0]/100000;
    $poscount = $poscount+1;

    my @set = ($arm, $ary[0]);
    $positioncode{$totalcount}=\@set;
    $totalcount = $totalcount+1
}
$sth->finish ();

my $seq_length = $poscount;

#####
#####
# Get RILS
# HASH WITH POSITION (IN 10*MB), ALLELE, COUNT

#####
#Loop through RILS
foreach my $test_ril (@RILnames){
    %prob_state = ();

    %RILS = ();

    my $rdat = $dbh->prepare ("SELECT A.pos, A.minor_allele_count,
A.major_allele_count FROM RILcountNEW A, RILposNEW B WHERE
A.chr=B.chr AND A.pos=B.pos AND A.chr = '$arm' AND A.genotype = '$test_ril'");
    $rdat->execute();
    while (my @ary = $rdat->fetchrow_array ()) {

```

```

    $RILS{$ary[0]/100000}{'A'}=$ary[1];
    $RILS{$ary[0]/100000}{'B'}=$ary[2];
}
$rdat->finish ();

#####
# HMM

#####
# Forward equations (alphas)
my %alphas = ();

# For each position ($t) get alpha: count forward
for (my $t=0; $t<=$seq_length-1; ++$t){
  # For first position
  if ($t == 0){
    my %emission = emission($t);
    foreach my $state (@statecodes){
      $alphas{$t}{$state} = elnproduct($initiation{$state}, $emission{$state});
    }
  }else{
    # get transition probabilities for position and previous position
    my %transitions = trans($t-1,$t,$arm);
    # Get emission probabilities for position
    my %emission = emission($t);

    foreach my $statal (@statecodes){
      my $sum = 'logzero';

      # Sum over founders
      foreach my $state2 (@statecodes){
        # The alpha is already eln from above etc.
        my $al_trans = elnproduct($alphas{$t-1}{$state2},
$transitions{$statal}{$state2});
        $sum = elnsum($sum, $al_trans);
      }

      $alphas{$t}{$statal}=elnproduct($sum, $emission{$statal});
    }
  }
}

#####
# Backward equations (betas)
my %betas = ();

# For each position ($t) get beta: count backwards
for (my $t=$seq_length-1; $t>=0; --$t){
  # For last position
  if ($t == $seq_length-1){
    foreach my $state (@statecodes){
      $betas{$t}{$state} = eln(1);
    }
  }else{
    # Get transition for position and position ahead
    my %transitions = trans($t,$t+1,$arm);

    # Get emission for position ahead

```



```

my %emission = emission($t+1);

foreach my $stater1 (@statecodes){
  my $sum = 'logzero';
  # Sum over founders
  foreach my $state2 (@statecodes){
    my $b_tm1 = elnproduct(elnproduct($betas{$t+1}{$state2},
$transitions{$stater1}{$state2}), $emission{$state2});
    $sum = elnsum($sum, $b_tm1);
  }
  $betas{$t}{$stater1}=$sum;
}
}

# Hash to store probability of state for each position
# Get Y for each position
for (my $t=0; $t<=$seq_length-1; ++$t){
  foreach my $stater1 (@statecodes){
    my $sum = 'logzero';
    foreach my $state2 (@statecodes){
      my $value = elnproduct($alphas{$t}{$state2},$betas{$t}{$state2});
      $sum = elnsum($sum, $value);
    }

    # Get Y out of log space
    $prob_state{$stater1}{$arm}{$positions{$t}*100000}{$stater1} =
eexp(elnproduct(elnproduct($alphas{$t}{$stater1},$betas{$t}{$stater1}), -$sum));
  }
}
for (my $j=0; $j<$totalcount; ++$j){
  my @dat = @{$positioncode{$j}};
  print PROBFIL "test_ril", ",", "$j", ",", "$dat[0]", ",", "$dat[1]", "," ;
  foreach my $state (@statecodes){
    print PROBFIL $prob_state{$test_ril}{$dat[0]}{$dat[1]}{$state}, ",";
  }
  foreach my $founder (@founders){
    my $addprob = additive($dat[1],$founder,$test_ril,$dat[0]);
    print PROBFIL "$addprob", ",";
  }
  print PROBFIL "\n";
}
} #close foreach for rils
} #close foreach for arms

close(PROBFIL);

#####
# Subroutines

#####
# Additive model
sub additive {
  my ($pos, $founder, $ril, $arm) = @_;
  my $addsum=0;
  foreach my $state (@statecodes){
    my $found1 = $statecodes{$state}->[0];

```

```

my $found2 = $statecodes{$state}->[1];
if ($found1 eq $founder && $found2 eq $founder){
  $addsum=$addsum+$prob_state{$ril}{$arm}{$pos}{$state};
}
elseif (($found1 eq $founder) || ($found2 eq $founder)){
  $addsum=$addsum+(0.5*$prob_state{$ril}{$arm}{$pos}{$state});
}
}
return $addsum;
}

# Transition probabilities
sub trans {
  my ($pos1, $pos2, $arm) = @_ ;

  my %transitions = ();

  my $trans_prob = recomb_prob($pos1,$pos2,$arm);
  foreach my $state (@statecodes){
    foreach my $stater1 (@statecodes){
      if ($state eq $stater1){
        # Probability of staying in state
        if ($statecodes{$state}->[0] eq $statecodes{$state}->[1]){
          $transitions{$state}{$stater1} = eln((1 - $trans_prob) +
(($prob_homo*$trans_prob)/$n_founders));
        }else{
          $transitions{$state}{$stater1} = eln((1 - $trans_prob) + 2 * (1-
$prob_transhomo) * $trans_prob * (1/(2*($n_founders-1))));
        }
      }else{
        #if original is homozygous:
        if ($statecodes{$state}->[0] eq $statecodes{$state}->[1]){
          #and if new is homozygous
          if ($statecodes{$stater1}->[0] eq $statecodes{$stater1}->[1]){
            #probability of moving to another homozygous state
            $transitions{$state}{$stater1} =
eln($trans_prob*$prob_homo*(1/($n_founders)));
          }else{
            #original homozygous, new is heterozygous with one copy same as original
            if (($statecodes{$state}->[0] eq $statecodes{$stater1}->[0]) or
($statecodes{$state}->[0] eq $statecodes{$stater1}->[1])){
              #probability of comment above
              $transitions{$state}{$stater1} = eln($trans_prob*(1-
$prob_homo)*(1/($n_founders-1)));
            }else{
              #all other moves from homozygous state involve 2 events, prob near 0
              $transitions{$state}{$stater1} = eln(0);
            }
          }
        }else{
          #if original heterozygous, new is one of homo states (only btw two
          represented in het)
          if (($stater1 eq $statecodes{$state}->[0].$statecodes{$state}->[0]) or
($stater1 eq $statecodes{$state}->[1].$statecodes{$state}->[1])){
            $transitions{$state}{$stater1} = eln($trans_prob * $prob_transhomo * 0.5);
          }else{
            #original heterozygous, new is het with one founder same as original het

```

```

        if (($statecodes{$$state1}->[0] eq $statecodes{$$state}->[0]) or
($statecodes{$$state1}->[0] eq $statecodes{$$state}->[1]) or ($statecodes{$$state1}-
>[1] eq $statecodes{$$state}->[0]) or ($statecodes{$$state1}->[1] eq
$statecodes{$$state}->[1])){
            $transitions{$$state}{$$state1} = eln($trans_prob * (1-$prob_transhomo) *
(1/(2*($n_founders-1))));
        }else{
            #all other moves from het involve two events, prob near zero
            $transitions{$$state}{$$state1} = eln(0);
        }
    }
}
}
}
}
return %transitions;
}
}

```

```

sub recomb_prob {
    #pass positions in Mb and arm
    my ($pos1, $pos2, $arm) = @_;
    #get distance between markers in bp
    my $dist = $positions{$pos2}*1E5 - $positions{$pos1}*1E5;
    #get midpoint between markers in 10*MB for lookup
    my $midpt = $positions{$pos1} + (($positions{$pos2} - $positions{$pos1})/2);

    #find positions for table lookup--uses RF TABLE

    my $LB = int($midpt);
    my $UB = $LB + 1;

    if ($midpt =~ /\^d+$/) {
        #if is at exact integer, just get from table
        my $rf = $recombdatt{$arm}{$midpt};
        #and multiply by distance
        my $trans_prob = (1-exp -($rf*$dist*$trans_scale));
        return $trans_prob;
    }else{
        #if not, calculate linear function slope and intercept and extrapolate
        my $slope = ($recombdatt{$arm}{$UB} - $recombdatt{$arm}{$LB})/($UB-$LB);
        my $intercept = -$slope*$UB+$recombdatt{$arm}{$UB};
        #get value for midpoint
        my $rf = $slope*$midpt + $intercept;
        #and multiply by distance
        my $trans_prob = (1-exp -($rf*$dist*$trans_scale));
        return $trans_prob;
    }
}
}

```

```

#####
# Emission probabilities
sub emission {
    #pass index
    my ($pos) = @_;
    my %emission;

    if (exists $RILS{$positions{$pos}}){

```

```

#get observed counts (k,N)
my $O_n1 = $RILS{$positions{$pos}}{'A'}; #k
my $O_n2 = $RILS{$positions{$pos}}{'B'};
my $O_N = $O_n1 + $O_n2; #N
#put in if for 0,0
if ($O_N>0){
  foreach my $state (@statecodes){
    #get founders for each state (homo or het)
    my $found1 = $statecodes{$state}->[0];
    my $found2 = $statecodes{$state}->[1];
    my $found1_n =
$founders{$found1}{$positions{$pos}}{'A'}+$founders{$found1}{$positions{$pos}}{'B'};
    my $found2_n =
$founders{$found2}{$positions{$pos}}{'A'}+$founders{$found2}{$positions{$pos}}{'B'};

    #get probability of obs "A" for each founder
    my $p1 =
$founders{$found1}{$positions{$pos}}{'A'}/($founders{$found1}{$positions{$pos}}{'A'}
+$founders{$found1}{$positions{$pos}}{'B'});
    my $p2 =
$founders{$found2}{$positions{$pos}}{'A'}/($founders{$found2}{$positions{$pos}}{'A'}
+$founders{$found2}{$positions{$pos}}{'B'});

    if ($p1 >0.995) {$p1=0.995;}
    if ($p1 <0.005) {$p1=0.005;}
    if ($p2 >0.995) {$p2=0.995;}
    if ($p2 <0.005) {$p2=0.005;}

    #get probability of obs k "A"'s in N observations given state.
    #sum over possible genotypes of ril
    my $prob_sum = eln(($p1 * $p2 * (1-$err)**$O_n1 * $err**($O_N-$O_n1))+
((1-$p1) * $p2 * (Math::Cephes::beta($alpha+$O_n1, $O_N+$beta-
$O_n1)/Math::Cephes::beta($alpha, $beta)))+
($p1 * (1-$p2) * (Math::Cephes::beta($alpha+$O_n1, $O_N+$beta-
$O_n1)/Math::Cephes::beta($alpha, $beta)))+
((1-$p1) * (1-$p2) * $err**$O_n1 * (1-$err)**($O_N-$O_n1)));
    #get binomial coefficient
    my $binom_coef = Math::Cephes::lgam($O_N + 1) - Math::Cephes::lgam($O_n1+1) -
Math::Cephes::lgam($O_N-$O_n1+1);
    #calculate probability
    if ($prob_sum eq 'logzero'){
      $emission{$state} = 'logzero';
    }else{
      $emission{$state} = $binom_coef + $prob_sum;
    }
  }#foreach close
}else{
  foreach my $state (@statecodes){
    $emission{$state} = eln(1);
  }
}
}#ifelse close
return %emission;
} #sub close

```

```

#####
#####
# Log Space

# Extended exp
sub eexp{
  # $X is a ln prob
  my ($X) = @_;
  if ($X eq 'logzero'){
    return 0;
  }else{
    return exp $X;
  }
}

# Entended ln
sub eln{
  # $X is raw prob
  my ($X) = @_;
  if ($X eq 0){
    return 'logzero';
  }else{
    return log $X;
  }
}

# Extended ln sum
sub elnsum{
  # $X, $Y are ln probs
  my ($X, $Y) = @_;

  if (($X eq 'logzero') or ($Y eq 'logzero')){
    if ($X eq 'logzero'){
      return $Y;
    }else{
      return $X;
    }
  }else{
    if ($X > $Y){
      #see Mann paper for equations below
      return $X + eln(1 + exp($Y-$X));
    }else{
      return $Y + eln(1 + exp($X-$Y));
    }
  }
}

# Extended product
sub elnproduct{
  # $X, $Y are ln probs
  my ($X, $Y) = @_;

  if (($X eq 'logzero') or ($Y eq 'logzero')){
    return 'logzero';
  }else{

```

```
    return $X + $Y;  
  }  
}
```

Simulated Phenotypes for the pA Inbred RILs

This is the R program to simulate phenotypes in the full pA panel of inbred RILs. It uses the founder SNP table and HMM results available at <http://flyrils.org/Data>.

```
library(RMySQL)

con <- dbConnect(MySQL(),
                 user = "username",
                 password = "password",
                 dbname = "database")

niter <- 200

devs <- c(rep(99, niter / 4),
          rep(39, niter / 4),
          rep(19, niter / 4),
          rep(9, niter / 4))
chrs <- c('X', '2L', '2R', '3L', '3R')

allpos <- data.frame('chr' = numeric(length = 0),
                    'pos' = numeric(length = 0),
                    'dev' = numeric(length = 0))

for (k in 1:length(chrs)) {
  if(chrs[k] == '3R'){
    query <- paste("CREATE TEMPORARY TABLE positions SELECT pos,
                  min(minor_count+major_count) AS MIN,
sum(minor_count)/sum(minor_count+major_count) AS MAF,
                  sum(minor_count/(minor_count + major_count) > 0.01 &&
minor_count/(minor_count + major_count) < 0.99) as Nhet
                  FROM SNPcount
                  WHERE chr= \"", chrs[k], "\"
                  AND pos > 1000000 AND pos<20000000
                  AND foundernum IN(1,18,4,6,9,10,14,15)
                  GROUP BY chr,pos;\"", sep = "")
  } else {
    query <- paste("CREATE TEMPORARY TABLE positions SELECT pos,
                  min(minor_count+major_count) AS MIN,
sum(minor_count)/sum(minor_count+major_count) AS MAF,
                  sum(minor_count/(minor_count + major_count) > 0.01 &&
minor_count/(minor_count + major_count) < 0.99) as Nhet
                  FROM SNPcount
                  WHERE chr= \"", chrs[k], "\"
                  AND pos > 1000000 AND pos<20000000
                  AND foundernum IN(1,3,4,6,9,10,14,15)
                  GROUP BY chr,pos;\"", sep = "")
  }

  send <- dbSendQuery(con, query)
  dbClearResult(send)
  query <- paste("SELECT pos FROM positions WHERE MIN>4 AND MAF > 0.05 AND
Nhet=0;", sep="")
  set <- dbGetQuery(con, query)
  rset <- data.frame('chr' = rep(chrs[k], niter))
}
```

```

rset$pos <- sample(set$pos, niter)
rset$dev <- sample(devs)
allpos <- rbind(allpos, rset)
send <- dbSendQuery(con, "DROP TABLE positions;")
dbClearResult(send)
}

rilnames <- dbGetQuery(con, "SELECT DISTINCT(A.ril) FROM HMMgenosAreg A, stocks B
WHERE B.gstatus=1 AND A.ril=B.ril;")
phenotype.matrix <- matrix(, nrow(rilnames), niter * 5)
row.names(phenotype.matrix) <- rilnames$ril
for (j in 1:nrow(allpos)) {
  if(allpos[j,1]=='3R'){
    query <- paste("SELECT foundernum,minor_count,major_count FROM SNPcount WHERE
foundernum IN(1,18,4,6,9,10,14,15) AND chr= \"",
allpos[j, 1], "\" AND pos= \"", allpos[j, 2], "\";\"", sep = "")
    founds <- dbGetQuery(con,query)
    founds$foundernum[founds$foundernum==18] <- 3
  } else {
    query <- paste("SELECT foundernum,minor_count,major_count FROM SNPcount WHERE
foundernum IN(1,3,4,6,9,10,14,15) AND chr= \"",
allpos[j, 1], "\" AND pos= \"", allpos[j, 2], "\";\"", sep = "")
    founds <- dbGetQuery(con, query)
  }

  founds$fma <- founds$minor_count / (founds$minor_count + founds$major_count)

  allpos$ffounds[j] <- mean(founds$fma)

  founds$fma[founds$fma > 0.995] <- 0.995
  founds$fma[founds$fma < 0.005] <- 0.005

  founds <- founds[order(founds$foundernum), ]

  # Need flanks
  position <- allpos[j, 2]
  position1 <- as.integer(position / 1e4) * 1e4
  position2 <- as.integer((position+1e4) / 1e4) * 1e4

  query1 <- paste("SELECT A.ril, A.AA1,A.AA2,A.AA3,A.AA4,A.AA5,A.AA6,A.AA7,A.AA8
FROM HMMgenosAreg A, stocks B
WHERE A.chr= \"", allpos[j, 1], "\" AND A.pos= \"", position1,
\"\" AND B.gstatus=1 AND A.ril=B.ril;\"", sep = "")
  hmmprob1 <- dbGetQuery(con, query1)

  query2 <- paste("SELECT A.ril, A.AA1,A.AA2,A.AA3,A.AA4,A.AA5,A.AA6,A.AA7,A.AA8
FROM HMMgenosAreg A, stocks B
WHERE A.chr= \"",allpos[j,1], "\" AND A.pos= \"", position2, "\"
AND B.gstatus=1 AND A.ril=B.ril;\"", sep = "")
  hmmprob2 <- dbGetQuery(con, query2)

  if(all.equal(hmmprob1$ril, hmmprob2$ril)){
    hmmprob <- (hmmprob1[ , -1] + hmmprob2[ , -1]) / 2
  } else { print(j) }

  probma <- apply(hmmprob, 1, function(x) sum(x * founds$fma))
  genos <- rbinom(length(probma), 1, probma)
  envs <- rnorm(length(genos), 0, sqrt(allpos[j, 3] * var(genos)))

```



```
if(TRUE %in% is.na(genos)) { print(j) }
phenotype <- genos + envs
if(all.equal(hmmprob1$ri1, row.names(phenotype.matrix))) {
  phenotype.matrix[ , j] <- phenotype
} else { print(j) }
allpos$frils[j] <- mean(genos)
}

save(allpos, file = 'SimPhenoPositions_wfreq.rda')
save(phenotype.matrix, file = 'SimulatedPhenotypes.rda')

dbDisconnect(con)
```

Simulated Phenotypes for the pA-pB cross

This is the R program to simulate phenotypes for the pA-pB crosses. It uses the founder SNP table and HMM results available at <http://flyrils.org/Data>.

```
library(RMySQL)

con <- dbConnect(MySQL(),
                 user = "username",
                 password = "password",
                 dbname = "database")

niter <- 200

devs <- c(rep(99, niter / 4),
          rep(39, niter / 4),
          rep(19, niter / 4),
          rep(9, niter / 4))
chrs <- c('X', '2L', '2R', '3L', '3R')

allpos <- data.frame('chr' = numeric(length = 0),
                    'pos' = numeric(length = 0),
                    'dev' = numeric(length = 0))

for (k in 1:length(chrs)){
  if(chrs[k] == '3R'){
    query <- paste("CREATE TEMPORARY TABLE positions SELECT pos,
                  min(minor_count+major_count) AS MIN,
sum(minor_count)/sum(minor_count+major_count) AS MAF,
                  sum(minor_count/(minor_count + major_count) > 0.01 &&
minor_count/(minor_count + major_count) < 0.99) as Nhet
FROM SNPcount
WHERE chr= \"", chrs[k], "\"
AND pos > 1000000 AND pos<20000000
AND foundernum IN(1,2,18,4,5,6,7,8,9,10,11,12,13,14,15)
GROUP BY chr,pos;\"", sep = "")
  } else {
    if(chrs[k] == '2L'){
      query <- paste("CREATE TEMPORARY TABLE positions SELECT pos,
                    min(minor_count+major_count) AS MIN,
sum(minor_count)/sum(minor_count+major_count) AS MAF,
                    sum(minor_count/(minor_count + major_count) > 0.01 &&
minor_count/(minor_count + major_count) < 0.99) as Nhet
FROM SNPcount
WHERE chr= \"", chrs[k], "\"
AND pos > 1000000 AND pos<20000000
AND foundernum IN(1,2,3,4,5,6,7,8,9,10,12,13,14,15)
GROUP BY chr,pos;\"", sep = "")
    } else {
      query <- paste("CREATE TEMPORARY TABLE positions SELECT pos,
                    min(minor_count+major_count) AS MIN,
sum(minor_count)/sum(minor_count+major_count) AS MAF,
```

```

        sum(minor_count/(minor_count + major_count) > 0.01 &&
minor_count/(minor_count + major_count) < 0.99) as Nhet
        FROM SNPcount
        WHERE chr= "\", chrs[k], "\"
        AND pos > 1000000 AND pos<20000000
        AND foundernum IN(1,2,3,4,5,6,7,8,9,10,11,12,13,14,15)
        GROUP BY chr,pos;\"", sep = "")
    }
}
send <- dbSendQuery(con, query)
dbClearResult(send)
query <- paste("SELECT pos FROM positions WHERE MIN>4 AND MAF > 0.05 AND
Nhet=0;\"",sep="")
set <- dbGetQuery(con, query)

if(chrs[k] == '2L'){
    noZero_2L <- dbGetQuery(con, "SELECT pos FROM T0_2L WHERE
minor_count+major_count>4;")
    set <- subset(set, pos %in% noZero_2L$pos)
}

rset <- data.frame('chr' = rep(chrs[k], niter))
rset$pos<- sample(set$pos, niter)
rset$dev <- sample(devs)
allpos <- rbind(allpos, rset)
send <- dbSendQuery(con, "DROP TABLE positions;")
dbClearResult(send)
}

Arilnames <- dbGetQuery(con, "SELECT DISTINCT(A.ril) FROM HMMgenosAreg A, stocks B
WHERE B.gstatus=1 AND A.ril=B.ril;")
Brilnames <- dbGetQuery(con, "SELECT DISTINCT(A.ril) FROM HMMgenosBreg A, stocks B
WHERE B.gstatus=1 AND A.ril=B.ril;")

phenotype.matrix <- matrix(, nrow(Arilnames), niter*5)
row.names(phenotype.matrix) <- Arilnames$ril
for (j in 1:nrow(allpos)){
    if(allpos[j, 1] == '2L'){
        query <- paste("SELECT foundernum,minor_count,major_count FROM SNPcount
            WHERE chr= "\", allpos[j, 1], "\"
            AND pos= "\", allpos[j, 2], "\"
            AND NOT foundernum=11;\"", sep = "")
        founds <- dbGetQuery(con, query)
        query <- paste("SELECT foundernum,minor_count,major_count FROM T0_2L
            WHERE chr= "\", allpos[j, 1], "\"
            AND pos= "\", allpos[j, 2], "\";\"", sep = "")
        founds2L <- dbGetQuery(con, query)
        founds <- rbind(founds, founds2L)
    } else {
        query <- paste("SELECT foundernum,minor_count,major_count FROM SNPcount WHERE
chr= "\",
            allpos[j, 1], "\" AND pos= "\", allpos[j, 2], "\";\"", sep = "")
        founds <- dbGetQuery(con, query)
    }
}
if(allpos[j, 1] == '3R') {
    founds$foundernum[founds$foundernum==1] <- 'A1'
    founds$foundernum[founds$foundernum==2] <- 'B1'
}

```

```

founds$foundernum[ founds$foundernum==3 ] <- 'D'
founds$foundernum[ founds$foundernum==4 ] <- 'A3'
founds$foundernum[ founds$foundernum==5 ] <- 'B2'
founds$foundernum[ founds$foundernum==6 ] <- 'A4'
founds$foundernum[ founds$foundernum==7 ] <- 'B3'
founds$foundernum[ founds$foundernum==8 ] <- 'B4'
founds$foundernum[ founds$foundernum==9 ] <- 'A5'
founds$foundernum[ founds$foundernum==10 ] <- 'A6'
founds$foundernum[ founds$foundernum==11 ] <- 'B5'
founds$foundernum[ founds$foundernum==12 ] <- 'B6'
founds$foundernum[ founds$foundernum==13 ] <- 'B7'
founds$foundernum[ founds$foundernum==14 ] <- 'A7'
founds$foundernum[ founds$foundernum==15 ] <- 'A8'
founds$foundernum[ founds$foundernum==16 ] <- 'D'
founds$foundernum[ founds$foundernum==17 ] <- 'D'
founds$foundernum[ founds$foundernum==18 ] <- 'A2'
} else {
founds$foundernum[ founds$foundernum==1 ] <- 'A1'
founds$foundernum[ founds$foundernum==2 ] <- 'B1'
founds$foundernum[ founds$foundernum==3 ] <- 'A2'
founds$foundernum[ founds$foundernum==4 ] <- 'A3'
founds$foundernum[ founds$foundernum==5 ] <- 'B2'
founds$foundernum[ founds$foundernum==6 ] <- 'A4'
founds$foundernum[ founds$foundernum==7 ] <- 'B3'
founds$foundernum[ founds$foundernum==8 ] <- 'B4'
founds$foundernum[ founds$foundernum==9 ] <- 'A5'
founds$foundernum[ founds$foundernum==10 ] <- 'A6'
founds$foundernum[ founds$foundernum==11 ] <- 'B5'
founds$foundernum[ founds$foundernum==12 ] <- 'B6'
founds$foundernum[ founds$foundernum==13 ] <- 'B7'
founds$foundernum[ founds$foundernum==14 ] <- 'A7'
founds$foundernum[ founds$foundernum==15 ] <- 'A8'
founds$foundernum[ founds$foundernum==16 ] <- 'D'
founds$foundernum[ founds$foundernum==17 ] <- 'D'
founds$foundernum[ founds$foundernum==18 ] <- 'D'
}

foundsdup <- subset(founds, foundernum=='A8')
foundsdup$foundernum <- 'B8'
founds <- rbind(founds, foundsdup)
founds <- subset(founds, foundernum != 'D')

founds$fma <- founds$minor_count / (founds$minor_count + founds$major_count)
allpos$ffounds[j] <- mean(founds$fma[1:15])
founds$fma[founds$fma > 0.995] <- 0.995
founds$fma[founds$fma < 0.005] <- 0.005

founds <- founds[order(founds$foundernum), ]
Afound <- founds[1:8, ]
Bfound <- founds[9:16, ]

## Need flanks
position <- allpos[j, 2]
position1 <- as.integer(position / 1e4) * 1e4
position2 <- as.integer((position + 1e4) / 1e4) * 1e4

query1 <- paste("SELECT A.ril, A.AA1,A.AA2,A.AA3,A.AA4,A.AA5,A.AA6,A.AA7,A.AA8
FROM HMMgenosAreg A, stocks B

```

```

WHERE A.chr= "\", allpos[j, 1], "\" AND A.pos= \", position1,
      "\" AND B.gstatus=1 AND A.ril=B.ril;", sep = "")
hmmprob1 <- dbGetQuery(con, query1)

query2 <- paste("SELECT A.ril, A.AA1,A.AA2,A.AA3,A.AA4,A.AA5,A.AA6,A.AA7,A.AA8
FROM HMMgenosAreg A, stocks B
      WHERE A.chr= "\", allpos[j, 1], "\" AND A.pos= \", position2,
      "\" AND B.gstatus=1 AND A.ril=B.ril;", sep = "")
hmmprob2 <- dbGetQuery(con, query2)

if(all.equal(hmmprob1$ril, hmmprob2$ril)){
  Ahmmprob <- (hmmprob1[, -1] + hmmprob2[, -1]) / 2
} else { print(j) }

query1 <- paste("SELECT A.ril, A.BB1,A.BB2,A.BB3,A.BB4,A.BB5,A.BB6,A.BB7,A.BB8
FROM HMMgenosBreg A, stocks B
      WHERE A.chr= "\", allpos[j, 1], "\" AND A.pos= \", position1,
      "\" AND B.gstatus=1 AND A.ril=B.ril;", sep = "")
hmmprob1 <- dbGetQuery(con, query1)

query2 <- paste("SELECT A.ril, A.BB1,A.BB2,A.BB3,A.BB4,A.BB5,A.BB6,A.BB7,A.BB8
FROM HMMgenosBreg A, stocks B
      WHERE A.chr= "\", allpos[j, 1], "\" AND A.pos= \", position2,
      "\" AND B.gstatus=1 AND A.ril=B.ril;", sep = "")
hmmprob2 <- dbGetQuery(con, query2)

if(all.equal(hmmprob1$ril, hmmprob2$ril)){
  Bhmmprob <- (hmmprob1[, -1] + hmmprob2[, -1]) / 2
} else{ print(j) }

Aprobma <- apply(Ahmmprob, 1, function(x) sum(x*Afound$fm))
Bprobma <- apply(Bhmmprob, 1, function(x) sum(x*Bfound$fm))
Agenos <- data.frame('genos' = rbinom(length(Aprobma), 1, Aprobma))
Bgenos <- data.frame('genos' = rbinom(length(Bprobma), 1, Bprobma))
Agenos$ril <- Arilnames$ril
Bgenos$ril <- Brilnames$ril

a1.n <- nrow(subset(Arilnames, ril < 12000))
a2.n <- nrow(subset(Arilnames, ril > 12000))
b1.n <- nrow(subset(Brilnames, ril < 22000))
b2.n <- nrow(subset(Brilnames, ril > 22000))

B1genos <- subset(Bgenos, ril < 22000)
B2genos <- subset(Bgenos, ril > 22000)

B1genos <- B1genos[1:a2.n, ]
B2genos <- B2genos[1:a1.n, ]

Bgenos <- rbind(B2genos, B1genos)

CROSS <- data.frame('patRIL' = Agenos$ril,
                    'matRIL' = Bgenos$ril)

genos <- (Agenos$genos + Bgenos$genos) / 2

allpos$frils[j] <- mean(genos)

```

```
  envs <- rnorm(length(genos), 0, sqrt(allpos[j, 3] * var(genos)))
  if(TRUE %in% is.na(genos)) { print(j) }
  phenotype <- genos + envs
  phenotype.matrix[ , j] <- phenotype
  cat(j, '\n')
}

save(phenotype.matrix, file = 'SimulatedPhenotypes_CROSS.rda')

dbDisconnect(con)
```

Simulated Phenotypes for the Multiallelic Scenario

This is the R program to simulate phenotypes for the multiallelic rare scenario. A small change to the code will produce the phenotypes for the multiallelic common scenario (see methods). It uses the HMM results available at <http://flyrils.org/Data>.

```
library(RMySQL)
con <- dbConnect(MySQL(),user="username",password="password",dbname = "database")

load(file='/home/eking/HMMtest/SimPhenoPositions.rda') #name allpos

allpos$dev <- rep(19,nrow(allpos))

rilnames <- dbGetQuery(con, "SELECT DISTINCT(A.ril) FROM HMMgenosAreg A, stocks B
WHERE B.gstatus=1 AND A.ril=B.ril;")
phenotype.matrix <- matrix(,nrow(rilnames),1000)
row.names(phenotype.matrix) <- rilnames$ril
for (j in 1:nrow(allpos)){
  position <- allpos[j,'pos']
  position1 <- as.integer(position/1e4)*1e4
  position2 <- as.integer((position+1e4)/1e4)*1e4
  query1 <- paste("SELECT A.ril, A.AA1,A.AA2,A.AA3,A.AA4,A.AA5,A.AA6,A.AA7,A.AA8
FROM HMMgenosAreg A, stocks B
WHERE A.chr= \"\",allpos[j,1],\"\" AND A.pos= \"\",position1,\"\" AND B.gstatus=1
AND A.ril=B.ril;\",sep="")

  hmmprob1 <- dbGetQuery(con,query1)
  query2 <- paste("SELECT A.ril, A.AA1,A.AA2,A.AA3,A.AA4,A.AA5,A.AA6,A.AA7,A.AA8
FROM HMMgenosAreg A, stocks B
WHERE A.chr= \"\",allpos[j,1],\"\" AND A.pos= \"\",position2,\"\" AND B.gstatus=1
AND A.ril=B.ril;\",sep="")

  hmmprob2 <- dbGetQuery(con,query2)
  if(all.equal(hmmprob1$ril,hmmprob2$ril)){
    hmmprob <- (hmmprob1[,-1]+hmmprob2[,-1])/2
  }else{print(j)}

  alleleprobs <- data.frame('a1'=hmmprob$AA1 + hmmprob$AA2 + hmmprob$AA3+hmmprob$AA4
+ hmmprob$AA5, 'a2'=hmmprob$AA6,'a3'=hmmprob$AA7, 'a4'=hmmprob$AA8)
  genos <- apply(alleleprobs, 1, function(x) which.max(x))
  gen.dev <- rnorm(3,-1,1)
  genos[genos==2] <- gen.dev[1]
  genos[genos==3] <- gen.dev[2]
  genos[genos==4] <- gen.dev[3]

  envs <- rnorm(length(genos),0,sqrt(allpos[j,3]*var(genos)))
  if(TRUE %in% is.na(genos)){print(j)}
  phenotype <- genos+envs
  if(all.equal(hmmprob1$ril,row.names(phenotype.matrix))){
    phenotype.matrix[,j] <- phenotype
  }else{
    print(j)
  }
}
```

```
    }  
  }  
  save(allpos,file='SimPhenoPositions_multi.rda')  
  save(phenotype.matrix,file='SimulatedPhenotypes_multitRARE.rda')  
  
  dbDisconnect(con)
```