COMP3212 Computational Biology - Tutorial 3

Pier Paolo Ippolito, ID: 28415981

1.1 The Fisher-Wright Model

Fisher-Wright Model = model what happens at a single locus (gene location) under selection, mutation and crossover

Examination of Fisher-Wright model where we consider a single locus with two alleles. The population size remains fixed. The mutant has a selection pressure of (1+s) meaning that the mutant produces (1+s) offspring for each offspring of the non-mutant. The forward mutation rate measures the chance of an non-mutate mutating to a mutant. The backward mutation rate is the probability of the reverse process.

- simulation.m

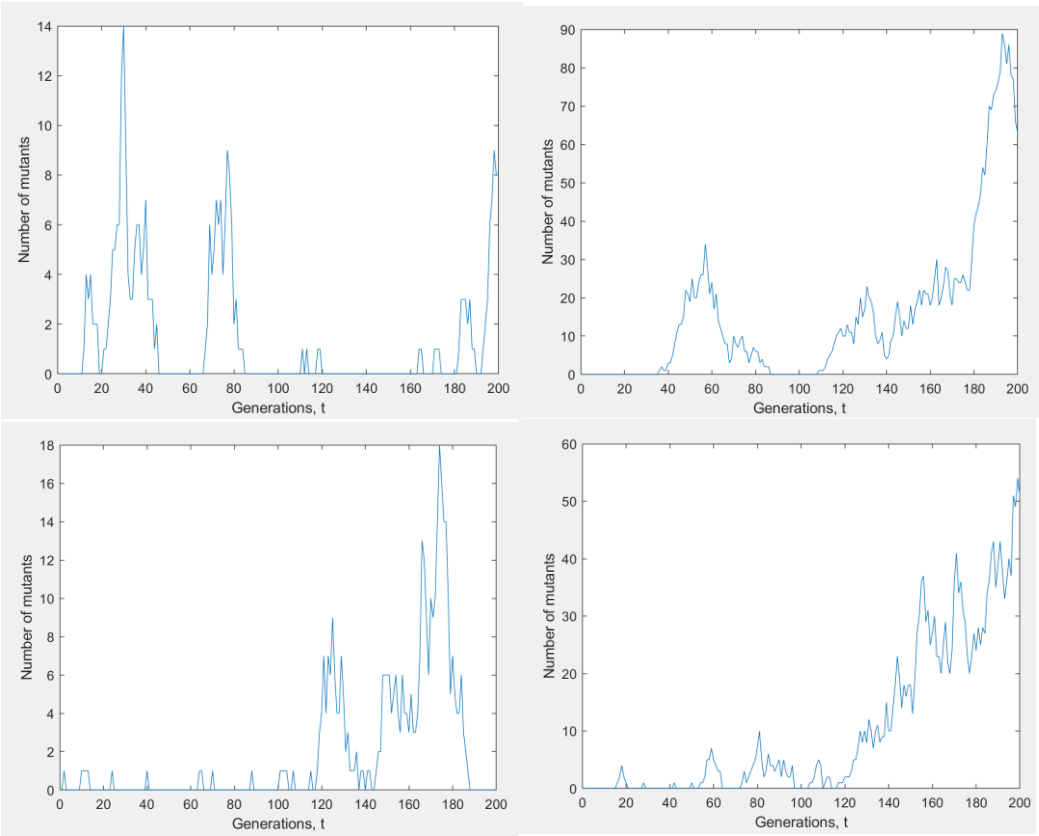
```
function simulation(P, s, u, v)
  t = 1;
  n(t) = 0;    % initialise number of mutants in popu-
lation
  while (n(t)<P+1  & t<200)
    % selection
    p_s = (1+s)*n(t)/(P+s*n(t));
    % mutations
    p_sm = (1-v)*p_s + u*(1.0-p_s);
    % sampling
    t = t + 1;
    n(t) = binomial_rnd(P, p_sm);
  end
  plot([1:t], n)
  % graceplot([1:t], n)
  xlabel('Generations, t')
  ylabel('Number of mutants')
  %z = [[1:t]',n'];
  %save -ascii "simulation.dat" z
```

- binomial_rnd.m

```
function [M] = binomial_rnd(N,p,r,c)
%BINRAND Create random numbers from binomial distribu-
tion.
%
% USE:
% [output] = binrand(N,p,r,c),
%
% where N = the number of trial,
% p = the probability of success per trial,
% r = the number of rows of the output, and
% c = the numbe of columns of the output.
%
% If r and c are unspecified, they will be set = 1.

% Tomo Eguchi
% 23 January 2000

if (nargin < 2),
   error('Not enough input arguments.  It needs at
least three.')
elseif (nargin > 4),
   error('Too many input arguments.  It needs at most
five.')
elseif (nargin == 3),
   c = 1;
elseif (nargin == 2),
   r = 1; c = 1;
end

M = zeros(r,c);

for c1 = 1:N,
   M1 = zeros(r,c);
   y = rand(r,c);
   i1 = find(y < p);
   if (length(i1)~=0),
      M1(i1) = 1;
   end
   M = M + M1;
end
```

simulation(P, s, u, v) = Gene frequency of a mutant allele in a population of size P, with a selective advantage of s (s=0 is no advantage s=1 produces twices as many children), a reverse mutation rate (i.e. from the mutant state to the original state) of v, and a forward mutation rate of u.

Using: simulation(100, 0.01, 0.001, 0.001), the following plots have been produced:



1. Experiment with the parameters.

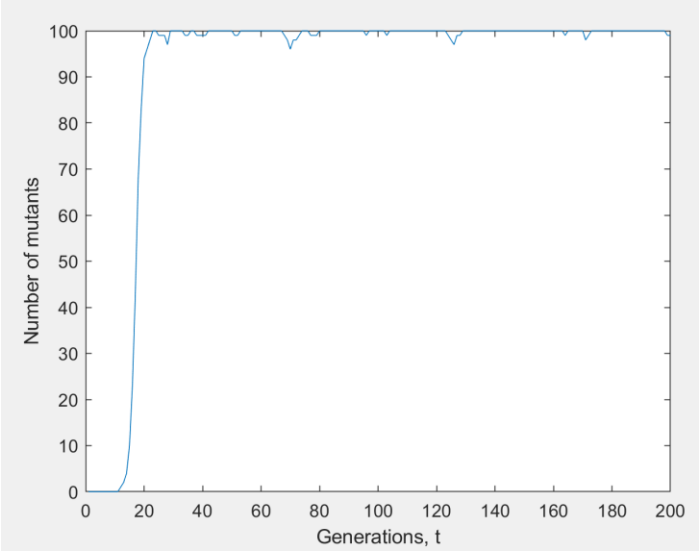
Increasing the population number, the number of mutants starts increasing more frequently when running the simulations multiple times, compared to when working with smaller populations. (there is less variance, the output becomes less noisy) Large population simulations are more consistent when running multiple times and less noisy.

Increasing the fitness the mutant population takes over faster.

Using a negative or equal to 0 selective advantage, the mutant population does not take over. There are some spikes of increase in mutant population, but they die out soon.

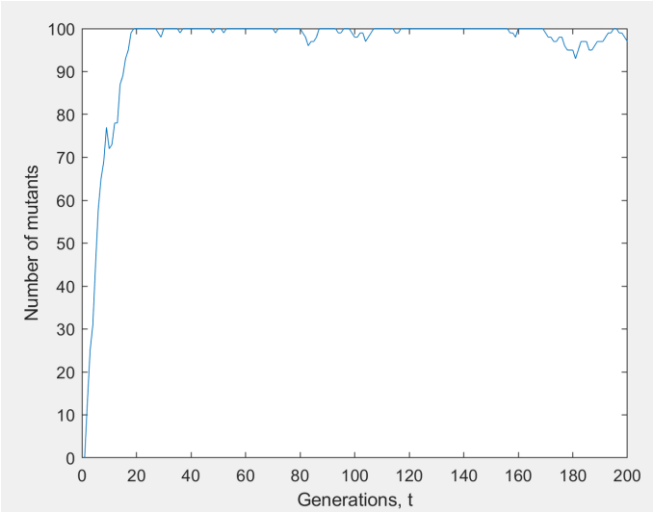
Using a selective advantage greater than 1 (produces twices as many children) the mutant population will constantly take over in every trial, the more we increase S the early the mutant population will take over.

Eg. simulation(100, 1.5, 0.001, 0.001)



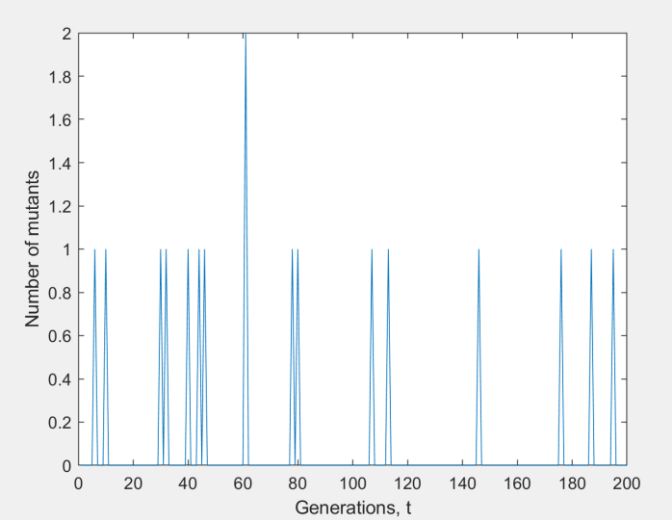
Increasing the forward mutation rate (u) the mutant population starts taking over drastically (faster than increasing the size of the selective advantage).

Eg. simulation(100, 0.01, 0.1, 0.001)



Increasing the reverse mutation rate (u), the number of mutants will just increase in spikes and soon die out. (No mutant population takeover)

Eg. simulation(100, 0.01, 0.001, 1)



2. When doesn't the mutant take over the population?

The mutant will only take over if it has a selective advantage (eg. s>0) and this is guaranteed only when there is a non-zero forward mutation (ie. v>0)

1.2 Markov Chain Analysis

- multisim.m

```
function av_n = multisim(P, s, u, v, NoCopies, T)
  n = zeros(NoCopies,1);
  average = zeros(T+1,1);
  clf
  hold on
  for t=0:T
    % plot distribution
    av_n(t+1) = mean(n);
    if (t>0)
      plot(x,h,'red')
    end
    [h,x] = hist(n,[0:P]);
    h = h/NoCopies;
    plot(x,h)
    xlabel('Number of mutants, n')
    ylabel('P(n)')
    str = strcat('t= ', num2str(t));
    title(str)
    drawnow

    % selection
    p_s = (1+s)*n./(P+s*n);
    % mutations
    p_sm = (1-v)*p_s + u*(1.0-p_s);
    % sampling
    t = t + 1;
    for i=1:NoCopies
      n(i) = binomial_rnd(P, p_sm(i));
    end
  end
  % plot average
  hold off
  plot([0:T], av_n)
  xlabel('Generations, t')
  ylabel('Average number of mutants, n')
```

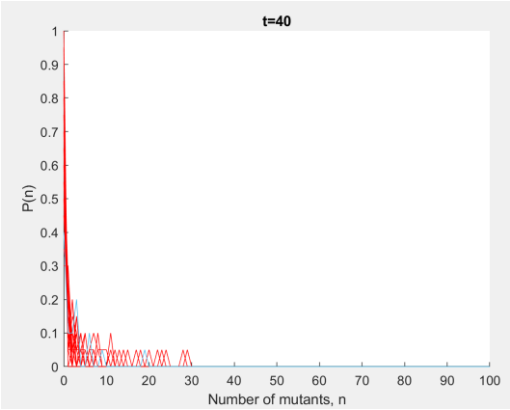
The function multisim(P, s, u, v, NoCopies, T), takes two additional arugments the number of simulations, NoCopies and the length of the simulations, T. It plots a histogram showing the number of mutants for each simulation and graphs the average number of mutants against time. It also

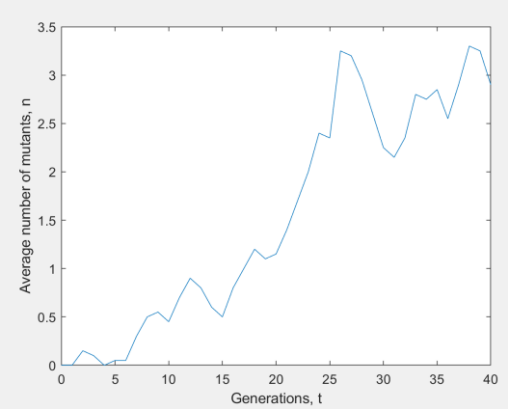
returns a vector containing the average number of mutants at different generations.

Here is a matlab function to generate a transition matrix for the Fisher-Wright model

```
function W = transition_matrix(P, s, u, v)
  W = zeros(P+1,P+1);
  for n = 0:P
    p_sm = ((1-v)*(1+s)*n + u*(P-n))/(P+s*n);
    lp = log(p_sm);
    lq = log(1.0-p_sm);
    x = P*lq;
    for np = 0:P
      W(np+1,n+1) = exp(x);
      x = x + lp - lq + log((P-np)/(np+1));
    end
  end
```

Eg. multisim(100, 0.01, 0.001, 0.001, 20, 40)






```
ans =

  Columns 1 through 10

         0         0    0.1500    0.1000         0    0.0500    0.0500    0.3000    0.5000    0.5500

  Columns 11 through 20

    0.4500    0.7000    0.9000    0.8000    0.6000    0.5000    0.8000    1.0000    1.2000    1.1000

  Columns 21 through 30

    1.1500    1.4000    1.7000    2.0000    2.4000    2.3500    3.2500    3.2000    2.9500    2.6000

  Columns 31 through 40

    2.2500    2.1500    2.3500    2.8000    2.7500    2.8500    2.5500    2.9000    3.3000    3.2500

  Column 41

    2.9000
```

1. Show that the matrix is stochastic

"In mathematics, a stochastic matrix is a square matrix used to describe the transitions of a Markov chain. Each of its entries is a nonnegative real number representing a probability. In the same vein, one may define a stochastic vector (also called probability vector) as a vector whose elements are nonnegative real numbers which sum to 1. Thus, each row of a right stochastic matrix (or column of a left stochastic matrix) is a stochastic vector." -Wikipedia (https://en.wikipedia.org/wiki/Stochastic_matrix)

W=transition_matrix(10,0.1,0.001, 0.001)

The matrix is stochastic because the columns sum to one.

```
W =

  Columns 1 through 10

    0.9900    0.3129    0.0874    0.0209    0.0041    0.0006    0.0001    0.0000    0.0000    0.0000
    0.0099    0.3855    0.2413    0.0987    0.0299    0.0066    0.0010    0.0001    0.0000    0.0000
    0.0000    0.2137    0.2996    0.2098    0.0987    0.0327    0.0072    0.0009    0.0000    0.0000
    0.0000    0.0702    0.2204    0.2642    0.1932    0.0958    0.0317    0.0061    0.0005    0.0000
    0.0000    0.0151    0.1064    0.2183    0.2481    0.1844    0.0913    0.0275    0.0038    0.0001
    0.0000    0.0022    0.0352    0.1237    0.2184    0.2433    0.1807    0.0847    0.0200    0.0011
    0.0000    0.0002    0.0081    0.0487    0.1336    0.2230    0.2482    0.1807    0.0729    0.0086
    0.0000    0.0000    0.0013    0.0131    0.0560    0.1401    0.2338    0.2644    0.1826    0.0482
    0.0000    0.0000    0.0001    0.0023    0.0154    0.0578    0.1445    0.2540    0.3000    0.1773
    0.0000    0.0000    0.0000    0.0002    0.0025    0.0141    0.0529    0.1445    0.2921    0.3862
    0.0000    0.0000    0.0000    0.0000    0.0002    0.0016    0.0087    0.0370    0.1280    0.3786

  Column 11

    0.0000
    0.0000
    0.0000
    0.0000
    0.0000
    0.0000
    0.0000
    0.0000
    0.0000
    0.0099
    0.9900
```

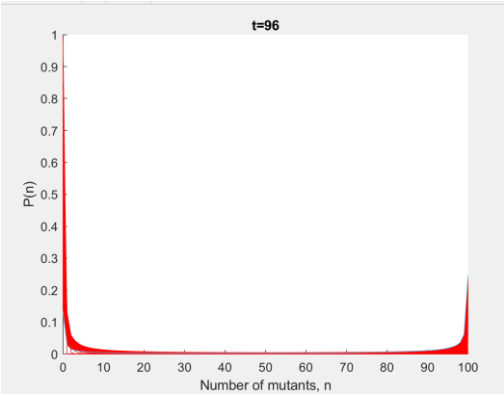
```
>> sum(W)

ans =

  Columns 1 through 10

    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000

  Column 11

    1.0000
```

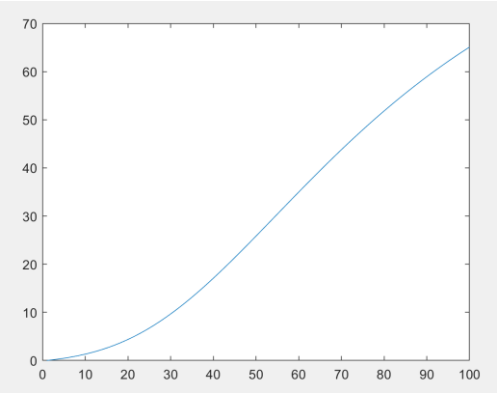
2. Generate the probabilities distributions p(t). Compare the predictions with Answer

"A univariate distribution gives the probabilities of a single random variable taking on various alternative values; a multivariate distribution (a joint probability distribution) gives the probabilities of a random vector – a list of two or more random variables – taking on various combinations of values." - Wikipedia (https://en.wikipedia.org/wiki/Probability_distribution)

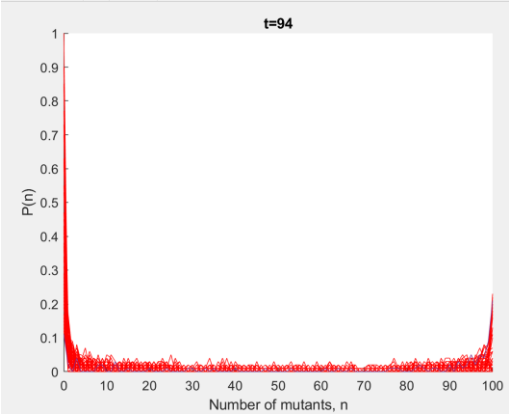
```
function markov(P, s, u, v, T)
  average = zeros(T,1);
  p = zeros(P+1,1);
  p(1) = 1;                % initialise probability
distribution
  W = transition_matrix(P, s, u, v);
  x = 0:P;
  hold on;
  plot(x,p);
  for t = 1:T
    pause(0.2);
    plot(x,p,'red');
    average(t) = mean(x*p);
    p = W*p;
    plot(x,p);
    xlabel('Number of mutants, n');
    ylabel('P(n)');
    str = strcat('t= ', num2str(t));
    title(str);
    drawnow;
  end
  hold off
  xlabel('Generations, t')
  ylabel('Average number of mutants, n')
  plot([1:T], average)
```

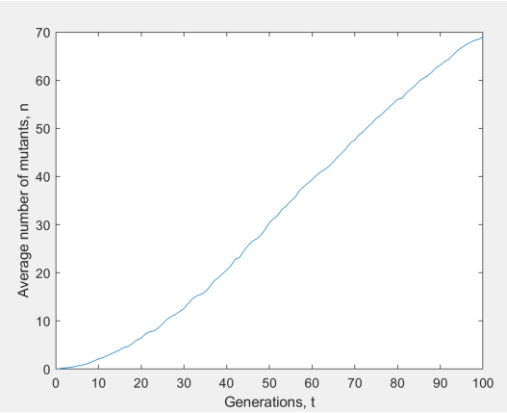
Using markov.m





Using multisim.m





The predictions plots are smoother than the answer counterpart, although the predictions are quite close to the actual results. The markov is plotting the mean of the distributions that are based on the transition probabilities (smooth). Multisim uses instead a stochastic process, that's why the output looks less smooth.

3. Compute the steady state distribution for P=100, s=0.1, v=0.01, u=0.01

"One question we can ask is for the probabilities that the system will be found in each of the states at a given future time. In general, these probabilities will depend on which future time you are asking about. In many, but not all, Markov chains, however, the probability for a particular one of the states will approach a limiting value as time goes to infinity. In other words, in the far

future, the probabilities won't be changing much from one transition to the next. These limiting values are called "stable probabilities".

If start off the system so that each state has probability equal to its stable probability, then these probabilities will persist for all time. The system will therefore be in a "steady state".

Some Markov chains do not have stable probabilities. For example, if the transition probabilities are given by the matrix

$$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

and if the system is started off in State 1, then the probability of finding the system in State 1 will oscillate between 0 and 1 forever.

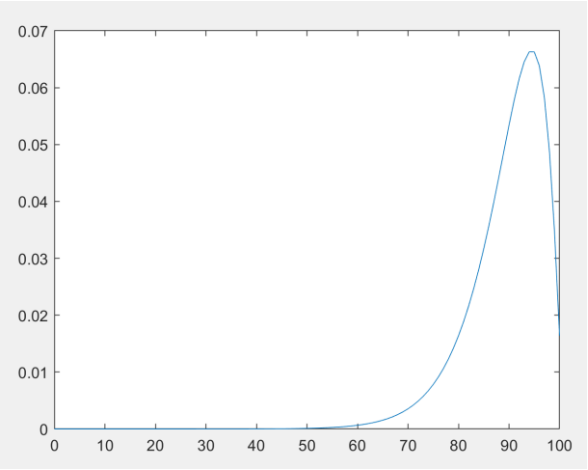
A steady state is an eigenvector for a stochastic matrix. That is, if I take a probability vector and multiply it by my probability transition step matrix and get out the same exact probability vector, it was a steady state. In other words, nothing changed after the step.

limiting distributions ≈ Steady state distributions for markov chains

The idea of a steady state distribution is that we have reached (or converging to) a point in the process where the distributions will no longer change. The distributions for this step are equal to distributions for steps from hereon forward. So if we have reached the limiting distribution and you are at a state ii, the probability you will "step" to an accessible state jj will be the same now and any time in the future. Steady states distributions are unique if they exist. “ - StackExchange (https://math.stackexchange.com/questions/133214/what-does-the-steady-state-represent-to-a-markov-chain)

The steady state distribution is given by the eigenvector with eigenvalue 1

```
W = transition_matrix(100, 0.1, 0.01, 0.01);
[V,L]=eig(W);
plot(0:100,V(:,1)/sum(V(:,1)));
```



The output is the same as if running the markov. This gaussian plot represents the mutant population distribution. On average, there will be around 95 mutants. This is the output, at the steady state point, if we move to the next future iteration, the output will not change much.

1.3 Diffusion Analysis

The following matlab program computes the steady state distribution using the diffusion analysis and the approximation to the diffusion analysis. It also compares it with the Markov chain analysis.

- stadystate.m

```
function steady_state(P,s,u,v)
  global sg;
  global vg;
  global ug;
  global Pg;
  Pg = P;
  vg = v;
  ug = u;
  sg = s;
  x = 0:1/P:1;
  y = zeros(size([0:1/P:1]));
  i=2;

  % Using Gauss-Legendre to perform integration
  xtab = zeros(1,8);
  xtab(1) = 0.0198550718;
```

```
  xtab(2) = 0.1016667613;
  xtab(3) = 0.2372337950;
  xtab(4) = 0.4082826788;
  xtab(5) = 0.5917173212;
  xtab(6) = 0.7627662050;
  xtab(7) = 0.8983332387;
  xtab(8) = 0.9801449282;
  weight = zeros(8,1);
  weight(1) = 0.0506142681;
  weight(2) = 0.1111905172;
  weight(3) = 0.1568533229;
  weight(4) = 0.1813418917;
  weight(5) = 0.1813418917;
  weight(6) = 0.1568533229;
  weight(7) = 0.1111905172;
  weight(8) = 0.0506142681;

  for xx = x
    if (xx==0) | (xx==1)
      y(i) = 0;
    else
      p_sm = ((1-vg)*(1+sg)*xx + ug*(1-xx))/(1+sg*xx);
      bsq = p_sm*(1-p_sm)/Pg;
      z = zeros(1,8);
      j = 1;
      xval = (xx-0.5)*xtab+0.5;
      for xxx = xval
    z(j) = integrand(xxx);
    j = j+1;
      end
      y(i) = exp((xx-0.5)*(z*weight))/bsq;
      %     numerical integration is slow
      %     y(i) = exp(quad('inte-
grand',0.5,xx,0.00001))/bsq;
      i = i+1;
    end
  end
  y = y/sum(y);

  % Compute the leading eigenvector of the transition
matrix
  W = transition_matrix(P,s,u,v);
  [V,L]=eig(W);
  % Use the approximate formula for the diffusion
equation
  z=x.^(2*P*u-1).*(1-x).^(2*P*v-1).*exp(2*P*s*x);
  z = z/sum(z)
```

```
  % plot results
  plot(x,y,x,V(:,1)/sum(V(:,1)),x,z)
  legend('diffusion','markov','approximate diffu-
sion',0)
```

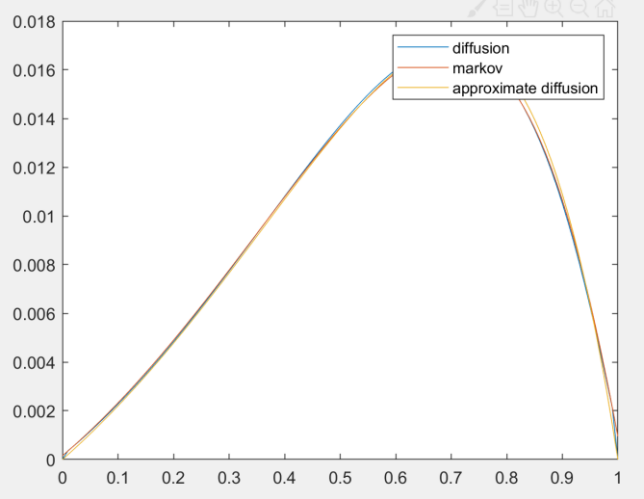
- integrand.m

```
function y=integrand(x)
  global sg;
  global vg;
  global ug;
  global Pg;
  p_sm = ((1-vg)*(1+sg)*x + ug*(1-x))/(1+sg*x);
  a = p_sm-x;
  bsq = p_sm*(1-p_sm)/Pg;
  y = 2*a/bsq;
```

1. Test the quality of the diffusion approximation

Most values work well

steady_state(100,0.01,0.01,0.01)



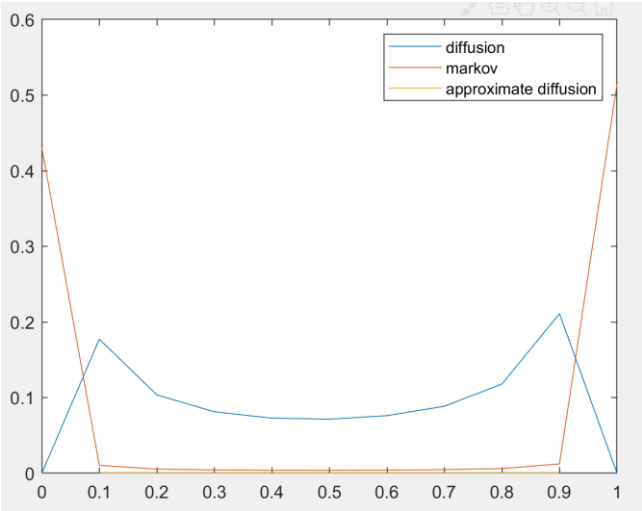
For small population sizes or large selection and mutation rates the diffusion model should begin to break down.

There are problems when 2 P u<1 or 2 P v<1 as there is a

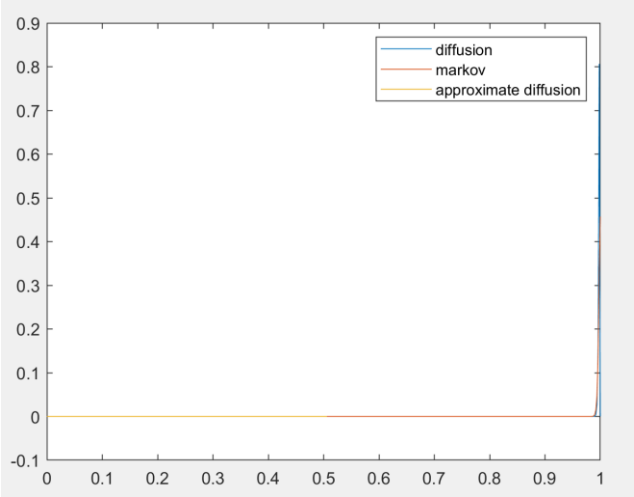
singularity at x=0 or x=1. This is only a problem because computing the normalisation.

Example: Small Population Size

```
>> steady_state(10,0.01, 0.001, 0.001)

z =

   NaN     0     0     0     0     0     0     0     0     0   NaN
```



Example: Large selective advantage
steady_state(500,1.4, 0.001, 0.001)

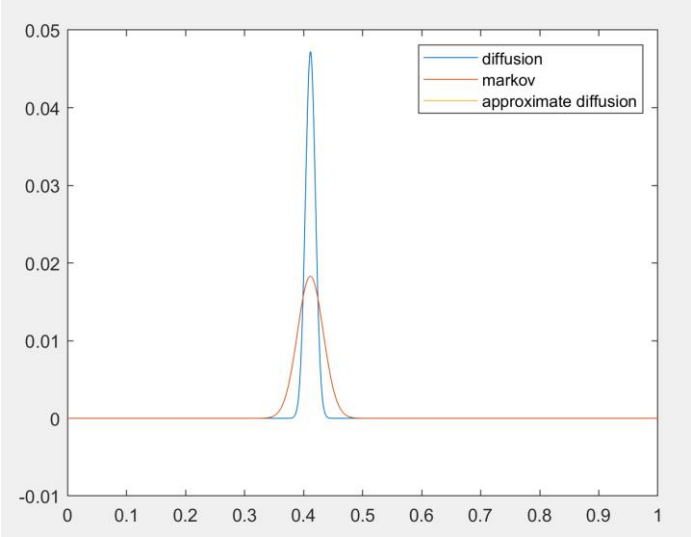


```
Columns 433 through 448

   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN

Columns 449 through 464

   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN

Columns 465 through 480

   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN

Columns 481 through 496

   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN

Columns 497 through 501

   NaN   NaN   NaN   NaN   NaN
```

Example: Large mutation rates

```
>> steady_state(1000,0.001, 1, 1)
Error using eig
Input matrix contains NaN or Inf.

Error in steady_state (line 57)
   [V,L]=eig(W);
```

steady_state(1000,0.001, 0.7, 1)



2. What does the diffusion equation give you which you can't get from the Markov analysis?

The Markov analysis becomes very slow as P incresases (you have to compute the eigenvector of a very large matrix).

1.4 Real Populations

The Fisher-Wright model models a population of binary strings undergoing selection and mutation kept in linkage equilibrium by crossover.

We can compare this with the evolution of a real population of binary strings undergoing selection, mutation and uniform crossover

- ga.m

```
function av_n = ga(P,s,u,v,L,T)
  pop=zeros(L,P);  % initialise population
  av_n = zeros(T+1,1);
  cu = 1/log(1-u);  % mutation magic number
  cv = 1/log(1-v);  % mutation magic number
  hold on
  for t=0:T
    % compute fitness
    n = sum(pop); % number of ones in each member of
the population
    av_n(t+1) = mean(n);
    fitness = (1+s).^n;

    % plot distribution of one at sites
    if t>0
      plot(x,h,'red')
    end
    [h,x] = hist(sum(pop,2),[0:L]);
    h = h/L;
    plot(x,h)
    xlabel('Number of mutants, n')
    ylabel('P(n)')
    str = strcat('t= ', num2str(t));
    title(str)
    drawnow
    pause(0.1) %%% NB this slows you down so you can
see the evolution
```

```
    % selection
    cum_fitness = cumsum(fitness);
    randindex = cum_fitness(P)*rand(P,1);  % roulette
wheels
    newpop=zeros(L,P);
    for mu=1:P
      % using binary search
      lower = 1;
      upper = P;
      while lower<upper
    middle = floor((lower+upper)/2);
    if (randindex(mu)<cum_fitness(middle))
      upper = middle;
    else
      lower = middle+1;
    end
      end
      newpop(:,mu)=pop(:,upper);
    end

    % mutation using a fast but non-obvious algorithm
    % forward mutations 0-->1
    mutu = cumsum(floor(cu*log(rand(3*u*L*P,1))));
    for i=1:length(mutu)
      mu = floor(mutu(i)/L)+1;
      if (mu>P)
    break;
      end
      l = rem(mutu(i),L)+1;
      if newpop(l,mu)==0;
    newpop(l,mu)=1;
      end
    end
    % backward mutation 1-->0
    mutv = cumsum(floor(cv*log(rand(3*v*L*P,1))));
    for i=1:length(mutv)
      mu = floor(mutv(i)/L)+1;
      if (mu>P)
    break;
      end
      l = rem(mutv(i),L)+1;
      if newpop(l,mu)==1;
    newpop(l,mu)=0;
      end
    end

    % uniform crossover
```

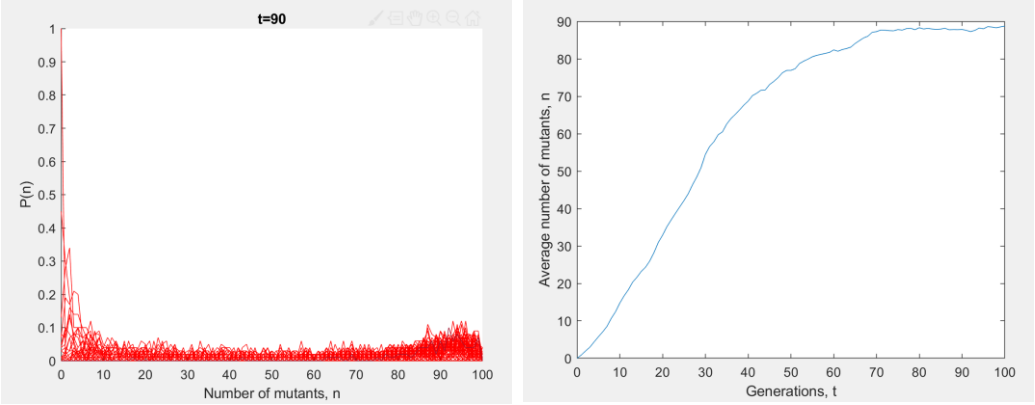
```
    for mu=1:2:P-1
      mask=(rand(L,1)>0.5);
      pop(:,mu) = (mask&newpop(:,mu))|(~mask&new-
pop(:,mu+1));
      pop(:,mu+1) = (~mask&newpop(:,mu))|(mask&new-
pop(:,mu+1));
    end
  end
  hold off
  % Show results
  plot(0:T,av_n)
  xlabel('Generations, t')
  ylabel('Average number of mutants, n')
```

1. Compare this with multisim(P, s, u, v, NoCopies, T) where each copy is run independently

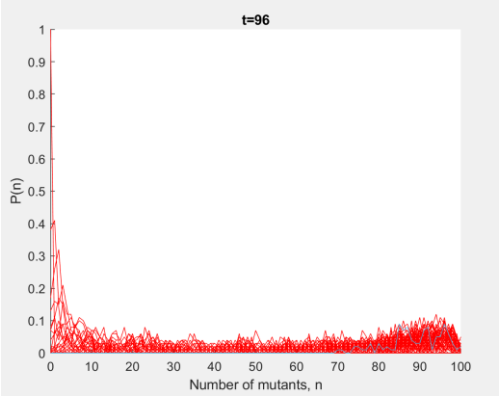
Try running the ga and mutlisim with the same set of parameter

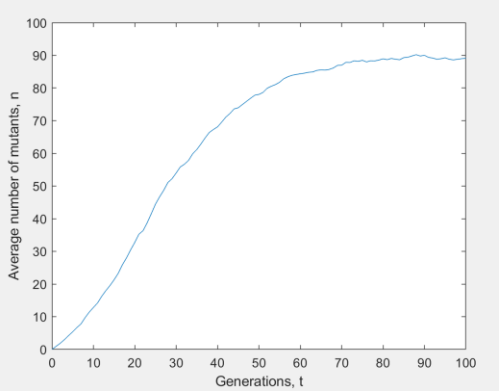
gaav = ga(100,0.1,0.01,0.01,100,100);
fw = multisim(100,0.1,0.01,0.01,100,100);

gavv:



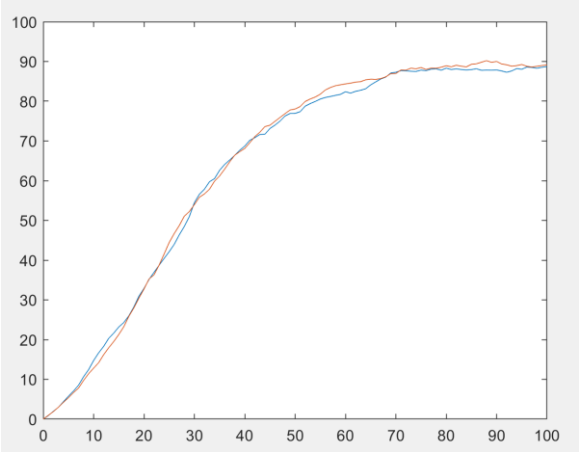
multisim:





The evolution looks very similar. Incidently the ga is much more efficient than multisim and had to be slowed down so you could view the evolution. To compare the average cost

plot(0:100,gaav,0:100,fw)



This shows that the GA is very nearly in linkage equilibrium.

As well as no epistasis the model assumes lots of crossover (compared to selection and mutation). Under this assumption we can treat the loci independently and concentrate on a single locus.