Population Genetic Analyses of Genomic Data 1

USN: 303039534

1 Exercises

1.1 Measurement of variance

	Sample size	Allele frequency
Selected population	124	0.43
Unselected population	176	0.34

Table 1: Collected data

a) Given the data in table 1 for allele frequencies of A in insects treated or untreated with a mild insectiside, we can calculate F_{ST}, quantifying the difference in degree of heterozygosity between the two populations. From the definition of F_{ST}, we have:

$$F_{ST} = 1 - \frac{2q_A^S q_a^S}{2q_A^T q_a^T} = 1 - \frac{N^{S_1} q_A^{S_1} q_a^{S_1} + N^{S_2} q_A^{S_2} q_a^{S_2}}{q_A^T q_a^T (N^{S_1} + N^{S_2})} \tag{1}$$

Inserting the data from table 1 gives

$$F_{ST} = 1 - \frac{124 * 0.43 * 0.57 + 176 * 0.34 * 0.66}{\frac{124 * 0.43 + 176 * 0.34}{300} * \frac{124 * 0.57 + 176 * 0.66}{300} * 300} = 0.008361 = 0.836\%$$

b) If I were to conduct a statistical test and it showed F_{ST} to be significantly greater than zero, I would conclude that treatment with insecticide is likely to enrich for the allele of interest in the insect population. This could have one of a number of different causes. The simplest is that one of the two allelles confers an evolutionary advantage/disadvantage in the presence of the insecticide, leading to direct selection for this allele in the presence but not the absence of insecticide, and hence the observed difference in heterozygosity suggested by $F_{ST} > 0$. An alternative explanation is that the locus of interest is genetically linked to a gene or genomic region that confers a selective advantage or disadvantage in the presence of the insecticide.

1.2 Modelling fitness in a diploid system

We now consider a set of diploid organisms with a fraction p_i infected by a parasite. Consider allele A with prequency p at locus A/a to confer a protective effect against the parasite and allele a at the same locus to have an allele frequency of $q = 1 - p$. We further specify that the fitness (here given as the 'growth rate' of the organism) is dependent on infection state and genetic composition at locus A/a as given by table 3

	AA	Aa	aa
Infected	a	b	c
Not infected	1	1	c

Table 2: Fitness of individuals

Here, $a, b, c \in]0, 1[$

a) The Hardy-Weinberg proportions specify the relative proportion of organisms with each genotype that would lead to a stable equilibrium given the allele frequencies. This is given by

$f(AA) = p^2$
$f(Aa) = pq + qp = 2pq$
$f(aa) = q^2$

This can be visualized as the relative areas of the diagonal and off-diagonal sections of the population Punnett square.

b) We now assume the following values:
$p = 0.5$
$q = 1 - p = 0.5$
$a = 0.8$
$b = 0.9$
$c = 0.7$
$p_i = 0.9$
This allows us to calculate the mean fitness of the population as the weighted mean of the fitnesses of the infected and uneffect populations. We denote the fitness F with subscripts i and u to denote infected and uninfected subpopulations.

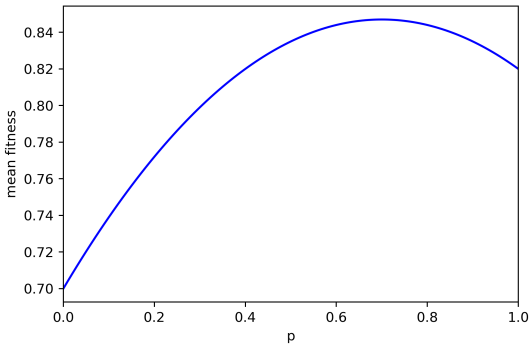
$$F_{mean} = p_i*(f(AA)*F_i(AA)+f(Aa)*F_i(Aa)+f(aa)*F_i(aa))+(1-p_i)*(f(AA)*F_i(AA)+f(Aa)*F_i(Aa)+f(aa)*F_i(aa))$$

$$F_{mean} = p_i * (p^2 * a + 2pq * b + q^2 * c) + (1 - p_i) * (p^2 + 2pq + q^2 * c) \tag{2}$$

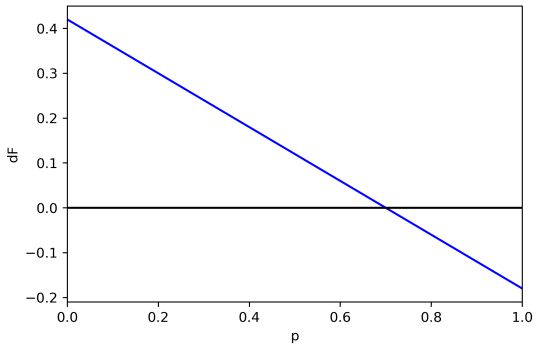
$$F_{mean} = 0.9 * (0.5^2 * 0.8 + 2 * 0.5^2 * 0.9 + 0.5^2 * 0.7) + 0.1 * (0.5^2 + 2 * 0.5^2 + 0.5^2 * 0.7) = 0.835$$

c)

To find the value of p that leads to maximum fitness while fixing the remaining parameters, we start by plotting the mean fitness of the population against the allele fraction p for these values of $a, b, c \,\&\, p_i$ in figure 1a. We see that the mean fitness has a maximum somewhere between p=0.6 and p=0.8.



(a) Plot of mean fitness against allele fraction p



(b) $\frac{dF_{mean}}{dp}$ against allele fraction p

We can find the optimum value of p analytically by solving

$$\frac{dF_{mean}}{dp} = 0$$

This gives

$$\frac{d}{dp} = p_i * (p^2 * a + 2(p - p^2) * b + (p^2 + 1 - 2p) * c) + (1 - p_i) * (p^2 + 2(p - p^2) + (p^2 + 1 - 2p) * c) = 0 \tag{3}$$

Inserting the parameter values specified in the question, we find

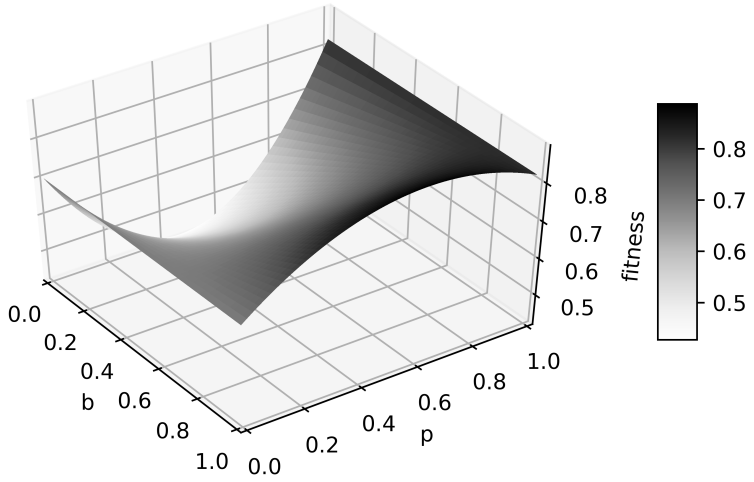
$$0.9 * (2 * p * 0.8 + 2(1 - 2 * p) * 0.9 + (2 * p - 2) * 0.7) + 0.1 * (2 * p + 2(1 - 2 * p) + (2 * p - 2) * 0.7) = 0$$

$$p(2*0.9*0.8-4*0.9*0.9+2*0.9*0.7+2*0.1-4*0.1+2*0.1*0.7)+(2*0.9*0.9-2*0.9*0.7+2*0.1-2*0.1*0.7)=0$$

This yields an optimum allele fraction of $p = 0.7$ as expected from figure 1b. We thus see that we optimize the mean fitness of a population in Hardy-Weinberg equilibrium with $p = 0.7$ with a monotonic decrease in fitness on either side of this optimum value.

d)

To find the values of b for which allele a is maintained in the population over long timescales, we start by plotting the mean population fitness as a function of both p and b for the values of $a, c\&p_i$ specified in the question (figure 3a).



(a) Mean fitness as a function of allele fraction p and $F_i(Aa) = b$

We assume a population in the limit of infinite size where genetic drift can be neglected. In order to maintain the allele a in the population over long timescales, we require that the maximum fitness occurs at $p \neq 1$ for a given b. We thus return to equation (3) and write $\frac{d}{dp}$ as a function of b and p.

$$\frac{d}{dp} = p(2*0.9*0.8-4*0.9*b+2*0.9*0.7+2*0.1-4*0.1+2*0.1*0.7)+(2*0.9*b-2*0.9*0.7+2*0.1-2*0.1*0.7)$$

$$\frac{d}{dp} = -3.6bp + 2.64p + 1.8b - 1.2 = (2.64 - 3.6b)p + (1.8b - 1.2) \tag{4}$$

We thus find an extremum in p when

$$p = \frac{1.2 - 1.8b}{2.64 - 3.6b}$$

We require this extremum to fall in $0 \leq p \leq 1$ for a local extremum within the given parameter space, and therefore find the limiting b values of biologically viable solutions as

$$1 = \frac{1.2 - 1.8b}{2.64 - 3.6b} => b = 0.8$$

$$0 = \frac{1.2 - 1.8b}{2.64 - 3.6b} => b = 0.67$$

Thus for $b > 0.8$ we have a maximum in p in the mean fitness landscape at $p < 1$, giving us a non-zero fraction of the a allele at long timescales. For $b < 0.67$, we have a minimum in the fitness landscape which we will discuss further below.

We now consider the case where $0.67 < b < 0.8$. Setting p = 0.5, equation (4) gives us

$$\frac{d}{dp} = -1.8b + 1.32 + 1.8b - 1.2 = 0.12 > 0$$

Since we do not cross a point at which $\frac{d}{dp} = 0$ for $0.67 < b < 0.8$, F_{mean} must be a monotonously increasing function of p in this range, and the equilibrium population composition contains only AA individuals, irrespective of the initial composition of the population. In this case, the a allele is not maintained over long timescales.

We can understand the above results in a more biological context by considering the relative fitness of different individuals for various values of b. The mean fitness of individuals with genotype AA is
$F_m(AA) = p_i * a + (1 - p_i) * 1 = 0.9 * a + 0.1 * 1 = 0.82$
Similarly,
$F_m(Aa) = 0.9 * b + 0.1$
$F_m(aa) = 0.9 * c + 0.1 * c = 0.7$

This is consistent with F_{mean} being invariant of b at p=0 where we only have aa individuals giving $F_{mean} = 0.7$, and at p=1 where we only have AA individuals giving $F_{mean} = 0.82$ (figure 3a).

Since AA individuals are always more fit than aa individuals, we only retain a in the population over long timescales when $F_m(Aa) > F_m(AA)$ which requires $b > \frac{0.82 - 0.1}{0.9} = 0.8$. I.e. we retain a in the population when $b > a$ since this implies that heterozygous individuals are more fit than either homozygote for the infected subpopulation, and equally fit to AA individuals in the non-infected subpopulation. This is consistent with our result above.

However, we also note that in the case of $F_m(Aa) < F_m(aa)$, we observe a local minimum of F_{mean} as a function of p, since starting from a population of aa individuals, introducing heterozygotes reduces the mean fitness. $F_m(Aa) < F_m(aa)$ corresponds to $b < \frac{0.7 - 0.1}{0.9} = 0.67$, again consistent with our result above. In this case, the genetic makeup of the population at long timescales depends on the initial population composition. If p_{init} is small enough, the population will move towards all individuals having the aa genotype, whereas if p_{init} is large, the population will move towards all individuals being AA.

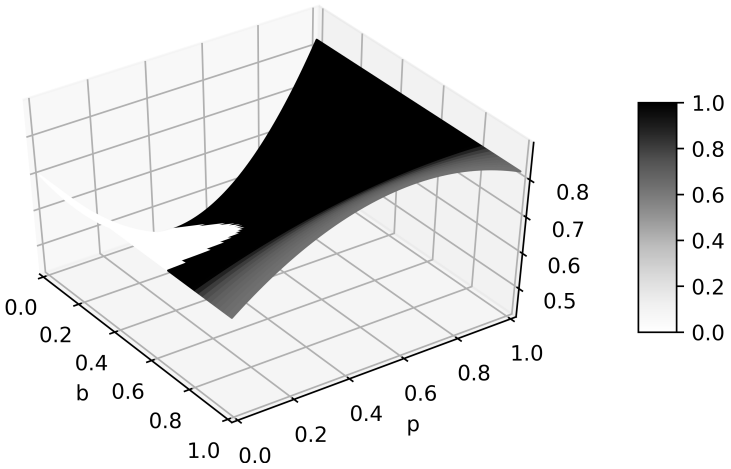
As before, the cutoff for these two scenarios is given by

$$p = \frac{1.2 - 1.8b}{2.64 - 3.6b}$$

This allows us to predict for any value of b and initial composition p, what the long term population composition will be in the limit of an infinitely large population. This can be summarized as follows:

1. $b < 0.67$ and $p_{init} < \frac{1.2 - 1.8b}{2.64 - 3.6b}$: The population will go to only aa individuals (the a allele *is* maintained over a long period of time)
2. $b < 0.67$ and $p_{init} > \frac{1.2 - 1.8b}{2.64 - 3.6b}$: The population will go to only AA individuals (the a allele *is not* maintained over a long period of time)
3. $0.67 < b < 0.80$: The population will go to only AA individuals (the a allele *is not* maintained over a long period of time)
4. $b > 0.80$: The population will go to a mixture of AA, Aa and aa individuals specified by $p_{eq} = \frac{1.2 - 1.8b}{2.64 - 3.6b}$ (the a allele *is* maintained over a long period of time)

This is illustrated graphically in figure 3a where the z-axis represents the mean fitness for a given b and p while the color scheme reflects the long-term allele frequency p.



(a) Plot of mean fitness (z-axis) against allele frequency p and fitness $F_i(Aa) = b$. The colour scheme represents the equilibirum composition p_{eq} that will be achieved over a long period of time if the population starts at a given initial point on the fitness landscape. This corresponds to the result of a steepest ascent search along the p axis. Black regions indicate initial points from which allele a *is not* maintained over a long period of time.

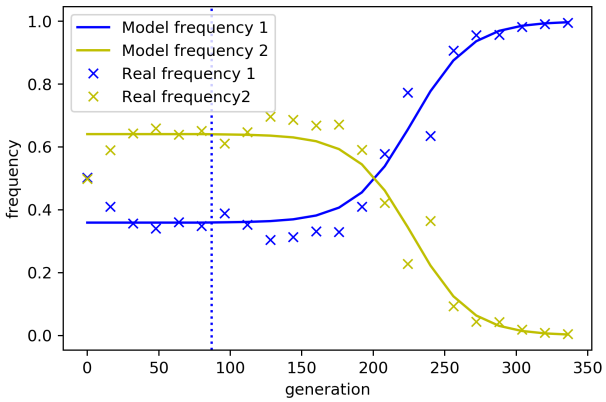
1.3 Identifying beneficial mutations in a neutral marker experiment

a) Optimist has been installed and compiled as instructed

b) For the data in Colour_freq.out when running 10 optimizations for a single mutation, every optimization gives a maximum log likelihood of $L = -92.5714$. The parameter space thus seems to be relatively well behaved as the algorithm consistently converges to the same maximum. The optimization yields a haplotype fitness of 0.049169, establishment time of 86.623 and initial frequencies of 0.359 and 0.641.

c) The systematic search in discrete time yields maximum log likelihood of -92.5725 with haplotype fitness 0.0492702, dicrete establishment time 87 and initial frequencies of 0.359 and 0.641. We thus see that this only yields a slight change in parameters from the initial optimization, with a slightly worse log likelihood due to being retricted to discrete establishment times.

We plot the model population composition together with the experimental frequencies in figure 4a and see that while they largely agree on the population evolution, the model fails to take into account the variation in frequencies prior to $t = 87$ when the mutation is introduced.



(a) Plot of model (line) and real (crosses) population composition at different generation times. Blue vertical line indicates $t = 87$ when a beneficial mutation arises in population 1.

d)

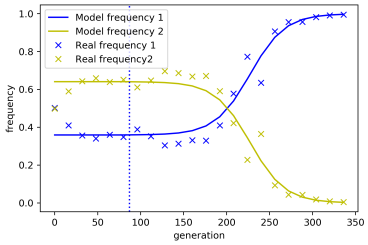
For two and three mutations, the parameter landscape in the initial optimization is less well-behaved, and we converge to maximum likelihoods ranging from -81.7427 to -81.7881 for 2 mutations and -81.1044 to -90.195 for three mutations.

Running the subsequent systematic optimization on the best intial optimization, we get the parameters specified in table 3.

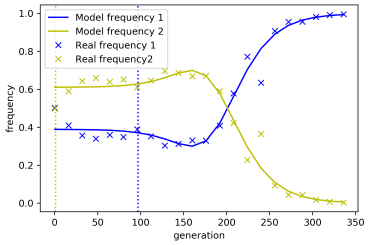
	1 mutation	2 mutations	3 mutations
log likelihood	-92.573	-81.750	-81.093
haplotype fitness	0.0493	0.0405	0.0442
clone	1	2	2
establishment time	87	1	1
haplotype fitness		0.0780	0.0651
clone		1	1
establishment time		97	74
haplotype fitness			0.0896
clone			1
establishment time			112

Table 3: Result of the systematic search algorithm for 1, 2 and 3 mutations.

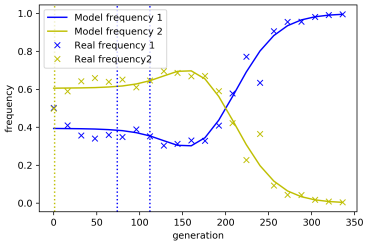
The model and real population compositions for 1-3 mutations are given in figures 5a-5c. In both of these cases, we initially get beneficial mutations in clone 2, followed by 1 or 2 mutations that increase the fitness of clone 1. This is consistent with the data which shows an initial dip in the frequency of clone 1 followed by a steep rise.



(a) plot of mean fitness



(b) plot of derivative



(c) plot of derivative

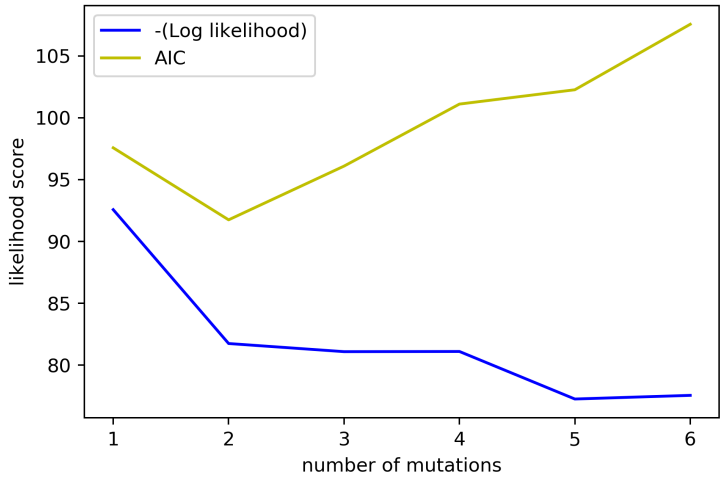
However, we also note that there is only a small increase in log likelihood from 2 to 3 mutations despite an increase in the number of free parameters, suggesting that introducing an extra mutation does not improve our ability to explain the data much. To investigate the balance of fit and number of parameters more systematically, we fit the data using 1-6 mutations and plot the optimized log likelihood in figure 6a.

We also introduce the Akaike Information Criterion $AIC = 2.5(p-1) - L$ used by Illingworth & Mustonen (Bioinformatics 2012) when they first described the *Optimist* software. Here, p is the number of parameters and L is the total log likelihood. The number of parameters is given by $1 + 2m$ where there is a constant parameter 1 specifying the initial distribution, and two additional parameters for each mutation specifying the establishment time and haplotype fitness. The AIC is also plotted against number of mutations in figure 6a.

We see that the log likelihood increases substantially from 1 to 2 mutations but then flattens out, leading to a clear minimum in AIC for 2 mutations. Together with the considerations above and plots of model and real frequencies, this suggests that two beneficial mutations occurred in the population over the course of the experiment, the first in clone 2 and the second in clone 1.

e) Mutations can confer antibiotic resistance by a number of different mechanisms. Three common mechanisms of resistance are given below:

1. If the antibiotic binds directly to a vital bacterial protein, a mutation that alters residues in the drug binding site can prevent binding and thus relieve inhibition. This will commonly lead to reduced bacterial fitness in the absence of the drug, particularly if the binding site is also the active site, since resistance by binding site mutation will then require alterations of the active site. An example of such resistance is seen in antibiotic resistance to penicillin. Penicillin and other β-lactam antibiotics inhibit cell-wall synthesis by irreversible inhibition of crosslinking enzymes, since the drug resembles the natural peptidoglycan substrate. However, the crosslinking enzymes may undergo mutations that increase the specificity for peptidoglycan over penicillin, leading to antibiotic resitance.



(a) model of mutation

2. Another common mechanism of antibiotic resistance is to clear the toxic compound from the cell using efflux pumps. These are either specific or general-purpose membrane proteins that utilize the energy stored in the proton-motive force to drive efflux of antibiotics and other toxic compounds. Mutations can alter the selectivity of such efflux pumps to improve clearance of antibiotics and thus improve resistance.

3. Antibiotic resistance can also evolve by inhibition of the antibiotic via phosphorylation or glycosylation. An example is the inactivation of the macrolide erythromycin, which is commonly phosphorylated at the 2'-O position, preventing it from binding to the bacterial ribosome and inhibiting translation. Such systems can evolve by altering the specificity of naturally occuring kinases and glycosylases.

Antibiotic resistance also commonly occurs via horizontal gene transfer between species, as is observed in the distribution of e.g. β-lactamases and other dedicated antibiotic defense systems across bacterial species. However, this is not a single mutation conferring resistance and does not occur in isolated populations, and is thus not as relevant to the present question

f)

Errors in a single datapoint are unlikely to significantly perturb the result when only modeling a few mutations as can for example be seen by the datapoint at 240 generations where frequency 2 appears to be significantly higher than the trend. However, consecutive or systematic errors will lead to erroneous models and thus erroneous inference of the time and effect of a given mutation, and possible even lead to inference of mutations that are not real to explain aberrant data.

It is stated in the documentation for *Optimist* that "the code assumes that this data is a sample of individuals, drawn randomly from the population, such that noise in the data is multinomial in nature". This is likely to be a reasonable assumption for human counting if this is done by subsampling a larger population and counting all individuals.

In the case of flow cytometry, on the contrary, there are known biases in terms of the size or fluorescent intensity of the populations. In this case if the effect of a mutation also alters the size or fluorescence of a subpopulation, this can also create artifacts in the inferred parameters. The unbiased multinomial model that is the basis of *Optimist* is thus unlikely to provide an accurate representation of the data.

Instead I would attempt to implement a different likelihood function that takes into account sampling bias with parameters determined at $t = 0$.

1.4 Inference of evolutionary parameters

Given the large population size of $N = 10^7$ we assume that genetic drift is negligible. We also assume a low rate of mutation such that selection over the course of the experiment is primarily on the genetic variation introduced through the two diverged ancestral strains rather than the emergence of novel mutations.

We consider data from a single segregating site at a read depth of 100 in a haploid population evolving over time under an artifical selection pressure. Data for this site is given in table 4 and is plotted in figure 7a.

time	0	1	2	3	4	5	6
$n_i^1(t)$	28	39	57	72	78	81	82

Table 4: allele counts at segregating site i over time

We start by considering the case where site i is under direct selection for allele 1. In this case, the rate of change in the frequency of allele 1 ($\frac{dq_i^1}{dt}$) depends on both the current frequency q_i^1, representing the number of individiduals that can provide allele 1, and $(1 - q_i^1)$ representing the number of individuals that can acquire allele 1. In this scenario, we can thus write down a differential equation (equation of motion) describing the change in allele frequency q over time.

$$\frac{dq_i^1}{dt} = \sigma q_i^1 (1 - q_i^1) \tag{5}$$

Here σ is a parameter specifying the fitness benefit of allele 1 over allele 0 in the presence of the evolutionary pressure. Solving this first order differential equation with boundary conditions $q_i^1(t = 0) = q_0$ gives

$$q_i^1(t) = \frac{q_0 \exp(\sigma t)}{1 - q_0 + q_0 \exp(\sigma t)}$$

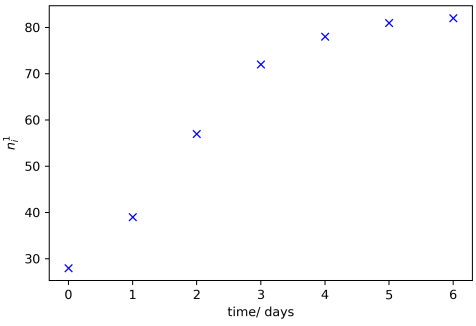
This will be our continous-time evolutionary model M. We can calculate the log likelihood of observing our data given a model M_{σ,q_0} with parameters σ and q_0 as

$$L = \sum_{k=0}^{6} \log P(n_i^1(t_k)|N, M_{\sigma,q_0})$$

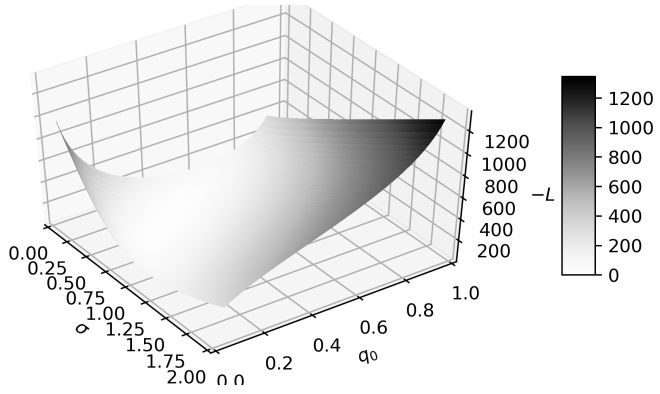
In the present case, $N = 100$, $n_i^0 = 100 - n_i^1$, $q_i^0 = 1 - q_i^1$ and we use a binomial model such that

$$P(n_i^1(t_k)|N, M_{\sigma,q_0}) = \frac{N!}{n_i^1(t_k)! n_i^0(t_k)!} q_i^1(t_k)^{n_i^1(t_k)} q_i^0(t_k)^{n_i^0(t_k)}$$

This allows us to estimate the probability of observing the data in table 4 given our model parameters by calculating $q_i^1(t_k)$ for $k \in [0 : 6]$. Since we only have two parameters, we can scan the entire log-likelihood surface and plot the result in 3 dimensions (figure 7b).



(a) n_i^1 at the 7 timepoints for which data is given.



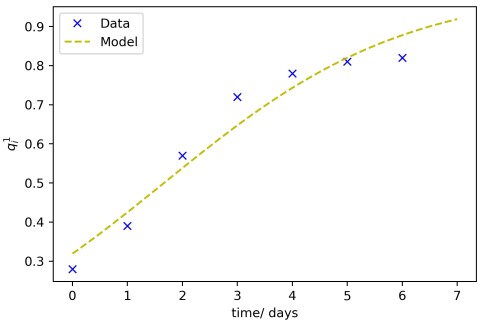
(b) Log likelihood of the observed data given different combinations of model parameters.

Figure 7

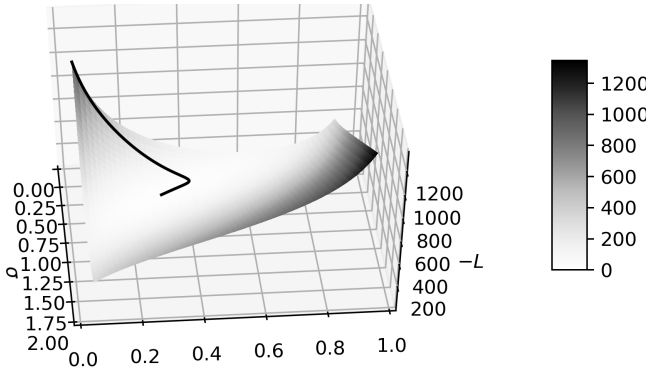
From figure 7b, we note firstly that a range of combinations of σ and q_0 appear to give reasonable approximations to the data, and secondly that the log-likelihood landscape is well behaved without multiple local minima. This allows us to write a numerical steepest descent algorithm to find the optimum parameter set (the maximum likelihood parameters). Note that all code used in this assignment is included in the appendix.

The steepest descent algorithm gives ML parameters of $\sigma = 0.455$ and $q_0 = 0.319$ with $L_{min} = -20.590$. This result is invariant of initial parameters, and the steepest descent path for initial parameters $\sigma = 0$ and $q_0 = 0.05$ has been projected onto the maximum likelihood surface in figure 8b as an example.

Given the optimum parameters, we can also estimate q_i^1 for $0 < t < 6$ and compare the model estimate with the fraction of allele 1 observed in the samples from table 4. This is plotted in figure 8a.



(a) q_i^1 of 100 samples at the 7 timepoints for which data is given, and q_i^1 versus time as predicted by the MLE model.



(b) Log likelihood of the observed data given different combinations of model parameters. Superimposed on this surface is the steepest descent path for a parameter optimization going to $(\sigma, q_0) = (0.455, 0.319)$. The same minimum was converged to with a range of different initial parameters.

Figure 8

We note that while there is some agreement between the model and the observed data, the model fit appears to assume less of a 'sigmoidal shape' than might have been expected, and asymptotes towards 1 much faster than the observed data which appears to asymptote to a lower maximum frequency.

In order to estimate the uncertainty in our parameters, we quantify a 99% confidence interval for our data. Since we are working with likelihoods, we can write $L_{min} = \log P_{min}$ where L_{min} is our optimum log likelihood and P_{min} is the corresponding non-log-transformed likelihood. For a 99% confidence interval, we then proceed to write

$$P_{thresh} = (1 - 0.99)P_{min} = 0.01P_{min}$$

Hence

$$L_{thresh} = \log P_{thresh} = \log\left[0.01P_{min}\right] = -\log 100 + L_{min} = -25.21$$

This region of uncertainty has been projected onto the log-likelihood landscape in figure 9a. For $L > -25.21$, we find that σ varies from 0.323 to 0.586 and q_0 from 0.237 to 0.414. However, we note that this variability takes an ellipsoid shape that is not aligned along the σ and q_0 axes, such that for example $(\sigma, q_0) = (0.586, 0.237)$ has $L = -24.8$ while $(\sigma, q_0) = (0.323, 0.237)$ has $L = -69.5$.

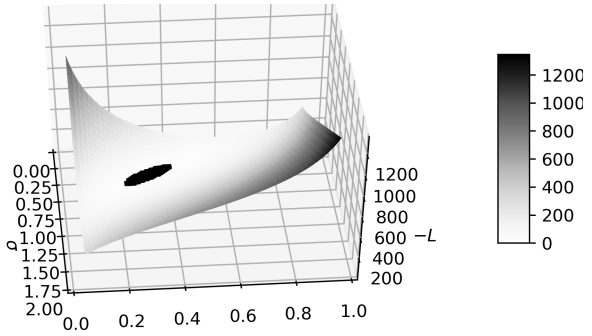
To get a better estimate of the uncertainty in our parameter space, we can perform a coordinate transformation to align our coordinate system with the directions of minimum and maximum variability in L. Near the minimum and setting $\delta = 0.001$, we can write

$$\nabla L \approx \begin{bmatrix} \dfrac{L(\sigma_{min} + \delta, q_{0,min}) - L(\sigma_{min}, q_{0,min})}{\delta} \\ \\ \dfrac{L(\sigma_{min}, q_{0,min} + \delta) - L(\sigma_{min}, q_{0,min})}{\delta} \end{bmatrix} = \begin{bmatrix} 0.684 \\ 1.45 \end{bmatrix}$$

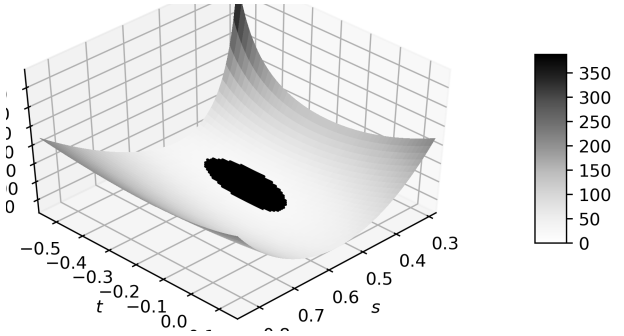
We therefore define new basis vectors as the normalized gradient vector $s_{vec} = (0.427, 0.904)$ and its orthogonal vector $t_{vec} = (-0.904, 0.427)$. We can transform between the two coordinate systems using

$$\begin{bmatrix} s \\ t \end{bmatrix} = \begin{bmatrix} 0.427 & 0.904 \\ -0.904 & 0.427 \end{bmatrix} \begin{bmatrix} \sigma \\ q_0 \end{bmatrix}$$

Similarly to figure 9a we can project the area of uncertainty onto the log likelihood landscape in our new coordinate system, and see that it aligns with the primary axes (figure 9b).



(a) Log likelihood landscape with 99% confidence interval (black ellipsoid) superimposed.



(b) Log likelihood in rotated coordinate system aligned with directions of maximum and minimum variability from the minimum. Black ellipsoid represents 99% confidence.

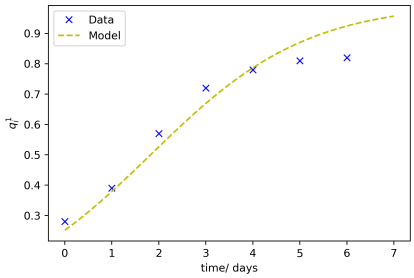
In this case, we get $s_{min} = 0.48 \pm 0.05$ and $t_{min} = -0.27 \pm 0.16$ and we have thus succesfully transformed our coordinate system into a high-variability axis s and a low-variability axis t, with movement along the t-axis only slightly perturbing the log likelihood. This also allows us to describe our full region of uncertainty using a simple equation for an ellipse in two dimensions given by

$$(\frac{s - 0.48}{0.05})^2 + (\frac{t + 0.27}{0.16})^2 = 1$$

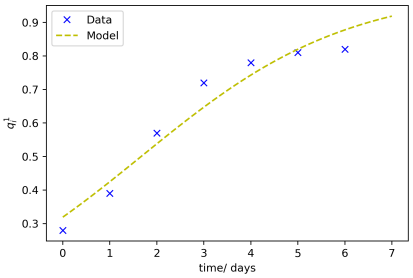
Which we can rewrite in terms of σ and q_0 as

$$(\frac{0.427\sigma + 0.904q_0 - 0.48}{0.05})^2 + (\frac{0.427q_0 - 0.904\sigma + 0.27}{0.16})^2 = 1 \tag{6}$$

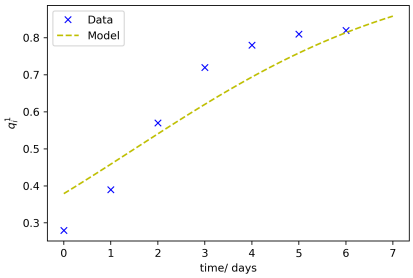
Equation (6) thus describes the extent of uncertainty in our original parameters. We can now consider what happens to the model fit as we move along the low-variability t-axis. We note that increasing t leads to reduced σ and increased q_0. To get a qualitative picture of what happens to our model, we plot the data from table 4 together with the models generated by fixing s at 0.48 and setting t to -0.43 (lowest end of confidence interval), -0.27 (optimum value) or -0.13 (highest end of confidence intercal) in figure 10a-10c.



(a) (s, t) = (0.48, -0.43)
(σ, q0) = (0.60, 0.25); L = -25.2



(b) (s, t) = (0.48, -0.27)
(σ, q0) = (0.45, 0.32); L = -20.6



(c) (s, t) = (0.48, -0.13)
(σ, q0) = (0.33, 0.38); L = -24.9

We find that the final term of the log likelihood sum at $t_k = 6$ dominates the log likelihood at smaller t, and reducing this term makes up for the worse fit at lower t_k when increasing t. If we were to compare the three fits in figure 10a-10c using an RMS error rather than binomial probability, we find fit 10b to be slightly better at 4.45 than 10a at 5.35, but both of them to be much better than 10c at 7.12. However, none of the parameter sets seem to capture all the features of the data.

These considerations raise two important points. The first is that our conclusions depend strongly on how we define our penalty function. The second is that in the present case, we do not seem to be able to find a fit that can account for the data at both low and high t_k, suggesting that our model is erroneous. This in turn suggests that equation (5) does not provide a good description of the change in frequency of q_i^1.

Alternative models, possibly involving multiple loci, are thus necessary. If, for example, the present locus is a passenger locus linked to a driver locus rather than undergoing selection itself, that might explain why the data asymptotes at 0.8-0.9 rather than at 1 as suggested by equation (5). A simple way to investigate this is to rewrite our equation of motion to include a threshold parameter γ .

$$\frac{dq_i^1}{dt} = \sigma q_i^1(\gamma - q_i^1) \tag{7}$$

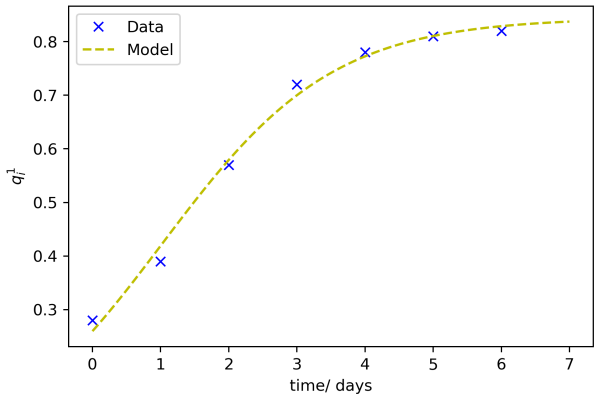
Here, γ specifies the threshold frequency, which is determined by how strongly linked our passenger locus is to a driver locus. Solving equation (7) yields an expression for $q_i^1(t)$.

$$q_i^1(t) = \gamma \frac{q_0 \exp(\sigma\gamma t)}{\gamma - q_0 + q_0 \exp(\sigma\gamma t)} \tag{8}$$

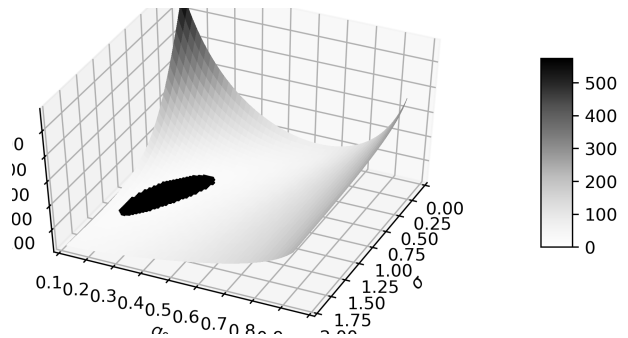
We note that setting $\gamma = 1$, this simplifies to our previous case of locus i being a driver itself. We now include γ as a third parameter of our model and learn optimum parameters by numerical steepest descent. This gives $\sigma = 0.943$, $q_0 = 0.260$, $\gamma = 0.844$ and $L_{min} = 17.217$

To determine whether this 3-parameter model gives a better description of the data, we use Baye's Information Criterion (BIC). The threshold model gives $BIC = k\log[n] - 2L = 3\log[7] - 2(-17.217) = 40.27$ compared to $BIC = 45.07$ for the previous 2-parameter model. We thus conclude that the threshold-model is a better fit to the data, as can also be seen qualitatively in figure 11a.

We also plot the log likelihood landscape for $\gamma = 0.844$ as a function of σ and q_0 in figure 11b and see that it is more uniform along the σ-axis than in figure 9a. This is to be expected since we have only 7 data points, so by fixing the the lower and upper bounds of the sigmoid via q_0 and γ, the shape of the interpolation as determined by σ is less important. We thus get high confidence in $q_0 = 0.26 \pm 0.10$ and $\gamma = 0.84 \pm 0.09$, but low confidence in $\sigma = 0.94 \pm 0.56$ when considering the full range of parameter sets correponding to $L > L_{thresh}$ in our original coordinate system.



(a) best fit three-parameter model for passenger mutation.



(b) best fit three-parameter model for passenger mutation.

In summary, we thus conclude that our locus of interest appears to be a passenger locus, the selection of which is driven by linkage to a driver locus. The strength of the linkage determines the threshold frequency γ at locus i and we find that $\gamma = 0.844$ suggesting relatively strong linkage. This allows us to fit our data using a simple model where the increase of allele 1 is proportional to its current frequency as well as its distance from threshold.

Appendix

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Thu Jan 31 20:00:55 2019

@author: kris
"""

import matplotlib
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt
import numpy as np
from matplotlib import cm
from scipy.misc import factorial


t = np.array(range(7))

n1 = np.array([28, 39, 57, 72, 78, 81, 82])
n0 = 100 - n1

N = 100

sigs = np.linspace(0, 2, N)
q0s = np.linspace(0.05,0.93,N)


prefactor = factorial(100) / (factorial(n1) * factorial(n0))

Ls = np.zeros((N,N))

def calcl(sig, q0, gamma = 1, t=t, prefactor=prefactor, n1=n1, n0=n0, ms = False):
    #q = (q0*np.exp(sig * t)) / (1 - q0 + q0*np.exp(sig * t))
    q = gamma * (q0*np.exp(sig * gamma * t)) / (gamma - q0 + q0*np.exp(sig *gamma* t))
    L = np.sum( np.log( prefactor * q**n1 * (1-q)**n0 ) )
    if ms:
        msq = (np.mean((100*q-n1)**2))**0.5
        print(msq)
        print(np.log( prefactor * q**n1 * (1-q)**n0 ))
    return L

for i, sig in enumerate(sigs):
    for j, q0 in enumerate(q0s):
        Ls[i,j] = calcl(sig, q0)

sigs = np.repeat(sigs,N).reshape(N,N)
q0s = np.transpose(np.repeat(q0s,N).reshape(N,N))

fig = plt.figure(figsize = (7,4))
ax = fig.add_subplot(111, projection='3d')

Ls = -Ls
surf = ax.plot_surface(sigs, q0s, Ls, cmap=cm.Greys, antialiased = False,
                       vmin = 0, vmax = np.amax(Ls))
fig.colorbar(surf, shrink=0.5, aspect=6, pad=0.05)
```

```
#make things look nice
ax.set_xlabel('$\sigma$')
ax.set_ylabel('$q_0$')
ax.set_zlabel('$-L$')
plt.xlim(0,2)
plt.ylim(0,1)
#save and show
ax.view_init(45, 325)
ax.dist = 10.5
plt.savefig('Lsurf.png', dpi=360)

plt.show()


def sdescent(q0 = 0.05, sig = 0, gamma=1, d = 10**(-5), g = 10**(-7),
             thresh = 10**(-12), pas = False):


    L = -calcl(sig, q0, gamma=gamma)
    gammas = [gamma]

    err = thresh+1

    sigs = [sig]
    q0s = [q0]
    Ls = [L]

    niter = 0
    print(niter, sig, q0, L)
    while err > thresh and niter < 10**9:
        niter += 1

        L0 = L

        newsig = sig+d
        newq = q0+d


        d1 = -calcl(newsig, q0, gamma=gamma)
        d2 = -calcl(sig, newq, gamma=gamma)

        if pas:
            newgam = gamma+d
            d3 = -calcl(sig, q0, gamma=newgam)
            gamma = gamma + (L - d3)/d*g
            gammas.append(gamma)

        sig = sig + (L - d1)/d*g
        q0 = q0 + (L - d2)/d*g

        L = -calcl(sig, q0, gamma=gamma)

        sigs.append(sig)
        q0s.append(q0)
        Ls.append(L)

        err = np.abs(L0-L)
```

```
        if niter % 10000 == 0: print(niter, sig, q0, gamma, L, err)


    if pas:
        print(sig, q0, gamma, L)
        return sig, q0, gamma, L, sigs, q0s, gammas, Ls

    else:
        print(sig, q0, L)
        return sig, q0, L, sigs, q0s, Ls

#sigmin, q0min, Lmin, sigsp, q0sp, Lsp = sdescent()
#sigmint, q0mint, gmint, Lmint, sigspt, q0spt, gspt, Lspt = sdescent(pas = True)

def fit_model(sig = sigmin, q0=q0min, gamma=1.0, ts = np.linspace(0,7,100),
              ext = '', pas=False):

    if pas:n = 100 * gamma * \
        (q0*np.exp(sig * gamma * ts)) / (gamma - q0 + q0*np.exp(sig *gamma* ts))
    else: n = (q0*np.exp(sig * ts)) / (1 - q0 + q0*np.exp(sig * ts))*100
    plt.plot(t, n1/100, 'bx')
    plt.plot(ts, n/100, 'y-')
    plt.xlabel('time/ days')
    plt.ylabel('$q_i^1$')
    plt.legend(['Data','Model'])
    plt.savefig('fit_model'+ext+'.png', dpi=360)
    plt.show()

fit_model()

plt.plot(t, n1, 'bx')
plt.xlabel('time/ days')
plt.ylabel('$n_i^1$')
plt.savefig('data.png', dpi=360)
plt.show()

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

fig = plt.figure(figsize = (7,4))
ax = fig.add_subplot(111, projection='3d')

surf = ax.plot_surface(sigs, q0s, Ls, cmap=cm.Greys, antialiased = False,
                       vmin = 0, vmax = np.amax(Ls))
fig.colorbar(surf, shrink=0.5, aspect=6, pad=0.05)

#make things look nice
ax.set_xlabel('$\sigma$')
ax.set_ylabel('$q_0$')
ax.set_zlabel('$-L$')
plt.xlim(0,2)
plt.ylim(0,1)
#save and show
ax.view_init(45, 355)
ax.dist = 10.5

plt.plot(sigsp, q0sp, np.array(Lsp)+1, 'k-')
```

```
plt.savefig('Lsurfproj.png', dpi=360)

plt.show()

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

fig = plt.figure(figsize = (7,4))
ax = fig.add_subplot(111, projection='3d')

surf = ax.plot_surface(sigs, q0s, Ls, cmap=cm.Greys, antialiased = False,
                       vmin = 0, vmax = np.amax(Ls))
fig.colorbar(surf, shrink=0.5, aspect=6, pad=0.05)

#make things look nice
ax.set_xlabel('$\sigma$')
ax.set_ylabel('$q_0$')
ax.set_zlabel('$-L$')
plt.xlim(0,2)
plt.ylim(0,1)
#save and show
ax.view_init(45, 355)
ax.dist = 10.5

Lthresh = Lmin+np.log(100)
coords = (Ls < Lthresh) #& (Ls > Lthresh-0.5)
sthresh = sigs[coords]
q0thresh = q0s[coords]
Lsthresh = Ls[coords]

print(np.amin(sthresh), sigmin, np.amax(sthresh))
print(np.amin(q0thresh), q0min, np.amax(q0thresh))

plt.plot(sthresh, q0thresh, Lsthresh, 'k-')

plt.savefig('Lsurfthresh.png', dpi=360)

plt.show()

#########

l0 = -calcl(sigmin, q0min)
delta = 0.001
ls = -calcl(sigmin+delta, q0min)
lq = -calcl(sigmin, q0min+delta)

print((ls-l0)/delta, (lq-l0)/delta)

svec = np.array([(ls-l0)/delta, (lq-l0)/delta])
tvec = np.array([(l0-lq)/delta, (ls-l0)/delta])
svec = svec/np.linalg.norm(svec); tvec = tvec/np.linalg.norm(tvec)
mat = np.array([svec, tvec])

np.dot(mat, np.array([0,0.05]))
np.dot(mat, np.array([2,0.05]))
np.dot(mat, np.array([0,0.93]))
np.dot(mat, np.array([2,0.93]))
```

```
minv = np.linalg.inv(mat)


N = 100

scoords = np.linspace(0.29,0.85, N)
tcoords = np.linspace(-0.552,0.15,N)


prefactor = factorial(100) / (factorial(n1) * factorial(n0))

Lnews = np.zeros((N,N))

for i, scoord in enumerate(scoords):
    for j, tcoord in enumerate(tcoords):
        sig, q0 = np.dot(minv, np.array([scoord,tcoord]))
        #print(scoord, tcoord, sig, q0)
        Lnews[i,j] = calcl(sig, q0)

ss_p = np.repeat(scoords,N).reshape(N,N)
ts_p = np.transpose(np.repeat(tcoords,N).reshape(N,N))

fig = plt.figure(figsize = (7,4))
ax = fig.add_subplot(111, projection='3d')

Lnews = -Lnews
surf = ax.plot_surface(ss_p, ts_p, Lnews, cmap=cm.Greys, antialiased = False,
                       vmin = 0, vmax = np.amax(Lnews))
fig.colorbar(surf, shrink=0.5, aspect=6, pad=0.05)

smin, tmin = np.dot(mat, np.array([sigmin, q0min]))

plt.plot([smin], [tmin], [Lmin], 'ko')

coords = (Lnews < Lthresh) #& (Ls > Lthresh-0.5)
ssthresh = ss_p[coords]
ttthresh = ts_p[coords]
LLsthresh = Lnews[coords]

print(np.amin(ssthresh), smin, np.amax(ssthresh))
print(np.amin(ttthresh), tmin, np.amax(ttthresh))

plt.plot(ssthresh, ttthresh, LLsthresh, 'k-')

#make things look nice
ax.set_xlabel('$s$')
ax.set_ylabel('$t$')
ax.set_zlabel('$-L$')
plt.xlim(0.29,0.85)
plt.ylim(-0.552,0.15)
#save and show
ax.view_init(45, 45)
ax.dist = 10.5
plt.savefig('Lsurfrot.png', dpi=360)

plt.show()
```

```
sig1, q01 = np.dot(minv, np.array([smin, tmin-0.16]))
fit_model(sig=sig1, q0=q01, ext = '1')
sig2, q02 = np.dot(minv, np.array([smin, tmin+0.14]))
fit_model(sig=sig2, q0=q02, ext = '2')
sig3, q03 = np.dot(minv, np.array([smin, tmin-0.32]))
fit_model(sig=sig3, q0=q03, ext = '3')

fit_model(sig=sigmint, q0=q0mint, gamma=gmint, pas=True, ext = 'pas')


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

sigst = np.linspace(0, 2, N)
q0st = np.linspace(0.12,0.93,N)

Lst = np.zeros((N,N))

for i, sig in enumerate(sigst):
    for j, q0 in enumerate(q0st):
        Lst[i,j] = calcl(sig, q0, gamma = gmint)

sigst = np.repeat(sigst,N).reshape(N,N)
q0st = np.transpose(np.repeat(q0st,N).reshape(N,N))

fig = plt.figure(figsize = (7,4))
ax = fig.add_subplot(111, projection='3d')

Lst = -Lst


Lthresht = Lmint+np.log(100)
coordst = (Lst < Lthresht) #& (Ls > Lthresh-0.5)
sthresht = sigst[coordst]
q0thresht = q0st[coordst]
Lsthresht = Lst[coordst]

print(np.amin(sthresht), sigmint, np.amax(sthresht))
print(np.amin(q0thresht), q0mint, np.amax(q0thresht))

surf = ax.plot_surface(sigst, q0st, Lst, cmap=cm.Greys, antialiased = False,
                       vmin = 0, vmax = np.amax(Lst))
fig.colorbar(surf, shrink=0.5, aspect=6, pad=0.05)
plt.plot(sthresht, q0thresht, Lsthresht, 'k-')
#make things look nice
ax.set_xlabel('$\sigma$')
ax.set_ylabel('$q_0$')
ax.set_zlabel('$-L$')
plt.xlim(0,2)
plt.ylim(0.1,1)
#save and show
ax.view_init(45, 25)
ax.dist = 10.5
plt.savefig('Lsurfpas.png', dpi=360)

plt.show()
```

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

'''
sigst = np.linspace(0, 2, N)
q0st = np.linspace(0.12,0.93,N)
gst = np.linspace(0.15,1.0,N)

Lst = np.zeros((N,N,N))

for i, sig in enumerate(sigst):
    print('new i', i)
    for j, q0 in enumerate(q0st):
        for k, gamma in enumerate(gst):
            Lst[i,j,k] = calcl(sig, q0, gamma = gamma)

print('filled out')

sigst =  np.repeat(sigst,N*N).reshape(N,N,N)
q0st = np.repeat( np.transpose(np.repeat(q0st,N).reshape(N,N)), N).reshape(N,N,N)
gst =  np.transpose(np.repeat(gst,N**2).reshape(N,N,N))

print('made arrays')

Lst = -Lst


Lthresht = Lmint+np.log(100)
coordst = (Lst < Lthresht) #& (Ls > Lthresh-0.5)
sthresht = sigst[coordst]
q0thresht = q0st[coordst]
gthresht = gst[coordst]
Lsthresht = Lst[coordst]

print('found threshs')

print(np.amin(sthresht), sigmint, np.amax(sthresht))
print(np.amin(q0thresht), q0mint, np.amax(q0thresht))
print(np.amin(gthresht), gmint, np.amax(gthresht))
'''


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


dirs = ['1_mut', '2_mut', '3_mut']
newlines = [ [87], [1, 97], [1,74,112], [], []  ]
cols = [ ['b'], ['y', 'b'], ['y', 'b', 'b'], [], [] ]

for n in range(1,6):

    f1 = open('/Applications/optimist/'+str(n)+'_mut/Model_frequencies.out', 'r')
    f2 = open('/Applications/optimist/'+str(n)+'_mut/Real_frequencies.out', 'r')

    dat = []
    ltypes = ['-', 'x']
    for i, f in enumerate([f1, f2]):
        ts = []
```

```
        fr1 = []
        fr2 = []
        for line in f:
            split = [float(val) for val in line.split()]
            ts.append(split[0])
            fr1.append(split[1])
            fr2.append(split[2])

        dat.append([ts, fr1, fr2])

        plt.plot(ts, fr1, 'b'+ltypes[i] )
        plt.plot(ts, fr2, 'y'+ltypes[i] )

    for i, val in enumerate(newlines[n-1]):
        plt.axvline(x=val, color = cols[n-1][i], linestyle = ':')

    plt.xlabel('generation')
    plt.ylabel('frequency')
    plt.legend(['Model frequency 1', 'Model frequency 2', 'Real frequency 1',
                'Real frequency2'])
    plt.savefig('fit_mut'+str(n)+'.png', dpi=360)
    plt.show()

    f1.close()
    f2.close()

muts = []
lls = []
with open('/Applications/optimist/systematic_lls', 'r') as f:
    for i, line in enumerate(f):
        muts.append(i+1)
        lls.append(float(line))

AIC = 2.5 * 2*np.array(muts) - np.array(lls)
plt.plot(muts, -np.array(lls), 'b-')
plt.plot(muts, AIC, 'y-')
plt.legend(['-(Log likelihood)', 'AIC'])
plt.xlabel('number of mutations')
plt.ylabel('likelihood score')
plt.savefig('lls.png', dpi = 360)
plt.show()



pi = 0.9
a = 0.8
b = 0.9
c = 0.7

p = np.linspace(0,1,1000)

f = pi*(p*p*a + 2*(1-p)*p*b + (1-p)*(1-p)*c) + (1-pi)*(p*p + 2*(1-p)*p + (1-p)*(1-p)*c)

plt.plot(p, f, 'b-')
plt.xlim([0,1])
plt.xlabel('p')
plt.ylabel('mean fitness')
```

```
#plt.savefig('fitness.png', dpi=360)
plt.show()

df=pi*(2*p*a + 2*(1-2*p)*b + (2*p-2)*c)+(1-pi)* (2*p + 2*(1-2*p) + (2*p-2)*c)

plt.plot(p, df, 'b-')
plt.plot(p,np.repeat(0.0, len(p)), 'k-')
plt.xlim([0,1])
plt.xlabel('p')
plt.ylabel('dF')
#plt.savefig('dfitness.png', dpi=360)
plt.show()

N = 1000

b = np.linspace(0,1,N)
b = np.repeat(b,N).reshape(N,N)
p = np.linspace(0,1,1000)
p = np.transpose(np.repeat(p,N).reshape(N,N))

f = pi*(p*p*a + 2*(1-p)*p*b + (1-p)*(1-p)*c) + (1-pi)*(p*p + 2*(1-p)*p + (1-p)*(1-p)*c)

fig = plt.figure(figsize = (7,4))
ax = fig.add_subplot(111, projection='3d')

surf = ax.plot_surface(b, p, f, cmap=cm.Greys, antialiased = False,
                       vmin = np.amin(f), vmax = np.amax(f))
fig.colorbar(surf, shrink=0.5, aspect=6, pad=0.05)

#make things look nice
ax.set_xlabel('b')
ax.set_ylabel('p')
ax.set_zlabel('fitness')
plt.xlim(0,1)
plt.ylim(0,1)
#save and show
ax.view_init(45, 325)
ax.dist = 10.5
plt.savefig('fsurf.png', dpi=720)

plt.show()

cols = np.zeros((N, N))

for i in range(N):
    for j in range(N):
        bc = b[i,0]
        pc = p[0,j]

        if bc < 2/3:
            if pc > (1.2 - 1.8*bc)/(2.64 - 3.6*bc): cols[i,j] = 1.0
            else: cols[i,j] = 0.0

        elif bc > 0.8:
            cols[i,j] = (1.2 - 1.8*bc)/(2.64 - 3.6*bc)

        else:
```

```
            cols[i,j] = 1.0

print('finished cols')

newfig = plt.figure()
newax = newfig.add_subplot(111, projection='3d')
surf1 = newax.plot_surface(b, p, cols, cmap=cm.Greys, antialiased = False,
                           vmin = 0, vmax = 1)
plt.show()

minn, maxx = cols.min(), cols.max()
norm = matplotlib.colors.Normalize(minn, maxx)
m = plt.cm.ScalarMappable(norm=norm, cmap=cm.Greys)
m.set_array([])
fcolors = m.to_rgba(cols)

fig = plt.figure(figsize = (7,4))
ax = fig.add_subplot(111, projection='3d')
print('about to plot')
surf = ax.plot_surface(b,p,f,facecolors=fcolors, vmin=minn, vmax=maxx, shade=False)
fig.colorbar(surf1, shrink=0.5, aspect=6, pad=0.05)
ax.view_init(45, 325)
ax.dist = 10.5


#fig = plt.figure()
#ax = fig.add_subplot(111, projection='3d')
#surf = ax.plot_surface(b, p, f, cmap=cm.Greys, antialiased = False,
#     vmin = np.amin(f), vmax = np.amax(f))

#make things look nice
ax.set_xlabel('b')
ax.set_ylabel('p')
#ax.set_zlabel('final composition')
plt.xlim(0,1)
plt.ylim(0,1)
#save and show
print('saving')
plt.savefig('fsurfinit.png', dpi=720, bbox_inches='tight')
plt.show()
```