John Sy
19 March 2008

Advanced Modelling in Biology

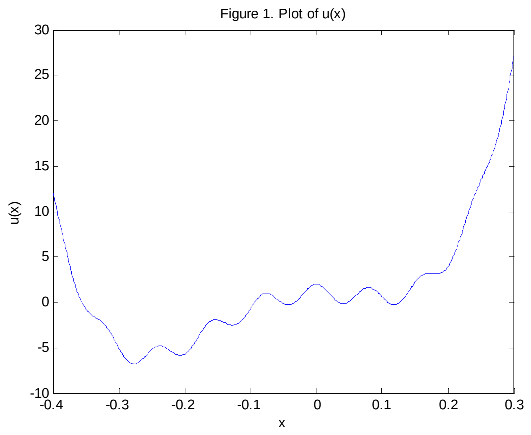
Assignment

Question 1.1: Global Optimization of Complex Functions

Local optimization (maximization or minimization) is computationally easy, especially when functions are convex and don't have any points of inflection around the point in question. However, finding global extrema, especially when the function is non-convex and has shallow minima, this can prove to be computationally much more difficult. We first consider the following example in the one-dimensional case where u(x) represents our "energy function" to be minimized.

$$u(x) = (1 + 0.5x - 130.1x^2 + 589x^3 + 2688x^4) + \cos\left(\frac{x\pi}{0.04}\right)$$

The first method of minimization one can use, and perhaps the easiest, is by simply plotting the function. This is shown in figure 1 below. Although at large values of *x* the function seems to be convex, zooming in shows that the function has multiple minima.



We can observe that there are 7 minima that we must consider, but we can also use the derivative of the energy function to determine how many minima the function has as is shown in the code below.

```
%Obtaining the number of minimia
j = 0;
for i = 2:length(x)
    if (assn1q1(x(i-1)) < 0) && (assn1q1(x(i)) > 0)
        j = j + 1;
    else
        j = j;
    end
end
```

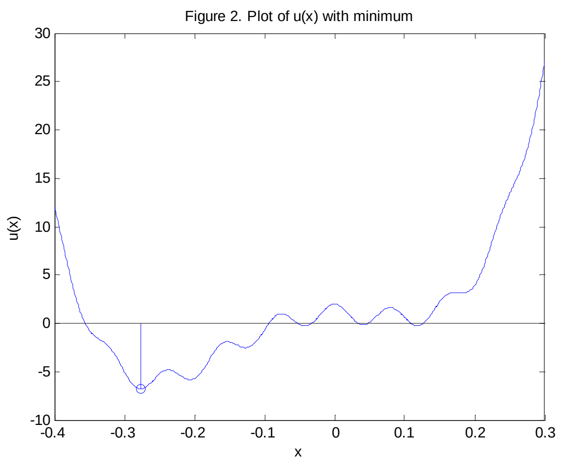
The code above yields 7 places where the derivative (assn1q1.m) changes from being negative to positive.

We can then utilize the fsolve function to get values of the minima using the approximate zeros from the code above as guesses. We can then evaluate the energy at each minima and select the lowest value to give us our global minimum. The fsolve function is an implementation of the steepest descent method. Although this does not always produce a convergent solution, having good guesses from our previous code will allow the function to converge. Furthermore, the benefit of using fsolve instead of relying on just finding when the gradient goes from a negative value to a positive value is that it is not dependent on having small step sizes. Fsolve can take any value around the minima and get the exact value for you. This allows long programs to run faster since we are minimizing the sampling rate of the derivative.

```
%Obtaining the number of minimia & global minimum value
j = 0;
for i = 2:length(x)
    if (assn1q1(x(i-1)) < 0) && (assn1q1(x(i)) > 0)
        j = j + 1;
        val(j) = fsolve(@assn1q1, x(i-1));
        en(j) = metroMC(val(j)); %metroMC is our energy function
    else
        j = j;
    end
end

%Selecting minimum x value and energy
minEN = 0;
for i = 1:length(val)
    if en(i) < minEN
        minEN = en(i);
        minX = val(i);
    end
end
minEN
minX
```

This slightly altered code gives us that the global minimum can be found when *x* = -0.2767 when the energy value is -6.7870. To ensure that we are indeed getting the minimum value, we can plot the results on top of the graph from figure 1. This result is displayed in figure 2.



One can also use ode45 to obtain the value of the global minima. This method works by integrating the negative gradient of the function with respect to time. This method is equivalent to standing at an initial condition and always going down the gradient of the energy function over time, similar to the steepest descent method. However, like the fsolve method, this does not guarantee that the global minimum is found. Probably the best we can do is sample several points and find the minimum energy that is reached.

If we now look to performing the same type of analysis on a two dimensional problem, this problem gets more computationally difficult since we have to take into consideration the minimum value of a surface. Now the function to be minimized is the following:

$$u(x,y) = (1 + 0.5x - 130.1x^2 + 589x^3 + 2688x^4) + \cos\left(\frac{x\pi}{0.04}\right) + (1 + 0.5y - 130.1y^2 + 589y^3 + 2688y^4) + \cos\left(\frac{y\pi}{0.04}\right)$$

However, we can consider the symmetry of our specific problem to simply get that the energy minimum value will be located at (-0.2767, -0.2767) with a value of -13.5740, knowing the results from the previous section. This is because the u(x,y) value is only dependent upon the sum of x terms and y terms and it is not interdependent. Although this argument might work for this problem, it cannot be generalized to more complex problems that are not symmetrical. For these cases, it is not possible to use the fsolve() function in matlab since that is only valid for one variable.

However, another similar function called fminsearch() is available for us to use which finds the local minima of a function. This time, we can again sample various points on a grid where we believe the global minimum lies and then save the lowest value and the x value which gives that lowest energy. This is implemented in the code below.

```
%To do the 2 dimensional minimization
myfunc = @(x)(1+0.5*x(1)-130.1*x(1)^2+589*x(1)^3+2688*x(1)^4+cos(x(1)*pi/0.04)+1+0.5*x(2)-
130.1*x(2)^2+589*x(2)^3+2688*x(2)^4+cos(x(2)*pi/0.04));
xguessvals = -0.4:0.01:0.3;
yguessvals = -0.4:0.01:0.3;
minval = 0;
minx = 0;
for i = 1:length(xguessvals)
    xguess = xguessvals(i);
    for i = 1:length(yguessvals)
        yguess = yguessvals(i);
        [x, fval] = fminsearch(myfunc, [xguess yguess]);
        if fval < minval
            minval = fval;
            minx = x;
        end
    end
end
minval
minx
```

This method gives us the value of the global minimum quite quickly, but if our space is too large to sample along a mesh, then we might to obtain the correct value.

To get around this problem, we need to use a heuristic search algorithm such as metropolis monte carlo or simulated annealing to search for the global minima. Pseudocode for a monte carlo heuristic search algorithm is shown below.

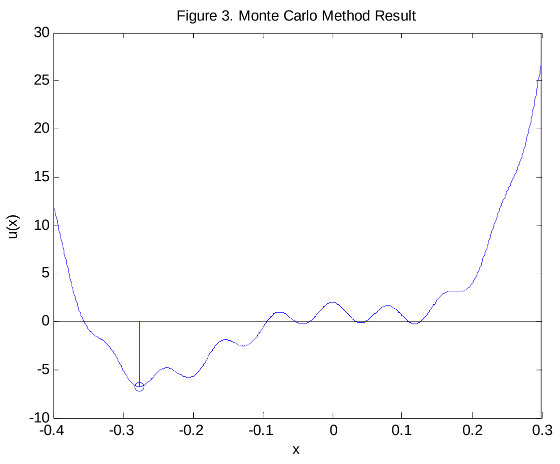
```
Pick a random number in a given range
Calculate the energy at that value
If new energy < old energy
        Accept change
Also accept given probability determined by exp(-dE/kT)
Else
```

```
        Do not accept change
Decrease the temperature over time
Repeat n times
```

Running the implementation below for the one dimensional case seems to work quite well to find the minima. The implementation pretty much throws many darts and allows changes to occur given a probability if the energy change is upward. This allows for the algorithm to escape local minima in search of the global minimum. As temperature decreases even further, the probability that large variations will be accepted decreases and only small variations in energy are accepted with successive iterations.

```
%We want to confine ourselves within a certain window where we know the
%minima lie:  -0.5 to 0.5
%Select a random number between -0.5 and 0.5 to initiate the algorithm
xvalue(1) = rand()-0.5;
%Calculate the energy of the point
en(1) = metroMC(xvalue(1));
%Initial Temperature
T = 5;
for i = 2:20000
    %Perturb the x value randomly up or down (random walk)
    xvaluenew = xvalue(i-1) + randn();
    %Clalculate the new energy
    ennew = metroMC(xvaluenew);
    %Accept if energy is less
    randnum = rand();
    if ennew < en(i-1)
        en(i) = ennew;
        xvalue(i) = xvaluenew;
    %Accept sometimes given a probability
    elseif randnum < exp(-(ennew-en(i-1))/T)
        en(i) = ennew;
        xvalue(i) = xvaluenew;
    %Otherwise, don't accept the change
    else
        en(i) = en(i-1);
        xvalue(i) = xvalue(i-1);
    end
    %Decrease temperature slowly to minimize jumpiness
    T = T*0.99;
end
Energy = en(length(en))
MinXValue = xvalue(length(xvalue))
```

We can again visualize the effectiveness of the algorithm by plotting the result on the graph as shown in figure 3.



Moving to two dimensions, this becomes slightly more difficult and it takes several runs to finally find the minimum. The code is shown below, implementing a similar algorithm to above except now our random walk is now in the direction of the gradient as it is better to have a directed approach.

```
%Monte carlo methods on a 2D system
%Set a  random initial guess on the interval -0.5 to 0.5, since we know
%that is where the minima lie
xvalue(1) = rand()-0.5;
yvalue(1) = rand()-0.5;
%Calculate energy (value of function)
en(1) = metroMC2D(xvalue(1),yvalue(1));

RT = 10; %Initial temperature
%Implementation of the Metropolis-Monte Carlo Algorithm
for i = 2:50000
    %Calculate gradient at point
    xgrad = assn1q1(xvalue(i-1));
    ygrad = assn1q1(yvalue(i-1));
    %Perturb the point (random walk simulation) on the gradient
    random1 = randn();
    newxval = xvalue(i-1) + xgrad*random1;
    newyval = yvalue(i-1) + ygrad*random1;
    %Evaluate the energy at this random number
    newen = metroMC2D(newxval,newyval);
    randnum = rand();
    if newen < en(i-1)
        en(i) = newen; %Always select if less
        xvalue(i) = newxval;
        yvalue(i) = newyval;
    %Accept with given probability sometimes for increases
    elseif randnum < exp(-(newen-en(i-1))/RT)
        en(i) = newen;
        xvalue(i) = newxval;
        yvalue(i) = newyval;
    else
        en(i) = en(i-1);
        xvalue(i) = xvalue(i-1);
        yvalue(i) = yvalue(i-1);
    end
    RT = RT*0.99; %Reduce temperature slowly
end
minxval = xvalue(length(xvalue))
minyval = yvalue(length(yvalue))
MinEn = en(length(en))
```

The minimum was found to be at the values expected when considering symmetry introduced earlier. Unfortunately, this method does not work as well for multidimensional problems as with one dimensional problems perhaps because there isn't enough iterations or the temperature is being decreased too quickly. For now, on two dimensional system which you can visualize, then it is best to use fminsolve, the standard function in matlab.

In general, for an N dimensional system similar to the one we just had, there will be 7^N number of minima that need to be considered.

Question 1.2: Multivariate Least Squares

The least squares regression method seeks to minimize the error of a curve of *n* dimensions with *m* number of points, where $m > n$. For a linear regression, we fit the curve to the general equation:

$$y_i = a_0 + a_1 x_i$$

Our m number of points can be generalized to the equation below:

$$\vec{y} = \begin{pmatrix} y_1 \\ \vdots \\ y_N \end{pmatrix} = a_0 \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix} + a_1 \begin{pmatrix} x_1 \\ \vdots \\ x_N \end{pmatrix} = \begin{pmatrix} 1 & x_1 \\ \vdots & \vdots \\ 1 & x_N \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \end{pmatrix} = \vec{x}\vec{A}$$

We can make a prediction to the line goes through the data points with y_i values being $\hat{y_i}$, and the task now becomes to minimize the error between y_i and $\hat{y_i}$. The mean squared error can then be defined below.

$$mse = e^T e = (y_i - \hat{y_i})^T (y_i - \hat{y_i})$$

Substituting $\vec{y} = \vec{x}\vec{A}$ and taking the derivative with respect to A, we get that:

$$\nabla E = -x^T y - x^T y + 2(x^T x)a$$

Setting this equation to zero to find the minimum, we get the relationship:

$$x^T y = (x^T x)a$$

And solving for *a*, the coefficients of our equation yields:

$$a = (x^T x)^{-1} x^T y$$

To ensure that we have a minimum, we can take the second gradient of the error and see if it is positive definite:

$$\nabla^2 \mathrm{E} = 2(x^T x) \geq 0$$

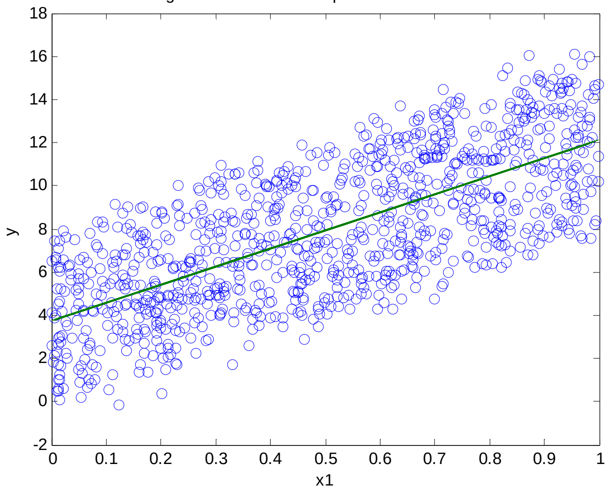
As we can see, multiplying the transpose of a vector with the vector is equivalent to taking the squares and this will always be positive. So indeed, we have succeeded in obtaining the minimum.

In matlab, this can be easily implemented with the code below to find the value of *b* for points1.dat.

```
%Least Squares on one dimension for fit 1
C = cat(2,ones(length(x1vals),1),x1vals);
b = ((transpose(C)*C)^(-1))*transpose(C)*yvals
ycalc = b(1) + b(2)*x1vals;
figure;
plot(x1vals,yvals,'o',x1vals,ycalc);
error = ycalc - yvals;
meansqerr = mean(error.^2)
% or meansqerr = mse(error)
```

We obtain that the best fit line for the points1.dat with the graph shown below

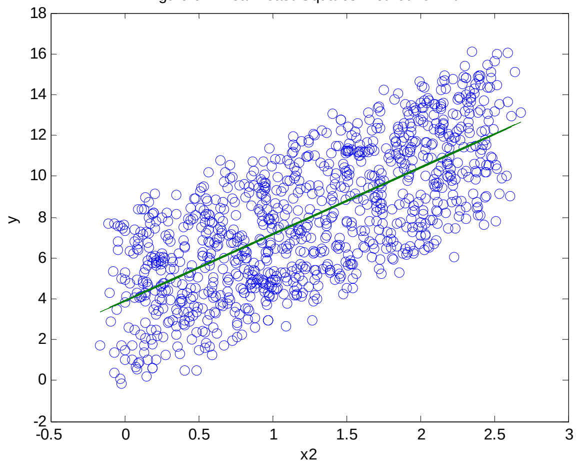
$$y = 3.7393 + 8.3811x$$



The mean squared error for this plot comes out to be 5.5609.

Similar, we can calculate the value of c with the other set of points in points1.dat. We get that the best fit line and corresponding plot is shown below. The mean squared error for this plot is 5.6367.

$$y = 3.8845 + 3.2663x$$



Now we can combine the two independent variables and say that *y* is dependent on both x_1 and x_2 and perform the linear regression correspondingly. We have to alter the code slightly to accommodate for this other variable as shown below.

```
C1 = cat(2,ones(length(x1vals),1),x1vals,x2vals);
A1 = ((transpose(C1)*C1)^(-1))*transpose(C1)*yvals

ycalc1 = A1(1) + A1(2)*x1vals + A1(3)*x2vals;
error1 = ycalc1 - yvals;
meansqerr1 = mse(error1)
```

And from this, we obtain that the best fit line is:

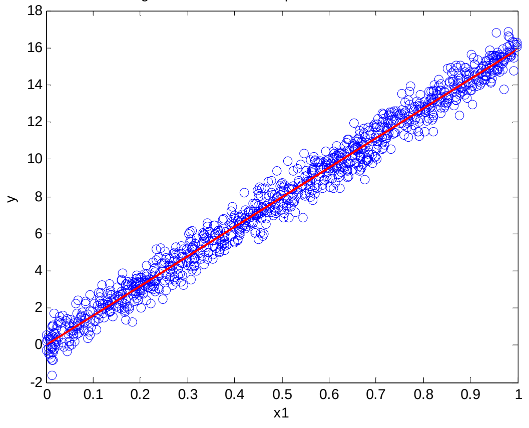
$$y = 0.0329 + 8.0221x_1 + 3.1245x_2$$

The corresponding mean squared error is 0.2654, which is much lower than any of the one-dimensional fits that we attempted. This suggests that both are principle components to the problem and both are required properly describe the data. We should remember that when we ignored one of the variables to perform the one-dimensional fit, we were taking the projection of the data along that plane. Now that we have another dimension, we can more accurately describe the data using two-dimensions.

The same methods can be applied to points2.dat. We have found the best fit line and have plotted the points and best fit line below. The mean squared error for this univariate fit is 0.3697.

$$y = 0.0005 + 15.9095x_1$$

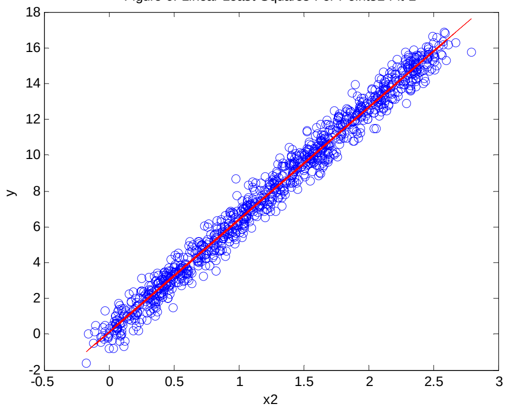
Figure 5. Linear Least Squares For Points2 Fit 1



For the second set of *x* values, we have also found the best fit line and plot. The mean squared value for this set is 0.3697.

$$y = 0.1030 + 6.2733x_2$$

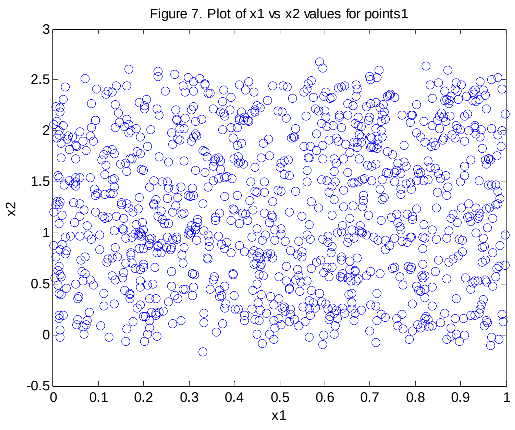
Figure 6. Linear Least Squares For Points2 Fit 2

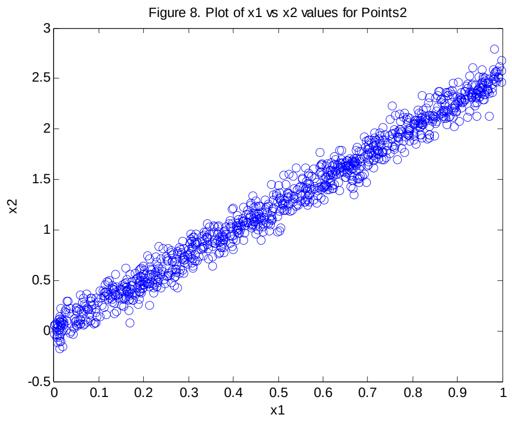


And now taking both x_1 and x_2 into consideration, we obtain that the best fit equation. The mean squared error this time is 0.2656.

$$y = 0.0127 + 7.9949x_1 + 3.1516x_2$$

Compared to the analysis on points1.dat, there is not a significant decrease in mean squared error when using a multivariate regression. We can get a further insight into this by plotting x1 vs x2 for each of the points1 and points2 and these are shown in figures 7 and 8.





As is seen in the above figures (7 and 8), in points1, the values of x1 and x2 do not show any correlation and this is confirmed when we get a large decrease in the mean squared error when we take into account both independent variables x1 and x2. In comparison, we can see in figure 8 that x1 and x2 are highly correlated. Because of this, we do not see a greater reduction in the mean squared error since either x1 depends on x2 or vice versa already. Thus, only one variable is necessary to explain the behavior of y in the system.

If we perform the singular value decomposition of the matrix with all of our data (not assuming any dependence relationships between any of the variables), we can hopefully gain a better understanding of what we see in the data and plots above. The singular values are just the eigenvalues of the covariance matrix.

To do this, we first subtract the mean value for our data sets. Then we calculate the eigenvalues and eigenvectors of the covariance matrix. For points 1, the corresponding eigenvalues and eigenvectors of the data are:

```
V =
   -0.9304    0.3617    0.0593
   -0.3492   -0.9239    0.1561
    0.1113    0.1245    0.9860


D =
    0.0034         0         0
         0    0.3008         0
         0         0   11.7707
```

And for points 2, the corresponding eigenvalues and eigenvectors of the data are:

```
V =
    0.9815    0.1814    0.0609
   -0.1888    0.9698    0.1546
   -0.0311   -0.1632    0.9861


D =
    0.0011         0         0
         0    0.0093         0
         0         0   22.1716
```

From the data above, we can see that there are two larger principal components in points 1 than in points 2 meaning that we could potentially discard 2 variables in points 2 but only one variable in points 1. Looking at the plot above, we recall that for points 1, x1 and x2 were not well correlated and the points were on the plane. Furthermore, by adding in both x1 and x2 into the least squares linear fit, we were able to get a significant decrease in the mean squared error. This means that it is not possible to discard 2 of the variables to describe the data, but can only discard one of the variables, confirmed by our principal component analysis. In points2, we recall that x1 and x2 were generally very well correlated and hence we didn't get a decrease in the mean squared error when we used both x1 and x2 to do our least squares fit. The fact that we got similar mean squared errors shows that we can discard two unnecessary variables in our data as confirmed by our principal component analysis.

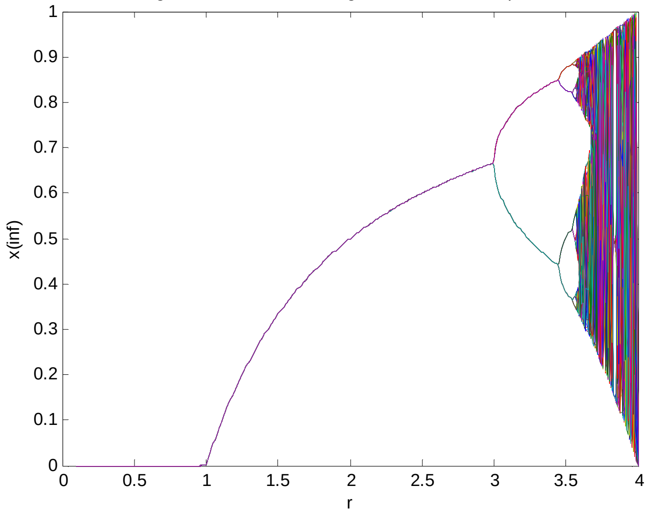
Question 2.1: A numerical exploration of the logistic map

In this problem, we consider a fishery whose population can be described by the difference equation shown below. In the equation, we have that the population at the next time step is dependent upon the population at the previous time step but is limited by the resources and so thus cannot have a population density greater than 1.

$$x_{k+1} = rx_k\left(1 - x_k\right)$$

The main parameter that is adjustable is the value of *r*, or the effective rate at which the population grows when not limited. We set the value of *r* to only lie between the values of 0 and 4. When *r* = 0, the population dies out at the next time step since there is no growth. Intuitively, we can also see that if the growth rate is less that 1, this means that there is not enough species going to the next time step and the population will eventually decrease to zero. But what happens at other values of *r*? To investigate this, we can first plot the bifurcation diagram with values of *r* versus values as *x* approaches infinity, x_∞. Since it is not computationally practical to actually obtain the value at infinity, we assume that the system will reach a steady state (if it ever reaches a steady state) at time step 1000. Graphical analysis has shown that the system can oscillate between different points, so to ensure that we have sampled enough time points to get all values *x* oscillates through, we take the last 200 time points from 800 to 1000. Below in figure 2.1a is the bifurcation diagram of the system along with the code used to produce the diagram.

Figure 2.1a. Bifurcation diagram for difference equation



```
rvals = 0.1:0.01:4;
for j = 1:length(rvals)
    r = rvals(j);
    x(j,1) = 0.5;
    for i = 2:1000
        x(j,i) = r*x(j,i-1)*(1-x(j,i-1));
    end
end

figure;
plot(rvals,x(:,800:1000))
axis([0 4 0 1])
title('Figure 2.1a. Bifurcation diagram for difference equation');
xlabel('r'); ylabel('x(inf)');
```

In figure 2.1a, we can see that the fixed point begins to move away from 0 as we increase the value of *r* and we observe different regimes of behaviour in the system. At *r* = 3, the system begins to oscillate between two fixed points (period 2) and as we increase *r* further past about 3.5, the system oscillates between 4 fixed points (period 4) and becomes chaotic soon after. This effect is known as period doubling. Even though it is not possible to see this or chaotic behaviour in one dimensional continuous systems, this effect is clearly seen in this seemingly simple one dimensional discrete system.

We can explore the function more analytically by finding the fixed points and performing stability analysis, similar to what we did with continuous systems last term. However to find the fixed points in discrete systems, we set $x_{t+1} = x_t$ and solve for the values of x^*. Doing this, we get that the fixed points are at $x^* = 0$ and $x^* = 1 - \frac{1}{r}$. If we constrain the value of *x* to be positive and real, then this puts a constraint on the stability of the fixed point at $x^* = 1 - \frac{1}{r}$. One can easily see that this fixed point is not valid for *r* less than 1. On the other hand, the fixed point at $x^* = 0$ will always exist as this is independent of the value of *r*. What about the stability of these two points? To perform linear stability analysis, we take the derivative of the equation and evaluate it at the fixed point. This procedure is derived similarly to deriving the linear stability analysis of continuous systems. Doing this for the system presented yields the following:

$$\left.\frac{df}{dx_k}\right|_{x^*} = r(1 - rx_k)$$

Instead of having limiting value at 0, the limiting value for difference equations is 1 such that stability is given by the condition $\left|\frac{df}{dx_k}\right| < 1$. Evaluating the above derivative at $x^* = 0$ gives $\left.\frac{df}{dx_k}\right|_{x^*=0} = r$. As *r* is also positive and real, this fixed point is only stable for values of *r* < 1.

Now we can also evaluate the derivative at $x^* = 1 - \frac{1}{r}$. This time we get that $\left.\frac{df}{dx_k}\right|_{x^*=1-\frac{1}{r}} = 2 - r$. Setting the condition for stability ($-1 < 2 - r < 1$), we obtain that this fixed point is only stable between 1 and 3. As seen in the bifurcation diagram in figure 2.1a, we indeed see that at *r* = 3, the behaviour of the system changes to being oscillatory. Using cobweb diagrams, we can gain a greater intuition of what is going on. Cobweb analysis allows us to visualize the time evolution of the system on the phase plane, similar to looking at the trajectories of the system.

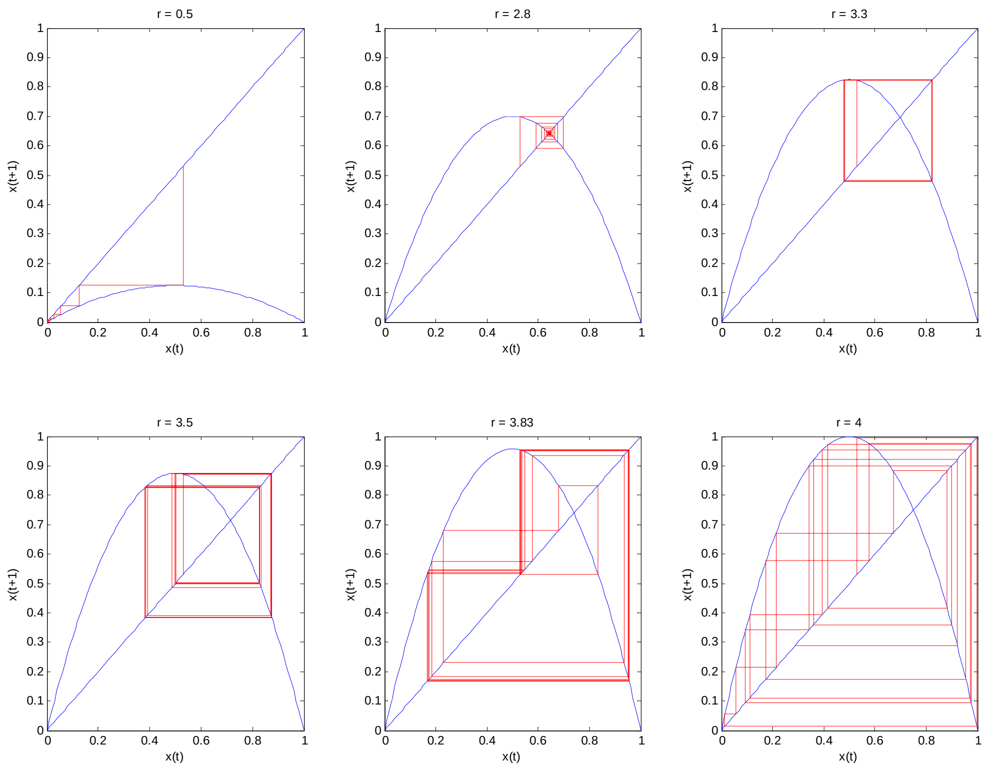
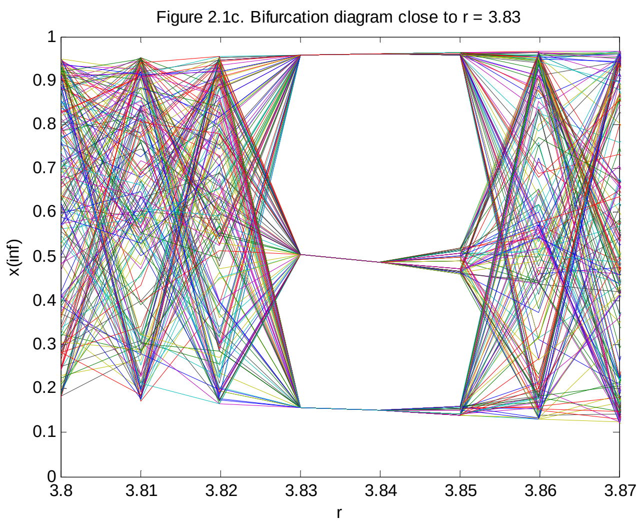


Figure 2.1b. Cobweb analysis for various values of *r*.

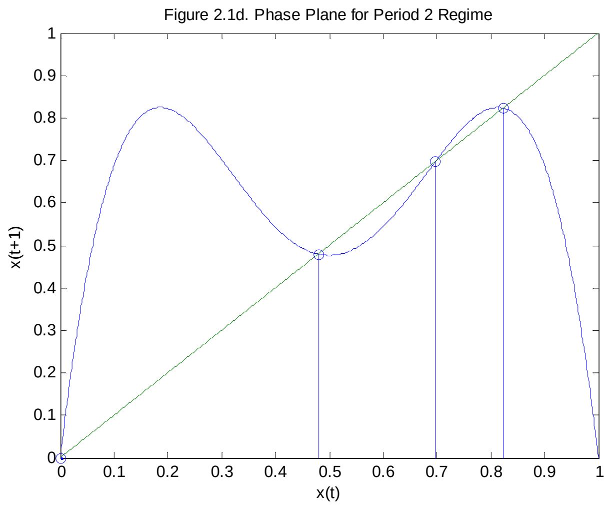
Up to values of *r* = 3.3, our analysis from above is confirmed. When *r* = 0.5, we are in the regime where the stable fixed point is at $x^* = 0$. For *r* = 2.8, we see that we have changed to the regime where the stable fixed point is at $x^* = 1 - \frac{1}{r} = 0.64$. Above the value of *r* = 3, we see oscillatory regimes beginning with period 2 (*r* = 3.3), period 4 (*r* = 3.5). When *r* = 3.83, we see something quite remarkable, a period 3 solution, and when *r* = 4, the behaviour becomes chaotic. For *r* = 3.3, 3.5, and 3.83, the attractor of the system is a limit cycle. If we look again at the bifurcation diagram close to the 3.83 value (Figure 2.1c), we can see that this regime is actually a window of periodicity within the chaotic regime.



Now looking back at r = 3.3, we can clearly see that this is within the period 2 oscillatory regime, but how can we actually determine the points at which it oscillates? We can make the assumption that for a period 2 oscillator $x_{k+2} = x_k$. If we define $x_{k+1} = f(x_k)$, we can then define $x_{k+2} = f(f(x_k))$. Then, we only need to find the fixed points of this system to find the points between which the system oscillates in this regime. We can do this with matlab both graphically and numerically. The equation we now need to solve for the fixed points is:

$$x_{k+2} = rrx_k(1-x_k)(1-rx_k(1-x_k))$$

We can use the fsolve function to find the values which satisfy $x_{k+2} = x_k$, which represent our fixed points. We see that we have 4 fixed points at $x^* = 0, 0.4794, 0.6970, \text{and } 0.8236$ which are graphically shown on the phase plane in figure 2.1d below.



We can also consider the stability of these points to see which of the 4 points the system oscillates between. We now use the criteria $\left|\frac{df(f(x_k))}{dx_k}\right| < 1$ to determine if a point is stable or not. For $x^* = 0$, we obtain that the derivative is 10.89, for $x^* = 0.4794$, the derivative is -0.29, for $x^* = 0.6970$, the derivative is 1.69, and for $x^* = 0.8236$, the derivative is -0.29.

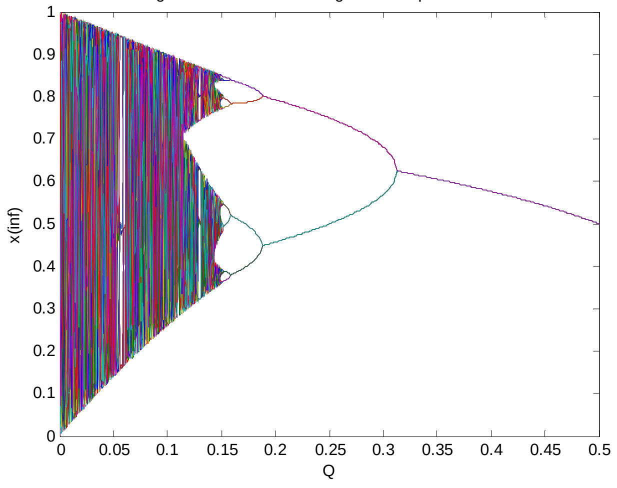
Only the fixed points $x^* = 0.4794$ and $x^* = 0.8236$ have evaluated derivatives whose absolute value is less than or equal to 1, and hence these values are the two values the system oscillates between when $r = 3.3$. This is shown graphically in figure 2.1b.

What happens when we start to harvest the fish at certain time points such that the equation now becomes:

$$x_{k+1} = rx_k(1 - x_k) - Q$$

In the equation above, Q is the depletion term and is constant for each time step. We can do a similar analysis to above, except now we fix the value of *r* to be 4 and see what happens as we change the parameter Q. The bifurcation diagram for sweeping values of Q is shown below for values between 0.1 and 0.5 and we can easily see the different regimes which we saw without the depletion term. However now, we see that the effect of Q is opposite to the effect of *r*. With increasing values of Q, we actually move away from the chaotic behaviour of the system and go into an oscillatory period followed finally by a steady state period, ultimately dying off when the value of Q goes beyond a certain point (related to the initial conditions of the system).

Figure 2.1e. Bifurcation diagram for Depletion model



Now that we've established the bifurcation diagram, we can more carefully look at time traces for each value of Q in which we are interested in. Below in figure 2.1f, we see the time evolution of the system as we change the value of Q.

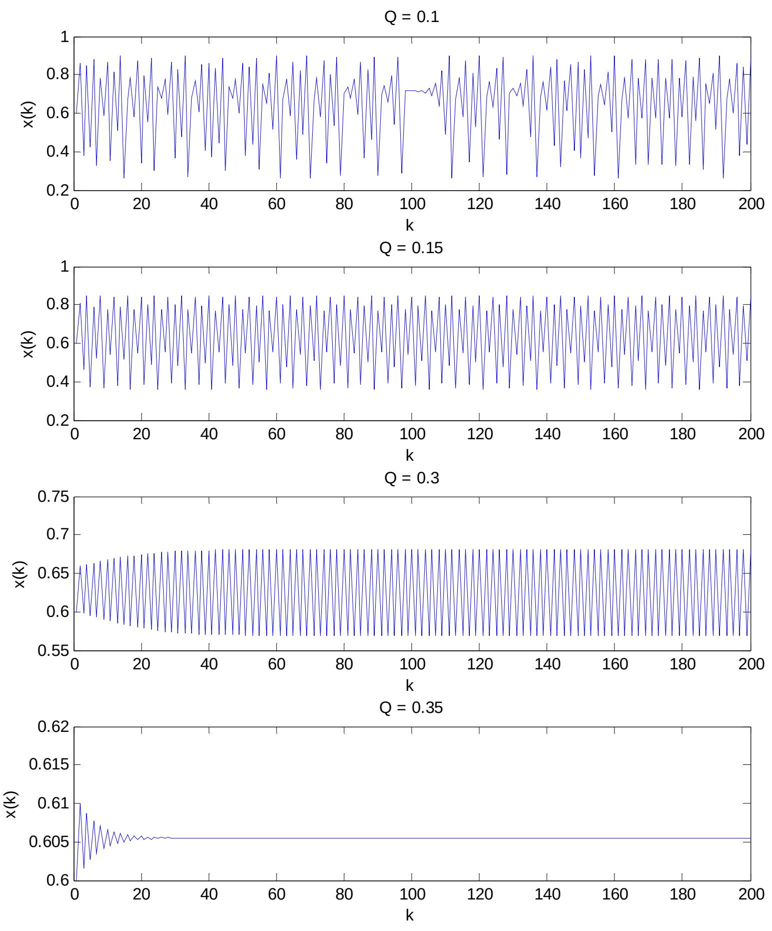


Figure 2.1f. Time evolution of system with various Q values

In figure 2.1f, we clearly see that we progress from chaotic behaviour (Q = 0.1) to a multiperiod behaviour (Q = 0.15) to a period 2 behaviour (Q = 3) and finally to a period 1 (steady state) behaviour (Q = 0.35).

In terms of operation of the fishery, this means that in all of the regimes, the fish will survive, given a high enough initial condition (if the initial conditions are set below the value of Q, it is observed that the fish die out after a period of time). Since a stable fish population is the most desirable and easiest to control in terms of feeding and caging, we would want to stay in the period 1 regime, observed above Q values of approximately 0.31. It would be best to not go into the period 2 oscillatory regime because although it produces predictable behaviour, it is not desirable to have fish populations which oscillate so dramatically. Since *x* is a population density, we also want to maximize the population density while getting a good yield since it is also costs less to fish when the population density is higher. Looking at the bifurcation diagram, the best point at which the fishery should be operating in would be slightly to the right at which the bifurcation occurs to turn the behaviour into a period 2 system (approximately *x* = 3.1). This allows leeway for in terms of harvesting and maintains the fish population at a constant level.

Question 2.2: A model for the evolution of a population of genes

We consider the discrete time evolution of a gene in a diploid organism. There are two alleles, the dominant allele A, and the recessive allele a. In the population, it follows that there are three genotypes that are possible: AA, Aa, and aa.

We define p and q to be the proportion of A and a alleles in the population, such that $p + q = 1$ (and hence $p^2 + q^2 + 2pq = 1$), and it follows that the proportion of AA genotypes is defined by p^2, the proportion of aa genotypes is q^2, and the proportion of Aa genotypes is $2pq$. The fitness, or the proportion of each genotype population that survives to the next time step is given by the following: Φ_{AA}, Φ_{Aa}, and Φ_{aa}. From this, we can deduce that the proportion of A alleles that will be present at the next step in time is given by the formula below.

$$p_{n+1} = \frac{p_n^2\Phi_{AA} + \frac{1}{2}(2p_nq_n\Phi_{Aa})}{p_n^2\Phi_{AA} + q_n^2\Phi_{aa} + 2p_nq_n\Phi_{Aa}}$$

We can rearrange this formula as follows

$$p_{n+1} = \frac{p_n^2\Phi_{AA} + p_n(1-p_n)\Phi_{Aa}}{p_n^2\Phi_{AA} + q_n^2\Phi_{aa} + (1 - p_n^2 - q_n^2)\Phi_{Aa}}$$

$$p_{n+1} = \frac{p_n\Phi_{Aa} - p_n^2(\Phi_{Aa} - \Phi_{AA})}{\Phi_{Aa} - p_n^2(\Phi_{Aa} - \Phi_{AA}) - q_n^2(\Phi_{Aa} - \Phi_{aa})}$$

Now we let $K = \frac{(\Phi_{Aa} - \Phi_{AA})}{\Phi_{Aa}}$ and $L = \frac{(\Phi_{Aa} - \Phi_{aa})}{\Phi_{Aa}}$, we note here that since the fitness levels are bounded between 0 and 1, the values of K and L are also bounded between -1 and 1.

This gives us the following relationship:

$$p_{n+1} = \frac{p_n - Kp_n^2}{1 - Kp_n^2 - Lq_n^2}$$

$$p_{n+1} = \frac{p_n(1 - Kp_n)}{1 - Kp_n^2 - L(1-p_n)^2}$$

To begin to analyze this system, we can first define the fixed points and perform linear stability analysis on these fixed points. We again set $p_{n+1} = p_n$ and sovle for the value of p_n^*. Through matlab or by hand, we can see that there are three fixed points at $p_n^* = 0$, $p_n^* = 1$, and $p_n^* = \frac{L}{K+L}$. We can differentiate the above equation with respect to p_n and substitute our fixed points to see if they are stable and when they become unstable.

The matlab code below does this for us nicely.

```
%Finding the fixed points
syms k l p
xstar = solve('((p*(1-k*p))/(1-k*p^2-l*(1-p)^2))-p','p');
```

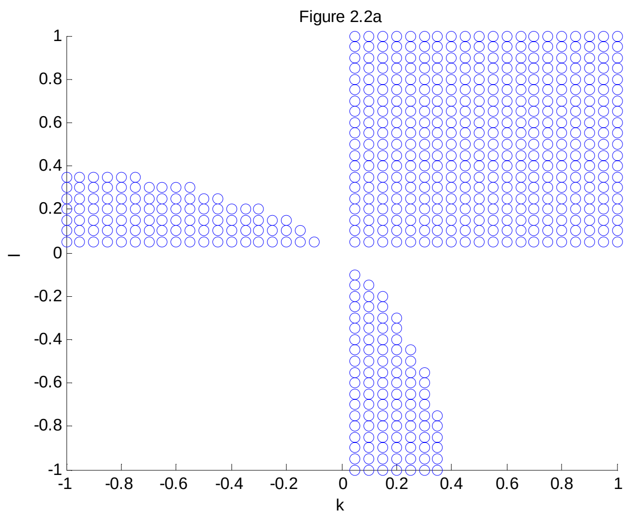
```
%Stability of fixed points
xdiff = diff('((p*(1-k*p))/(1-k*p^2-l*(1-p)^2))','p');

for i = 1:length(xstar)
    stabfp(i) = subs(xdiff,{p},{xstar(i)});
end
```

For the fixed point $p_n^* = 0$, we obtain the differential to be $\left.\frac{df}{dp_n}\right|_{p_n^*=0} = \frac{1}{1-L}$. Given the definition of stability for the absolute value of the differential to be less than 1, this means that the fixed point at $p_n^* = 0$ will only be stable for values of $L < 0$ and $L > 2$. Since the value of L is bounded, this means that this fixed point will only be stable for $-1 < L < 0$.

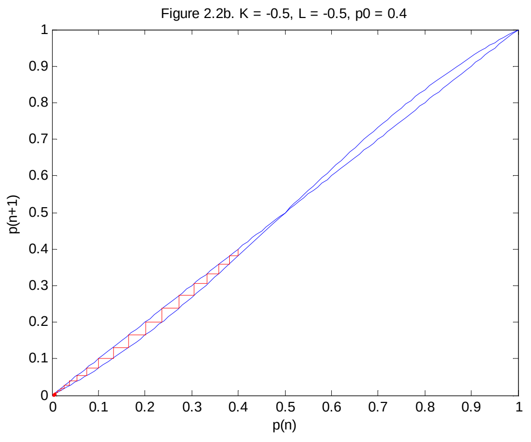
For the fixed point $p_n^* = 1$, we obtain the differential to be $\left.\frac{df}{dp_n}\right|_{p_n^*=1} = \frac{1+K}{1-K}$. With similar analysis to that done above, we conclude that this fixed point will only be stable for $-1 < K < 0$.

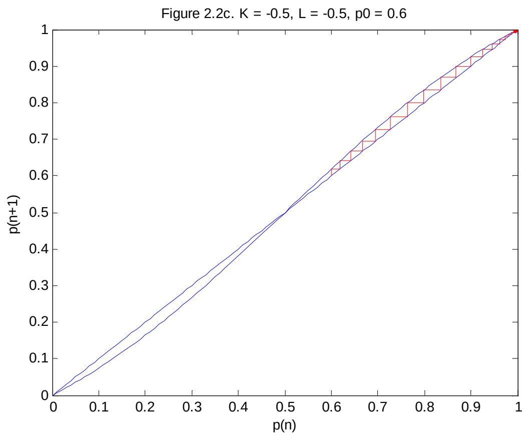
For the fixed point $p_n^* = \frac{L}{K+L}$, the differential is much more complicated, so we can instead do numerical analysis and scan the values of L and K as shown in figure 2.2a. The circles correspond to values at which the fixed point should be stable.



However, if we put the condition such that the value of p is bounded between 0 and 1, then the only values for which the fixed point $p_n^* = \frac{L}{K+L}$ is stable are those for which both K and L are positive (upper right quadrant).

With all of the above in mind, we can now split the final behavior of the system into four categories which correspond to the four quadrants of the Cartesian coordinate system bounded from -1 to 1 on both the L and K axis. In the upper right quadrant, the fixed point $p_n^* = \frac{L}{K+L}$ dominates and will be the attractor of the system. In the upper left quadrant, the fixed point $p_n^* = 1$ dominates and will be the attractor of the system. Similarly, for the lower right quadrant, the fixed point $p_n^* = 0$ dominates and will be the attractor of the system given corresponding values of *K* and *L*. In the lower left quadrant, the system depends on the initial conditions, as seen in the following two cobweb diagrams (figures 2.2b and 2.2c) that represent the time dynamics of the system.





In term of the allele frequencies of the gene, being in different regimes will yield different proportions of A and a alleles. In the upper left quadrant, we know that the attractor of the system is $p_n^* = 1$, so all of the a alleles will die out leaving A alleles. In the lower right quadrant, the opposite occurs since $p_n^* = 0$ and all of the A alleles will die out. In the lower left quadrant (both K and L negative), the final proportion of A will either be 0 or 1, depending upon the initial conditions of the system as is seen in figures 2.2b and 2.2c. So in this regime, only one of the alleles will survive, dependent upon the relative fitness of each. However, we have a unique situation where get a steady proportion of both alleles seen as a fixed point in figure 2.2b and 2.2c. This point actually corresponds to the fixed point $p_n^* = \frac{L}{K+L}$. As is seen in the figures above, this is an unstable fixed point, but still exists for negative values of *L* and *K*. Any small perturbation away from this fixed point will cause the system to go to either 0 or 1.

In the upper right, we see that the proportion of the different alleles are dependent upon the values of both *K* and *L*, but will not die out as in the previous other two cases. In this regime, the fixed point $p_n^* = \frac{L}{K+L}$ is stable and will be the attractor of the system. Although the other two fixed points are still present, they are unstable and will only exist if the initial conditions begin there.

In terms of the entire population, when one is at the fixed point $p_n^* = 1$, only AA genotypes will be present, when at the fixed point $p_n^* = 0$, only aa genotypes will be present, and finally at the fixed point $p_n^* = \frac{L}{K+L}$, all the genotypes will be present at amounts depending upon the relative fitness values.