1 Part 1

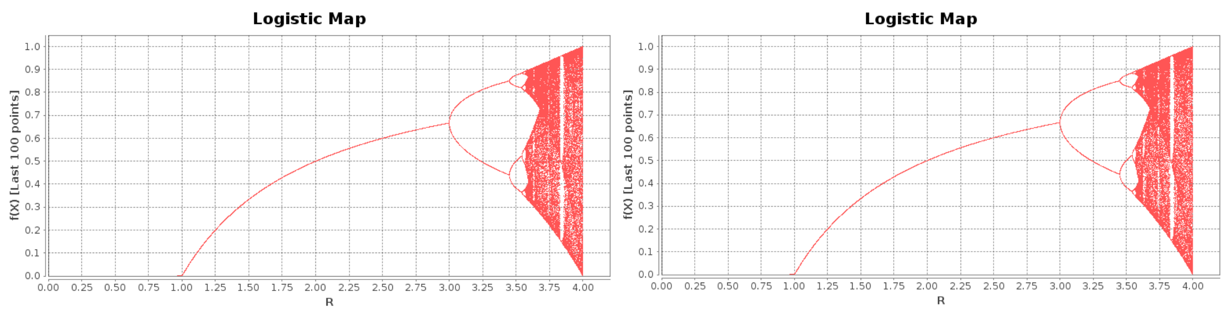
I studied the two functions

$$\begin{aligned} f_1(x) &= r * x * (1 - x) \\ f_1(x) &= r * x * (1 - x) * (2 - x) * (3 - x) \end{aligned}$$

Which both satisfy the following properties:

1. $f(0) = f(1) = 1$
2. $f''(x) < 0$
3. $f_x = r * f_0(x)$
4. $f : [0, 1] \to [0, 1]$

Using my code in `LogisticMap2.java`, I generated the following two images:



(a) Iterated results for $f_1(x; r)$

(b) Iterated results for $f_2(x; r)$

It is evident that there is very little different in the high-iteration appearance of the iterated maps f_1 and f_2. I produced these plots by iterating through r, and at each value of r, iterating f 1000 times. In order to clean up the output, I only returned the last 100 results of iteration using a buffer that was filled once the iteration counter passed 900. The rational behind this is that the early behavior of iteration are strongly related to the initial value of x (which was set randomly for me). However, due to the nature of the maps f_i used, convergence to the fixed points (or cycles) was quick.

2 Part 2

For the map $f(x) = rx(1 - x)$, we can analytically solve for the bifurcation points in r.

Begin with the hypothesis that there is an $r = r_2$ such that f develops a 2-cycle. This corresponds to the onset of a stable fixed point of $f^{(2)}$, and the occurrence of an instability in f. From our derivations in class, the instability in f corresponds to the point at which $f'(x^\star) = -1$.

Solving for $f(x^\star) = x^\star$ we find that

$$(x^\star) = 1 - \frac{1}{r}.$$

Using this, we can solve for r_2:

$$\begin{aligned} -1 &= f'(x^\star) \\ &= \frac{d}{dx}(rx(1-x))|_{x^\star} \\ &= r_2(1-2x^\star) \\ &= r_2(1-2(1-\frac{1}{r_2})) \\ &= -r_2+2 \end{aligned}$$

Hence,

$$r_2 = 3$$

and plugging back in, we see that at r_2,

$$x^\star|_{r_2} = \frac{2}{3} = .666.$$

3 Part 3

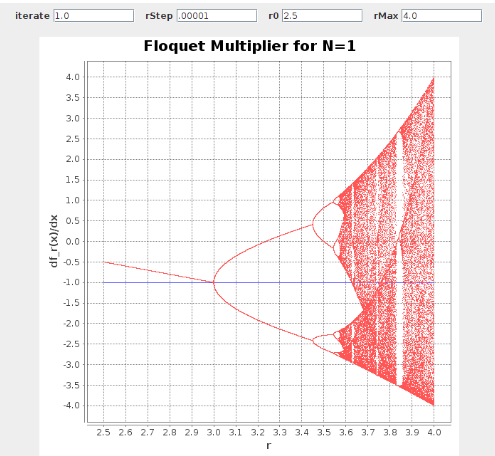
Using the Newton-Raphson method, and r values taken from the above plot of f_1, I numerically determined the fixed points of f when it has stable 2 and 4 cycles. The guesses for the method were the result of iterating the function 10 times from a random start value.

n-cycle	r	guess x_0	converged value $x^\star$
2	3.2	0.513	0.513
		0.799	0.798
4	3.5	0.840	0.501
		0.471	0.827
		0.872	0.875
		0.391	0.383

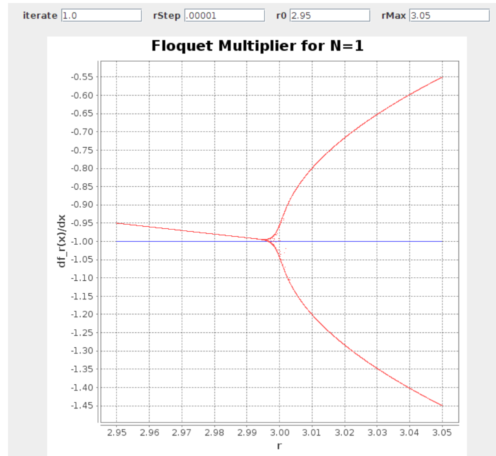
As is evident by some of the above (guess,converged value) pairs, the iterations converge very quickly to the fixed points. The convergence tolerance for Newton-Raphson was $1e-9$. The delta used for derivatives was $1e-6$.

4 Part 4

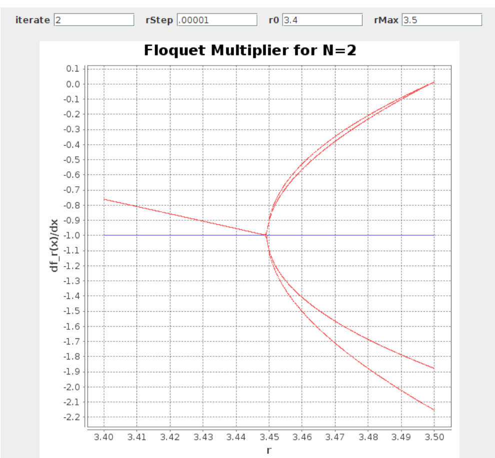
In `Floquet.java`, I plot the Floquet Multiplier $\Lambda|_{f^{(N)}} = \frac{df^{(N)}}{dx}|_{x^\star}$ (L in code) against r. As expected, at each bifurcation of $f^{(N)}$, $\Lambda|_{f^{(2N)}} = -1$. See the figure below for $N = 2, 4, 8$. As you can see, at each bifurcation, $f(N)$ splits into an equal number of stable and unstable fixed points.



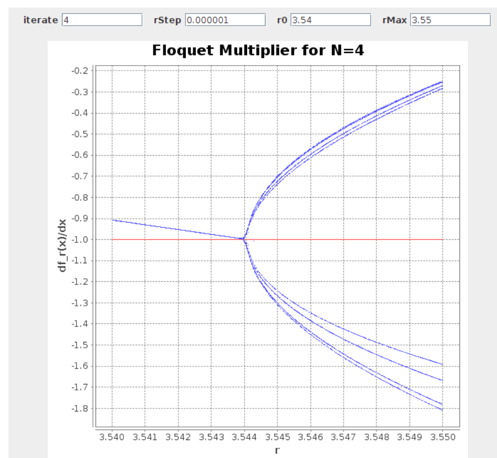
(a) Floquet Multiplier of f



(b) Bifurcation of f at r_2



(c) Double Bifurcation of $f^{(4)}$ at r_4

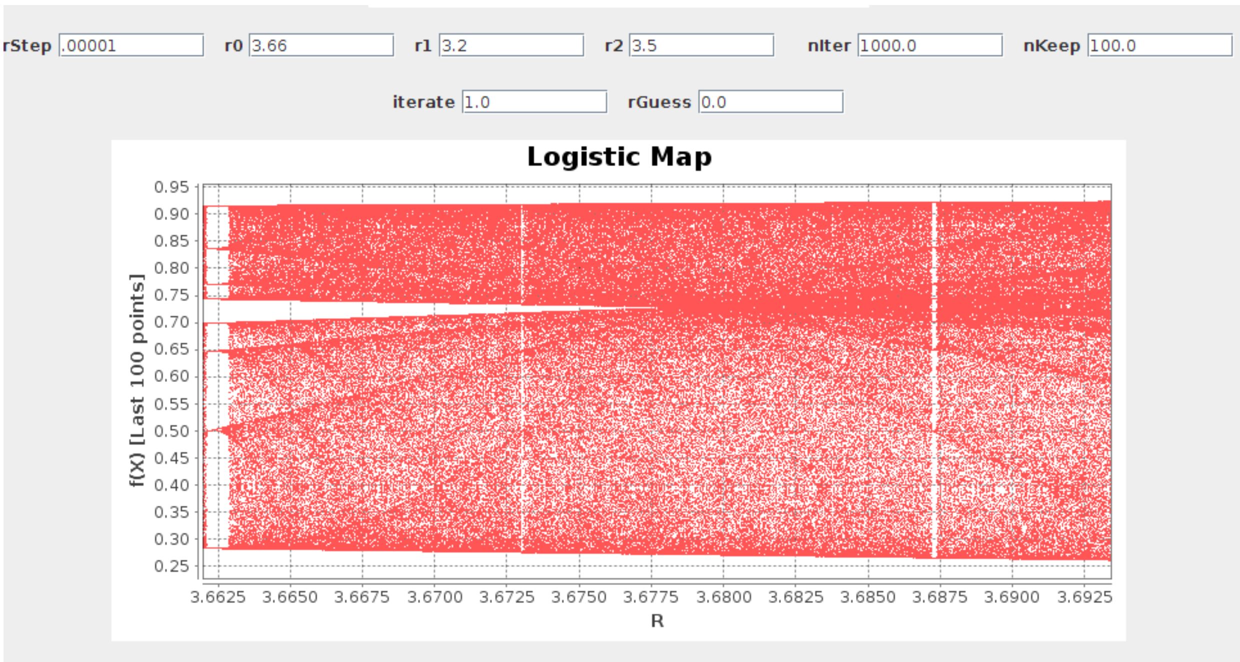


(d) Quad Bifurcation of $f^{(8)}$ at r_8

5 Part 5

I looked at the behavior of f at 4 values of r where the logistic map curve exhibited interesting behavior. In each of the following subsections, there is a picture of the region around the interesting r as well as a brief discussion of the occurrences there.

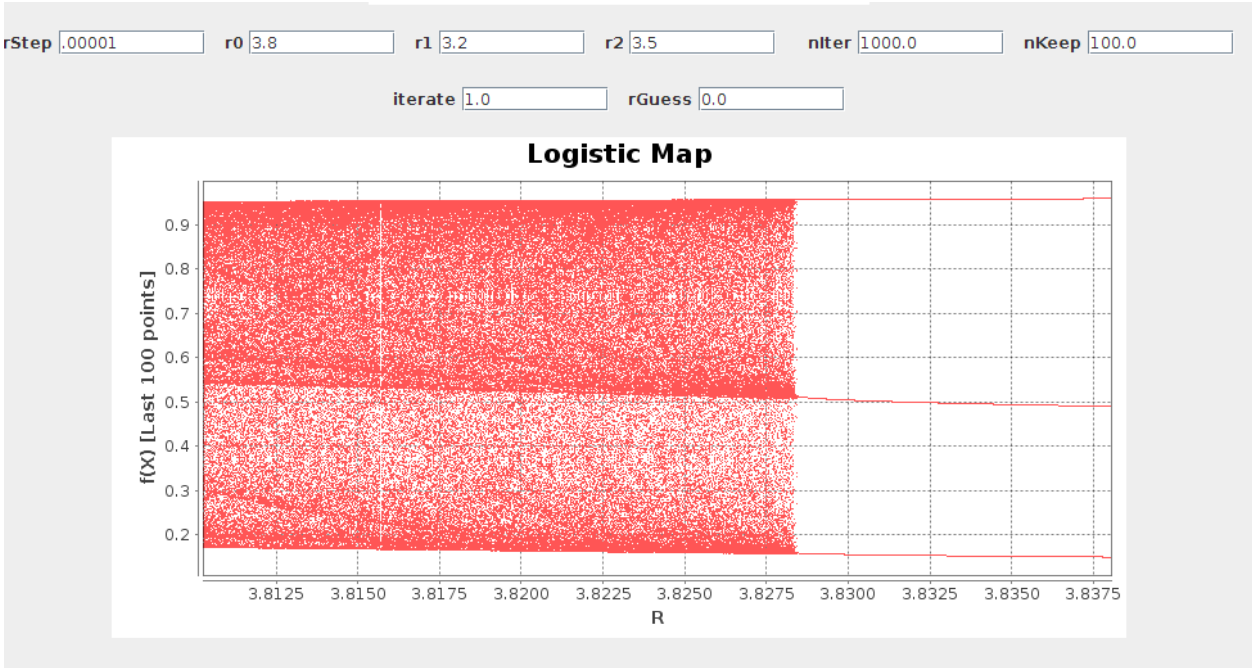
5.1 $r \approx 3.68$



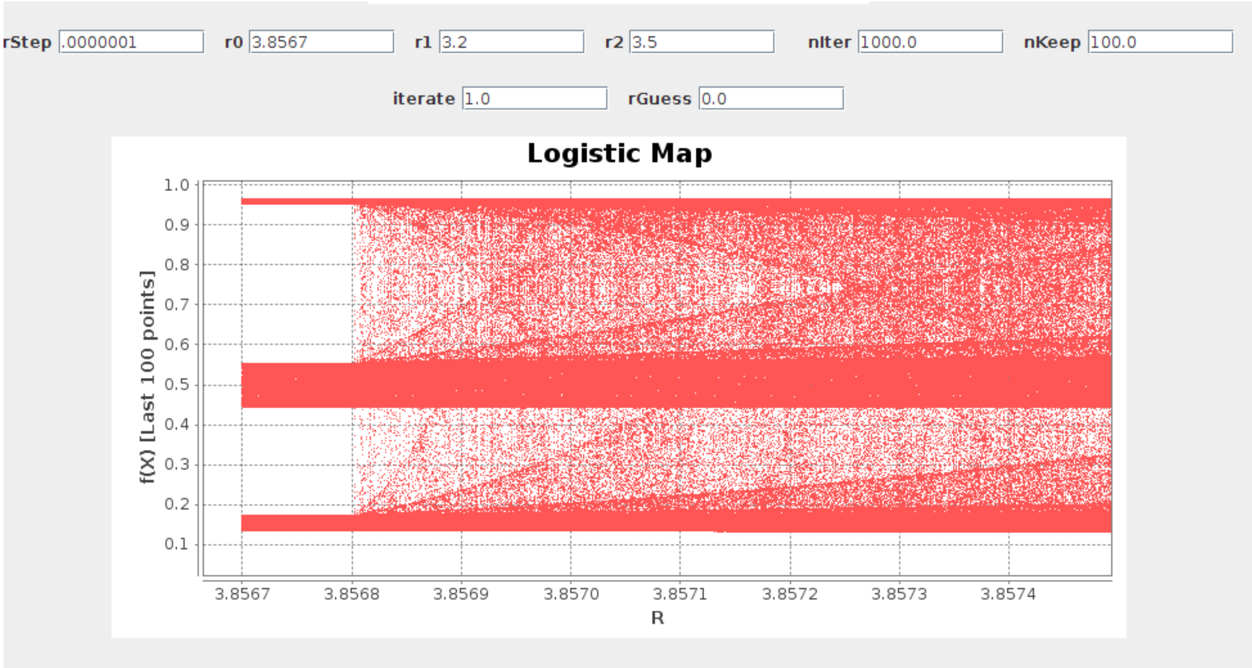
At this point, r_∞, the function is involved in an infinite cycle. For smaller r, the cycles were finite in length, growing like 2^N. Visually, two halves meet that had originally split from the instability as $N = 1 \to N = 2$.

5.2 $r \approx 3.83$

Here, the mess that had emerged after r_∞ seems to have converged back into a stable 3-cycle. This is rather confusing at first glance. Why should something that went infinite spontaneously reemerge as finite. Moreover, what sets that transition value $r_{strange}$?

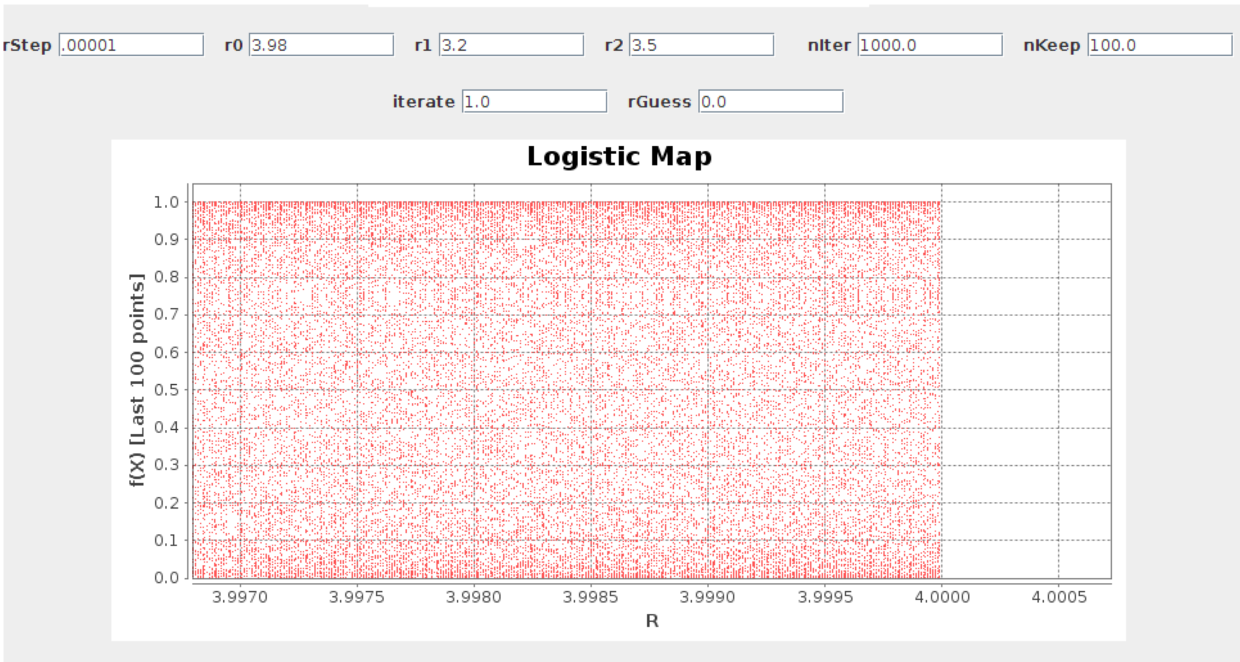


5.3 $r \approx 3.8568$



The stable 3 cycle found in the previous plot began to bifurcate repeatedly again. However, before the outermost bifurcation paths intersected again, a “wall of cycles” occurs again. Because of the resolution of my plot and the lack of any buildup, this feature seems to be one of the Logistic map and not an artifact of the numerics. Once again, this spontaneous phase transition is perplexing - what sets the r_{crit} and what is the length of the cycle that it jumps to there?

5.4 $r = 4$



For our logistic map function, $r = 4$ is the maximum allowed r value. That being the case, it is also the only value of r for which 1 is in the range of f. Naturally, this seems like a reasonable point to see some sort of new behavior, whether it is convergence or something else. However, upon inspection, the lefthand limit reveals no such interesting behavior. Hence I mark this as an interesting point because the features of the curve do not appear to change as $r \to 4$ from the left, so there is either a discontinuity in the behavior of f at $r = 4$ or it is continuous and there is a seemingly "normal" behavior beyond what is allowed in our system.

../LogisticMap2.java

```
// LogisticMap.java
// Thanasi's Logistic Map Class
// cleaned up a bit

import javax.swing.*;
import P251.*;

public class LogisticMap2 extends P251Applet {

    /**********************/
    /****** VARIABLES ******/
    /**********************/

    /****** Global Variables ******/
    double defaultTolerance = 1e-9;  // tolerance for N-R convergence
    double derivDelta = 1e-6;        // delta for use in numerical deriv
    int iterMax = 1000000;           // max number of iterations in N-R

    double rMax = 4;    // maximum value for r to be checked against

    /****** Initial values for user-input variables ******/
    double rStep = 1e-3;        // step size for iterations through r
    double R0 = 1;              //
    double R1,R2;               // for use in finding fixed points of cycles
    int nIter = 1000;           // number of iterations of f(x) to run for each r
    int nKeep = 100;            // number of iterations of f(x) to plot for each r

    int iterate = 1;            // iterate of f
    double rGuess = 0;          // initial guess for N-R

    /****** Calculation variables and arrays ******/
    double R;      // to calculate the map
    double X;
    double X0 = Math.random();

    double [] XX; // to calculate X vs R plot
    double [] YY;
    double [] RR;
    double [] RR2 = new double [2];
    double [] RR4 = new double [4];

    /****** Panels ******/
    private inputPanel ip1, ip2, ip3;
    private graphPanel gp1, gp2;


    /*******************/
    /****** METHODS ******/
    /*******************/

    /***** Custom Math Functions *****/

    double f (double x) {
    return R * x * (1 - x);
    // return R * (-1 * Math.abs(x-.5) + .5);
```

```
    // return R * .25 * x * (1 - x) * (2 - x) * (3 - x);
    }

    double iterf (double x, int n) {
    // return n-th iterate of f
    if (n>1) return f(iterf(x,n-1));
    else return f(x);
    }

    double dfdx (double x, int n, int i) {
    // take the numerical i-th derivative
    // of the n-th iterate of f
    // at the point x, using small (globally set) delta

    if (i>1) return (dfdx(x+derivDelta,n,i-1)-dfdx(x,n,i-1)) / derivDelta;
    else return (iterf(x+derivDelta, n)-iterf(x,n)) / (derivDelta);
    }

    double dfdx (double x, int n) {
    // overload with default i=1
    return dfdx(x,n,1);
    }

    double root (double x0, int n, double tol) {
    // use Newton-Raphson to find the fixed points of the
    // n-th iterate of f
    // iterate until tolerance of tol is reached
    double x = x0;
    double delta;
    int i = 0;
    do {
        // subtract x and 1 from the function and the derivative
        // because for fixed points, we want to find the roots of
        // iterf(x, n) = x
        delta = -(iterf(x, n)-x) / (dfdx(x, n)-1);

        // update x (we'd be here a while if we didn't)
        x += delta;

        if (i >= iterMax) break;  // if we've tried too hard, then give up
        i++;
    } while (Math.abs(delta)>tol);

    if (i==iterMax) x = Double.NaN;

    System.out.println(String.format("Found fixed point of the %d iterate of f for
        R=%4.3f:\t%6.4f --> %6.4f", n, R, x0, x));

    return x;
    }

    double root (double x0, int n) {
    // overload with default tolerance
    return root(x0, n, defaultTolerance);
    }
```

```
    double [] getLastValues(int n, int num, int nit) {
    // get the last num values of the nth iterate of f
    // after iterating nit times
    double [] output = new double[num];
    X = X0;
    for (int i=0; i<nit; i++) {
        X = iterf(X, n);
        if (i>nit-num-1) {
        output[i + num - nit] = X;
        //      System.out.println("getLastValues " + n + " " + num + " " + nit + "
            >> " + X);
        }
    }
    return output;
    }

    double [] findFixedPoints(double [] guess) {
    // find the fixed points of an iterative of f
    // the iterative is determined by the length of the guess array
    int N = guess.length;
    double [] ans = new double[N];
    for (int i = 0; i<N; i++) {
        ans[i] = root(guess[i], N);
    }

    return ans;
    }

    /****** Custom plotting functions ******/

    public void vLine(double x, double ymin, double ymax, graphPanel gp) {
    // draw a vertical line on graph panel gp
    for (int i=0; i<800; i++) {
        double [] xx = {x};
        double [] yy = {ymin + i*(ymax-ymin)/800};
        gp.addData(xx, yy, "vline");
    }
    }

    public void hLine(double y, double xmin, double xmax, graphPanel gp) {
    // draw a horizontal line on graph panel gp
    for (int i=0; i<800; i++) {
        double [] yy = {y};
        double [] xx = {xmin + i*(xmax-xmin)/800};
        gp.addData(xx, yy, "hline");
    }
    }

    public void plotFunc(double xmin, double xmax) {
    // calculate and plot the desired iterate of f over its domain,
    // along with the line y=x

    int resolution = 10000;
    double step = (xmax - xmin) / resolution;
    double [] x = new double[resolution];
    double [] y = new double[resolution];
```

```

    // initialize arrays
    x[0] = 0;
    y[0] = f(x[0]);

    int [] it = {1,2,4}; // iterates to plot

    // calculate and plot f
    gp1.clear();
    for (int j=0; j<3; j++){
        for (int i=1; i<resolution; i++) {
        x[i] = x[i-1] + step;
        y[i] = iterf(x[i], it[j]);
        }

        gp1.setDotSize(1);
        gp1.addData(x, y, "fplot"+it[j]);    // plot f
        gp1.addData(x, x, "xxplot");  // plot y=x for comparison
    }
    }

    void plotXvsR() {
    // plot the last nKeep values of f(X) iterated nIter times
    double X = X0;
    for (int i=0; i<nIter; i++){
        X = iterf(X, iterate);
        if (i>nIter-nKeep){
        // once we're in the 'keepable' number of iterations
        // save them to the keeping array
        RR[i + nKeep - nIter - 1] = R;
        XX[i + nKeep - nIter - 1] = X;
        }
    }
    gp2.setDotSize(1);
    gp2.addData(RR,XX,"xvsr");
    }

    /****** P251Applet Methods ******/

    public void fillPanels() {

    // set up input list
    ip1 = new inputPanel();
    ip2 = new inputPanel();
    ip3 = new inputPanel();
    ip1.addField("rStep", rStep);    // step size for R
    ip1.addField("r0", 2.5);         //  minimum for x vs R plot
    ip1.addField("r1", 3.2);
    ip1.addField("r2", 3.5);

    ip2.addField("nIter", 1000);
    ip2.addField("nKeep", 100);
    ip3.addField("iterate", 1);
    ip3.addField("rGuess", 0);

    // set up function graph panel
```

```
    gp1 = new graphPanel(400, 400, false);
    gp1.setXLabel("x");
    gp1.setYLabel("f(x)");
    gp1.setTitle("Iterated Function");
    addPanel(gp1);

    // set things up to look good
    addPanel(ip1);
    addPanel(ip2);
    addPanel(ip3);

    // set up (x vs r) graph panel
    gp2 = new graphPanel(800, 400, false);
    gp2.setXLabel("R");
    gp2.setYLabel("f(X) [Last " + nKeep + " points]");
    gp2.setTitle("Logistic Map");
    addPanel(gp2);
    }

    public void initValues() {
    // set up initial values
        // for when the button is pressed
    R0 = 2.5;
    R1 = 3.2;
    R2 = 3.5;
    R = 2.5;
    nIter = 1000;
        nKeep = 100;
    rStep = 1e-3;

    X0 = Math.random();
    iterate = 1;
    rGuess = 0;

    // re-initialize arrays to new sizes

    XX = new double[nKeep];
    YY = new double[nKeep];
    RR = new double[nKeep];

        gp1.clear();
        gp2.clear();
        plotFunc(0,1);
    }

    public void readValues() {
    // get input values
    rStep = ip1.getValue(0);
    R0 = ip1.getValue(1);
    R1 = ip1.getValue(2);
    R2 = ip1.getValue(3);
    nIter = (int) ip2.getValue(0);
    nKeep = (int) ip2.getValue(1);
    iterate = (int) ip3.getValue(0);
    rGuess = ip3.getValue(1);

```

A. Athanassiadis


```
    // re-initialize arrays to new sizes
    XX = new double[nKeep];
    YY = new double[nKeep];
    RR = new double[nKeep];

    // re-initialize R variable to restart plot
    R = R0;
    X0 = Math.random();


    // clear plots
    gp1.clear();
    gp2.clear();

    plotFunc(0,1);
    }

    public void compute(){

    // first task
    // iterate through R to create x vs r plot for desired range of r
    R = R0;

    while (R<rMax) {
        plotXvsR();
        R += rStep;
        if (Thread.interrupted()) return;
    }

    // second task
    // find stable values of 2-cycles and 4-cycles
    // determine guesses for X by iterating the function nIter times
    // then run it through Newton-Raphson and plot to verify
    R = R1;
    double [] cycle2Guess = getLastValues(1,2,10);
    double [] cycle2 = findFixedPoints(cycle2Guess);

    R = R2;
    double [] cycle4Guess = getLastValues(1,4,10);
    double [] cycle4 = findFixedPoints(cycle4Guess);

    // vLine(R1, 0,1, gp2);
    // vLine(R2, 0,1, gp2);


    // fourth task


    // fifth task
   }


}
```

../Floquet.java

```java
// Floquet.java
// find Floquet Multiplier as function of r
// for the N-th iterate of f
// remember floquet multiplier is:
// L_i = [f(N)]'(x*)



import javax.swing.*;
import P251.*;

public class Floquet extends P251Applet {

    /**********************/
    /****** VARIABLES ******/
    /**********************/

    /****** Global Variables ******/
    double defaultTolerance = 1e-9;  // tolerance for N-R convergence
    double derivDelta = 1e-6;        // delta for use in numerical deriv
    int iterMax = 1000000;           // max number of iterations in N-R
    int nIter = 1000;


    /****** Initial values for user-input variables ******/
    double rStep = 1e-3;        // step size for iterations through r
    double R0 = 1;              //
    double rMax = 4;     // maximum value for r to be checked against


    int nR = (int) ((rMax-R0) / rStep); // the number of R values to look at
    int iterate = 1;            // iterate of f

    /****** Calculation variables and arrays ******/
    double R;      // to calculate the map
    double X;

    /****** Panels ******/
    private inputPanel ip1;
    private graphPanel gp1;


    /*******************/
    /****** METHODS ******/
    /*******************/

    /***** Custom Math Functions *****/

    double f (double x) {
    return R * x * (1 - x);
    // return R * (-1 * Math.abs(x-.5) + .5);
    // return R * .25 * (1-x) * (2-x) * (3-x);
    }

    double iterf (double x, int n) {
```

```
    // return n-th iterate of f
    if (n>1) return f(iterf(x,n-1));
    else return f(x);
    }

    double dfdx (double x, int n, int i) {
    // take the numerical i-th derivative
    // of the n-th iterate of f
    // at the point x, using small (globally set) delta

    if (i>1) return (dfdx(x+derivDelta,n,i-1)-dfdx(x,n,i-1)) / derivDelta;
    else return (iterf(x+derivDelta, n)-iterf(x,n)) / (derivDelta);
    }

    double dfdx (double x, int n) {
    // overload with default i=1
    return dfdx(x,n,1);
    }

    double [] getLastValues(int n, int num, int nit) {
    // get the last num values of the nth iterate of f
    // after iterating nit times
    double [] output = new double[num];
    X = Math.random();

    for (int i=0; i<nit; i++) {
        X = iterf(X, n);
        if (i>nit-num-1) {
        output[i + num - nit] = X;
        }
    }

    return output;
    }

    /****** Custom plotting functions ******/

    public void vLine(double x, double ymin, double ymax, graphPanel gp) {
    // draw a vertical line on graph panel gp
    for (int i=0; i<800; i++) {
        double [] xx = {x};
        double [] yy = {ymin + i*(ymax-ymin)/800};
        gp.addData(xx, yy, "vline");
    }
    }

    public void hLine(double y, double xmin, double xmax, graphPanel gp) {
    // draw a horizontal line on graph panel gp
    for (int i=0; i<800; i++) {
        double [] yy = {y};
        double [] xx = {xmin + i*(xmax-xmin)/800};
        gp.addData(xx, yy, "hline");
    }
    }

    /****** P251Applet Methods ******/
```

```
    public void fillPanels() {

    // set up input list
    ip1 = new inputPanel();
    ip1.addField("iterate", 1);
    ip1.addField("rStep", rStep);    // step size for R
    ip1.addField("r0", 2.5);         //  minimum for x vs R plot
    ip1.addField("rMax", 4);
    addPanel(ip1);

    // set up function graph panel
    gp1 = new graphPanel(600, 600, false);
    gp1.setXLabel("r");
    gp1.setYLabel("df_r(x)/dx");
    gp1.setTitle("Iterated Function");
    addPanel(gp1);
    }

    public void initValues() {
    // set up initial values
        // for when the button is pressed
    R0 = 2.5;
    rMax = 4;
    R = 2.5;

    rStep = 1e-3;

    iterate = 1;

    nR = (int) ((rMax-R0) / rStep);

    // re-initialize arrays to new sizes
        gp1.clear();
    gp1.setTitle("Floquet Multiplier for N=" + iterate);

    }

    public void readValues() {
    // get input values
    iterate = (int) ip1.getValue(0);
    rStep = ip1.getValue(1);
    R0 = ip1.getValue(2);
    rMax = ip1.getValue(3);
    nR = (int) ((rMax-R0) / rStep);

    // re-initialize R variable to restart plot
    R = R0;

    // clear plots
    gp1.clear();
    gp1.setTitle("Floquet Multiplier for N=" + iterate);
    }

    public void compute(){

```

```
    double [] RR = new double[nR];
    double [] LL = new double[nR];
    R = R0;
    for (int i = 0; i<nR; i++) {
        X = getLastValues(iterate, 1, nIter)[0]; // iterate the function until convergence
        RR[i] = R;
        LL[i] = dfdx(X, iterate); // get floquet multiplier
        // System.out.println(String.format("%02d: %3.4f %3.4f", iterate, R, LL[i]));

        R += rStep;

    }


    gp1.addData(RR,LL, "Floquet");
    hLine(-1, R0, rMax, gp1);

  }


}
```