A. Athanassiadis

1 Parts A & C

2 Part E

3 Part F

../IterMap.java

```
// IterMap.java
// Study the convergence of r_k in the logistic map

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

public class IterMap extends P251Applet {

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

    double tol = 1e-8;        // tolerance for N-R convergence
    int iterMax = 1000000;    // max number of iterations in N-R
    double epsilon = 1e-6;    // delta for use in numerical deriv

    int nR;                 // number of rsk to look for
    double delta;           // scaling parameter for r
    double alpha;           // scaling parameter for y

    double [] roots;        // discovered roots
    double [] y;            // value of iterf half way through a 2^k cycle

    double xMax = .5;       // x where maximum occurs for logistic map

    inputPanel ip1;

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

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

    double f (double x, double r) {
    // logistic map
    return r * x * (1 - x);
    }

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

    double F (double r, int k) {
    // function used for optimization of r when x=xMax
        //
    double ans = iterf(.5, r, (int) Math.pow(2,k)) - xMax;
    for (int i=0; i<nR; i++){
        ans /= (r-roots[i]);
    }

    return ans;

    }
```

```

    double dFdr (double r, int k){
    // take the derivative of F WRT r
    return (F(r+epsilon, k) - F(r, k)) / epsilon;
    }

    double rootR (double rg, int k){
    // use Newton's method to find r_{si}

    double r = rg;
    double delta;
    int i = 0;

    do {
        // find derivative towards zero
        delta = -F(r,k) / dFdr(r,k);
        // update r
        r += delta;

        if (i>=iterMax) break;
        i++;
    } while(Math.abs(delta)>tol);

    // if we timed out, then return NaN
    if (i==iterMax) {
        r = Double.NaN;
        System.out.println("Tried too hard to find rootR");
    }


    return r;
    }

    /****** P251Applet Methods ******/
    public void fillPanels() {
    // define the panels for human interaction
    ip1 = new inputPanel();

    ip1.addField("nR", 10);
    addPanel(ip1);
    initValues();

    }

    public void initValues() {
    // set up initial values

    nR = 10;
    roots = new double[nR];
    y = new double[nR];
    // set roots to 1 so as not to cause overflow on divide
    for (int i=0; i<nR; i++) roots[i] = 1;

    }

    public void readValues() {
```

```java
    // read input panel values
    nR = (int) ip1.getValue(0);
    roots = new double[nR];
    y = new double[nR];
    // set roots to 1 so as not to cause overflow on divide
    for (int i=0; i<nR; i++) roots[i] = 1;

    }

    public void compute() {

    double rg; // r guess for root finding

    // solved for first two analytically
    roots[0] = 2;
    roots[1] = 1 + Math.sqrt(5);

    System.out.println("\nk\tr_sk\td\ty\ta");

    // solve details of k=0,1 cases outside of the loop
    int k = 0;
    y[k] = iterf(.5, roots[k], (int) Math.pow(2,k-1));
    System.out.println(String.format("%02d\t%4.4f\t--\t%4.4f\t—", k+1, roots[k],
        y[k]));

    k = 1;
    y[k] = iterf(.5, roots[k], (int) Math.pow(2,k-1));
    System.out.println(String.format("%02d\t%4.4f\t--\t%4.4f\t—", k+1, roots[k],
        y[k]));



    // for each k, figure out r_sk, delta, y, alpha
    for (k=2; k<nR; k++) {
        rg = roots[k-1] + .1 * (roots[k-1] - roots[k-2]);
        roots[k] = rootR(rg, k);
        delta = (roots[k-1] - roots[k-2]) / (roots[k]-roots[k-1]);
        y[k] = iterf(.5, roots[k], (int) Math.pow(2,k-1));
        alpha = - (y[k-1] - y[k-2]) / (y[k] - y[k-1]);
        System.out.println(String.format("%02d\t%4.4f\t%4.4f\t%4.4f\t%4.4f", k+1,
            roots[k], delta, y[k], alpha));

        if (Thread.interrupted()) return;

    }

    }

}
```

../IterMap2.java

```
// IterMap2.java
// Study the convergence of r_k in an iterative map

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

public class IterMap2 extends P251Applet {

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

    double tol = 1e-8;       // tolerance for N-R convergence
    int iterMax = 1000000;   // max number of iterations in N-R
    double epsilon = 1e-6;   // delta for use in numerical deriv

    int nR;                  // number of rsk to look for
    double delta;            // scaling parameter for r
    double alpha;            // scaling parameter for y

    double [] roots;         // discovered roots
    double [] y;             // value of iterf half way through a 2^k cycle

    double X0 = .8;      // initialize X0 for iteration at startup

    double xMax = .8;  // x where maximum occurs for f
    double rMin = 2.6; // just before the first bifurcation
    double rMax = 3.5; // once f leaves the range of interest
    double rStep;      // how finely to look at r in the bifurcation map

    inputPanel ip1;
    graphPanel gp1;


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

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

    double f (double x, double r) {
    // Gr (RP3.8)
    return r * x*x * Math.sqrt(1 - x);
    }

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

    double F (double r, int k) {
    // function used for optimization of r when x=xMax
        //
    double ans = iterf(xMax, r, (int) Math.pow(2,k)) - xMax;
```

```
    for (int i=0; i<nR; i++){
        ans /= (r-roots[i]);
    }
    return ans;
    }

    double dFdr (double r, int k){
    // take the derivative of F WRT r
    return (F(r+epsilon, k) - F(r, k)) / epsilon;
    }

    double rootR (double rg, int k){
    // use Newton's method to find r_{si}

    double r = rg;
    double delta;
    int i = 0;

    do {
        // find derivative towards zero
        delta = -F(r,k) / dFdr(r,k);
        // update r
        r += delta;

        if (i>=iterMax) break;
        i++;
    } while(Math.abs(delta)>tol);

    // if we timed out, then return NaN
    if (i==iterMax) {
        r = Double.NaN;
        System.out.println("Tried too hard to find rootR");
    }


    return r;
    }

    double[] getLastValues(double r, int num, int nit) {
    // get the last num values of f
    // after iterating nit times
    double[] output = new double[num];
    double X = X0;
    for (int i=0; i<nit; i++) {
        X = f(X, r);
        if (i>nit-num-1) {
        output[i + num - nit] = X;
        }
    }
    return output;
    }


    /****** Plotting Methods ******/
    void plotBifurcations(double rStep, graphPanel gp, int nKeep, int nIter){
    gp.clear();
```

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

    double R = rMin;
    while (R<=rMax) {
        XX = getLastValues(R, nKeep, nIter);
        for (int i = 0; i<nKeep; i++) RR[i] = R;

        gp.addData(RR, XX, "bifurcation");
        R += rStep;
        if (Thread.interrupted()) return;
    }
    }

    /****** P251Applet Methods ******/
    public void fillPanels() {
    // define the panels for human interaction
    ip1 = new inputPanel();

    ip1.addField("nR", 10);
    ip1.addField("rStep", .001);
    addPanel(ip1);
    initValues();

    gp1 = new graphPanel(600, 300, false);

    gp1.setXLabel("R");
    gp1.setYLabel("f(X)");
    gp1.setTitle("Bifurcation Diagram");
    addPanel(gp1);

    }

    public void initValues() {
    // set up initial values

    nR = 10;
    rStep = .001;
    roots = new double[nR];
    y = new double[nR];
    // set roots to 1 so as not to cause overflow on divide
    for (int i=0; i<nR; i++) roots[i] = 1;

    }

    public void readValues() {
    // read input panel values
    nR = (int) ip1.getValue(0);
    rStep = ip1.getValue(1);
    roots = new double[nR];
    y = new double[nR];
    // set roots to 1 so as not to cause overflow on divide
    for (int i=0; i<nR; i++) roots[i] = 1;

    }

```

```java
    public void compute() {

    double rg; // r guess for root finding

    // solved for first two analytically
    roots[0] = 2.79508;
    roots[1] = 3.15783;

    System.out.println("\nk\td\ta");

    // solve details of k=0,1 cases outside of the loop
    int k = 0;
    y[k] = iterf(xMax, roots[k], (int) Math.pow(2,k-1));
    System.out.println(String.format("%02d\t--\t--", k+1));

    k = 1;
    y[k] = iterf(xMax, roots[k], (int) Math.pow(2,k-1));
    System.out.println(String.format("%02d\t--\t--", k+1));

    // for each k, figure out r_sk, delta, y, alpha
    for (k=2; k<nR; k++) {
        rg = roots[k-1] + .1 * (roots[k-1] - roots[k-2]);
        roots[k] = rootR(rg, k);
        delta = (roots[k-1] - roots[k-2]) / (roots[k]-roots[k-1]);
        y[k] = iterf(xMax, roots[k], (int) Math.pow(2,k-1));
        alpha = - (y[k-1] - y[k-2]) / (y[k] - y[k-1]);
        System.out.println(String.format("%02d\t%4.4f\t%4.4f", k+1, delta, alpha));

        if (Thread.interrupted()) return;

    }

    plotBifurcations(rStep, gp1, 100, 1000);




    }

}
```

../IterMap3.java

```
// IterMap3.java
// Study the convergence of r_k in the cusped map

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

public class IterMap3 extends P251Applet {

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

    double tol = 1e-8;        // tolerance for N-R convergence
    int iterMax = 1000000;    // max number of iterations in N-R
    double epsilon = 1e-6;    // delta for use in numerical deriv

    int nR;                   // number of rsk to look for
    double delta;             // scaling parameter for r
    double alpha;             // scaling parameter for y

    double [] roots;          // discovered roots
    double [] y;              // value of iterf half way through a 2^k cycle

    double X0 = .5;     // initialize X0 for iteration at startup

    double xMax = .5;    // x where maximum occurs for f
    double rMin = .9;   // just before the first bifurcation
    double rMax = 2.82; // once f leaves the range of interest
    double rStep;        // how finely to look at r in the bifurcation map

    inputPanel ip1;
    graphPanel gp1;


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

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

    double f (double x, double r) {
    // cusp map
    return r * ( Math.pow(.5, 1.5) - Math.pow(Math.abs(x-.5), 1.5) );
    }

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

    double F (double r, int k) {
    // function used for optimization of r when x=xMax
        //
    double ans = iterf(xMax, r, (int) Math.pow(2,k)) - xMax;
```

```
    for (int i=0; i<nR; i++){
        ans /= (r-roots[i]);
    }
    return ans;
    }

    double dFdr (double r, int k){
    // take the derivative of F WRT r
    return (F(r+epsilon, k) - F(r, k)) / epsilon;
    }

    double rootR (double rg, int k){
    // use Newton's method to find r_{si}

    double r = rg;
    double delta;
    int i = 0;

    do {
        // find derivative towards zero
        delta = -F(r,k) / dFdr(r,k);
        // update r
        r += delta;

        if (i>=iterMax) break;
        i++;
    } while(Math.abs(delta)>tol);

    // if we timed out, then return NaN
    if (i==iterMax) {
        r = Double.NaN;
        System.out.println("Tried too hard to find rootR");
    }


    return r;
    }

    double [] getLastValues(double r, int num, int nit) {
    // get the last num values of f
    // after iterating nit times
    double [] output = new double[num];
    double X = X0;
    for (int i=0; i<nit; i++) {
        X = f(X, r);
        if (i>nit-num-1) {
        output[i + num - nit] = X;
        }
    }
    return output;
    }


    /****** Plotting Methods ******/
    void plotBifurcations(double rStep, graphPanel gp, int nKeep, int nIter){
    gp.clear();
```

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

    double R = rMin;
    while (R<=rMax) {
        XX = getLastValues(R, nKeep, nIter);
        for (int i = 0; i<nKeep; i++) RR[i] = R;

        gp.addData(RR, XX, "bifurcation");
        R += rStep;
        if (Thread.interrupted()) return;
    }
    }

    /****** P251Applet Methods ******/
    public void fillPanels() {
    // define the panels for human interaction
    ip1 = new inputPanel();

    ip1.addField("nR", 10);
    ip1.addField("rStep", .001);
    addPanel(ip1);
    initValues();

    gp1 = new graphPanel(600, 300, false);

    gp1.setXLabel("R");
    gp1.setYLabel("f(X)");
    gp1.setTitle("Bifurcation Diagram");
    addPanel(gp1);

    }

    public void initValues() {
    // set up initial values

    nR = 10;
    rStep = .001;
    roots = new double[nR];
    y = new double[nR];
    // set roots to 1 so as not to cause overflow on divide
    for (int i=0; i<nR; i++) roots[i] = 1;

    }

    public void readValues() {
    // read input panel values
    nR = (int) ip1.getValue(0);
    rStep = ip1.getValue(1);
    roots = new double[nR];
    y = new double[nR];
    // set roots to 1 so as not to cause overflow on divide
    for (int i=0; i<nR; i++) roots[i] = 1;

    }

```

A. Athanassiadis
Due 11/9/2012

```
    public void compute() {

    double rg; // r guess for root finding

    // solved for first two analytically
    roots[0] = Math.pow(2, .5);

    System.out.println("\nk\td\ta");

    // solve details of k=0,1 cases outside of the loop
    int k = 0;
    y[k] = iterf(xMax, roots[k], (int) Math.pow(2,k-1));
    System.out.println(String.format("%02d\t--\t--", k+1));

    k = 1;
    rg = roots[k-1] * 1.1;
    roots[k] = rootR(rg, k);
    y[k] = iterf(xMax, roots[k], (int) Math.pow(2,k-1));
    System.out.println(String.format("%02d\t--\t--", k+1));

    // for each k, figure out r_sk, delta, y, alpha
    for (k=2; k<nR; k++) {
        rg = roots[k-1] + .1 * (roots[k-1] - roots[k-2]);
        roots[k] = rootR(rg, k);
        delta = (roots[k-1] - roots[k-2]) / (roots[k]-roots[k-1]);
        y[k] = iterf(xMax, roots[k], (int) Math.pow(2,k-1));
        alpha = - (y[k-1] - y[k-2]) / (y[k] - y[k-1]);
        System.out.println(String.format("%02d\t%4.4f\t%4.4f", k+1, delta, alpha));

        if (Thread.interrupted()) return;

    }

    plotBifurcations(rStep, gp1, 100, 1000);




    }

}
```