A Java implementation of the Logistic Map

Daniel Kallin

1 August 2007

1 Historic background

Self organizing systems such as the natural fauna around us can be approximated with mathematical models of varying complexity. These systems can exhibit chaotic and surprisingly complex behaviour despite their simplicity. Well known examples are Conway's Game of Life and the Sandpile model. As a rule some property of such self-organizing systems follow fractal distributions. Fractals and self organization are two recently discovered disciplines but the links between the two were not discovered simultaneously with the discovery of the phenomenons themselves.

Chaotic dynamical systems were first discovered by Poincaré in the end of the 19th century during his attempts to find an exact solution to the three-body problem.[1] The discovery of non-linear behaviour in celestial mechanics was contradictory to the contemporary view on mathematics and the natural sciences as a whole. But the effects of the discovery was limited, the scientific and mathematical community lacked the tools to properly explore behaviours that lacked analytical solutions. The chaotic dynamic systems were "rediscovered" and popularized more than 170 years later with the advent of computers. It was not a mathematician but a meterologist - Edward Lorenz - who in 1961 discovered nonlinear behaviour in computer simulations of atmosphere dynamics.[2]

Figure 1: Robert May

During the seventhies the biologist Lord Robert May studied an simplified model of population growth. In his simulations he discovered chaotic behaviour. The model, called the Logistic Map, is extremely simple and yet contain several key attributes of complex, chaotic behaviour.

[1] http://www.skanaar.com/kaos.pdf, 9 August 2007.
[2] http://en.wikipedia.org/wiki/Chaos_theory, 9 August 2007.

2 The Logistic Map

In the population growth interpretation of the model, x is the population size normalized to the interval $(0, 1)$. The parameter r is a combination of the death and birth rates in the population. The model is discrete and can only predict the population size in iterations starting with an inital population size x_0.

$$0 < x_0 < 1$$

$$1 < r < 4$$

$$x_{n+1} = rx_n(1 - x_n)$$

It is interesting to study x_n's asymptotical behaviour's dependence on the parameter r. A plot of the asymptotical behaviour of x_n against the parameter r is in this report called a *bifurcation diagram*. For every fixed value of r along the horizontal axis in the graph each asyptotic value that x_n can attain is marked. If a vertical scanline crosses two lines in the graph the r-value that corresponds to that location along the horizontal axis give rise to an asymptotic periodic behaviour between two different x-values for every initial $x_0 \in (0, 1)$.

See Section 3, A Java Implementation, for a description of how the bifurcation diagrams in this report have been generated.

Figure 2:pane A shows the bifurcation diagram. Pane B is a magnification of the area enclosed in a rectangle in the left part of pane A. C and D are further magnifications. In the diagrams we can see how each line branches - bifurcates - until the details grow too small to be visible. Upon magnification we see the basic structure being repeated. This scale invariance is one of the fundamental defining properties of a fractal.[3]

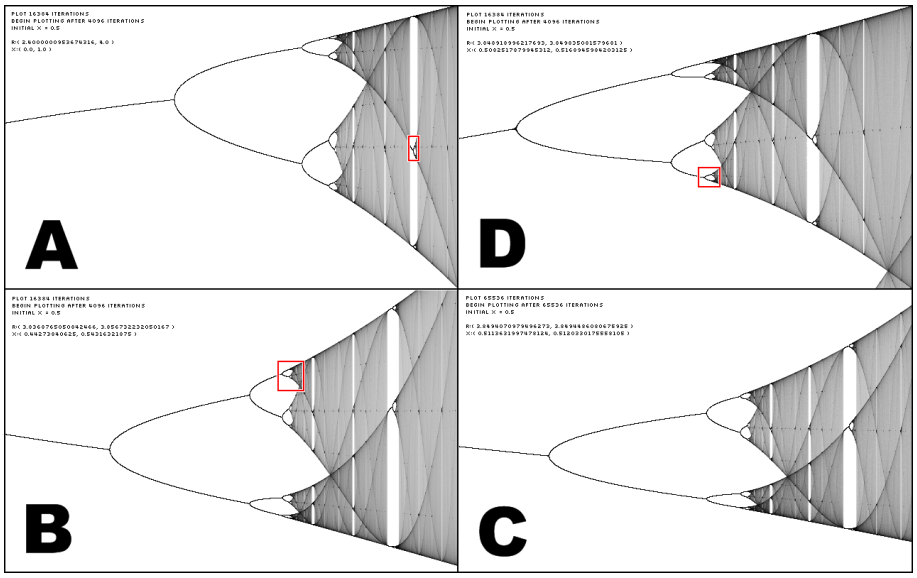

Figure 2: The bifurcation diagram under increasing magnification.

[3]Mandelbrot, B. B. (1982). *The Fractal Geometry of Nature*. W. H. Freeman and Company. ISBN 0-7167-1186-9.

3 A Java Implemetation

To study this system I have implemented the logistic map as two different programs written in Java from Sun using the Processing framework by Ben Fry and Casey Reas (*www.processing.org*). The first program, the *Initial value explorer*, visualizes the n^{th} iteration by plotting the values of x_n against the range of possible initial x_0. The second program, the *Logistic map explorer*, plots bifurcation diagrams by visualizing an entire range of iterations $x_n, n \in [n_{min}, n_{max}]$ against a range of values for the parameter r.

These programs together with source code can (as of August 2007) be found at *www.skanaar.com/chaos*.

3.1 Initial value explorer

This program allows the user to select a threshold n, value for r and a set $\mathcal{X}$ of N equally distanced values in the interval $(0, 1)$. It then calculates and plots the n^{th} iteration for each initial $x_0 \in \mathcal{X}$.

Pseudo code

```
// x0 is the initial x value
// N is the number of x0 values in the range (0,1)

x0 = 0
loop N number of times{

  x0 = x0 + 1/N
  x = x0

  loop n number of times
    x = r*x*(1-x)

  drawPoint( x0, x )
}
```

3.2 Logistic map explorer

This program allows the user to select a range and a resolution for r and then visualizes the bifurcation diagram by drawing several iteration values after an initial deep iteration sequence. For values of r in the selected range and resolution, x_0 is set to an inital value and the logistic equation is iterated n times. After this the resulting x_n is used as the inital value for m iterations that are all drawn to the screen. In this way a possible periodic behaviour can be detected. The program was made highly configurable and the following settings can be adjusted in real time:

- The method for selecting the inital x_0-value can be set to either a fixed value of 0.5 or a randomly selected value in the range $(0, 1)$.

- The method for selecting r-values out of the selected range can be set to sequential incrementing with a set resolution that corresponds to the individual pixels in the output image. A second mode is to choose a random value out of the entire range.
- The inital iteration count n and the visualized iteration count m is configurable.
- The range (r_{min}, r_{max}) can be selected through simple rubber rectangle selection in the actual output image.
- The range of visualized values (x^n_{min}, x^n_{max}) can be selected through simple rubber rectangle selection in the actual output image. This results in clipping of the output data as values for x^n that falls outside this range are ignored in the visualization.

To attain a high output image quality without aliasing effects the image is rendered in several passes with each pass slightly shifted in r values. This incremental enhancement of the image is very noticeable when running the program. A balance between speed and visual feedback was chosen in that a set number of vertical scanlines were calculated and stored and then, as a single batch, drawn to the screen at a much lower frequency than the calculation of the individual scan lines.

Pseudo code

```
loop with r in (r_min, r_max){

  x = initialValue()

  loop n number of times
    x = r*x*(1-x)

  loop m number of times
    x = r*x*(1-x)
    drawPoint( r , x )
}
```

3.3 Numerical approximation

The bifurcation diagram is filled with minute detail (infinitisemal actually, since the self-similarity is unbounded). This is one of the fractal properties of the logistic map. The hardware to compute and present a graph over the bifurcation diagram is by its nature discrete. Displays are divided into pixels and number representations in a computer have bounded precision. The latter can be alleviated by using sufficently high number precision so that other constraints manifest themselves before the inexactness of the numerical calculations are evident. One such constrain is time. When using 32-bit floating point variables one can clearly see the inexactness of the calculations. They manifest themselves as a third branch leaving a bifurcation point (see Figure 3). Similar artifacts

still exist when using 64-bit double precision floating point variables. But these should teoretically manifest themselves at a smaller scale. Floating-point arithmetic introduce a known maximum relative error during each operation. This *guaranteed maximum relative error* of a *double precision* number is known to be a factor $\frac{1}{2}^{29}$ smaller than for a *single precision* since the data-type double have 29 additional mantissa bits.[4]

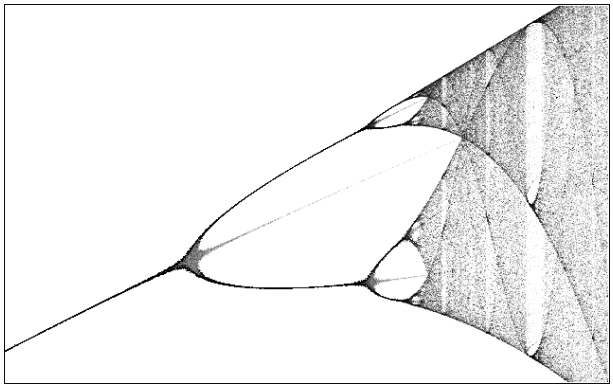

Figure 3: Deep zoom portion of the logistic diagram computed with 32-bit precision floating point arithemtic. Notice fuzzy third branch caused by rounding errors.

We let ε represent the guaranteed maximum relative error of a single precision number. After a fixed number n of floating point operations we experience a worst case relative error using single precision arithmetic of $(1+\varepsilon)^n$. When using double precision arithmetic the same maximum relative error becomes $(1+\varepsilon\frac{1}{2}^{29})^n$.

4 Results

4.1 Period Doubling

Robert May studied which conditions caused the logistic map's asymptotical behaviour to stabilize into a periodic behaviour. For small r the population size variable x quickly approached a stable state. But when r was increased past 3, x never stabilized, it assumed a oscillating behaviour, with increasing amplitude with increasing r. The population would oscillate between two values until r was increased beyond $1+\sqrt{6} \approx 3.45$ when the population would begin to oscillate between four values (see Figure 5). This behaviour called period doubling is one common attribute of chaotic systems.[5]

4.2 Chaos

At a certain threshold $r \approx 3.57$ the period doubling interpretation of the behaviour of x_n fails. The periodicity of the logistic map has constantly increased with more and more densely packed period doublings. At this point - called the

[4] http://en.wikipedia.org/wiki/IEEE_754, 9 August 2007.
[5] http://mathworld.wolfram.com/Chaos.html, 9 August 2007.

accumulation point - we expect to see a period of infinite length. At this point the models shows no periodicity but instead a chaotic unrepeating behaviour.

Beyond this threshold periodicity once again arises for certain intervals of r. These islands of stability have different lengths and also different periodic lengths. At $r = 1 + 2\sqrt{2}$ (shown by Myrberg in 1958)[6] we see a periodic oscillation not between a power of 2 number of values, but instead oscillation between 3 values. At other points it is possible to find oscillation between 5, 7, 9 or 11 values. The widest (measured in r) of these stable islands start with the 3-cycle and thus occupy the region $3.828427 < r < 3.85680$.

4.3 Initial value sensitivity

The logistic map's conformity to the characteristics of a chaotic system first observed by Poincaré also includes a sensitivity to initial values. When $r < 3.5699$ this sensitivity is only present at certain positions. In the following diagrams (Figures 4, 5 and 6) the i^{th} iterated value $f^i(x)$ is plotted against the inital x_0. The gray line ($x_n = x_0$) intersects the red function graph where we should find branches in the logistic map diagram for that specific r value. In the $r \approx 3.02$ diagram (Figure 4) we observe tree intersections instead of the two which we expect since $r \approx 3.02$ corresponds to a 2-cycle. However the vertical parts of the red graph have a derivative that diverges when the iteration number i increases. There is only one value of x_0 that corresponds to this intersection, every other value of $x_0 \in (0, 1)$ will produce either one of the two other values. The sensitivity to initial values is only exhibited at the points where the red graph is vertical.

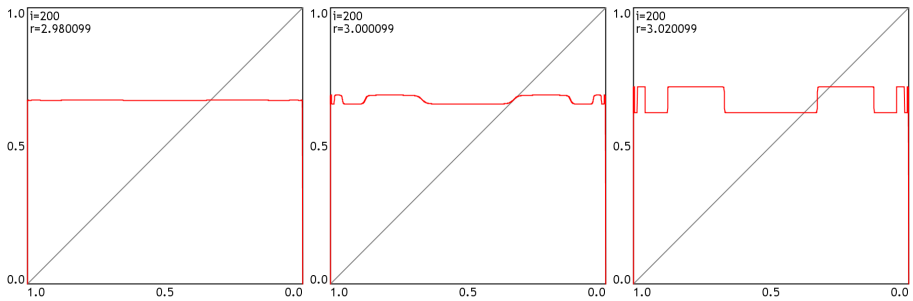

Figure 4: Sensitivity to initial values around 1^{st} bifurcation $r_2 = 3$. In the graph the 200^{th} iteration value is plotted against the initial value.

The plot when r has passed beyond the accumulation point ($r > 3.57$) has distinct differences as compared to the plots where $r < 3.57$. The graph in Figure 6 corresponds to a position where the logistic map exhibit a 3-cycle. But in contrast to the 2 and 4-cycles the iterated value exhibit an extreme sensitivity to inital conditions at points all over the interval $(0, 1)$. There appears to be a fundamental difference between the periodic behaviour that occurs on the right and the left side of the accumulation point. This different is caused by the fact that the periodicity beyond the accumulation point is caused by *mode locking*.[7]

[6] http://mathworld.wolfram.com/LogisticMap.html, 9 August 2007.

[7] http://mathworld.wolfram.com/LogisticMap.html, http://mathworld.wolfram.com/ModeLocking.html, 9 August 2007.

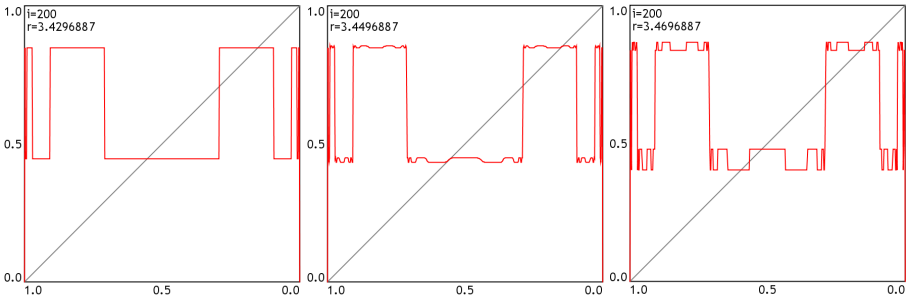

Figure 5: Sensitivity to initial values around 2^{nd} bifurcation $r_4 = 1 + \sqrt{6}$. In the graph the 200^{th} iteration value is plotted against initial value.

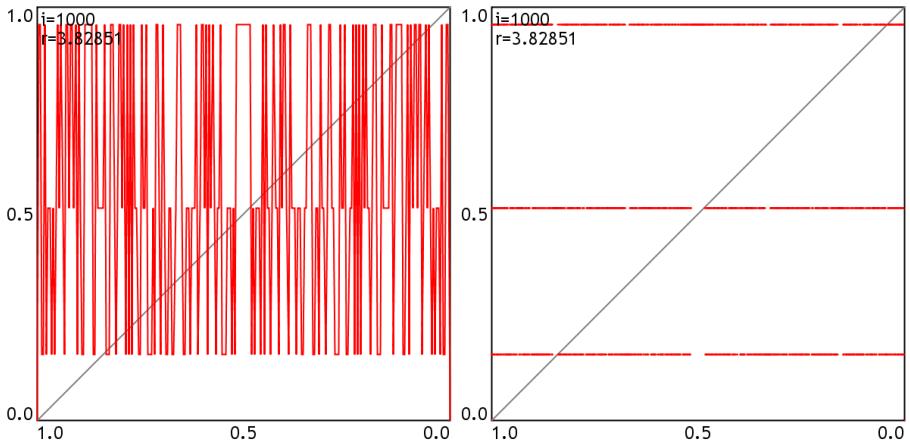

Figure 6: Response to initial values beyond $r_3 = 1 + 2\sqrt{2} \approx 3.8284$. In the graph the 1000^{th} iteration value plotted against initial value. *Left)* Continous line between datapoints. *Right)* Plotted using discrete points.

4.4 The Feigenbaum Constant

The successive bifucations manifest themselves with decreasing intervals. We know from the litterature [8] that the ratio between successive intervals approach a fixed value with increasing bifurcations. From the plots in my software the r position of the first nine bifurcations have been recorded. From the following table we see how the ratio are close to and approaches the Feigenbaum constant $d = 4.66920$.

[8] Briggs, Keith M. (1997) *Feigenbaum scaling in discrete dynamical systems* University of Melbourne (*http://keithbriggs.info/documents/Keith_Briggs_PhD.pdf*, 9 August 2007).

bifurcation	coordinate r	difference	ratio
1	3	-	-
2	3.449490	0.44949	-
3	3.544090	0.09460	4.7515
4	3.564407	0.02032	4.6559
5	3.568759	0.00435	4.6687
6	3.5696915	0.00093	4.6670
7	3.56989126	0.00020	4.6681
8	3.569934017	0.00004	4.6720
9	3.569943174	0.00001	4.6693

Table 1: Convergence towards the Feigenbaum constant bifurcation coordinate r difference ratio.

5 Appendix - source code

5.1 Initial value sensitivity explorer

Processing source code.

```
LogisticDiagram logMap;
PFont font;
boolean drawPoints=false;
//-----------------
void setup(){
  size(600,600);
  font = loadFont("TrebuchetMS-24.vlw");
  textFont(font);
  logMap = new LogisticDiagram();
  logMap.setCanvas(50,5,width-55,height-55);
  background(128);
  logMap.draw();
}
void draw(){}
void keyPressed(){
  if(key==' ') saveFrame("logmap-##.tif");
  if(key=='x') logMap.r+=0.02;
  if(key=='z') logMap.r-=0.02;
  if(key=='s') logMap.r+=0.0001;
  if(key=='a') logMap.r-=0.0001;
  if(key=='w') logMap.iterationDepth++;
  if(key=='q') logMap.iterationDepth--;
  if(key=='r') logMap.xRes*=0.5;
  if(key=='e') logMap.xRes*=2;
  if(key=='d') drawPoints = !drawPoints;
  if(logMap.iterationDepth==0) logMap.iterationDepth=1;
  if(logMap.xRes<1/4096f) logMap.xRes = 1/4096f;
  println("xRes: "+logMap.xRes);
  logMap.draw();
}
class LogisticDiagram{
  float xOff, yOff, w, h, r=2, xRes=0.002;
  int iterationDepth = 15;
  void setCanvas(float x, float y,float w, float h){
    this.xOff = x;
    this.yOff = y;
    this.w = w;
    this.h = h;
  }
  void draw(){
    background(255);  smooth();
    pushMatrix();     translate(xOff,yOff);
    fill(255);        stroke(0);            strokeWeight(2);  rect(0,0,w,h);
    stroke(128);      line(0,h,w,0);
    stroke(255,0,0);  strokeWeight(2);      noFill();

    if(drawPoints) beginShape(POINTS);
    else beginShape();
    for(float x=0,x0=0;x0<1;x=x0=x0+xRes){
      for(int i=0;i<iterationDepth;i++)
        x = r*x*(1-x);
      vertex( w*x0, h-h*x );
    }
    vertex( w, h );
    endShape();

    fill(0);           textAlign(RIGHT);
    text("1.0",-5,0+20);  text("0.5",-5,h/2+10);  text("0.0",-5,h);
    textAlign(LEFT);    text("1.0",0,h+25);
    textAlign(CENTER);  text("0.5",w/2,h+25);
    textAlign(RIGHT);   text("0.0",w,h+25);
    textAlign(LEFT);    text("i="+iterationDepth,5,25);    text("r="+r,5,50);

    popMatrix();
  }
}
```

5.2 Logistic map explorer

Processing source code.

```
ImgBuffer imgBuffer;
PFont font;
int infinity = 1024;
int periodChunk = 1024;
int parameterChunk = 64;
int scanChunk = 64; // must divide width
int scanline = 0;
double rMin=2.4f, rMax=4f;
double xMin=0f, xMax=1f;
int mouseDownX, mouseDownY;
int spectrum=0;
boolean fixedInitialValue = true;
boolean stochasticRendering = false;
//-----------------
void setup(){
  size(640,400);
  imgBuffer = new ImgBuffer( width, height );
  font = loadFont("font.vlw");
  textFont( font );
}
//-----------------
void draw(){
  if(!mousePressed){
    if(stochasticRendering)
      for(int i=0;i<parameterChunk;i++)
        logisticMap( rMin + (rMax-rMin)*Math.random() );
    else
      if(scanline<width){
        double rFactor = (rMax-rMin)/width;
        for(int i=0;i<scanChunk;i++)
          logisticMap( Math.random()*rFactor + rMin + (scanline++)*rFactor );
        if(scanline>=width) scanline=0;
      }
  }

  imgBuffer.drawBuffer();

  if(mousePressed) {
    noFill();
    stroke(255,0,0);
    rect(mouseDownX,mouseDownY,mouseX-mouseDownX,mouseY-mouseDownY);
  }

  text( "plot "+periodChunk+" iterations", 10, height-60 );
  text( "begin plotting after "+infinity+" iterations", 10, height-50 );

  if(fixedInitialValue) text( "initial x = 0.5", 10, height-40);
  else text( "initial x randomized", 10, height-40);

  double mr = rMin+mouseX*(rMax-rMin)/width;
  double mx = xMin+mouseY*(xMax-xMin)/height;
  text( "mouse :( "+mr+", "+mx+" )", 10, height-30);

  text( "r:( "+rMin+", "+rMax+" )", 10, height-20 );
  text( "x:( "+xMin+", "+xMax+" )", 10, height-10 );
}
//-----------------
void logisticMap(double r){
  double x, rFactor = width/(rMax-rMin);
  double xFactor = height/(xMax-xMin);
  x = fixedInitialValue ? (double)0.5 : (double)random(0,1);

  for(int i=0;i<infinity;i++)
    x = r*x*(1-x);

  for(int i=0;i<periodChunk;i++){
    imgBuffer.drawPoint( rFactor*(r-rMin) , xFactor*(x-xMin) );
    x = r*x*(1-x);
  }
}
//-----------------
```

```
void clearMap(){
  imgBuffer.clearBuffer();
  scanline = 0;
}
//-----------------
void mousePressed(){
  mouseDownX = mouseX;
  mouseDownY = mouseY;
}
//-----------------
void mouseReleased(){
  int l,t,r,b;
  if(mouseButton==LEFT){
    // flips the rubber rectangle if it has been drawn upsidedown/mirrored
    r = (mouseX>mouseDownX) ? mouseX     : mouseDownX ;
    l = (mouseX>mouseDownX) ? mouseDownX : mouseX ;
    b = (mouseY>mouseDownY) ? mouseY     : mouseDownY ;
    t = (mouseY>mouseDownY) ? mouseDownY : mouseY ;
    if( min(r-l,b-t)>4 ) zoom(l, t, r, b);
  }
  else if( mouseButton==RIGHT)
    zoom(2.0);
}
//-----------------
void zoom(float factor){
  rMax = factor*0.5*(rMax-rMin) + (rMax+rMin)/2;
  xMax = factor*0.5*(xMax-xMin) + (xMax+xMin)/2;
  rMin = -factor*0.5*(rMax-rMin) + (rMax+rMin)/2;
  xMin = -factor*0.5*(xMax-xMin) + (xMax+xMin)/2;
    clearMap();
}
//-----------------
void zoom(float l, float t, float r, float b){
  double t1 = (rMax-rMin)*r/(double)width + rMin;
  double t2 = (xMax-xMin)*b/(double)height + xMin;
  rMin = (rMax-rMin)*l/(double)width + rMin;
  xMin = (xMax-xMin)*t/(double)height + xMin;
  rMax = t1;
  xMax = t2;
  clearMap();
}
//-----------------
void keyPressed(){
  if(!online && key=='s'){
    saveFrame("bifurcation-##.tif");
    return;
  }
  if(!online && key=='a'){
    imgBuffer.drawBuffer();
    saveFrame("bifurcation-##.tif");
    return;
  }
  else if(key=='f')
    fixedInitialValue = !fixedInitialValue;
  else if(key=='r')
    stochasticRendering = !stochasticRendering;
  else if(key=='x')
    infinity = infinity*2;
  else if(key=='z')
    infinity = (infinity+1)/2;
  else if(key=='m')
    periodChunk = periodChunk*2;
  else if(key=='n')
    periodChunk = (periodChunk+1)/2;
  else if(key==' '){
    rMin=2.4;
    rMax=4f;
    xMin=0f;
    xMax=1f;
  }
  clearMap();
}
```