Math 411 project 1: Method for Calculating Feigenbaum's Scaling Constant δ for One-Dimensional Discrete Maps, Using the Points of Superstable Cycles

Anton Nikiforov
Department of Physics and Astronomy
Brigham Young University, Provo, UT

April 28, 2006

1 Why it is a nice project?

Idea of the project was to implement the knowledge of numerical methods gained in the class 411 into a working computer program written in Matlab, which, actually, solves a real scientific problem.

The scientific problem that seemed interesting to me was calculation of Feigenbaum's scaling constant δ, suggested by K.Briggs[1]. The way people calculate it in regular textbooks is a little bit handwavy and is not exact.

In the program I used Newton's method to find a solution of equation (8).

2 Why bother doing that? (Why Feigenbaum constants are important)

The Feigenbaum constant δ is a universal constant for functions approaching chaos via period doubling. It was discovered by Mitchell Feigenbaum in late 1970's while studying the fixed points of iterated functions in the form

$$x_{n+1} = f(x_n, r), \tag{1}$$

and characterizes the geometric approach of the bifurcation parameter to its limiting value as the parameter r is increased for fixed x.

Amazingly, the Feigenbaum constant is "universal" for all one-dimensional maps if has a single locally quadratic maximum [3].

Discrete maps are not just an abstract theoretical thing. For systems with continuous time, it is possible to construct a Poincare section, which literally is a discrete map. Procedure of building Poincare section is often used as an instrument for research of nonlinear systems' dynamics. Discussing nonlinear systems in terms of Poincare section, we can draw general conclusions about the systems, described by differential equations (both autonomous and non-autonomous), as well as about discrete (iterative) maps. For example, multiple period-doubling bifurcations reveal one of possible ways of how turbulence flows might be created [2]

3 Some rigorous theory

To begin with, let me remind you some facts about Mitchell Feigenbaum's Universality Theory. Continuous mapping of a segment into itself is called unimodal, if inside of the segment there's an extremum x_0, and astride of it the mapping is strictly monotone. We'll further consider just unimodal mappings of the form (1).

If set x_n for a certain r consists of n points, we'll call this set n-cycle, so that

$x_1 = f(x_0), x_2 = f(x_1), ..., x_n = f(x_{n-1})$, or $x_n = f^n(x_0)$

The points of a cycle that satisfy

$$x^* = f(x^*) \tag{2}$$

is called fixed.

Multiplicator $\Lambda(R_i) = \prod_{i=0}^{n} \frac{df_{R_i}(x_i)}{dx}$ determines stability of a n-cycle and is called stability ([1], p.121). n-cycle is called stable if $|\Lambda| < 1$.

n-cycle that contains x_0 is called superstable. For that cycle $|\Lambda| = 0$ will be satisfied.

Let's denote by r_n such values of parameter r, that number of points in the cycle doubles (becomes equal to 2^n). In 1978 M.Feigenbaum showed [3], that values of r_n satisfy the following scaling relation:

$$\delta = \lim_{n \to \infty} \frac{r_n - r_{n-1}}{r_{n+1} - r_n}, \tag{3}$$

which is often written in literature as

$$r_k - r_\infty \propto -C_0[\delta(d)]^{-k}, n \gg 1(see[1]) \tag{4}$$

or $\Delta r_k = r_k - r_\infty \propto C_0[\delta(d)]^{-k}$, [2]. Intervals d_n from $x = x_0$ (where x_0 is an extremum of a given discrete map. E.g. on the picture 1, $x_0 = \frac{1}{2}$) to the closest cycle point satisfy the following relation:

$$\frac{d_n}{d_{n+1}} = -\alpha, n \gg 1 \tag{5}$$

Feigenbaum constant have the following value: $\delta = 4.6692016091\ldots$

Isn't that amazing, that for any unimodal mappings Feigenbaum constant is exactly the same!

Table 1: Input and output data

INPUT DATA	OUTPUT DATA
Start iterations of function f with $f_0(R) = x_0$ Iterate derivative of function starting with $f'_0(R)$ Initial approximations for two values of parameter R: R_0^0, R_1^0 Reasonable initial approximation for the constant: $\delta_{min} < \delta_1 < \delta_{max}$	δ

4 Algorithm for calculation of Feigenbaum constant

It is worthwhile to note, that R_i points also can be used to calculate $\Lambda(r)$, and we will use them from now on. At points R_i multiplicator $\Lambda(r)$ is always zero, which means that those cycles are stable:

$\Lambda(R_i) = \prod_{i=0}^{n} \frac{df_{R_i}(x_i)}{dx} = f'(x_0) \cdot f'(x_1) \cdot \ldots \cdot f'(x_{n-1}) = 0 < 1$

For example, for a cycle of period 2:

$f'(x_0, R_1) \cdot f'(x_1, R_1) = 0$, where $x_1 = f(x_0, R_1)$ $f(x_1, R_1) = f(f(x_0), R_1) = x_0$, so that

$$f_{2^1}(x_0, R_1) - x_0 = 0 \tag{6}$$

For a cycle of period 4: $f'(x_0, R_2) \cdot f'(x_1, R_2) = 0$, where $x_1 = f(x_0, R_2)$ $f(x_1, R_2) = f(f(x_0), R_2) = x_0$, so that

$$f_{2^2}(x_0, R_2) - x_0 = 0 \tag{7}$$

For 2^n-cycles we can write:

$$f_{2^n}(x_0, R_2) - x_0 = 0 \tag{8}$$

Equation (8) can easily be solved for R_n, for example, with Newton's method:

$$R_n^{i+1} = R_n^i - \frac{f_{2^n}(x_0, R_n^i) - x_0}{f'_{2^n}(x_0, R_n^i)}, \tag{9}$$

where i is the number of iteration. So we see, that the calculation of constant δ is reduced to finding the values of parameter R, at which bifurcation diagram crosses $x = x_0$ line. To do that, we need to solve (8) by iterating it 2^n times.

The process can be described with the following expressions:

$$R_n^0 = R_{n-1} + \frac{R_{n-1}}{\delta_{n-1}}, n = 2, 3, 4...$$

$$R_n^{i+1} = R_n^i + \frac{f_{2^n}(x_0, R_n^i) - x_0}{f'_{2^n}(x_0, R_n^i)}, n = 0, 1, 2...$$

$$f_k(R)$$

$$\frac{f_k(R)}{R}$$

$$R_n = \lim_{n\to\infty} R_n^i$$

$$\delta = \frac{R_{n-1} - R_{n-2}}{R_n - r_{n-1}}$$

$$\delta = \lim_{n\to\infty} \delta_n$$

Let's consider some examples and see how these formulas can be used for calculation.

Example 1:

$$x \leftarrow r - x^2$$

$$x_{n+1} = f(x_n) = r - x_n^2 \tag{10}$$

For a given value of x_n, f will depend just on constant r. Let's denote this function as $\tilde{f}(R)$. Then (10) can be rewritten as

$$\tilde{f}_k(R) = R - [\tilde{f}_{k-1}(R)]^2$$

$$\frac{\partial \tilde{f}_k(R)}{\partial R} = 1 - 2 \cdot \tilde{f}_{k-1}(R)$$

$$x_0 = 0$$

Example 2:

$$x \leftarrow r \cdot x \cdot (1 - x)$$

$$x_{n+1} = \varphi(x_n) = r \cdot x_n - r \cdot x_n^2$$

$$\tilde{\varphi}_k(R) = R \cdot \tilde{\varphi}_{k-1}(R) - R \cdot \tilde{\varphi}_{k-1}(R) \cdot \tilde{\varphi}_{k-1}(R)$$

$$\frac{\partial \tilde{\varphi}_k(R)}{\partial R} = \tilde{\varphi}_{k-1}(R) + R \cdot \tilde{\varphi'}_{k-1}(R) - [\tilde{\varphi^2}_{k-1}(R) + 2 \cdot R \cdot \tilde{\varphi'}_{k-1}(R) \cdot \tilde{\varphi'}_{k-1}(R)]$$

$$x_0 = \frac{1}{2}$$

Example 3:

$$x \leftarrow r \cdot sin(\pi x)$$

$$x_{n+1} = \psi(x_n) = r \cdot sin(\pi x_n)$$

$$\tilde{\psi}_k(R) = R \cdot sin(\pi \tilde{\psi}_{k-1}(R))$$

$$\frac{\partial \tilde{\psi}_k(R)}{\partial R} = sin(\pi \tilde{\psi}_{k-1}(R)) + R \cdot \tilde{\psi'}_{k-1}(R) \cdot \pi \cdot sin(\pi \cdot \psi_{k-1}(R))]$$

$$x_0 = \frac{1}{2}$$

The program shows a fairly good accuracy (6 digits after decimal point)

Table 2: Dependence of calculated δ on the number of iterations i

i	δ
1	6.9032539091...
2	4.7443094689...
3	4.6744478277...
4	4.6707911502...
5	4.6694616483...
6	4.6692658098...
...	...
11	4.66920173800930...

5 Conclusions

The program that calculates Feigenbaum constant, using the method suggested by K.Briggs[1], was written. It's the most accurate and fast method known to the author. However, I'm still in the search for other methods of calculation scaling constants.

References

[1] K.Briggs ”Feigenbaum Scaling in Discrete Dynamical Systems”, PhD thesis, 1997

[2] L.D.Landau, E.M.Lifshitz, ”Fluid Mechanics, Second Edition : Volume 6 (Course of Theoretical Physics)”, (1987)

[3] M.Feigenbaum. Universal behavior in nonlinear systems. Physica D, 7:16, 1983

[4] H.G. Shuster, Deterministic Chaos, (Springer, Heidelberg, 1982)

[5] E. Vul, Ya. Sinai, K. Khanin, Feigenbaum universality and the thermodynamic formalism,Russ. Math. Surv. 39 (1984), 1-40. 82

[6] N. Kalitkin, Numerical Methods. Textbook, Moscow, ”Science”, 1978

```
function project1

delta = calculateDelta(0, 0, 1, 0, 0, 3.2);
fprintf('Feigenbaum constant delta=%d\n', delta);

   function [f, df] = getFunctions(numberOfCycles, r, fInitial, fInitialDerivative)
    % function: getFunctions
    % getFunctions calculates values of derivatives of maps and maps themselves

    fOld = fInitial;
    fOldDerivative = fInitialDerivative;

    iLast = round(2^numberOfCycles);

    for i = 1:iLast
      fNew = func(r, fOld);
      fNewDerivative = dFunc(fOld, fOldDerivative);
      fOld = fNew;
      fOldDerivative = fNewDerivative;
    end

    f = fNew;
    df = fNewDerivative;

end;


    function out = calculateRi(r, f0, df0, i)
    % function: calculateRi
    % calculateRi calculates points of bifurcations

    rOld = r;

    for n = 1:1000
      [f,df] = getFunctions(i, rOld, f0, df0);
      rNew = rOld - f/df;
      rOld = rNew;
    end

    out = rNew;

end;

    function out = calculateDelta(x0, R00, R01, f0, df0, delta)
    % function: calculateDelta
    % calculateSigma calculates Feigenbaum constant delta for
    % a discrete map defined by func(r, x)
    % INPUT:
    % x0 - extremum of a discrete map
    % R00, R01 - Initial approximations for two values of parameter R
    % f0, df0 - Start iterations of function func with f0, and initial
    %           approximation to the derivative df0
    % delta - Reasonable initial approximation for the constant
    % OUTPUT:
    % out - value of the constant

    R1 = calculateRi(R00, f0, df0, 0)
    R2 = calculateRi(R01, f0, df0, 1)

    for i = 2:10
      R3 = calculateRi(R2+(R2-R1)/delta, f0, df0, i);
      delta = (R2-R1)/(R3-R2);
      R1 = R2;
      R2 = R3;
      fprintf('delta=%d\n', delta);
    end

    out = delta;

end;


function out = func(r, x)
    % function: func
    % defines a discrete map
    % INPUT:
    % x -  value of x we want to calculate function at
    % r -  parameter
    % OUTPUT:
    % out - function
         out = r - x^2;
end; % of function func(x)

function out = dFunc(f, df)
    % function: dFunc
    % defines a derivative of a discrete map

    out = 1 - 2*df*f;
end; % of function dFunc(x)

end % of project1
```

Matlab output:

```
R1 =

     0


R2 =

     1

delta=3.218511e+000
delta=4.385678e+000
delta=4.600949e+000
delta=4.655130e+000
delta=4.666112e+000
delta=4.668549e+000
delta=4.669061e+000
delta=4.669172e+000
delta=4.669195e+000
Feigenbaum constant delta=4.669195e+000
```