Problem Set Number 07, (18.353/12.006/2.050)j
MIT (Fall 2023)

Rodolfo R. Rosales (MIT, Math. Dept., room 2-337, Cambridge, MA 02139)

November 29, 2023

Due December 5, 2023 (turn it in via the canvas course website).
I may add one more problem before Friday December 1, noon. Check to see.

Contents

1 Liapunov exponents for 1-D maps #02

Statement: Liapunov exponents for 1-D maps #02

Compute the Liapunov exponent, and produce a figure analog to figure 10.5.2 in Strogatz's book (see example 10.5.3 there) **for the 1-D maps $x_{n+1} = f(x_n)$ below.** In all the cases, **given the range of r selected, justify the selected range for x.**

Meaning of "justify". Show that the x-region is such that it may contain an attractor, because either: (a) It is trapping; an orbit starting there stays there; or (b) Orbits starting outside the region diverge to infinity, so that any attractor has to be inside.

1. The **cosine map** $f(x) = r\,\cos(x)$, with $-5 \leq r \leq 5$ and $-5 \leq x \leq 5$.
2. The **quartic map** $f(x) = r\,(1 - (2\,x - 1)^4)$, with $0 \leq r \leq 1$ and $0 \leq x \leq 1$.
3. The **cusp075 map** $f(x) = r\,(1 - z^\mu)$, with $0 \leq r \leq 1$ and $0 \leq x \leq 1$, where $z = |2\,x - 1|$ and $\mu = 3/4$. **Important.** $\mathrm{d}f/\mathrm{d}x$ for this map involves $z^{\mu-1}$. To **avoid a potential division by zero,** when calculating use $z = |2\,x - 1| + \epsilon$, where ϵ is very small; e.g.: $\epsilon = 10^{-200}$.

In all cases, plot not just the region listed above, but **do a detail of the r-ranges where the exponent transitions from negative to positive. See also the optional task below, after the hints.**

Hints and related.

h1. The **process to follow to calculate the Liapunov exponent** is explained in example 10.5.3 of Strogatz book [also in the lectures]. At any rate, see the *sample/sketch MatLab script* at the end of the problem statement.

h2. If you use MatLab, "vectorize" the operation, so that you do **all** the r's simultaneously.

h3. In MatLab the command "print -dpng FigureName" added to the script will save the figure as a **small png file** — it is also more reliable saving the figure using the GUI in the figure window. **Please be careful with the figure sizes, do not upload monster size answers.** Just to give you a reference: in my answer the pictures take about 30kb each.

Optional task. For the case of the **cusp075** map, plot the Liapunov exponent versus r for the region where the Liapunov exponent transitions from negative to positive — this is, roughly $0.52 < r < 0.68$. Use for this a program following the outline of the "sample/sketch MatLab script" below. Run the program a few times (without changing any parameters) and do a few plots. **What do you see?** (you should be seeing something that "should not" be, remember that you are looking at a deterministic system). **Explain why you see what you see.** Note: *The explanation is simple and clean. If you find yourself making convoluted arguments, you are on the wrong track!* **Hints:** (i) Do an orbit/bifurcation diagram for the map. (ii) Initialize the iterations that compute the Liapunov exponent with $x_0 = 0.51$. (iii) Initialize the iterations that compute the Liapunov exponent with $x_0 = 0.01$.

Sample/sketch MatLab script. These are the parameters that you will need to assign values to:
r1 = Lower value of the parameter r range to explore.
r2 = Upper value of the parameter r range to explore.
N = Number of r-values to use between r1 and r2. Because the Liapunov exponent as a function of r can be very "wiggly", take N large, say: N = 1000, or larger.
x1 = Lower value of x considered.
x2 = Upper value of x considered.
nb = Number of map iterations before the calculation starts. This should be a fairly large number, say nb = 5000, to allow the iterates to settle on the attractor.
np = Number of iterations used to calculate the Liapunov exponent. Again, a fairly large number, to obtain an accurate calculation. Note that the size of the error is, typically, $O(1/np)$**!**
In addition, **you will need two sub-scripts,** y = Fun(r, x) and y = dFun(r, x), which compute the function $f(x)$, and the absolute value of its derivative, $|df/dx|$. The basic script is then:

```
r = r1 + (r2 - r1)*(0:N)/N;
x = x1 + (x2 -x1)*rand(size(r)); % This initializes the iteration.
for j=1:nb; x = Fun(r, x); end; % Iterate nb times to approach the attractor.
nf = nb + np;
le = zeros(size(r)); % Will contain the Liapunov exponent for each value in the array r.
for j=(nb+1):nf
    x = Fun(r, x);
    le = le + (1/np)*log(dFun(r, x));
end
```

Now all that remains to do is plot le versus r.

2 Newton's method in the complex plane #01

Statement: Newton's method in the complex plane #01

Suppose that you want to solve an equation, $g(x) = 0$. Then you can use *Newton's method*, which is as follows: Assume that you have a "reasonable" guess, x_0, for the value of a root. Then the sequence $x_{n+1} = f(x_n)$, $n \geq 0$, where converges (very fast) to the root.

$$f(x) = x - \frac{g(x)}{g'(x)}, \tag{2.1}$$

Remark 2.1 (The idea). Assume an approximate solution $g(x_a) \approx 0$. Then write $x_b = x_a + \delta x$ to improve it, where δx is small. Then $0 = g(x_a + \delta x) \approx g(x_a) + g'(x_a)\,\delta x \;\Rightarrow\; \delta\, x \approx -\frac{g(x_a)}{g'(x_a)}$, and (2.1) follows.
Of course, if x_0 is not close to a root, the method may not converge. Even if it converges, it may converge to a root that is far away from x_0, not necessarily the closest root. In this problem **we investigate the behavior of Newton's method in the complex plane, for arbitrary starting points.** ♣

Consider iterations of the map generated by Newton's method for the roots of $z^3 - 1 = 0$. i.e.: where $0 < |z_0| < \infty$ is arbitrary, and the z_n are **complex numbers.**

$$z_{n+1} = f(z_n) = \left(\frac{2}{3} + \frac{1}{3\,z_n^3}\right) z_n, \quad n \geq 0, \tag{2.2}$$

Note that are the roots of $z^3 = 1$.

$$\zeta_1 = 1, \quad \zeta_2 = e^{i\,2\,\pi/3} = \frac{1}{2}(-1 + i\,\sqrt{3}), \quad \text{and} \quad \zeta_3 = e^{i\,4\,\pi/3} = \frac{1}{2}(-1 - i\,\sqrt{3}), \tag{2.3}$$

Your tasks: Write a computer program to calculate the orbits $\{z_n\}_{n=0}^{\infty}$. Then, for every initial point z_0, draw a colored dot at the position of z_0, where **the colors are picked as follows:**

$$z_n \to \zeta_1,\ \textbf{green.} \qquad z_n \to \zeta_2,\ \textbf{red.} \qquad z_n \to \zeta_3,\ \textbf{blue.} \qquad \textbf{No convergence, black.} \tag{2.4}$$

What do you see? Do blow ups of the limit regions between zones.

Hints and practical numerical considerations.

h1. Divide the region where the initial data z_0 will be picked [I suggest the square $-2 \leq \mathrm{Re}(z_0),\ \mathrm{Im}(z_0) \leq 2$] into pixels, then pick a z_0 at the center of each pixel, and color the pixel according to (2.4).

h2. If you use MatLab, **do not plot points.** As suggested in item **h1** plot pixels — use the command image$(x,\ y,\ C)$ to plot, where $x = \mathrm{Re}(z_0)$ and $y = \mathrm{Im}(z_0)$. **Why?** Because using points leaves a lot of unpainted space in the figure, and **gives huge file sizes** if you use enough pixels to get a good picture.

h3. **Deciding convergence.** Deciding that the sequence converges is easy: once z_n gets "close enough" to one of the roots, then the very design of Newton's method guarantees convergence. Thus, given a z_0, compute z_N for some large N, and check if $|z_N - \zeta_j| < \delta$ for one of the roots and some "small" tolerance δ — which does not have to be very small, in fact $\delta = 0.25$ is good enough. If this criteria is not satisfied for any of the roots, then classify the sequence starting at z_0 as "non-convergent".

You can get reasonable pictures with $N = 50$ iterations on a 150×150 grid — a larger N is needed when refining near the boundary between zones. For the answer I used a 500×500 grid and $N = 100$ iterations — which I increased to $N = 200$ and $N = 300$ for the blow ups of details.

h4. **Compute in parallel.** If you use MatLab, make sure to do all the sequences (one for each pixel) in parallel, using vector/matrix operations. This is much faster than a "for loop".

h5. **Avoid division by zero.** Note that (2.2) ceases to make sense if $z_n = 0$ — classify this as non-convergence. This can cause a problem if you are computing all the sequences in parallel, because this requires all of them to be computed from z_0 to z_N. One way to get around this (in MatLab) is as follows: Place all the iterates in a complex matrix **Zn**, where the entry $(p,\ q)$ corresponds to z_n for the sequence starting in the $(p,\ q)$ pixel. Then, before computing the next iterate, execute: **Zn = Zn + del*(Zn == 0)**, where **del = 1e-30**.† After this sequences with $z_n = 0$ will produce a very large z_{n+1}, which is guaranteed not return to the vicinity of the roots ζ_j for many iterations (more than 300), resulting in "effective" non-convergence.‡

† This replaces zero entries in **Zn** by **del**, because the logical operator **(Zn == 0)** yields zero for all non-zero entries in **Zn**, and one for zero entries.

‡ The result will be $z_{n+1} \approx (1/3)10^{60}$, while for z_n large (2.2) reduces to $z_{n+1} \approx (2/3)\, z_n$. Hence returning to $z_{n+M} = O(1)$ requires, roughly, $(2/3)^M\, 10^{60} = O(1)$.

THE END.