

Fractals and Chaos - opt.2606 - Practical Problems

R. Willingale and D.J. Raine

September 26, 2011

Contents

1	Introduction	3
2	Syllabus	3
2.1	Fractals and processes which produce them	3
2.1.1	Julia sets	3
2.1.2	Iterated Function Systems	3
2.2	Dynamical Systems	3
2.2.1	Discrete systems	3
2.2.2	Systems with continuous variables	3
2.2.3	Routes to chaos	4
3	Books and References	4
4	Problems	4
4.1	Problem 1 - Calculating and plotting Julia sets	5
4.2	Problem 2 - Calculating and plotting the Mandelbrot set	6
4.3	Problem 3 - Generating a fern picture using affine transformations	7
4.4	Problem 4 - Plotting the 4 affine transformations from the fern IFS	8
4.5	Problem 5 - Plotting the bifurcation diagram of the logistic map	8
4.6	Problem 6 - Plotting the Lyapunov exponent of the logistic map	8
4.7	Problem 7 - Plotting time series and phase portraits for a driven LCR circuit	9
4.8	Problem 8 - Plotting a bifurcation diagram for a non-linear LCR circuit	10
4.9	Problem 9 - A practical demonstration of the varactor LCR circuit	11

1 Introduction

The problems described below form part of the opt.2606 course entitled Fractals and Chaos. These problems will teach you how to generate fractals and how to simulate chaotic systems using a computer. They are to be completed in 8 hours of workshop sessions distributed throughout the course. You will be required to complete and modify skeleton programs which are written in C. You will also be able to compare the behaviour of a real chaotic system with a computer simulation.

Details about the workshops including this script can be found at URL:

<http://www.star.le.ac.uk/~rw/fractals/>

2 Syllabus

2.1 Fractals and processes which produce them

2.1.1 Julia sets

key concepts; iteration, simple formulae leading to a very high level of complexity, fractal geometry, self-similarity, scaling, fractal dimensions, connectivity of sets - connected (one piece) or a Cantor set (dust of infinitely many points) - the Mandelbrot set

2.1.2 Iterated Function Systems

modelling fractals in the real world; Sierpinski triangle, affine transformations, clouds, smoking chimneys and ferns, deterministic and random systems, Koch curves, landscapes, mountains, rivers and coastlines, perimeter-area relation for fractal objects

2.2 Dynamical Systems

2.2.1 Discrete systems

Iteration of 1-D and 2-D maps, regular, quasi-periodic and chaotic behaviour, logistic equation, period doubling, bifurcation sequences, sensitivity to initial conditions, Lyapunov exponents, attractors

2.2.2 Systems with continuous variables

equations of motion, solution by iteration of difference equations, phase space, phase portraits, Poincaré sections, connection with discrete systems, basins of attraction, Fourier analysis, - power spectra, dimensionality of dynamical systems, delay coordinates

2.2.3 Routes to chaos

Forced motion (dissipative systems), Hamiltonian systems, universality, Feigenbaum's number, strange attractors, the Lorenz system, fractal dimensions, non-linear dynamics, feedback, coupled equations, examples of chaotic physical systems

3 Books and References

The Science of Fractal Images, Eds. Heinz-Otto Peitgen and Dietmar Saupe, Springer-Verlag, 1988, ISBN 0-387-96608-0

Fractals, Jens Feder, Plenum NY and London, 1988, 0306428512

Chaos and Fractals, H.O. Peitgen, H. Jurgens, D. Saupe, Springer-Verlag NY, 1992, 3540979034

Fractal Images and Chaos, H. Lauwerier, Penguin, 1991, 0140144110

Chaotic Dynamics, G.L. Baker and J.P. Gollub, CUP, 1990, 0521476852

Fractals and Chaos, P.S. Addison, IoP, 1997, 0750304006

Period doubling and chaotic behaviour in a driven anharmonic oscillator, P.S. Linsay, Phys. Rev. Lett. 47, 1349, 1981

Evidence for the universal chaotic behaviour of a driven nonlinear oscillator, J. Testa, José Pérez and C. Jeffries, Phys. Rev. Lett. 48, 714, 1982

Chaotic Dynamics and Instabilities in solids, C.D. Jeffries, Physica Scripta Vol. T9, 11, 1985

4 Problems

The skeleton C code for the problems must be copied onto your own directory. For example, the source code file for problem 1 is called `fractal1.c` and must be copied using the `cp` command. Note that the final dot in the command line puts the file into the current directory.

```
$ cp /home/z/zrw/under/fractal1.c .
```

The other skeleton source code files should be copied in the same way, as required. You can edit the source code using a text editor in the same way as in the 1st and 2nd year C programming workshops.

Instructions for compiling and running the programs are provided within the source code. The programs use the PGPLOT graphics routines to plot graphs and pictures.

When you run the programs they will prompt for a plotting device. If you type `?` then all the device names will be listed. The device name `/xs` should be used to produce a plot on your terminal screen. You can also produce a gif file using `filename.gif/gif` or a postscript file using `filename.ps/cps` (landscape) or `filename.ps/vcps` (portrait) and you can view the postscript file using `gv`:

```
$ gv filename.ps &
```

The postscript files can also be printed as hardcopy:

```
$ lp -dA4 filename.ps
```

As you proceed through the problems you MUST keep notes recording what you do, results you obtain and new ideas and concept which you learn. If you try things which go beyond the scope of the script record these as well. At the end of the lectures and workshops your notes should provide a complete record of the course and what you have learnt.

4.1 Problem 1 - Calculating and plotting Julia sets

The skeleton program source file is `fractal1.c`. This implements an algorithm known as the Level Set Method for producing a rendering or picture of a Julia Set (or more strictly a filled-in Julia set).

The Julia set is built around a mapping of the form:

$$z \rightarrow f_c(z)$$

where c is a constant. The simplest functional form which produces complicated Julia sets is:

$$f_c(z) = z^2 + c$$

Repeated application of this mapping produces an orbit in the complex plane:

$$z \rightarrow f_c(z) \rightarrow f_c(f_c(z)) \rightarrow \dots \rightarrow f_c^k(z)$$

After k iterations we end up at the point:

$$z_c^k = f_c^k(z)$$

If $f_c^k(z)$ never reaches ∞ , even as $k \rightarrow \infty$, then the initial point z is a member of the filled-in Julia set characterised by the constant c . On the other hand, if the orbit escapes to ∞ then z is not a member of the Julia set.

In practise we cannot iterate an infinite number of times and we cannot measure if the orbit truly escapes to infinity. We therefore choose a large number N_{max} and define some target set which comprises all the points at a distance greater than some radius R from the origin. If the orbit enters this target set for $k < N_{max}$ it is deemed to have escaped to infinity and the level of point z is defined as the number of iterations required to escape. On the other hand if the orbit remains at a radius $< R$ after N_{max} iterations the point is deemed to be in the filled-in Julia set.

We take a square grid of points spanning part of the complex plane, usually centred on the origin. For each point we iterate as described above. If the orbit escapes we label the point with the level k . If not we label the point with N_{max} indicating the point is very close to, or within, the set.

If we increase N_{max} or R we expect the result to be a better approximation to the limiting set.

Four lines are missing from the skeleton program. These lines perform the quadratic mapping given above. The variable vr is the real part of z and the variable vi is the imaginary part of z . Similarly the constant c has real and imaginary components represented by the variables cr and ci respectively.

i) Replace the missing lines and run the program.

ii) Run the program for different values of the constant c . Try and find examples of connected Julia sets and Cantor Julia sets.

Julia sets can be defined for transcendental functions and not just the quadratic mapping above.

iii) Modify the program to calculate the filled-in Julia set of the mapping defined by:

$$f_c(z) = \lambda \cos(z)$$

where the constant λ is real in the range $2.9 < \lambda < \pi$. Note that the program must calculate the cosine of a complex number. You will have to work out what this corresponds to in terms of the real and imaginary parts of z . The functions \exp , \cos , \sin , \cosh and \sinh are all available in `c` (from the `math.h` library which is already included in the programs). If $\lambda \sim 2.96$ the resulting pattern exhibits many of the characteristics expected of a fractal.

4.2 Problem 2 - Calculating and plotting the Mandelbrot set

The Mandelbrot set is a sort of catalogue of the Julia sets associated with the quadratic mapping. A point z on the complex plane is a member of the Mandelbrot set if the Julia set for $c = z$ is connected (one continuous island in the complex plane). If the point is not a member of the Mandelbrot set then the corresponding Julia set is a Cantor set (a dust of infinitely many unconnected points).

We can calculate the Mandelbrot set using the Level Set Method described above. The only difference in the algorithm is that the constant c is replaced by the starting value z , strange but true!

i) Create a file `fractal2.c` by copying your version of `fractal1.c`:

```
$ cp fractal1.c fractal2.c
```

Edit the new source file `fractal2.c` so that it calculates the Mandelbrot set. You will capture the complete set if the real axis covers the range $-2.5 < real < +1.0$ and the imaginary axis covers the range $-1.75 < imag < +1.75$.

ii) Using the plot obtained from the `fractal2` program you can find points which correspond to Julia sets that are connected or Cantor sets. Use the program `fractal1` to check that this is correct.

iii) Zoom into the edge of the Mandelbrot set by changing the area of the complex plane plotted. How large can the zoom factor be before the algorithm for rendering the set starts to break down? What do you think is limiting the algorithm?

4.3 Problem 3 - Generating a fern picture using affine transformations

Fractals which resemble naturally occurring patterns can be produced using Iterated Function Systems (IFS). A particularly rich form of IFS uses a combination of affine transformations. An affine transformation in two dimensional space is defined by:

$$w \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} a_{11}x + a_{12}y + b_1 \\ a_{21}x + a_{22}y + b_2 \end{bmatrix}$$

The a_{ij} 's and b_i 's are real constants that form a matrix A and vector B . Thus the affine transformation is specified by six real numbers. We can re-write the components of the affine transformation as:

$$A = \begin{bmatrix} r \cos \theta & -s \sin \phi \\ r \sin \theta & s \cos \phi \end{bmatrix}$$

$$B = \begin{bmatrix} h \\ k \end{bmatrix}$$

where h and k are translations, θ and ϕ are rotations and r and s are scalings. A fixed point of an affine transformation is a point (x, y) which remains the same on applying the transformation.

The IFS consists of taking a group of affine transformations (4 say) and allotting a probability to each one. Starting with a fixed point of one of the transformations you iterate, choosing the next transformation at random in accordance with the probabilities. The point will move about in the XY plane slowly covering a closed set of points under the transformations. The geometry of the pattern is prescribed by the transformations while the density of points in particular areas of the pattern is determined by the probabilities. If you don't start with a fixed point the iteration may wander off in a random or open orbit.

The source file `fractal3.c` will plot a fractal fern-like pattern.

i) Use the coefficients of the affine transformations in the source code to calculate the translations, rotations and scalings that make up the 4 mappings which determine the shape of the fern. You can do this either by adding code to the program (the elegant way) or doing it by hand using a calculator (the clunky way). Note that you can determine the order of the coefficients by looking at the function `affine_trans`.

ii) The function `affine_fixed` should return a fixed point of the transformation defined by the 6 coefficients supplied by the first argument `double a[6]`. Some of the code is missing. You must replace the missing code to make the program work properly. If there is a fixed point return the result in `*xf` and `*yf` and return the function value of 0 (zero). If there is no fixed point for the transformation return function value 1 to indicate that the function has failed.

4.4 Problem 4 - Plotting the 4 affine transformations from the fern IFS

The program fractal4.c plots the effect of the 4 affine transformations from Problem 3 on a test triangle. Try running the program. This will give you a better idea of how the fern geometry is determined by the transformations.

- i) How can you alter the transformations so that the fern bends over to the right more? Try out the new transformations in fractal3.c and fractal4.c to check that your suggested solution is correct.
- ii) How do the probabilities associated with each of the transformations effect the final picture? Try running the fractal3.c program with equal probability for all the transformations. How can you use the plot produced by the fractal4 program to estimate the optimum probabilities to use?

4.5 Problem 5 - Plotting the bifurcation diagram of the logistic map

The logistic map has the quadratic form:

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

It is very similar to the quadratic mapping used for generating the Mandelbrot set except that x is real rather than complex. If the constant μ lies in the range $0 < \mu < 4.0$ successive values of x_n are bounded $0 < x_n < 1.0$. If the mapping is iterated a sufficiently large number of times the sequence of x_n 's lie on a well defined orbit. The structure and complexity of this orbit depends on the value of μ .

The program chaos1.c is designed to plot the orbit values x_n for NPARAM values of μ . Immediately after the value of μ is changed the iteration is allowed to proceed NSPIN times before a sequence of NPOINT orbit values are stored. Eventually a total of NPARAM*NPOINT values of x_n are stored along with their associated μ values. The complete sequence is then plotted as dots to form a so-called bifurcation diagram.

- i) The lines of code which decrement the value of μ and perform the iteration are missing. You must provide these lines to make the program work properly.
- ii) Change the ranges of μ values and x values plotted so that the diagram zooms in on the first bifurcation sequence. Use the plot to provide an estimate of Feigenbaum's number:

$$\delta = \lim_{k \rightarrow \infty} \frac{\mu_k - \mu_{k-1}}{\mu_{k+1} - \mu_k}$$

where μ_k is the value of the constant at the k th bifurcation. Also estimate the value of $\mu_{k \rightarrow \infty}$ for the first bifurcation sequence.

4.6 Problem 6 - Plotting the Lyapunov exponent of the logistic map

For a 1-D map the Lyapunov exponent is given by:

$$\lambda = \lim_{n \rightarrow \infty} \frac{1}{n} \sum_{i=0}^{n-1} \log_e |f'(x_i)|$$

where $f'(x_i)$ is the first derivative of the map function. This exponent is the stretching rate per iteration (divergence or convergence) averaged over the orbit. Regions of periodic behaviour of the logistic map correspond to intervals in μ over which $\lambda < 0$. At bifurcations $\lambda \rightarrow 0$ and if the motion becomes chaotic $\lambda > 0$.

Program `chaos2.c` calculates and plots this exponent for a range of μ values.

i) The lines of code which perform the summation to calculate λ are missing. Add these lines to make the program work.

ii) Modify the program so that you can calculate Feigenbaum's number δ and $\mu_{k \rightarrow \infty}$ from the first bifurcation sequence, as accurately as possible. Exactly how you do this is up to you. You can set the plotting ranges to zoom in on the bifurcation sequence. You may wish to select bifurcation points using the cursor. The plotting function `cpgcurs(float *x, float *y, char *ch)` which is available in the PGPLOT library will return the position (x, y) from the cursor. The character *ch* returned depends on which button or key is pressed when using the cursor. You could also search for values of the Lyapunov exponent which are very close to zero.

4.7 Problem 7 - Plotting time series and phase portraits for a driven LCR circuit

The LCR circuit under consideration comprises an inductance L , a varactor diode instead of a conventional capacitor and a resistance R , in series, driven by a sinusoidal voltage. The diode has a capacitance which depends on the applied voltage. If the drive voltage is given by:

$$V(t) = V_0 \sin(\omega t)$$

with the frequency $\omega/2\pi$ reasonably close to resonance, the equations of motion of the current and voltage for the diode are approximately:

$$\frac{dI}{dt} = \frac{V_0 \sin(\omega t) - RI - V}{L}$$

$$\frac{dV}{dt} = \frac{I - I_d(V)}{C(V)}$$

where the capacitance has two components, C_j the junction differential capacitance and C_s the storage differential capacitance:

$$C_j(V) = C_{j0} \left(1 - \frac{V}{\phi_c}\right)^{-1/2}$$

$$C_s(V) = C_{s0} \exp\left(\frac{V}{\phi}\right)$$

and the junction current is:

$$I_d(V) = I_0 \left(\exp\left(\frac{V}{\phi}\right) - 1 \right)$$

The parameters C_{j0} , ϕ_c , C_{s0} , ϕ and I_0 are measurable parameters for a given junction.

We can numerically integrate these equations, using a fourth order Runge-Kutta algorithm or similar, to give the diode current and voltage as a function of time. The program `chaos3.c` performs such an integration. In the program the variable `cd` is C_{s0} and the variable `ct` is C_{j0} . The other parameters are self explanatory. The function `fmotion()` should represent the equations of motion. The array components of double `y[NVAR]` are the dynamical variables, `y[0]`, the diode current and `y[1]`, the diode voltage. The function should return the drive voltage `vd` at time `t`, and the derivatives of the current and voltage in the array double `yprime`, also at time `t`, according to the equations above.

- i) Look up further details and background to driven nonlinear oscillators in the Linsay (1981), Testa et al. (1982) and Jeffries (1985).
- ii) Lines of code are missing from `fmotion()` in the source file `chaos3.c` although the comment lines have been left in for guidance. You are required to add the missing lines to make the function work correctly. The program should plot out a time series and two phase portraits of the motion. With $V_0 = 1.5$ the motion should be chaotic. Demonstrate that this is the case.
- iii) When the program is working use it to characterise the motion for drive amplitudes $V_0 = 1.0$, $V_0 = 2.05$ and $V_0 = 4.0$.
- iv) Change the function `fmotion()` so that the capacitance is NOT a function of applied voltage. Demonstrate that if this is the case the motion is regular and not chaotic whatever the drive amplitude.

4.8 Problem 8 - Plotting a bifurcation diagram for a non-linear LCR circuit

We can plot a bifurcation diagram for the non-linear LCR circuit described in Problem 7 above. Instead of plotting successive iterations of a 1-D mapping as we did for the logistic map in the `chaos1.c` program we now plot the amplitudes of successive maxima of the oscillation. To do this we must sample the integrated time series once every cycle. We can change the phase of the sampling by changing the start time of the integration sequence. This can be used to compensate for any phase difference between the driving voltage and the diode current.

The program `chaos4.c` is designed to do this. It is very similar to `chaos1.c` except that instead of iterating the logistic equation we integrate the equations of motion of the LCR circuit through 1 cycle.

- i) The `for()` loops that perform the sampling are missing in `chaos4.c`. Using `chaos1.c` as a guide add in the necessary code. You will also have to include your version of the `fmotion()` function from the `chaos3.c` program.
- ii) Using the program `chaos4.c` estimate the convergence ratio:

$$\delta = \lim_{k \rightarrow \infty} \frac{V_{0k} - V_{0k-1}}{V_{0k+1} - V_{0k}}$$

for the first bifurcation sequence. How does your answer compare with Feigenbaum's number?

iii) Find the drive voltage amplitudes that yield period 5 and period 3 oscillations. Are there any other periodic windows you can identify within the chaotic region? How do these windows compare with the bifurcation sequence of the logistic map?

4.9 Problem 9 - A practical demonstration of the varactor LCR circuit

The LCR circuit is supplied in a black box. If you take the top off you will see the pertinent components. An op-amp is used to monitor the diode current. The input must be provided using a signal generator. The frequency should be set in the range 20-60 kHz. The X-output is the drive voltage and the Y-output is the diode current as seen by the op-amp. The jump leads on the top of the box allow you to select different L and R values.

i) Set the circuit up and display the X and Y outputs as separate traces on a scope triggering off the Y output. Try out various combinations of L , R and drive frequency until you find a reasonable looking bifurcation sequence. Don't forget to turn on the op-amp using the switch on the box!

ii) Change the oscilloscope setting to plot the X and Y outputs as a Lissajous figure. Using this display it is very easy to view the bifurcation sequence. Try following the sequence as carefully as possible through to chaos noting down the drive voltage values at each bifurcation. Also record the voltage when the sequence falls to chaos. You can use a digital meter to measure the input voltages reasonably accurately.

iii) Continue increasing the drive voltage and look for period 5 and period 3 windows. Draw a sketch of the bifurcation sequence and compare it with the simulated plot from the chaos4 program.

iv) Make a quantitative comparison of the simulated LCR bifurcation diagram and the observed LCR bifurcation sequence with the results from program chaos1 for the logistic equation.