Computational Physics
Homework 3

Kieran Finn

October 1, 2014

1 Problem 1

Write a program to implement the logistic map $f(x) = ax(1-x)$. Explore the dynamical behaviour for $a = 0.9$, 0.99 and 1. Prove that $x = 0$ is a globally attractive fixed point and calculate the rate of convergence.

1.1 Description of program

This program is called *main.py* in the directory. I wrote a function called *iterate*, which is in the *functions.py* file which performs a given number of iterations on a given function. It has options for verbosity from only outputting the final number to stopping after each iteration and returning the full list.

The program asks for a value of a and x_0 and then iterates in 2 steps. The first step has no output whatsoever and is used to remove transience. the second one produces a plot and an output txt file of the process.

1.2 Experimetntal results

figure 1 shows the output of the program.

1.3 Theoretical results

1.3.1 proof that 0 is a globally attractive fixed point

consider the function $g(x) = f(x) - x$. this has a maximum at

$$g'(x) = a - 1 - 2ax = 0 \qquad (1)$$

$$\Rightarrow x = \frac{a-1}{2a}. \tag{2}$$

However, for $0 \leq a \leq 1$, this is at $x \leq 0$, which is out of range, so that the maximum value of g within the range allowed is at $x = 0$. But $g(0) = 0$, which means

$$g(x) \leq 0 \Rightarrow f(x) \leq x \tag{3}$$

(where the equality sign is only for $x = 0$). We also know that $f(x) \geq 0$. This means that x is constantly decreasing, but is bounded from below, so therefore it must tend towards a fixed point at $x = 0$.

1.3.2 Rate of convergence $a < 1$

assume we start a distance ϵ from the fixed point, the subsequent iterations look like

$$\begin{aligned} x_0 &= \epsilon & (4) \\ x_1 &= a\epsilon(1-\epsilon) = a\epsilon + O(\epsilon^2) & (5) \\ x_2 &= a^2\epsilon + O(\epsilon^2) & (6) \\ &\ldots \\ x_n &= a^n\epsilon + O(\epsilon^2) & (7) \end{aligned}$$

so it converges as $\epsilon_n = a^n \epsilon_0$.

1.3.3 Rate of convergence for a=1

The previous way doesn't work since $a^n = 1 \forall n$ so we must include the second order terms

$$\begin{aligned} x_0 &= \epsilon & (8) \\ x_1 &= \epsilon(1-\epsilon) = \epsilon - \epsilon^2 & (9) \\ x_2 &= \epsilon - 2\epsilon^2 + O(\epsilon^3) & (10) \\ &\ldots \\ x_n &= a^n\epsilon - n\epsilon^2 + O(\epsilon^3) & (11) \end{aligned}$$

so it converges as $\epsilon_n = \epsilon_0 - n\epsilon_0^2$.

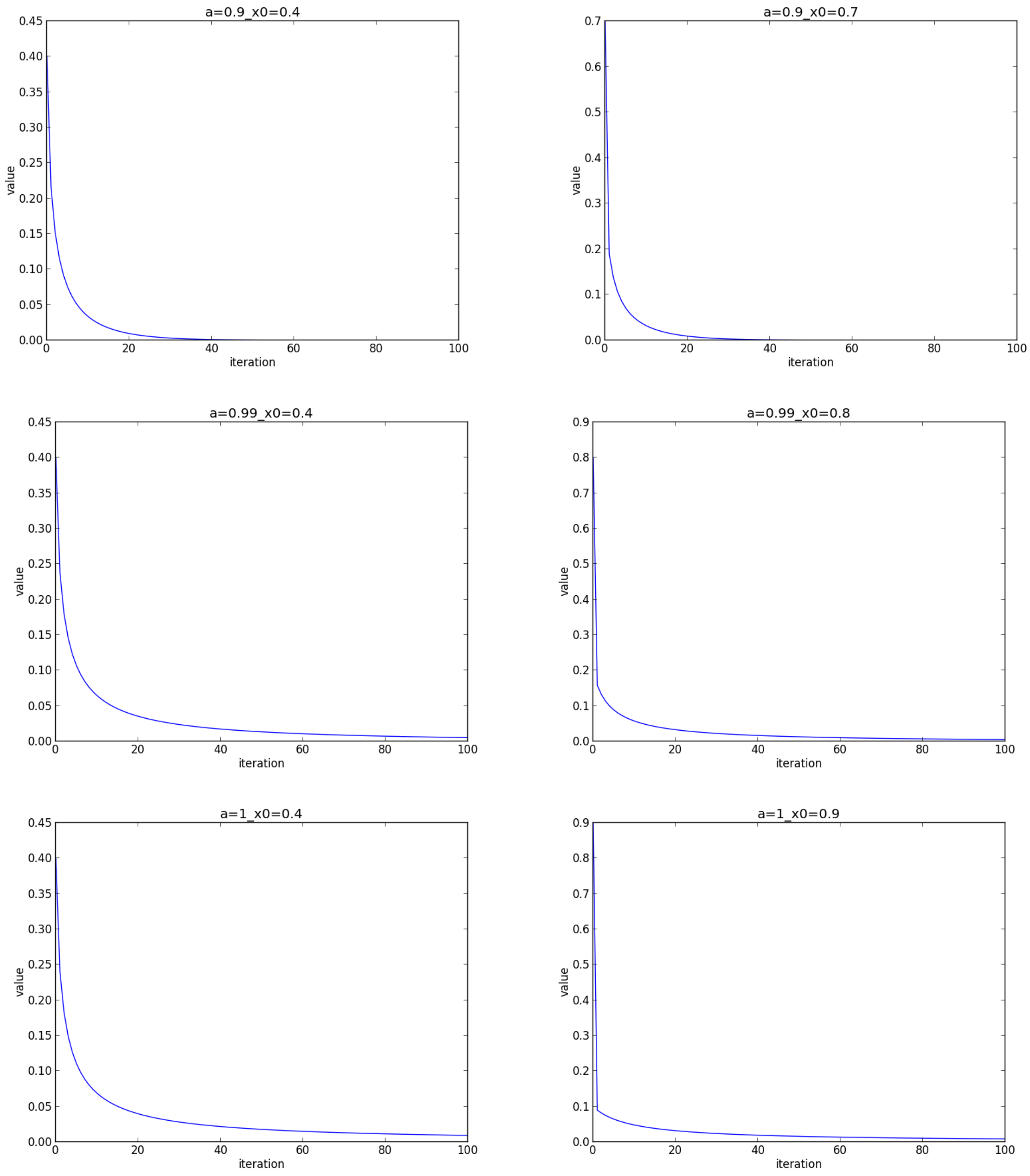

Figure 1: Iterations with $0 \leq a \leq 1$ showing 0 is a globally attracting fixed point

2 Problem 2

Explore the dynamical behaviour for $a = 1.01, 2, 2.9, 2.99, 3$. Calculate the new fixed point and the rate of convergence.

2.1 Description of program

Same program as the last question.

2.2 Experimetntal results

figure 2 shows the output of the program.

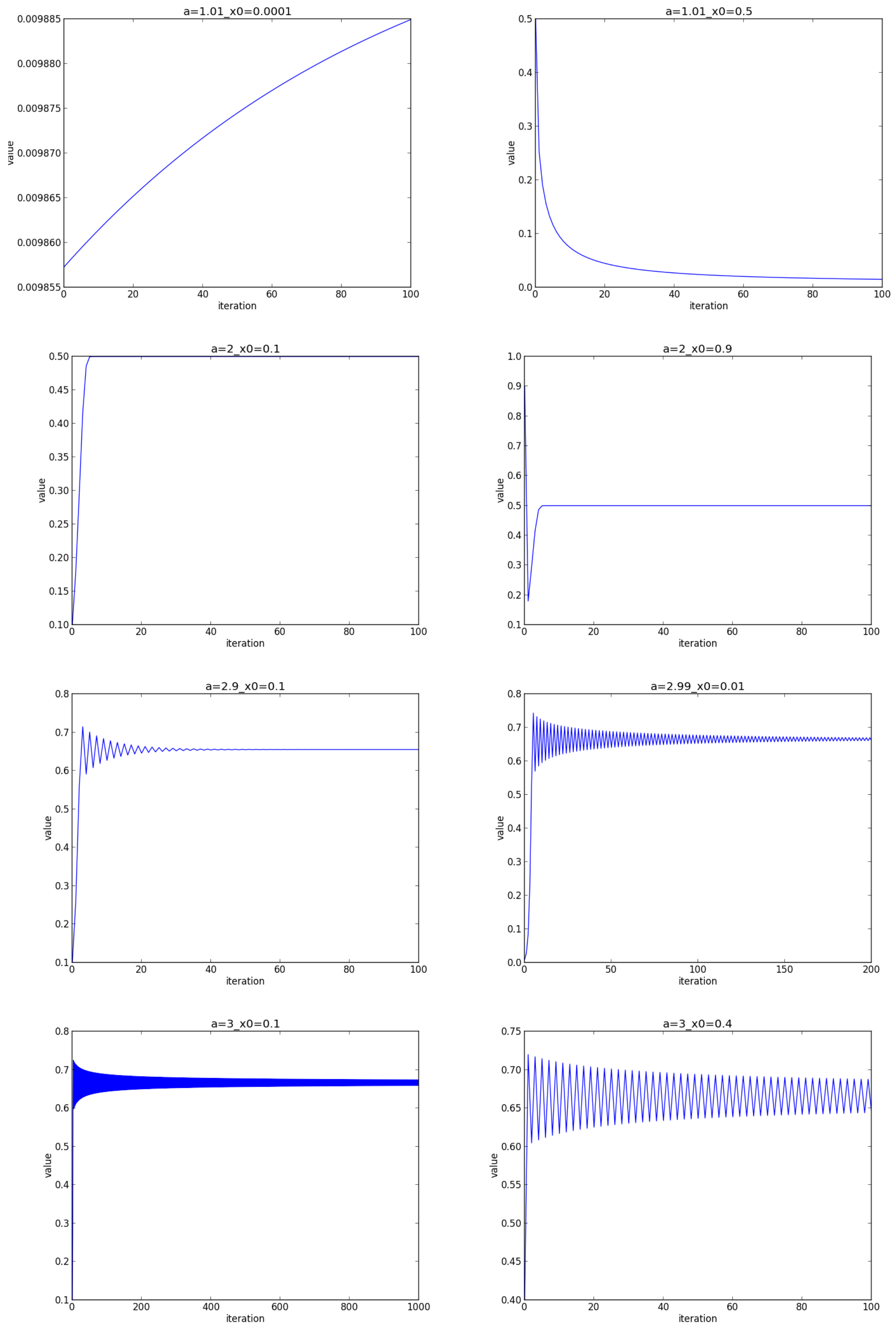

Figure 2: Iterations with $1 < a \leq 3$ showing a globally attracting fixed point

2.3 Theoretical results

2.4 Stability of $x = 0$ fixed point

If we look back at equation 11 we see that now, since $a > 1$, the error *diverges* as

$$\epsilon_n = a^n \epsilon_0 \tag{12}$$

and thus the fixed point $x = 0$ is unstable.

2.4.1 Value of fixed point

To find the fixed point we simply solve:

$$\begin{aligned} f(x*) = ax^*(1 - x^*) &= x^* && (13) \\ \Rightarrow x^* = 1 - \frac{1}{a} & && (14) \end{aligned}$$

2.4.2 Rate of convergence and stability for $1 < a < 3$

Again, assume we start a distance ϵ from the fixed point. The error for the second iteration is

$$\begin{aligned} \epsilon_1 &= f(x^* + \epsilon) - x^* && (15) \\ &= f(x^*) + \epsilon f'(x^*) + O(\epsilon^2) - x^* && (16) \\ &= a\epsilon(1 - 2x^*) + O(\epsilon^2) && (17) \\ &= \epsilon(2 - a) && (18) \end{aligned}$$

where we have used the definition of the fixed point, $f(x^*) = x^*$, in moving from the 2nd to 3rd line. We can easily repeat this process to see what the general error is

$$\begin{aligned} \epsilon_0 &= \epsilon && (19) \\ \epsilon_1 &= \epsilon(2 - a) + O(\epsilon^2) && (20) \\ x_2 &= (2 - a)^2 \epsilon + O(\epsilon^2) && (21) \\ \ldots & && \\ x_n &= (2 - a)^n \epsilon + O(\epsilon^2) && (22) \end{aligned}$$

so it converges as $\epsilon_n = (2 - a)^n \epsilon_0$.

2.4.3 Rate of convergence for a=3

Again, this approach doesn't work for the limiting case $a = 3$ so we must go to higher order. We also look at the twice-iterated map $f(f(x))$

$$\begin{aligned}
\epsilon_1 &= \epsilon(2-a) - a\epsilon^2 && (23)\\
&= -\epsilon - 3\epsilon^2 && (24)\\
\epsilon_2 &= -(-\epsilon - 3\epsilon^2) - 3(-\epsilon - 3\epsilon^2)^2 && (25)\\
&= \epsilon + 3\epsilon^2 - 3(\epsilon^2 + 6\epsilon^3) + O(\epsilon^4) && (26)\\
&= \epsilon - 18\epsilon^3 + O(\epsilon^4) && (27)
\end{aligned}$$

Since the leading term is just ϵ, we can easily repeat this to find the general rule

$$\begin{aligned}
x_0 &= \epsilon && (28)\\
x_2 &= \epsilon - 18\epsilon^3 + O(\epsilon^4) && (29)\\
x_4 &= \epsilon - 36\epsilon^3 + O(\epsilon^4) && (30)\\
&\ldots\\
x_2n &= a^n\epsilon - 18n\epsilon^3 + O(\epsilon^4) && (31)
\end{aligned}$$

so it converges as $\epsilon_2 n = \epsilon_0 - 18n\epsilon_0^3$.

3 Problem 3

Explore the dynamical behaviour for $a = 3.01, 3.1, 3.4, 3.44, 3.449$. Show the bifurcation into orbit of period 2.

3.1 Description of program

Same program as the last question.

3.2 Experimetntal results

figure 3 shows the output of the program.

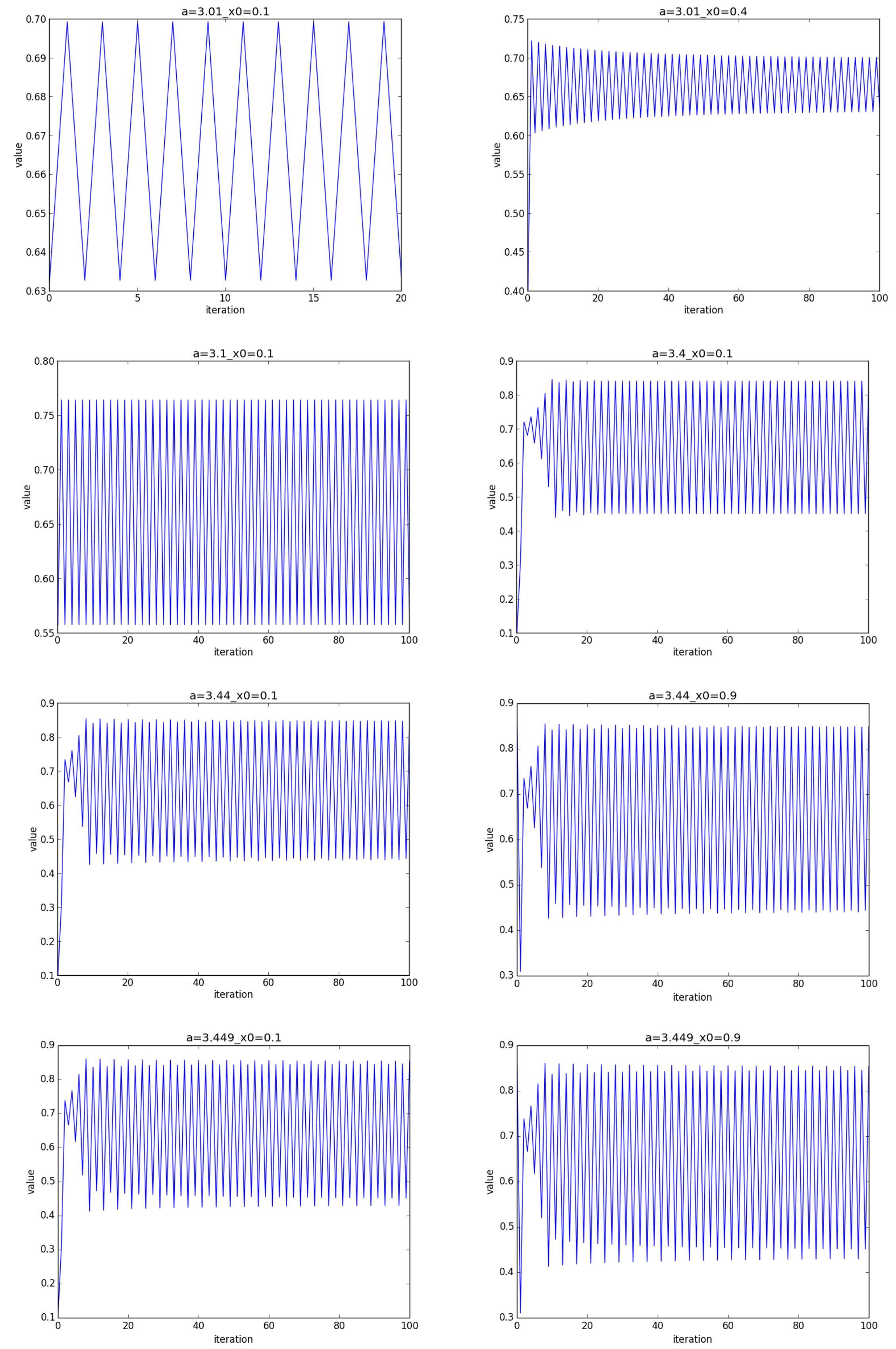

Figure 3: Iterations with $1 < a \leq 3$ showing a globally attracting fixed point

3.2.1 what happens as we approach 3.449489742783178

We get a second bifurcation and the orbit becomes of period 4.

3.3 Theoretical results

3.4 Derivation of period 2 orbit

We want to solve

$$f(f(x)) = x \tag{32}$$

This leads us to the 4th order equation (see the mathematica file)

$$(a^2-1)x - (a^2+a^3)x^2 + 2a^3x^3 - a^3x^4 = 0. \tag{33}$$

In general, this would be difficult to solve. However, we already know two of the roots of this equation; the fixed point solutions. We can therefore divide through by $(x-0)(x-x^*)$ to get a quatratic equation for x. The equation is

$$a^2x^2 - (a+a^2)x + (1+a) = 0 \tag{34}$$

which has solutions

$$x_\pm = \frac{a+1 \pm \sqrt{a^2-2a-3}}{2a}. \tag{35}$$

The first thing to notice is that these solutions only exist for $a \geq 3$, which explains why the bifurcation occurs here.

3.4.1 stability of the period-2 orbits

If we again consider a point a distance ϵ from the orbit then we have the following

$$\begin{aligned} x_0 &= x_+ + \epsilon && (36)\\ x_1 &= f(x_+) + \epsilon f'(x_+) + O(\epsilon^2) = x_- + \epsilon f'(x_+) + O(\epsilon^2) && (37)\\ x_2 &= f(x_-) + \epsilon f'(x_+)f'(x_-) + O(\epsilon^2) = x_+ + \epsilon f'(x_+)f'(x_-) + O(\epsilon^2) && (38)\\ & && (39)\end{aligned}$$

for stability, we need $|x_2 - x_+| < |x_0 - x_+|$. This leads to the condition

$$|f'(x_+)f'(x_-)| < 1 \tag{40}$$

but $f'(x) = a(1-2x)$ so that

$$f'(x_\pm) = -1 \mp \sqrt{a^2-2a-3} \tag{41}$$

plugging this into equation 40 gives us the condition

$$\left|1-(a^2-2a-3)\right|<1 \tag{42}$$

the two critical as are

$$\begin{aligned} 1-(a^2-2a-3) &= 1 \quad (43)\\ \Rightarrow a &= 3 \quad (44)\end{aligned}$$

which is where the orbit first appears, and

$$\begin{aligned} & 1-(a^2-2a-3)-1 \quad (45)\\ \Rightarrow \quad & a=1+\sqrt{6}=3.4494897... \quad (46)\end{aligned}$$

which is where the orbit becomes unstable and we get a second bifurcation.

4 Problem 4

Increase a gradually and look for successive bifurcations

4.1 Description of program

This uses the program *increase.py*

Choose the a0, a1 and da from user input. Also get the starting x, the number of iterations and the number of suppressed iterations which will be the same for all graphs. Create a new directory to store all the plots in and plot them from a=a0 to a1 in steps of da. Finally string all the plots together into a video stored in the main directory and destroy the subdirectory.

4.2 Experimetntal results

see videos. These videos are useful for finding the approximate points of the bifurcations

5 Problem 5

Write a program to find the bifurcation points more precisely

5.1 Description of program

The programs for this part are *find_bifurcations.py* which finds the bifurcation for a given order and *find_all_bifurcations.py*, which repeats this for all powers of 2 within a given range.

This program is more complicated and made up of smaller parts

5.1.1 get_fp_and_f

This calculates the values of $f_m(x)$ and its derivative for a given order, m, and value, x. It does this by using the function iterate to generate a list of $x, f(x), f(f(x))$... and then calculates the derivative using the chain rule

$$f'_m(x) = \prod_{k=0}^{m-1} f'(f_k(x)). \tag{47}$$

5.1.2 Newton

Simply applies Newton's algorithm to find the root of $f_m(x) - x = 0$ using values of $f_m(x)$ and $f'_m(x)$ calculated using the algorithm above. Has an automatic stopping procedure that defines success when both $|x_n - x_{n+1}| < x_{tol}$ and $|f(x) - x| < y_{tol}x$. There is also a maximum number of iterations (set at 2000) before it declares failure.

5.1.3 findroot

A more clever algorithm for finding the root of $f_m(x) - x = 0$. From the initial value of x given, the program will iterate the map a given number of times (set at 100) to get close to the root. It then uses the function Newton (described above) to calculate the position of the root more precisely. If Newton fails, the map is iterated again and this is repeated until the maximum (set at 10) is reached.

What the subroutine actually returns is the value of $f'_m(x^*) + 1$, as this is the function we need the root of to find the bifurcation.

5.1.4 secant_with_brackets

This subroutine is used to find the root of $f'_m(x^*, a) + 1 = 0$, which is the location of the bifurcation. The LHS is calculated using the findroot algorithm described above. The function $f(x)$ referred to below is $f'(x^*) + 1$ with $a = x$.

Since findroot sometimes fails, especially for large a and large m, care is needed to find the initial bracket used for the bisection/secant method described below. The left hand side of the initial bracket is assumed to be good and the first stage is to find a suitable right hand side (x_r) for the bracket. There are two reasons for rejecting a given x_r. Firstly if the findroot algorithm fails, in this case we need to move x_r in a bit so that we can actually calculate y values to use in the secant method. We update our smallest value that fails $x_{r0} = x_r$ and then choose a new x_r at the midpoint of the current bracket. The other reason to reject x_r is if it is on the lhs of the root, i.e. if $f(x_r) > 0$. In this case we choose a new x_r half way between this and x_{r0}. This repeats until we find an x_r that falls on the rhs of the root and doesn't cause findroot to fail. Interestingly, since we know $f_m(x^*) < 1$ when the orbit is stable, if we find $f(x_r) > 2$ we are on the rhs of the bifurcation and are simply in a regime where findroot is not as trustworthy. Therefore, we can treat $f(x_r) > 2$ in the same way as we treat $f(x_r) < 0$ in adapting our bracket.

Although it may seem simpler to treat findroot failing as indication that we are on the rhs of the root and use bisection accordingly, this method is actually quicker (I have tested this, see *find_bifurcations_simpler.py*). This is because it takes a long time to evaluate findroot in the regions where it fails and the above method ensures that, once a bracket is found, this never happens and subsequent iterations are very quick.

Once a suitable bracketing interval has been found the algorithm proceeds as follows. Calculate x_n from x_{n-1} and x_{n-2} according to the secant method. If this lies outside the current bracket then switch to bisection and instead use the midpoint of the bracket as x_n. Calculate $f(x_n)$ and update the bracket accordingly remembering that $f(x) > 2$ is equivalent to $f(x) < 0$. Once the bracket is narrower than a given tolerance we have found the bifurcation.

5.2 Experimental Results

Figure 4 shows the bifurcation points found using this method.

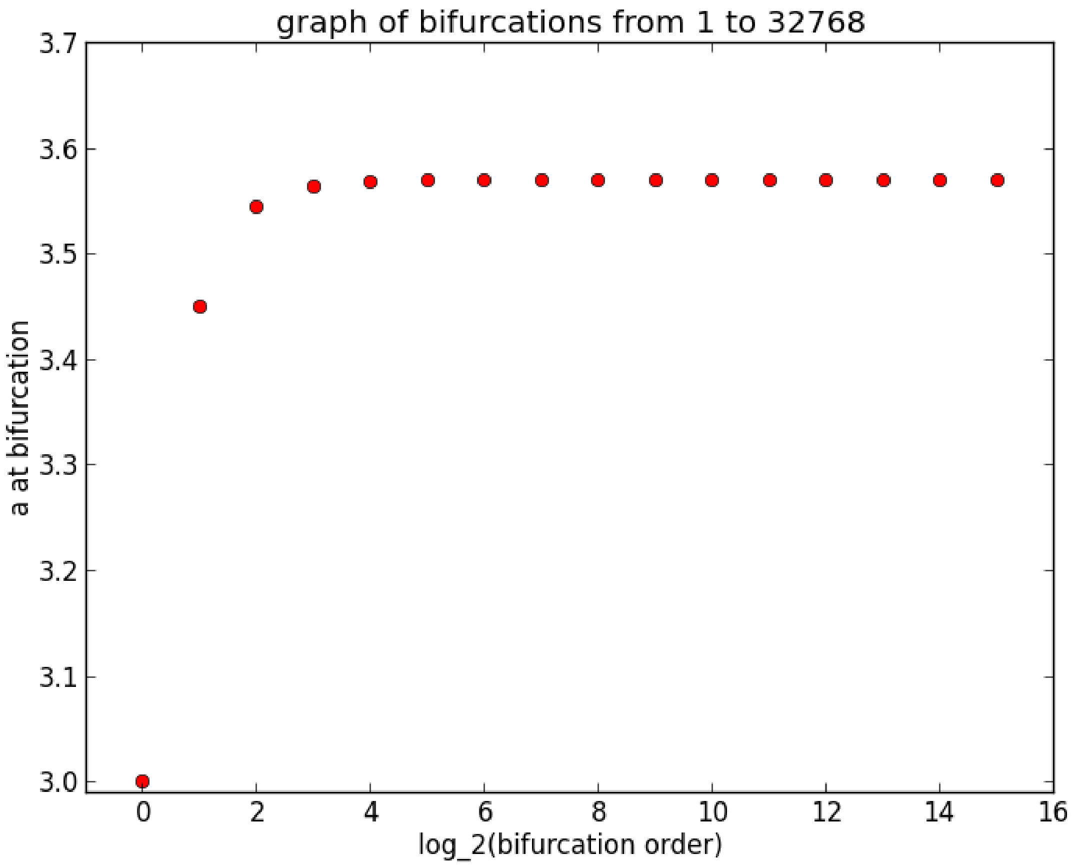

Figure 4: Locations of successive bifurcations

6 Problem 6

Analyse the convergence of the bifurcation points. and calculate the Feigenbaum constant.

6.1 Description of the program

This question used *bifurcation_points.py.* We extract the positions of the bifurcations from the output of the previous program. We assume that a_∞, the value we are converging to is simply the final bifurcation calculated and use that to calculate the errors. These are then plotted on a log scale and a straight line is fitted. Only the first 6 points (up to orbits of period 32) are used to calculate the line since after that we run into problems with roundoff error as well as the fact that we are not using the exact value of a_∞.

6.2 Experimental Results

Figure 5 shows a plot of the errors as well as the straight line fitted to the first 6 entries.

6.3 Theoretical Results

The points are converging to

$$a_\infty \approx 5.699456. \tag{48}$$

We get a straight line in log-log space which means

$$\begin{aligned} \ln(a_n - a_\infty) \equiv \ln(\epsilon_n) &= mn + c & (49) \\ \Rightarrow \epsilon_n &= \epsilon_0 \alpha^{-n} & (50) \end{aligned}$$

By fitting the straight line we find

$$\begin{aligned} \epsilon_0 &= 0.567 & (51) \\ \alpha &= 4.68 & (52) \end{aligned}$$

so we see α is the Feigenbaum constant, correct to almost 3 significant figures.

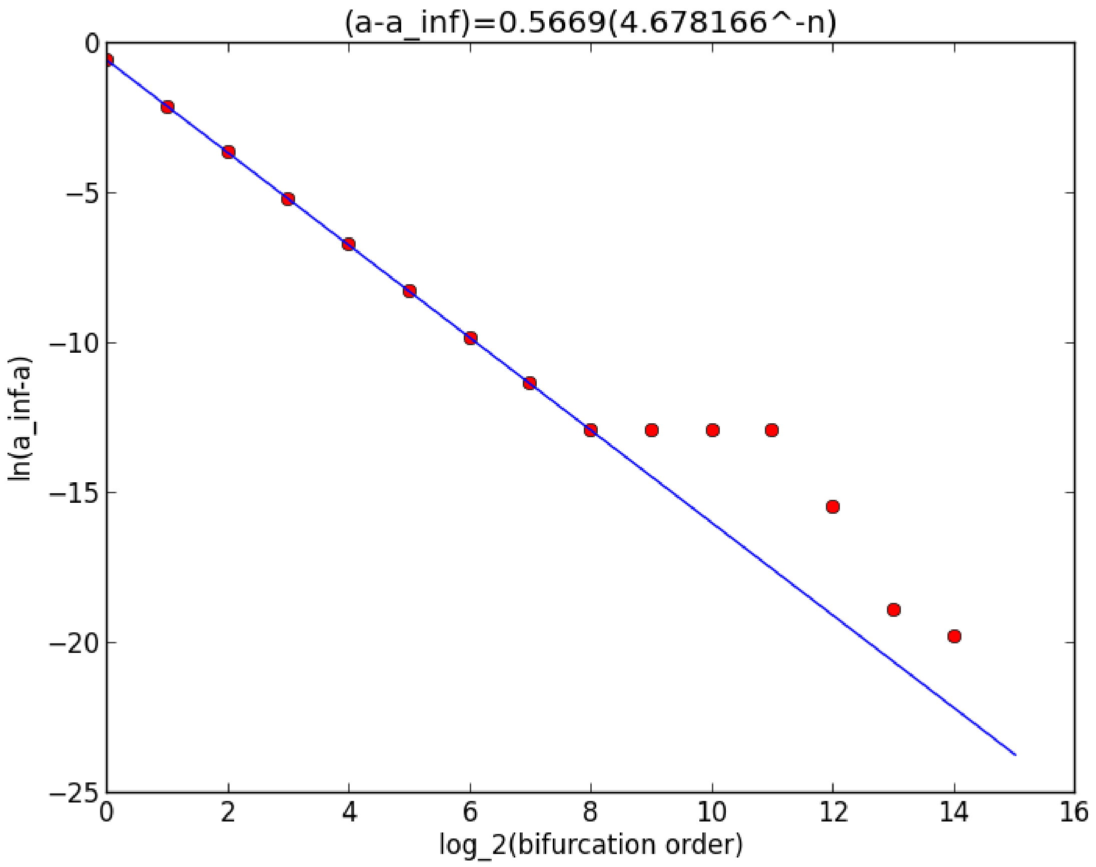

Figure 5: Graphical derivation of Feigenbaum's constant

7 Problem 7

Repeat the analyses for the map $f(x) = a\sin(\pi x)$

7.1 Description of the program

The same programs as above but with the function changed. Stored in the directory *different_function*.

7.2 Experimental Results

Figure 6 shows the two previous figures but with $f(x) = a\sin(\pi x)$ instead

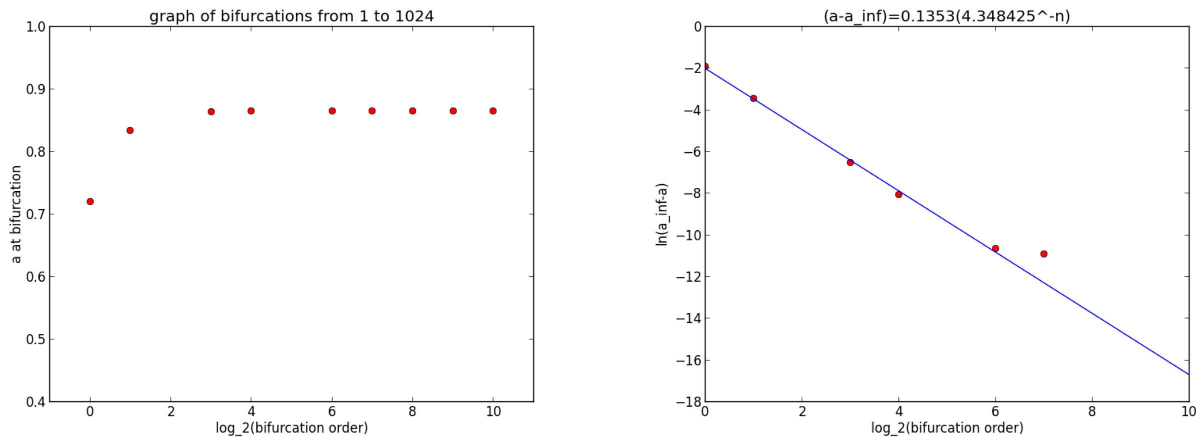

Figure 6: bifurcation points and convergences for the function $f(x) = a\sin(\pi x)$

7.3 Theoretical Results

Although this calculation is even less accurate than the previous one. the constant is still within 10% of Feigenbaum's number.

8 Bonus problem

Produce the bifurcation graph.

8.1 Description of the program

In order to produce a graph that scales more easily I drew the bifurcation diagram in a different way to how it is usually done. I used the program *bif_graph.py*

The first step is to collect the location of the bifurcation points from the output of the previous programs. I could then split the a-axis into regions in which I knew what the periodicity, m, of the stable orbit. I made a grid of 100 points in each of these regions and used findroot (described above) to calculate the fixed points for $f_m(x) = x$ at each value of a in the grid and then used iterate to find all the other points on the stable orbit. I then sorted these and plotted the m curves of the bifurcation diagram.

8.2 Experimental Results

Figure 7 shows the bifurcation diagram.

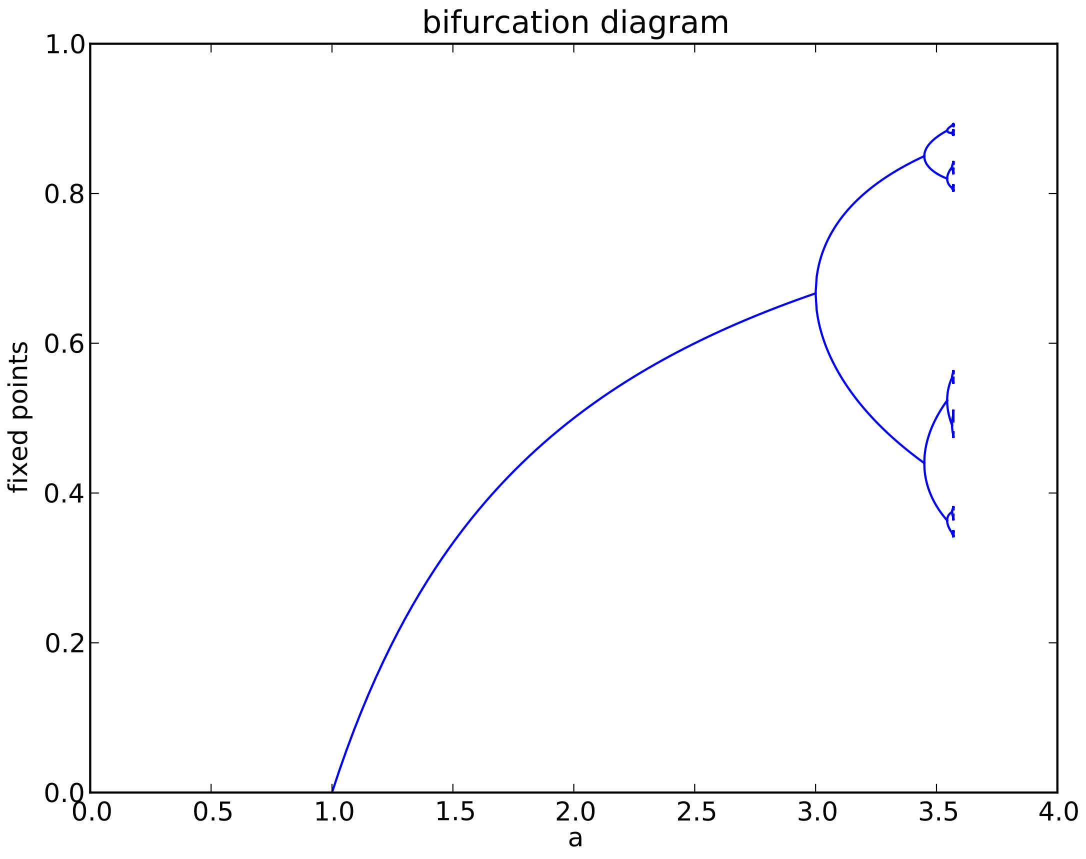

Figure 7: Bifurcation diagram