

Thursday, April 26th, 2001

Due 12 noon, Friday, May 11th, 532a Clark Hall.

Physics 209: Final Assignment (#12) (Take-home Exam on Chaos)

This assignment must be handed in before the deadline of noon, Friday, May 11th, in my (David Mermin's) office, 532a Clark Hall. If you are in the habit of not meeting deadlines, please set yourself an earlier deadline. If there are compelling medical or religious reasons why you cannot make the May 11th deadline please let me know immediately so we can work out a plan to deal with the problem. Be sure to budget enough time for this project. It is the equivalent of two full problem assignments. It ought to take many hours for the numerical explorations and several hours more for writing up your discoveries.

* * *

Suppose that in the summer of 1975 an astronomer had recorded what might or might not be a message from a source that might or might not be natural, a series of binary blips which when translated into decimal turned out to be the number 4.669201609...repeated over and over again. The scientific world would have expressed some disappointment that the signal wasn't 3.141592653... because it would have stretched the imagination to argue that π was just a coincidence... [But there would have been] nothing significant about 4.669201609. The astronomers must have found a natural source, a periodic vibration of some distant neutron star, radiation from a black hole. However, had the same signal been received in 1976... .

Ian Stewart, *Does God Play Dice?*, p. 195.

* * *

This serves as the take-home final examination on the chaos part of the course. You are to do it on your own, without consulting anybody else. If you run into problems, however, feel free to consult me, Trevor Olson, or Richard Helms by phone, e-mail, office hours, or appointment. The assignment consists of some rather open-ended numerical investigations. What you should hand in is a short *essay* (about eight pages: three to five for each of the two Parts of the Assignment) reporting on the results of those investigations.

The report should contain clearly labeled tables of the various numbers you have found, explaining lucidly and persuasively the significance of those numbers.

Do not attach masses of unexplained raw data. Include all data that is relevant to your discussion, and no data that is not. Make the relevance of the data that you do list clear.

Your explanation should be a concise and coherent piece of English prose. Though you can assume you are addressing a knowledgeable reader¹ you should make clear precisely what it is you are describing, and what convinced you that your description is correct.

The first part of the assignment is to use the sine map,²

$$x \rightarrow \frac{a}{4} \sin(\pi x). \quad (1)$$

to calculate the Feigenbaum number, $\delta = 4.669\dots$, discovered about 25 years ago by Mitchell Feigenbaum. Feigenbaum's number characterizes the manner in which chaos is approached through a series of period doublings. The range of the parameter a for which things settle down to a stable cycle of period 2, 4, 8, 16, 32, 64, etc., decreases with each successive period doubling by a factor that gets closer and closer to 4.669... as the length of the period gets bigger and bigger. The remarkable thing about Feigenbaum's number is that it is *exactly* the same for an enormous variety of different maps, of which the logistic and the sine map are just two examples. You might have expected that even if the same type of behavior held for different maps, the precise value of any number that characterized the approach to chaos through period doubling would differ from one map to the next. The fact that exactly the same number always shows up is an example of what it is now fashionable among physicists to call "universality." The other remarkable thing is that Feigenbaum discovered his number in the age of supercomputers, while playing with a programmable pocket calculator. So shall you.

The most efficient way to get at δ is not to look for the values of a at which 2^m cycles give birth to 2^{m+1} cycle, but to look for values of a between the values that give such period doublings, where the cycles are superstable. All you have to know about a superstable cycle is that one of the points on the cycle is the point $x_0 = 0.5$ at which $f_a(x)$ has its biggest possible value.³ We seek values of a at which x_0 is on an n -cycle—i.e. for

¹ Thus you don't have to explain terms like "map", "iterate", "fixed point", "cycle".

² Be sure the calculator has been set to run in radian mode (do 61 24 to make the little RAD appear in the display if it is not there).

³ I use $f_a(x)$ to stand for either $\frac{a}{4} \sin(\pi x)$ or $ax(1-x)$, depending on whether we are talking about the sine map or the logistic map. Much of the discussion that follows applies to either case.

which $f_a^{(n)}(x_0) = x_0$.⁴

So our problem is to find a value of a that solves the equation $g(a) = 0$, where

$$g(a) = f_a^{(n)}(0.5) - 0.5. \quad (2)$$

Newton figured out an iterative way to do this three hundred years ago, though doing it effortlessly had to await the advent of programmable computers and calculators.

The first part of the program described below⁵ uses Newton's method to find a value of z at which a specified function $g(z)$ is zero. This is then applied to the case where z is just the control parameter a , and the function g is as in Equation (2) above. To run the Newton's method program you place your guess for a in the display, after storing a (rough) estimate of the amount d_0 by which that guess for a is likely to differ from the true value you are seeking. In applying the full program to finding a value of a that gives a superstable n -cycle you must store n in location 8, and you must store 0.5 in location 4. So you start with:⁶

1. d_0 in location 1. This should be your guess at how far the correct value of a might be from your guess for a (see 4 below).
2. n in 8. This is just the order of the cycle you are investigating. If you are interested in 16-cycles, for example, you should put 16 in location 8.
3. 0.5 in location 4. This guarantees that the value of a we come up with gives a *superstable* n -cycle. You can then forget about location 4. (If you put a different number in location 4 the program will try to find an n -cycle that has that number on it.)
4. Your guess for a in the display.

If you then do **XEQ B**, The program will run (for several minutes, if n is large,) before stopping with a value of a in the display (and in location 9) that gives an n -cycle. *Beware of one crucial point:* If you have found, for example, the value a_4 that gives a superstable

⁴ By $f_a^{(n)}(x)$ we mean $f_a(x)$ iterated n times. For example:

$$f_a^{(2)}(x) = f_a(f_a(x)), \quad f_a^{(3)}(x) = f_a(f_a(f_a(x))), \text{ etc.}$$

⁵ **XEQ B** is how you start the whole business. The listing begins on page 10 below.

⁶ The contents of locations 4 and 8 are left unchanged by the program, so if you want to run it again for the same n you need only put the new guess for d_0 in 1 and the new guess for a in the display, before doing **XEQ B**. (You never need to change the contents of location 4 from 0.5.) You will have to run the program again with the same n if you have converged to an a giving the old value for a 2^{m-1} cycle when you were looking for the new value giving a 2^m cycle, as remarked upon shortly.

4-cycle and are now seeking the value a_8 of a that gives a superstable 8-cycle, the program may just give you back the value a_4 that you already know (since a 4-cycle also repeats after 8 iterations). In that case you have to move your initial guess a little farther above a_4 and, perhaps, also decrease the value of d_0 so that the program does not fall into the trap of searching for a_8 too near the irrelevant value of a_4 .⁷

The second part of the program⁸ (page 11) is a subroutine labeled **F** which starts with z in the display and ends with $g(z)$ in the display.⁹ Its last instruction is **RTN** (61 26). This subroutine is listed after the main program. The main program¹⁰ **B** calls up **F** in the case in which the variable z is the parameter a appearing in the logistic or the sine map, and the function g is $f_a^{(n)}(0.5) - 0.5$. Subroutine **F** evaluates $f_a^{(n)}(0.5) - 0.5$. (It does this by running its own subroutine **2** over and over, n different times.) **F** should be put into program memory immediately after the first part of the program. Subroutine **F** assumes that a is in the display, but the main program puts it there before calling up the subroutine, so you don't have to worry about this. **F** stores a in 9, gives x the initial value 0.5 and then calls up the subsidiary subroutine **C** (logistic map) or **E** (sine map) to iterate $f_a(x)$ n times. Finally, Subroutine **F** subtracts 0.5, leaving $f_a^{(n)}(0.5) - 0.5$ in the display, after which it returns control to the main program, which continues with Newton's method.

Important: To change from the logistic map to the sine map you need only find the instruction **XEQ C** in Subroutine **F** and change it to **XEQ E**. (To make the change, with **XEQ C** in the display press the back-arrow button (35), which deletes it, and then enter

⁷ This difficulty arises because the program does not check to make sure that no value repeats itself in the course of the cycle. All it is concerned with is that the 8-th value is the same as the first. Therefore as far as the program is concerned 4-cycles, 2-cycles, and even fixed points qualify as 8-cycles. As a result, you have to apply some artistry in finding n -cycles whenever n is a product of two or more integers. Usually it is obvious when the program has given you the wrong answer, since the result will be recognizable as a value of a you have already computed for a cycle of lower order. But later on you should beware, for example, of getting a 3-cycle when what you were looking for was a 9-cycle. It's easy enough to check: just start with 0.5, iterate 9 times, and make sure that 0.5 doesn't show up on the 3rd and 6th iterations as well as on the 9th.

⁸ This paragraph is only to give you a sense of how the whole program operates. You don't have to follow all these details to use it. But I hope it will be of some help if you want to program your own computer or a different calculator to do these things.

⁹ Nothing is actually displayed while the program is running, but you can see what the program currently has in the display at any stage if you stop it in the middle of its work with **R/S**.

¹⁰ More precisely a subroutine of the main program; **B** does some preliminary book-keeping, and then hands the iterative part of the Newton's-method procedure over to Subroutine **1** which repeats over and over until it has an answer.

XEQ E as its replacement. Then leave programming mode at once (by doing [*b*] **PRGM** (51 26)) before you have a chance to do any accidental damage.) If you load the whole program starting at location 0 in program memory, then **XEQ C** (or **XEQ E**) will be at location 43. To get quickly to location 43 do **GTO . 43**. This works whether or not you are in programming mode. If you are in programming mode make sure you include the “.” or you will introduce the disastrous instruction **GTO 43** into your program! A prudent person will do **GTO . 43** only when *not* in program mode.

The very last part of the program (subroutine **3**) is used by the main program **B** to check that your guesses for a and d_0 have not been so unlucky as to give you a subsequent estimate for a which is outside of the acceptable range between 0 and 4. This rare misfortune actually befell a Physics 209 student a few years ago, who came up with a value for a_{13} which, upon checking, she discovered did *not* give a 13-cycle starting at 0.5. She received much glory for noticing and reporting this. In the current version of the program, in the highly unlikely event that this misfortune afflicts you, the program will not lead you down the garden path to a wrong answer, but will simply stop with 0 in the display. You should then resume what you were doing before with a slightly different pair of guesses for a and d_0 . It will then be necessary to start the main program with **XEQ B** (rather than with **R/S**). As you might imagine, her discovery caused us all to think long and hard. I hope the program is now bug-free, but complicated programs rarely are, so please do note any other anomalous behavior that you encounter.

It is worth noting that the main program and the various subroutines make use of all memory locations except 3 and 7. If you want to stop while the main program is running (with **R/S**) to do some subsidiary calculation before starting up again, do not use any memory locations except 3 and 7, or the program will not function properly when you resume (again with **R/S**).¹¹

To test that you have loaded correctly everything on pages 10-13, you should check that things work as expected when $n = 1$ — i.e. when the map is simply applied once, not iterated. To do this store 1 in location 8, and 0.5 in location 4. We know that the value of a we seek is just 2 for either map, since either map has 0.5 as a (superstable) fixed point when $a = 2$. Enter 0.1 for d_0 , storing it in location 1, and some number in the neighborhood of 2 in the display. Then do **XEQ B**, and after a short while you should get 2 in the display, indicating that the map indeed has a superstable fixed point at $a = 2$. To be quite sure you should probably also check things for $n=2$: store 2 in location 8, store 0.1 in location 1, enter 3.2 in the display, do **XEQ B**, and check that you indeed get a

¹¹ If indeed you want the program to function properly when you resume, it is also essential to store whatever appears in the display when you stopped the program in location 3 (or 7) and then, after you have finished your calculation, to do **RCL 3** (or **RCL 7**) immediately before starting up the program again with **R/S**.

number that results in a 2-cycle containing $a = 0.5$, by running the appropriate map (sine or logistic) twice¹² with that value of a in location 9 and 0.5 in the display.

In Part I you investigate the sine map, so you must change **XEQ C** to **XEQ E** in the second part of the program.¹³

Note: *If you do the computations that follow using a calculator or computer other than the HP20S, identify in your paper the calculator or computer that you did use. If you used a computer (a) give a listing of your program with brief explanatory comments and (b) comment on how far beyond a_{128} you can get. If you used a different calculator comment on how you had to modify the programs I list below for the HP20S to make them work on your calculator.*

¹² As noted in a footnote on page 7, to iterate the sine (logistic) map you now have to do **XEQ E** (**XEQ C**) each time. **R/S** will get you into trouble.

¹³ When in program mode you delete the instruction in the display by pressing the back-arrow button (35). You can then enter the instruction you want to replace it, in the usual way.

WARNING: Be sure to read the instructions on the preceding pages before you try to answer the questions that follow!

PART I.

A. Feigenbaum's δ .

Hunt down the values of a_2, a_4, a_8, \dots at which the sine map¹⁴ has superstable 2, 4, 8, 16, 32, 64, and 128-cycles. It is necessary to be wise in guessing values for a , since you will discover that the program is annoyingly willing to give you the old a_{32} back again when what you want is a_{64} . The program is being perfectly honest, since a 32-cycle is also a 64 cycle, which just happens to repeat itself in the middle.

Here is a good way to proceed: List in a column on the left of a piece of scratch paper your successive results starting with $a_1 = 2$, which we already know. Start a second column to the right of the first, which lists the differences between a_2 and a_1 , a_4 and a_2 , a_8 and a_4 . Make a third column which shows the ratio of successive differences:

$$\frac{a_2 - a_1}{a_4 - a_2}, \quad \frac{a_4 - a_2}{a_8 - a_4}, \quad (2)$$

and so on.¹⁵ You will discover that these differences are all numbers a little bigger than 4.5, getting closer and closer to 4.669 as you go further down the list. Once you have noticed this, you have a very good hint for what to take for your guess for the next value of a . For example, once you have found a_2 and a_4 a good assumption about a_8 is that the ratio

$$\frac{a_8 - a_4}{a_4 - a_2}$$

is very close to

$$\frac{a_4 - a_2}{a_2 - a_1}.$$

Consequently a_8 will be bigger than a_4 by roughly $1/4.6$ of the amount that a_4 was bigger than a_2 . This is what you should take as your guess for a_8 .¹⁶ You should also make sure

¹⁴ I stress, the *sine* map, *not* the logistic map, for which the corresponding numbers are listed in the Appendix below.

¹⁵ You can, of course, use the calculator to do all this arithmetic for you. The analogous table for the logistic map is shown in the Appendix on page 14. (I computed everything in this table using only the HP20S in the same way I'm asking you to use it.)

¹⁶ *Warning:* Other 2^m -cycles lurk in the chaotic region at still higher values of a . If the calculator hands you a value of a that is suspiciously high — for example a value of a_{16} that lies above a_8 by significantly more than 25% of the amount by which a_8 lies above a_4 — then the calculator has given you the wrong 2^m cycle and you should readjust your initial guesses.

that your guess for d_0 is significantly smaller than the distance between a_4 and your guess for a_8 . If you don't, there is a significant chance that your program may just give you a_4 over again.¹⁷ Thus, for example, if the last value you found for a is 0.001 less than your guess for the next value, then you should not take d_0 to be much bigger than 0.0001.

I found in this way that I could go all the way up to¹⁸ a_{128} . The program ran for about 5 minutes before stopping with its final value in that last case. If you become alarmed at such dithering you can press **R/S** to stop the program. Immediately store the contents of the display in 3 and alter no storage registers, though you can look to see what's in them.¹⁹ When you are ready to resume the calculation do **RCL 3** to put back the contents of the display and press **R/S** to resume execution of the program at the point you stopped it.

You should make a table for the sine map looking like the table I give below in the Appendix for the logistic map. Two important points:

(1) *If you were not able to get at least to a_{128} you should write a couple of paragraphs describing what difficulties prevented you from going beyond the point you were able to reach.*

(2) *Do not quote numbers to greater precision than your calculation justifies, unless you wish to make a specific point with the extra incorrect figures.* Note, though, that the calculator gives you values of a_n to its full 12-place accuracy (including the hidden digits) and that it is necessary to use that full accuracy for large values of n , when the changes in a_n are only in the last several digits.

If you are in doubt, you can easily check the validity of any of your results and find the n different values x goes through on its n -cycle by iterating the sine map the appropriate number of times with the value of a the computer gave you. Simply put the value of a into 9 (where it is anyway at the end of the program that gives it to you), put $x_0 = 0.5$ into the display, do **XEQ E** n times,²⁰ and check to see that you do indeed end up back at

¹⁷ This can become irritating when you're searching for a_{64} or a_{128} (or, if you're ambitious, a_{256}) when the program can run for many minutes before giving you the answer you're not interested in.

¹⁸ In fact I could go further, but the values of a were getting so close together that going further did not improve my estimate of the Feigenbaum number δ .

¹⁹ If location 1 contains a very tiny number (like $4.3678...E -27 = 4.3678... \times 10^{-27}$) then chances are all is well; you just became too impatient.

²⁰ Warning: You can use Subroutines **E** (or **C**) to apply the sine (or logistic) map to the number in the display but if you want to iterate them you have to use **XEQ E** (or **XEQ C**) each time, rather than just using **R/S** for subsequent iterations. If you try to run it again with **R/S** then when the subroutine is finished it transfers control not back to you, but back to the main program, and you can find yourself with a lengthy and nonsensical calculation in progress.

0.5 after (*but not before!*) exactly n iterations.

B. Feigenbaum's α .

Another universal number discovered by Feigenbaum is $\alpha = 2.5029\dots$, which is easily estimated once you have the list of values of a at which the superstable 2^m cycles occur. If a is such a value and you iterate to the half-way point in the cycle (i.e. 2^{m-1} times), starting with $x = 0.5$ then you will almost (but not quite) get back to 0.5. (If you iterate another 2^{m-1} times you will, of course, get 0.5 exactly.)²¹

1. Make a table of the amount by which you miss, for all the a values you found for the sine map in Part 1 above. Thus the first entry in your table will be the amount by which the value of x in the superstable 2-cycle that is one step away from 0.5 differs from 0.5; the next entry will be the amount by which the value of x in the superstable 4-cycle that is two steps away from 0.5 differs from 0.5; the next entry will be the amount by which the value of x in the superstable 8-cycle that is four steps away from 0.5 differs from 0.5; and so on. These differences will get smaller and smaller, and the amount by which they shrink between successive period doublings gets closer and closer to Feigenbaum's α . To how many decimal places can you estimate α for the sine map?

2. Make the same series of estimates for the logistic map. (A table of the values of a giving superstable 2^m -cycles of the logistic map is given in the Appendix at the end of this Assignment.) Do you seem to be getting the same number α for both maps?

PART II.

A. Logistic map

The second part of the assignment is to explore some of the other peculiar things that happen with the logistic and sine maps for values of a even higher than the one at which the period doubling has given rise to chaotic behavior. In particular there turn out to be superstable cycles for all *odd* values of n . Indeed there are many for each n . I have a conjecture (which could be false). When n is an odd number bigger than 1, let a_n be the *smallest* value of a for which there is a superstable n -cycle. My conjecture is that the

²¹ Hitting **XEQ C** or **XEQ E** 64, 128, or even 256 times can get to be a bore, and can be more than a little irritating if you lose count. You can use Subroutine 2 to iterate the logistic map (or, the sine map, if you change the instruction **XEQ C** in location 43 to **XEQ E**) N times by (a) storing N in location 5, (b) storing your starting point x_0 (in this case 0.5) in location 6, (c) storing the value of the control parameter a (as usual) in location 9, and then (d) doing **XEQ 2**. The program will end with 0 in the display but if you then do (e) **RCL 6**, you can see the result of iterating the map N times starting at x_0 . (The next time you do this trick, of course, you have to store the starting value of x_0 in location 6 again.)

values $a_3, a_5, a_7, a_9, \dots$ form a decreasing series characterized by a Feigenbaum δ of their own which is about 2.8 for the logistic map.

See how well you can do hunting down this series of a_n 's. To get started, note that I found a_3 to be around 3.83, a_5 to be around 3.74, and a_7 to be around 3.70. Using the program you can easily get these values to the full accuracy of the calculator. From them you can estimate the effective δ , the amount by which the separation between successive a_n for odd n drops, and go on to find a_9, a_{11} , etc. to the full accuracy of the calculator. They start closing in on a value of a that I found to be somewhat less than 3.68. I was able to track them all the way to a_{29} , but at that point I found that when I iterated the logistic map 29 times for my value of a_{29} starting with $a = 0.5$, the 29th iterate differed enough from 0.5 to make me doubt that I would get reliable information by going to still higher n .²²

See how well you can do in discovering this sequence and how accurately you can estimate the value of δ that characterizes it. Better yet, see if you can prove me wrong, by constructing the sequence along the lines described above, and then finding a stable odd n -cycle at a value a below²³ the value in my sequence. (I'd be surprised if you can do this, but I haven't played around with these numbers for very long.) *Be very careful* not to settle for any old odd cycle the calculator comes up with. *There are enormous numbers of them in the chaotic region.* You really have to search systematically for the decreasing sequence in the manner I have described above. But be suspicious! It is possible that all I've discovered is a procedure for coming up with what I want to come up with. Adopt a critical attitude toward your findings and mine.

B. Sine map

Having done this for the logistic map, see if you can discover a similar sequence for the sine map. To avoid prejudicing your data this time I'll only tell you that the sequence I found has a_3 at about 3.75. Compare the Feigenbaum δ you find in this way for the odd a_n with the one you found for the logistic map. Does it display the kind of universality you found for the δ and α associated with period doubling?

End of Take-home Exam; Programs Follow

²² On the other hand you can see that the value of a_{29} is still quite accurate, since if you alter it even by 1 in the last decimal place and iterate 29 times, you miss getting back to 0.5 by considerably more! *Feel free to give some illustrations of this behavior for such large odd cycles in your essay.* As noted in Part I.B, you can save yourself some pushes of buttons by using Subroutine 2 to iterate the logistic map N times (or, the sine map, if you change the instruction **XEQ C** in location 43 to **XEQ E**).

²³ I stress *below*. You can find *lots* of them at *higher* values of a , but they are of no interest for this investigation.

Programs

The programs listed on the pages that follow contain everything you need for this assignment. They almost fill the entire program memory of the HP-20S. You should probably clear program memory before entering them in the order in which I list them. It took me about five minutes to do this. Be very careful. If you have trouble with the first few simple tests of the program described on page 5, check again to make sure you entered it correctly, by stepping through it and comparing what's in the display with what's in the program listings here. Be careful when stepping through not to introduce spurious 7's and 8's into the program by failing to use the blue shift key with 7 and with 8 to step down and up.

First Part of the Program:
Newton's method for finding roots of $g(z) = 0$.

	Key name	Key code	Comments
[y]	LBL B	61 41 B	Labels program B.
	STO 0	21 0	Puts guess z_0 in 0.
	XEQ F	41 F	Calculates $g(z_0)$
	STO 2	21 2	and puts result in location 2.
	RCL 0	22 0	Puts z_0 back in display.
	XEQ 1	41 1	Updates z_n (in 0) and d_n (in 1) until convergence.
	RCL 0	22 0	Display the solution.
	R/S	26	Stop to admire the answer.
[b]	GTO B	51 41 B	Do it again.
[y]	LBL 1	61 41 1	1 Labels the program that iterates.
	+	75	Adds to z_n
	RCL 1	22 1	d_n
	=	74	to get z_{n+1}
	STO 0	21 0	which is stored in location 0.
	XEQ 3	41 3	Check for a rare error
	XEQ F	41 F	Find g_{n+1} .
	STO \times 1	21 55 1	Put $d_n g_{n+1}$ in location 1.
	-	65	Subtract from g_{n+1}
	RCL 2	22 2	g_n , to get.
	=	74	$g_{n+1} - g_n$.
[y]	$x = 0?$	61 43	If $g_{n+1} = g_n$
[y]	RTN	61 26	then we are finished.
	STO + 2	21 75 2	Otherwise put g_{n+1} in location 2.
	$1/x$	15	Find $1/(g_{n+1} - g_n)$.
	+/-	32	Change its sign.
	STO \times 1	21 55 1	And put $d_{n+1} = d_n g_{n+1}/(g_n - g_{n+1})$ into 1.
	RCL 0	22 0	Put z_{n+1} in the display.
[b]	GTO 1	51 41 1	And iterate.

Second Part of the Program:
The Subroutine F that Does the Iterating
To Get $g(a) = f_a^{(n)}(x_0) - x_0$

Key name	Key code	Comments
[y] LBL F	61 41 F	Subroutine F.
STO 9	21 9	Put a in 9.
RCL 4	22 4	Put a copy of x_0
STO 6	21 6	in location 6.
RCL 8	22 8	Put a copy of n
STO 5	21 5	in location 5.
XEQ 2	41 2	Iterate $f_a(x)$ for n times.
RCL 6	22 6	Take updated value of x
–	65	and subtract from it
RCL 4	22 4	x_0
=	74	leaving $f^{(n)}(x_0) - x_0$ in the display.
[y] RTN	61 26	Go back to main program.
[y] LBL 2	61 41 2	Subroutine 2: iterates $f_a(x)$ for n times.
RCL 6	22 6	Take x ,
XEQ C	41 C	evaluate logistic map $f_a(x)$ [Change this to XEQ E to do the sine map instead]
STO 6	21 6	and put that back in 6 as the new x .
RCL 5	22 5	Take the number of iterations remaining
–	65	and subtract from it
1	1	1.
=	74	
[y] $x = 0?$	61 43	Is the result 0?
[y] RTN	61 26	If so, we're finished.
STO 5	21 5	If not, put the reduced number back in 5
[b] GTO 2	51 41 2	and iterate.

**Third Part of the Program:
The Subroutine C that Actually Calculates The Logistic Map**

Key name	Key code	Comments
[y] LBL C	61 41 C	Subroutine C. Starts with x in display
(33	
(33	
–	65	and subtracts from it
[b] x^2	51 11	x^2
)	34	to get $x(1 - x)$
×	55	which is then multiplied by
RCL 9	22 9	a
)	34	to get $f_a(x) = ax(1 - x)$,
[y] RTN	61 26	and that's it.

**Alternative Third Part²⁴ of the Program:
The Subroutine E that Actually Calculates The Sine Map**

Key name	Key code	Comments
[y] LBL E	61 41 E	Subroutine E. Starts with x in display,
(33	
(33	
×	55	multiplies it by
[y] π	61 22	π ,
)	34	
sin	23	and evaluates $\sin(\pi x)$
×	55	which is then multiplied by
RCL 9	22 9	a
÷	45	and divided
4	4	4
)	34	to get $f_a(x) = (a/4) \sin(\pi x)$.
[y] RTN	61 26	

²⁴ Enter both this and the preceding part into program memory. Which part the program actually uses depends on whether the instruction listed on page 13 is **XEQ C** or **XEQ E**.

**Subroutine 3: Checks that $0 \leq z_{n+1} \leq 4$:
If not, stops program, with 0 in display.**

	Key name	Key code	Comments
[y]	LBL 3	61 41 3	Subroutine 3. Starts with z_{n+1} in display.
	-	65	Subtract 2,
	2	2	
	=	74	
[b]	ABS	51 44	take the magnitude,
	-	65	subtract
	2	2	2 again,
	=	74	
	+/-	32	change the sign,
	+	75	and add to what you get
[b]	ABS	51 44	its absolute value.
	=	74	The result will be zero if and only if
			$z_{n+1} \leq 0$ or $z_{n+1} \geq 4$.
[y]	$x = 0?$	61 43	If the result is zero
	R/S	26	abort the program (with 0 in display).
	RCL 0	22 0	Otherwise put z_{n+1} back in display
[y]	RTN	61 26	and go on with the calculation.

Appendix

Here is a summary of my numerical explorations of the logistic map that lead to an estimate for Feigenbaum's δ , accurate to about 6 decimal places. In Part I.A you are asked to get the same kind of information for the sine map. In Part I.B you are asked to use the information you found in Part I.A and the information given in this Appendix to estimate Feigenbaum's α for the sine and logistic maps.

n	a_n	$D_n = a_n - a_{n-1}$	$\delta_n = D_{n-1}/D_n$
1	2.000000000000	—	—
2	3.23606797749	1.23606797749	—
4	3.49856169932	0.26249372183	4.708943013
8	3.55464086278	0.05607916346	4.680770996
16	3.56666737986	0.01202651708	4.662959616
32	3.56924353162	0.00257615176	4.66840396
64	3.56979529374	0.00055176212	4.6689536
128	3.56991346539	0.00011817165	4.6691581
256	3.56993877420	0.00002530881	4.669190
512	3.56994419457	0.00000542037	4.66920
1024	3.56994535553	0.00000116096	[4.66886...]

Here a_n is the value of the control parameter a at which the logistic map ($x \rightarrow ax(1-x)$) has an n -cycle that includes the point $x = 0.5$ (a so-called “superstable n -cycle”.) Note that when one reaches the superstable 1024-cycle the values of a_n are getting so close together that their differences are no longer given accurately enough to result in further improvement of the estimates δ_n to δ . To get it to more than six decimal places you need a computer that works to more than 12 place accuracy.