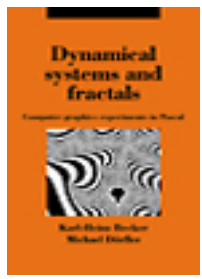


Dynamical Systems and Fractals

Computer Graphics Experiments with Pascal

Karl-Heinz Becker, Michael Dörfler, Translated by I. Stewart

Book DOI: http://dx.doi.org/10.1017/CBO9780511663031

Online ISBN: 9780511663031

Hardback ISBN: 9780521360258

Paperback ISBN: 9780521369107

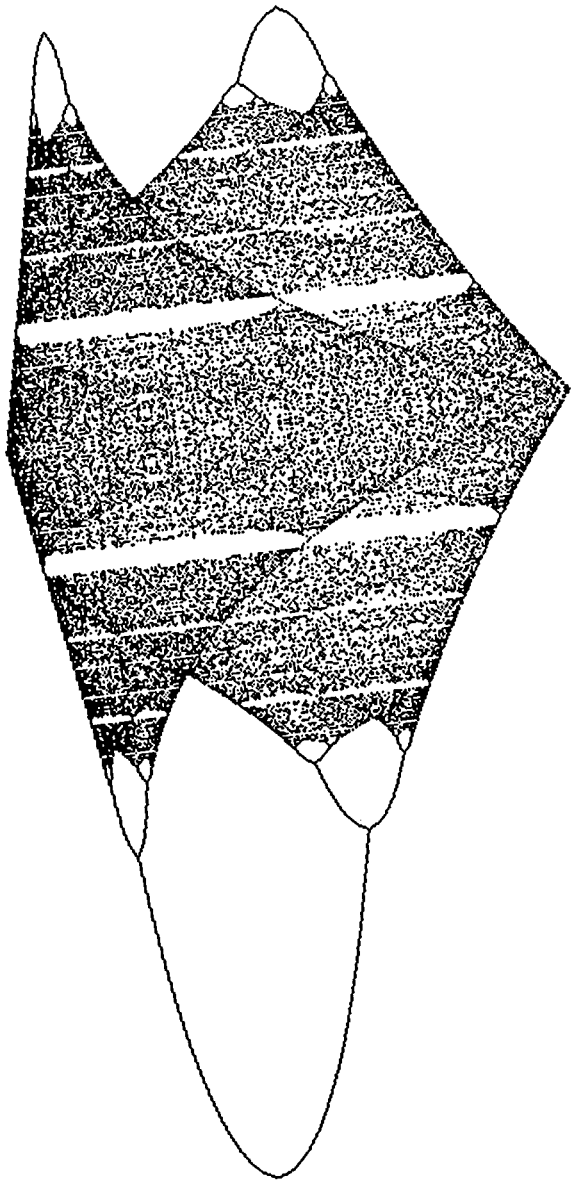
Chapter

2 - Between Order and Chaos: Feigenbaum Diagrams pp. 17-54

Chapter DOI: http://dx.doi.org/10.1017/CBO9780511663031.004

Cambridge University Press

2 Between Order and Chaos: Feigenbaum Diagrams



2.1 First Experiments

One of the most exciting experiments, in which we all take part, is one which Nature carries out upon us. This experiment is called *life*. The rules are the presumed laws of Nature, the materials are chemical compounds, and the results are extremely varied and surprising. And something else is worth noting: if we view the ingredients and the product as equals, then each year (each day, each generation) begins with exactly what the previous year (day, generation) has left as the starting-point for the next stage. That development is possible in such circumstances is something we observe every day.

If we translate the above experiment into a mathematical one, then this is what we get: a fixed rule, which transforms input into output; that is, a rule for calculating the output by applying it to the input. The result is the input value for the second stage, whose result becomes the input for the third stage, and so on. This mathematical principle of re-inserting a result into its own method of computation is called *feedback* (see Chapter 1).

We will show by a simple example that such feedback is not only easy to program, but it leads to surprising results. Like any good experiment, it raises ten times as many new questions as it answers.

The rules that will concern us are mathematical formulas. The values that we obtain will be real numbers between 0 and 1. One possible meaning for numbers between 0 and 1 is as percentages: $0\% \leq p \leq 100\%$. Many of the rules that we describe in this book arise only from the mathematician's imagination. The rule described here originated when researchers tried to apply mathematical methods to growth, employing an interesting and widespread model. We will use the following as an example, taking care to remember that not everything in the model is completely realistic.

There has been an outbreak of measles in a children's home. Every day the number of sick children is recorded, because it is impossible to avoid sick and well children coming into contact with each other. *How does the number change*?

This problem corresponds to a typical dynamical system - naturally a very simple one. We will develop a mathematical model for it, which we can use to simulate an epidemic process, to understand the behaviour and regularities of such a system.

If, for example, 30% of the children are already sick, we represent this fact by the formula $p = 0.3$. The question arises, how many children will become ill the next day? The rule that describes the spread of disease is denoted mathematically by $f(p)$. The epidemic can then be described by the following equation:

$$f(p) = p+z.$$

That is, to the original p we add a growth z.

The value of z, the increase in the number of sick children, depends on the number p of children who are already sick. Mathematically we can write this dependence as $z \approx p$, saying that 'z is proportional to p'. By this proportionality expression we mean that z may depend upon other quantities than p. We can predict that z depends also upon the number of well children, because there can be no increase if all the children

are already sick in bed. If 30% are ill, then there must be 100% - 30% = 70% who are well. In general there will be $100\%-p = 1-p$ well children, so we also have $z \approx (1-p)$. We have thus decided that $z \approx p$ and $z \approx (1-p)$. Combining these, we get a growth term $z \approx p*(1-p)$. But because not all children meet each other, and not every contact leads to an infection, we should include in the formula an *infection rate k*. Putting all this together into a single formula we find that:

$$z = k*p*(1-p),$$

so that

$$f(p) = p+k*p*(1-p).$$

In our investigation we will apply this formula on many successive days. In order to distinguish the numbers for a given day, we will attach an index to p. The initial value is p_0, after one day we have p_1, and so on. The result $f(p)$ becomes the initial value for the next stage, so we get the following scheme:

$$f(p_0) = p_0+k*p_0*(1-p_0) = p_1$$
$$f(p_1) = p_1+k*p_1*(1-p_1) = p_2$$
$$f(p_2) = p_2+k*p_2*(1-p_2) = p_3$$
$$f(p_3) = p_3+k*p_3*(1-p_3) = p_4$$

and so on. In general we have

$$f(p_n) = p_n+k*p_n*(1-p_n) = p_{n+1}.$$

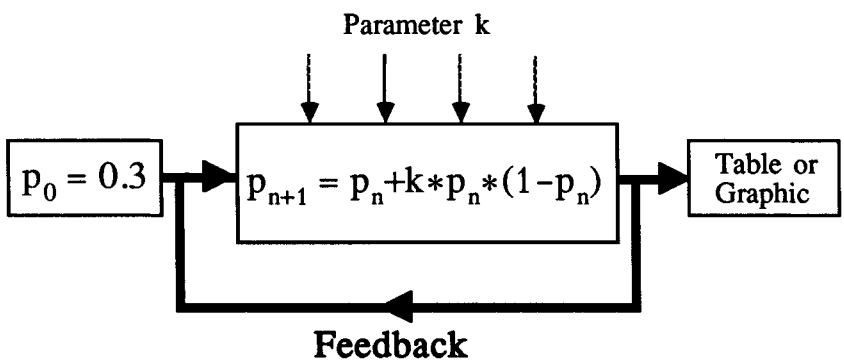


Figure 2.1-1 Feedback scheme for 'Measles'.

In other words this means nothing more than that the new values are computed from the old ones by applying the given rule. This process is called *mathematical feedback* or *iteration*. We have already spoken of this iterative procedure in our fundamental considerations in Chapter 1.

For any particular fixed value of k we can calculate the development of the disease from a given starting value p_0. Using a pocket calculator, or mental arithmetic, we find that these function values more or less quickly approach the limit 1; that is, all children fall sick. We would naturally expect this to occur faster, the larger the factor k is.

Table 2-1

	A	B	C	D	E
1	p n	k	1 - p n	k*pn*(1-pn)	pn+1
2	0.3000	0.5000	0.7000	0.1050	0.4050
3	0.4050	0.5000	0.5950	0.1205	0.5255
4	0.5255	0.5000	0.4745	0.1247	0.6502
5	0.6502	0.5000	0.3498	0.1137	0.7639
6	0.7639	0.5000	0.2361	0.0902	0.8541
7	0.8541	0.5000	0.1459	0.0623	0.9164
8	0.9164	0.5000	0.0836	0.0383	0.9547
9	0.9547	0.5000	0.0453	0.0216	0.9763
10	0.9763	0.5000	0.0237	0.0116	0.9879
11	0.9879	0.5000	0.0121	0.0060	0.9939

Figure 2.1-2 Development of the disease for $p_0 = 0.3$ and $k = 0.5$.

Table 2-2

	A	B	C	D	E
1	p n	k	1 - p n	k*pn*(1-pn)	pn+1
2	0.3000	1.0000	0.7000	0.2100	0.5100
3	0.5100	1.0000	0.4900	0.2499	0.7599
4	0.7599	1.0000	0.2401	0.1825	0.9424
5	0.9424	1.0000	0.0576	0.0543	0.9967
6	0.9967	1.0000	0.0033	0.0033	1.0000
7	1.0000	1.0000	0.0000	0.0000	1.0000
8	1.0000	1.0000	0.0000	0.0000	1.0000
9	1.0000	1.0000	0.0000	0.0000	1.0000
10	1.0000	1.0000	0.0000	0.0000	1.0000
11	1.0000	1.0000	0.0000	0.0000	1.0000

Figure 2.1-3 Development of the disease for $p_0 = 0.3$ and $k = 1.0$.

In order to get a feeling for the method of calculation, get out your pocket calculator. Work out the results first yourself, for the k-values

$$k_1 = 0.5,\ k_2 = 1,\ k_3 = 1.5,\ k_4 = 2,\ k_5 = 2.5,\ k_6 = 3$$

using the formula

$$f(p_n) = p_n + k*p_n*(1-p_n) = p_{n+1}$$

to work out p_1 up to p_5. Take $p_0 = 0.3$ in each case. So that you can check your results, we have given the calculation in the form of six tables (see Figures 2.1-2 to 2.1-7). In each table ten values per column are shown. In column A are the values p_i, in column E the values p_{i+1}.

Table 2-3

	A	B	C	D	E
1	p n	k	1 - p n	k*pn*(1-pn)	pn+1
2	0.3000	1.5000	0.7000	0.3150	0.6150
3	0.6150	1.5000	0.3850	0.3552	0.9702
4	0.9702	1.5000	0.0298	0.0434	1.0136
5	1.0136	1.5000	-0.0136	-0.0207	0.9929
6	0.9929	1.5000	0.0071	0.0105	1.0035
7	1.0035	1.5000	-0.0035	-0.0052	0.9983
8	0.9983	1.5000	0.0017	0.0026	1.0009
9	1.0009	1.5000	-0.0009	-0.0013	0.9996
10	0.9996	1.5000	0.0004	0.0007	1.0002
11	1.0002	1.5000	-0.0002	-0.0003	0.9999

Figure 2.1-4 Development of the disease for $p_0 = 0.3$ and $k = 1.5$.

Table 2-4

	A	B	C	D	E
1	p n	k	1 - p n	k*pn*(1-pn)	pn+1
2	0.3000	2.0000	0.7000	0.4200	0.7200
3	0.7200	2.0000	0.2800	0.4032	1.1232
4	1.1232	2.0000	-0.1232	-0.2768	0.8464
5	0.8464	2.0000	0.1536	0.2600	1.1064
6	1.1064	2.0000	-0.1064	-0.2354	0.8710
7	0.8710	2.0000	0.1290	0.2248	1.0957
8	1.0957	2.0000	-0.0957	-0.2098	0.8859
9	0.8859	2.0000	0.1141	0.2021	1.0880
10	1.0880	2.0000	-0.0880	-0.1916	0.8965
11	0.8965	2.0000	0.1035	0.1857	1.0821

Figure 2.1-5 Development of the disease for $p_0 = 0.3$ and $k = 2.0$.

Table 2-5

	A	B	C	D	E
1	p n	k	1 - p n	k*pn*(1-pn)	pn+1
2	0.3000	2.5000	0.7000	0.5250	0.8250
3	0.8250	2.5000	0.1750	0.3609	1.1859
4	1.1859	2.5000	-0.1859	-0.5513	0.6347
5	0.6347	2.5000	0.3653	0.5797	1.2143
6	1.2143	2.5000	-0.2143	-0.6507	0.5637
7	0.5637	2.5000	0.4363	0.6149	1.1785
8	1.1785	2.5000	-0.1785	-0.5260	0.6525
9	0.6525	2.5000	0.3475	0.5669	1.2194
10	1.2194	2.5000	-0.2194	-0.6687	0.5507
11	0.5507	2.5000	0.4493	0.6186	1.1692

Figure 2.1-6 Development of the disease for $p_0 = 0.3$ and $k = 2.5$.

Table 2-6

	A	B	C	D	E
1	p n	k	1 - p n	k*pn*(1-pn)	pn+1
2	0.3000	3.0000	0.7000	0.6300	0.9300
3	0.9300	3.0000	0.0700	0.1953	1.1253
4	1.1253	3.0000	-0.1253	-0.4230	0.7023
5	0.7023	3.0000	0.2977	0.6272	1.3295
6	1.3295	3.0000	-0.3295	-1.3143	0.0152
7	0.0152	3.0000	0.9848	0.0449	0.0601
8	0.0601	3.0000	0.9399	0.1694	0.2295
9	0.2295	3.0000	0.7705	0.5305	0.7600
10	0.7600	3.0000	0.2400	0.5473	1.3072
11	1.3072	3.0000	-0.3072	-1.2048	0.1024

Figure 2.1-7 Development of the disease for $p_0 = 0.3$ and $k = 3.0$.

The tables were computed using the spreadsheet program 'Excel' on a Macintosh. Other spreadsheets, for example 'Multiplan', can also be used for this kind of investigation. For those interested, the program is given in Figure 2.1-8, together with the linking formulas. All diagrams involve the mathematical feedback process. The result of field E2 provides the starting value for A3, the result of E3 is the initial value for A4, and so on.

Measles Spreadsheet

	A	B	C	D	E
1	p n	k	1 - p n	k*pn*(1-pn)	pn+1
2	0.3	3	=1-A2	=B2*A2*C2	=A2+D2
3	=E2	=B2	=1-A3	=B3*A3*C3	=A3+D3
4	=E3	=B3	=1-A4	=B4*A4*C4	=A4+D4
5	=E4	=B4	=1-A5	=B5*A5*C5	=A5+D5
6	=E5	=B5	=1-A6	=B6*A6*C6	=A6+D6
7	=E6	=B6	=1-A7	=B7*A7*C7	=A7+D7
8	=E7	=B7	=1-A8	=B8*A8*C8	=A8+D8
9	=E8	=B8	=1-A9	=B9*A9*C9	=A9+D9
10	=E9	=B9	=1-A10	=B10*A10*C10	=A10+D10
11	=E10	=B10	=1-A11	=B11*A11*C11	=A11+D11

Figure 2.1-8 List of formulas.

Now represent your calculations graphically. You have six individual calculations to deal with. Each diagram, in a suitable coordinate system, contains a number of points generated by feedback.

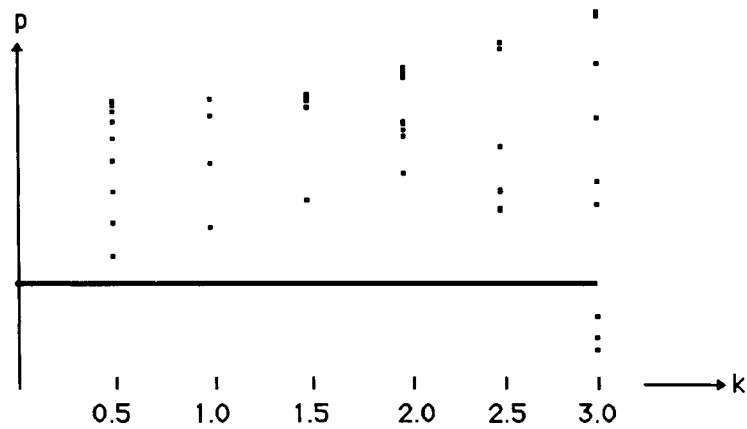


Figure 2.1-9 Discrete series of 6 (k_i, p_i)-values after 10 iterations.

We can combine all six diagrams into one, where for each k_i-value (k_i = 0.5, 1.0, 1.5, 2.0, 2.5, 3.0) we show the corresponding p_i-values (Figure 2.1-9).

You must have noticed how laborious all this is. Further, very little can be deduced from this picture. To gain an understanding of this dynamical system, it is not sufficient to carry out the feedback process for just 6 k-values. We must do more: for each k_i-value $0 \leq k_i \leq 3$ that can be distinguished in the picture, we must run continuously through the entire range of the k-axis, and draw in the corresponding p-values.

That is a tolerably heavy computation. No wonder that it took until the middle of this century before even such simple formulas were studied, with the help of newfangled computers.

A computer will also help us investigate the 'measles problem'. It carries out the same tedious, stupid calculation over and over again, always using the same formula.

When we go on to write a program in Pascal, it will be useful for more than just this problem. We construct it so that we can use large parts of it in other problems. New programs will be developed from this one, in which parts are inserted or removed. We just have to make sure that they fit together properly (see Chapter 11).

For this problem we have developed a Pascal program, in which only the main part of the problem is solved. Any of you who cannot finish the present problem, given the program, will find a complete solution in Chapters 11ff.

Program 2.1-1

```
PROGRAM MeaslesNumber;
  VAR
    Population, Feedback : real;
```

```
     MaximalIteration : integer;
(*-------------------------------------------------*)
(*    BEGIN : Problem-specific procedures *)
     FUNCTION f (p, k : real) : real;
     BEGIN
        f := p + k * p * (1 - p) ;
     END;

     PROCEDURE MeaslesValue;
        VAR
           i : integer;
     BEGIN
        FOR i := 1 to MaximalIteration DO
        BEGIN
           Population := f(Population, Feedback);
           writeln('After' , i , 'Iterations p has the
                        value :',
           Population : 6 : 4);
        END;
     END;
   (* END Problem-specific procedures *)
(* -------------------------------------------------*)
(* BEGIN: Useful subroutines *)
(* see Chapter 11.2 *)
(* END  : Useful subroutines *)

(* BEGIN : Procedures of main program *)
   PROCEDURE Hello;
   BEGIN
      InfoOutput ('Calculation of Measles-Values');
      InfoOutput ('---------------------------');
      Newlines (2);
      CarryOn ('Start : ');
      Newlines (2);
   END;

   PROCEDURE Application;
   BEGIN
      ReadReal ('Initial Population p (0 to 1) >',
         Population);
      ReadReal ('Feedback Parameter k (0 to 3) >',
```

```
        Feedback);
      ReadInteger ('Max. Iteration Number >',
        MaximalIteration);
  END;

  PROCEDURE ComputeAndDisplay;
  BEGIN
    MeaslesValues;
  END;

  PROCEDURE Goodbye;
  BEGIN
    CarryOn ('To stop : ');
  END;
(* END : Procedures of Main Program)


BEGIN (* Main Program *)
  Hello;
  Application;
  ComputeAndDisplay;
  Goodbye;
END.
```

We have here written out only the main part of the program. The 'useful subroutines' are particular procedures to read in numbers or to output text to the screen (see Chapters 11ff.).

When we type in this Pascal program and run it, it gives an output like Figure 2.1-10. In Figure 2.1-10 not all iterations are shown. In particular the interesting values are missing. You should now experiment yourself: we invite you to do so before reading on. Only in this way can you appreciate blow by blow the world of computer simulation.

We have now built our first measuring instrument, and we can use it to make systematic investigations. What we have previously accomplished with tedious computations on a pocket calculator, have listed in tables, and drawn graphically (Figure 2.1-9) can now be done much more easily. We can carry out the calculations on a computer. We recommend that you now go to your computer and do some experimenting with Pascal program 2.1-1.

A final word about our 'measuring instrument'. The basic structure of the program, the main program, will not be changed much. It is a kind of standard tool, which we always construct. The useful subroutines are like machine parts or building blocks,

which we can use in future, without worrying further. For those of you who do not feel so sure of yourselves we have an additional offer: complete tested programs and parts of programs. These are systematically collected together in Chapter 11.

```
Initial Population p (0 to 1)       >0.5
Feedback Parameter k (0 to 3)       >2.3
Max. Iteration No.                  >20

After 1  Iterations p has the value : 1.0750
After 2  Iterations p has the value : 0.8896
After 3  Iterations p has the value : 1.1155
After 4  Iterations p has the value : 0.8191
After 5  Iterations p has the value : 1.1599
After 6  Iterations p has the value : 0.7334
After 7  Iterations p has the value : 1.1831
After 8  Iterations p has the value : 0.6848
After 9  Iterations p has the value : 1.1813
After 10 Iterations p has the value : 0.6888
After 11 Iterations p has the value : 1.1818
After 12 Iterations p has the value : 0.6876
After 13 Iterations p has the value : 1.1817
After 14 Iterations p has the value : 0.6880
After 15 Iterations p has the value : 1.1817
After 16 Iterations p has the value : 0.6879
After 17 Iterations p has the value : 1.1817
After 18 Iterations p has the value : 1.6879
After 19 Iterations p has the value : 1.1817
After 20 Iterations p has the value : 0.6879
```

Figure 2.1-10 Calculation of measles values.

Computer Graphics Experiments and Exercises for §2.1

Exercise 2.1.-1

Implement the measles formula using a spreadsheet program. Generate similar tables to those shown in Figures 2.1-1 to 2.1-7. Check your values against the tables.

Exercise 2.1-2

Implement Program 2.1-1 on your computer. Carry out 30 iterations with 6 data sets. For a fixed initial value $p_0 = 0.3$ let k take values from 0 to 3 in steps of 0.5.

Exercise 2.1-3

Now experiment with other initial values of p, vary k, etc.

Once you've got the program `MeaslesNumber` running, you have your first measuring instrument. Find out for which values of k and which initial values of p the resulting series of p-values is

(a) simple (convergence to $p = 1$),

(b) interesting, and

(c) dangerous.

We call a series ' dangerous' when the values get larger and larger - so the danger is that they exceed what the computer can handle. For many implementations of Pascal the following range of values is not dangerous: $10^{-37} < |x| < 10^{38}$ for numbers x of type `real`.

By the interesting range of k we mean the interval from $k = 1.8$ to $k = 3.0$. Above this range it is dangerous; below, it is boring.

Exercise 2.1-4

Now that we have delineated the boundaries of the k-regions, we can present the above results acoustically. To do this you must change the program a little.

Rewrite Program 2.1-1 so that the series of numerical values becomes audible as a series of musical tones.

Exercise 2.1-5

What do you observe as a result of your experiments?

2.1.1 It's Prettier with Graphics

It can definitely happen that for some k-values no regularity can be seen in the series of numbers produced: the p-values seem to be more or less disordered. The experiment of Exercise 2.1-4 yields a regular occurrence of similar tone sequences only for certain values of p and k. So we will now make the computer sketch the results of our experiments, because we cannot find our way about this 'numerical salad' in any other manner. To do that we must first solve the problem of relating a cartesian coordinate system with coordinates x,y or k,p to the screen coordinates. Consider Figure 2.1.1-1 below.

Our graphical representations must be transformed in such a way that they can all be drawn on the same screen. In the jargon of computer graphics we refer to our mathematical coordinate system as the *universal coordinate system*. With the aid of a transformation equation we can convert the universal coordinates into screen coordinates.

Figure 2.1.1-1 shows the general case, in which we wish to map a *window*, or rectangular section of the screen, onto a projection surface, representing part of the screen. The capital letter U represents the universal coordinate system, and S the screen

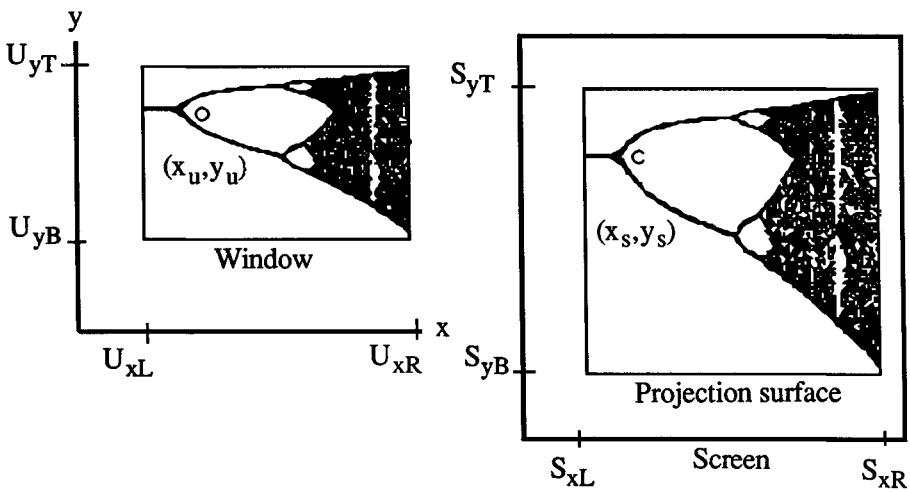


Figure 2.1.1-1 Two coordinate systems.

coordinate system.

The following transformation equations hold:

$$x_S = \frac{S_{xR}-S_{xL}}{U_{xR}-U_{xL}}(x_u-U_{xL}) + S_{xL} ,$$

$$y_S = \frac{S_{yT}-S_{yB}}{U_{yT}-U_{yB}}(y_u-U_{yB})+S_{yB} .$$

Here *L*, *R*, *B*, *T* are the initials of 'left', 'right', 'bottom', 'top'. We want to express the transformation equation as simply as possible. To do this, we assume that we wish to map the window onto the entire screen. Then we can make the following definitions:

- U_{yT} = `Top` and S_{yT} = `Yscreen`
- U_{yB} = `Bottom` and S_{yB} = `0`
- U_{xL} = `Left` and S_{xL} = `0`
- U_{xR} = `Right` and S_{xR} = `Xscreen`.

This simplifies the transformation equation:

$$x_S = \frac{\text{Xscreen}}{\text{Right-Left}}(x_u-\text{Left})$$

$$y_S = \frac{\text{Yscreen}}{\text{Top-Bottom}}(y_u-\text{Bottom}) .$$

On the basis of this formula we will write a program that is suitable for displaying the

measles values. Observe its similar structure to that of Program 2.1-1.

Program 2.1.1-1

```
PROGRAM MeaslesGraphic;
   (* Possible declaration of graphics library *)
   (* Insert in a suitable place *)
CONST
   Xscreen = 320;  (* e.g. 320 pixels in x-direction *)
   Yscreen = 200;  (* e.g. 200 pixels in y-direction *)
VAR
   Left, Right, Top, Bottom, Feedback : real;
   IterationNo : Integer;
(* BEGIN:  Graphics Procedures *)
PROCEDURE SetPoint (xs, ys : integer);
BEGIN  (* Insert machine-specific graphics commands here*)
END;

PROCEDURE SetUniversalPoint (xu, yu: real);
   VAR
      xs, ys : real;
BEGIN
   xs := (xu - Left) * Xscreen / (Right - Left);
   ys := (yu - Bottom) * Yscreen / (Top - Bottom);
   SetPoint (round(xs), round(ys));
END;

PROCEDURE TextMode;
BEGIN
   (* Insert machine-specific commands: see hints *)
   (* in Chapter 11 *)
END;

PROCEDURE GraphicsMode;
BEGIN
   (* Insert machine-specific commands: see hints *) in
   (* Chapter 11 *)
END;
PROCEDURE EnterGraphics;
   (* various actions to initialise the graphics *)
   (* such as GraphicsMode etc.                    *)
      GraphicsMode;
```

```
END;
PROCEDURE ExitGraphics;
BEGIN
(* Actions to end the graphics, e.g. : *)
   REPEAT
      (* Button is a machine-specific procedure *)
   UNTIL Button;
   TextMode;
END;
(* END:  Graphics Procedures *)
(* ------------------------------------------------------ *)
(* BEGIN : Problem-specific Procedures *)
FUNCTION f (p, k : real) : real;
BEGIN
   f := p + k * p * (1 - p);
END;

PROCEDURE MeaslesIteration;
   VAR
      range, i: integer
      population : real
      deltaxPerPixel: real;
BEGIN
   deltaxPerPixel := (Right - Left) / Xscreen;
   FOR range := 0 TO Xscreen DO
   BEGIN
      Feedback := Left + range * deltaxPerPixel;
      population := 0.3
      FOR i := 0 to IterationNo DO
      BEGIN
         SetUniversalPoint (Feedback, population);
         population := f ( population, Feedback );
      END;
   END;
END;
(* END:  Problem-specific Procedures *)
(*-------------------------------------------------------*)
(* BEGIN  Useful Subroutines *)
(* See Program 2.1-1, not given here *)
(* END : Useful Subroutines *)
(* BEGIN: Procedures of Main Program *)
```

```
PROCEDURE Hello;
BEGIN
   TextMode;
   InfoOutput ('Diagram of the Measles Problem');
   InfoOutput ('-----------------------------');
   Newlines (2);
   CarryOn ('Start : ');
   Newlines (2);
END;

PROCEDURE Initialise;
BEGIN
   ReadReal ('Left                    >', Left);
   ReadReal ('Right                   >', Right);
   ReadReal ('Top                     >', Top);
   ReadReal ('Bottom                  >', Bottom);
   ReadInteger (Iteration Number      >', IterationNo);
END;

PROCEDURE ComputeAndDisplay;
BEGIN
   EnterGraphics;
   MeaslesIteration;
   ExitGraphics;
   END;

PROCEDURE Goodbye;
BEGIN
   CarryOn ('To end : ');
END
(*END : Procedures of Main Program *)

BEGIN (* Main Program *)
   Hello;
   Initialise;
   ComputeAndDisplay;
   Goodbye;
END.
```

We suggest that you now formulate Program 2.1.1-1 as a complete Pascal program and enter it into your machine. The description above may help, but you may have developed your own programming style, in which case you can do everything differently if you

wish. Basically Program 2.1.1-1 solves the 'algorithmic heart' of the problem. The machine-specific components are discussed in Chapter 12 in the form of reference programs with the appropriate graphics commands included.

`TextMode`, `GraphicsMode`, and `Button` are machine-specific procedures. In implementations,`TextMode` and `GraphicsMode` are system procedures. This is the case for Turbo Pascal on MS-DOS machines and for UCSD Pascal (see Chapter 12).

`Button` corresponds to the `Keypressed` function of Turbo Pascal. The 'useful subroutines' have already been described in Program 2.2-1. By comparing Programs 2.1-1 and 2.1.1-1 you will see that we have converted our original 'numerical' measuring-instrument into a 'graphical' one. Now we can visualise the number flow more easily.

The development of the program mostly concerns the graphics: the basic structure remains unchanged.

Something new, which we must clarify, occurs in the procedure `MeaslesIteration` (see Program 2.1.1-1):

```
deltaxPerPixel := (Right - Left) / Xscreen;
   FOR range := 0 TO Xscreen DO
   BEGIN
      Feedback := Left + range * deltaxPerPixel;
                          ...
```

Compare this with the transformation formula:

$$x_S = \frac{\text{Xscreen}}{\text{Right-Left}}(x_u\text{-Left})$$

Solve this equation for x_S.

When we give the screen coordinate x_S the value 0, then the universal coordinate must become `Left`. Setting the value `Xscreen` for the maximal screen coordinate x_S, we get the value `Right`. Every other screen coordinate corresponds to a universal coordinate between `Left` and `Right`. The smallest unit of size that can be distinguished on the screen is one pixel. The corresponding smallest unit of size in universal coordinates is thus `deltaxPerPixel`.

After this brief explanation of the graphical representations of the measles data with the aid of Program 2.1.1-1, we will describe the result, produced by the computer program in the form of a graphic. See Figure 2.1.1-2, to which we have added the coordinate axes.

How do we interpret this graphic? From right to left the factor k changes in the range 0 to 3. For small values of k (in particular $k = 0$) the value of p changes by little or nothing. For k-values in the region of 1 we see the expected result: p takes the value 1 and no further changes occur.

The interpretation for the model is thus: if the infection rate k is sufficiently large, then soon all children become ill ($p = 100\%$). This occurs more rapidly, the larger k is.

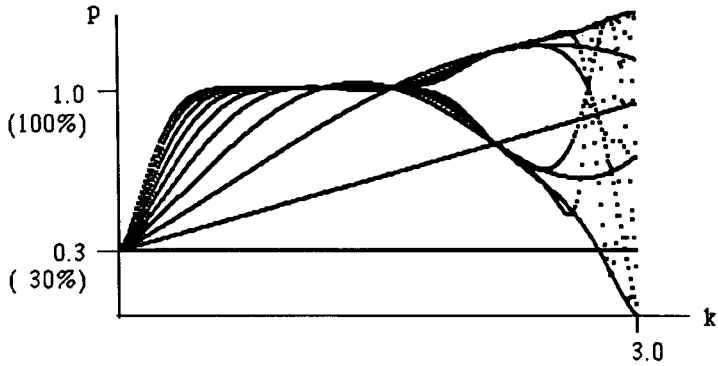


Figure 2.1.1-2 Representation of the measles epidemic on the screen, `IterationNo = 10`.

You can also see this result using the values computed by pocket calculator, e.g. Figures 2.1-1 to 2.1-7.

For k-values greater than 1 something surprising and unexpected happens: p can become larger than 1! Mathematically, that's still meaningful. Using the formula you can check that the calculation has proceeded correctly. Unfortunately it illustrates a restriction on our measles example, because more than 100% of the children cannot become ill. The picture shows quite different results here. Might something abnormal be going on?

Here we find ourselves in a typical experimental situation: the experiment has to some extent confirmed our expectations, but has also led to unexpected results. That suggests new questions, which possess their own momentum. Even though we can't make sense of the statement that 'more than 100% of children become sick', the following question starts to look interesting: how does p behave, if k gets bigger than 2?

Figure 2.1.1-2 provides a hint: p certainly does not, as previously, tend to the constant value $p = 1$. Apparently there is no fixed value which p approaches, or, as mathematicians say, towards which the sequence p *converges*.

It is also worth noting that the sequence does not *diverge* either. Then p would increase beyond all bounds and tend towards $+\infty$ or $-\infty$. In fact the values of p jump about 'chaotically', to and fro, in a range of p between $p = 0$ and $p = 1.5$. It does not seem to settle down to any particular value, as we might have expected, but to many. What does that mean?

In order better to understand the number sequences for the population p, we will now take a quick look at the screen print-out of Figure 2.1-10 (calculation of measles values) from Chapter 2.1.

We can use the program again in Exercises 2.1-1 to 2.1-4, which we have already given, letting us display the results once more on the screen (see Figure 2.1-10). As an

additional luxury we can also make the results audible as a series of musical tones. The melody is not important. You can easily tell whether the curve tends towards a single value or many. If we experiment on the `MeaslesNumber` program with $k = 2.3$, we find an ' oscillating phase' jumping to and fro between two values (see Figure 1.2-10). One value is > 1, and the other < 1. For $k = 2.5$ it is even more interesting. At this point you should stop hiding behind the skirts of our book, and we therefore suggest that, if you have not done so already, you write your first program and carry out your first experiment now. We will once more formulate the task precisely:

Computer Graphics Experiments and Exercises for §2.1.1

Exercise 2.1.1-1

Derive the general transformation equations for yourself with the aid of Figure 2.1.1-1. Check that the simplified equation follows from the general one. Explain the relation between them.

Exercise 2.1.1-2

Implement the Pascal program `MeaslesGraphic` on your computer. Check that you obtain the same graphic displays as in Figure 2.1.1-2. That shows you are on the right track.

Exercise 2.1.1.3

Establish the connection between the special transformation formula and the expression for `deltaxPerPixel`.

Exercise 2.1.1-4

Use the program `MeaslesGraphic` to carry out the same investigations as in Exercise 2.1-3 (see Chapter 2.1) - this time with graphical representation of the results.

2.1.2 Graphical Iteration

It may perhaps have occurred to you that the function

$$f(x) = x + k * x * (1-x)$$

- for so we can also write the equation - is nothing other than the function for a parabola

$$f(x) = -k * x^2 + (k+1) * x$$

This is the equation of a downward-opening parabola through the origin with its vertex in the first quadrant. It is clear that for different values of k we get different parabolas. We can also study the 'feedback effect' of this parabola equation by *graphical iteration*.

Let us explain this important concept.

Feedback means that the result of a calculation is replaced into the same equation as

a new initial condition. After many such feedbacks (iterations) we establish that the results run through certain fixed values. By *graphical feedback* we refer to the picture of the function in an x,y-coordinate system (x stands for p, y for $f(x)$ or $f(p)$).

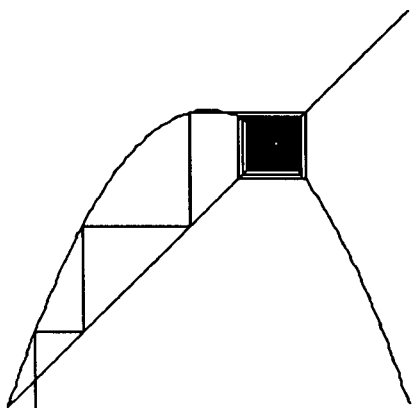
Graphical iteration takes place in the following way. Beginning with an x-value, move vertically up or down in the picture until you hit the parabola. You can read off $f(x)$ on the y-axis. This is the initial value for the next stage of the feedback. The value must be carried across to the x-axis. For this purpose we use the diagonal, with equation y = x. From the point on the parabola (with coordinates (x, $f(x)$) we draw a line horizontally to the right (or left), until we encounter the diagonal (at the coordinates ($f(x)$, $f(x)$)). Then we draw another vertical to meet the parabola, a horizontal to meet the diagonal, and so on.

This procedure will be explained further in the program and the pictures that follow it.

Program Fragment 2.1.2-1

```
(* ------------------------------------------------ *)
(* BEGIN : Problem-specific Procedures *)
FUNCTION f (p, k : real) : real;
BEGIN
   f := p + k * p * (1 - p);
END;

PROCEDURE ParabolaAndDiagonal(population,feedback : real) ;
   VAR
      xCoord, deltaxPerPixel : real;
BEGIN
   DeltaxPerPixel : = (Right - Left) / Xscreen;
   SetUniversalPoint (Left, Bottom);
   DrawUniversalLine  (Right, Top);
   DrawUniversalLine (Left, Bottom);
   xCoord  := Left;
   REPEAT
      DrawUniversalLine (xCoord, f(xCoord, feedback));
      xCoord := xCoord + deltaxPerPixel;
   UNTIL (xCoord > Right);
   GoToUniversalPoint (population, Bottom);
END;

PROCEDURE GraphicalIteration;
(* Version for graphical iteration *)
   VAR
```

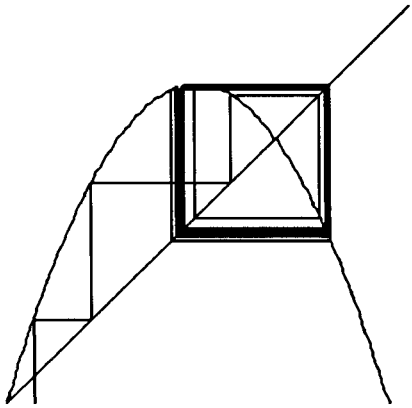
```
        previousPopulation : real;
    BEGIN
       ParabolaAndDiagonal (population, feedback);
       REPEAT
          DrawUniversalLine (population, population);
          previousPopulation := population;
          population := f(population, feedback);
          DrawUniversalLine (previousPopulation, population);
       UNTIL Button;
    END;

    (* END : Problem-specific Procedures *)
    (* ------------------------------------------------ *)

    (* DrawUniversalLine (x,y) draws a line from the *)
    (* current position to the point with universal coordinates
(x,y). *)
```



```
Graphical Iteration
-------------------

Start: hit <RETURN> key

Left        >0
Right       >1.5
Bottom      >0
Top         >1.5
Population  >0.1
Feedback    >1.99

End: hit <RETURN> key
```

Figure 2.1.2-1 Initial value $p = 0.1$, $k = 1.99$, a limit point, with screen dialogue.

For each given k-value we get distinct pictures. If the final value is the point $f(p) = 1$, we obtain a spiral track (Figure 2.1.2-1). In all other cases the horizontal and vertical lines tend towards segments of the original curve, which correspond to limiting p-values. Clearly the two vertical lines in Figure 2.1.2-2 represent two different p-values.



```
Graphical Iteration
-------------------

Start: hit <RETURN> key

Left        >0
Right       >1.5
Bottom      >0
Top         >1.5
Population  >0.1
Feedback    >2.4

End: hit <RETURN> key
```

Figure 2.1.2-2 Initial value $p = 0.1$, $k = 2.4$, two limiting points.

The distinct cases (limiting value 1, or n-fold cycles, Figures 2.1.2-1, 2.1.2-2) are thus made manifest. For an overview it can be useful to carry out the first 50 iterations without drawing them, after which 50 iterations are carried out and drawn.

This process of graphical iteration can also be applied to other functions. In this way we obtain rules, about the form of the graph of a function, telling us which of the above two effects it will produce.

Computer Graphics Experiments and Exercises for §2.1.2

Exercise 2.1.2-1

Develop a program for graphical iteration. Try to generate Figures 2.1.2-1 and 2.1.2-2. Experiment with the initial value $p = 0.1$ and $k = 2.5, 3.0$. How many limiting values do you get?

Exercise 2.1.2-2

Devise some other functions and apply graphical iteration to them.

2.2 Fig-trees Forever

In our experiments with the program `MeaslesGraphic` you must surely have noticed that the lines in the range $0 \leq k \leq 1$ get closer and closer together, if we increase the number of iterations (see Program 2.1.1-1). Until now we have computed with small values, in order not to occupy too much of the computer's time. But now we will make our first survey of the entire range. Figure 2.2.-1 shows the result of 50 iterations for comparison with Figure 2.1.1-2.

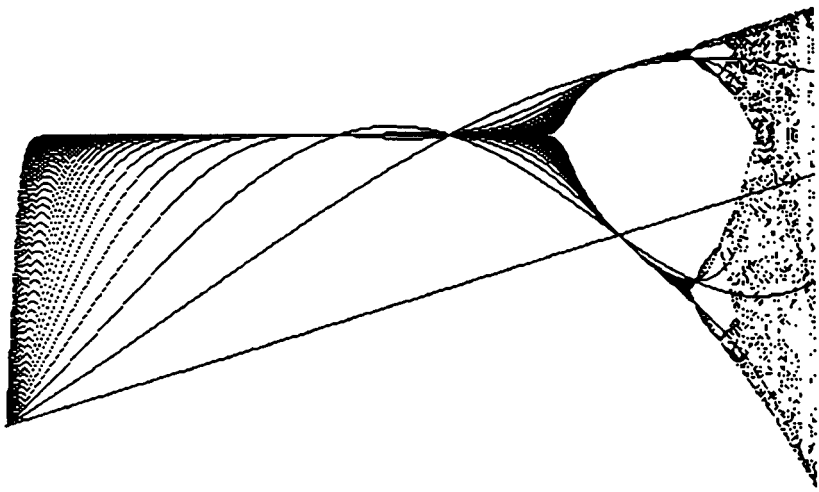


Figure 2.2-1 Situation after the onset of oscillations, iteration number = 50.

Obviously some structure comes to light when we increase the accuracy of our measurements (that is, the number of iterations). And it is also clear that the extra lines in the range $0 \leq k \leq 1$ are *transient effects*. If we first carry out the iteration procedure for a while (say 50 iterations) without drawing points, and then continue to iterate while plotting the resulting points, the lines will disappear.

The above remarks are in complete agreement with our fundamental ideas in the simulation of dynamical systems. We are interested in the 'long-term behaviour' of a system under feedback (see Chapter 1). Program 2.2.-1 shows how easily we can modify our program `MeaslesGraphic`, in order to represent the long-term behaviour more clearly.

Program Fragment 2.2-1

```
(*  BEGIN:   Problem-specific procedures *)
FUNCTION f (p, k : real) : real;
BEGIN
   f := p + k * p * (1 - p);
END;
PROCEDURE FeigenbaumIteration;
   VAR
      range, i :integer;
      population, deltaxPerPixel : real;
BEGIN
   deltaxPerPixel := (Right - Left) / Xscreen;
   FOR range := 0 TO Xscreen DO
```

```
      BEGIN
         Feedback := Left + range*deltaxPerPixel;
         population := 0.3;
         FOR i := 0 to Invisible DO
            BEGIN
               population := f(population, Feedback);
            END
         FOR i := 0 TO Visible DO
            BEGIN
               SetUniversalPoint (Feedback, population);
               population := f(population, Feedback);
            END;
      END;
END;
(* END: Problem-specific procedures *)
(* ------------------------------------------------------*)
PROCEDURE Initialise;
BEGIN
   ReadReal ('Left            >', Left);
   ReadReal ('Right           >', Right);
   ReadReal ('Bottom          >', Bottom);
   ReadReal ('Top             >', Top);
   ReadInteger ('Invisible        >', Invisible);
   ReadInteger ('Visible          >'), Visible);
END;

PROCEDURE ComputeAndDisplay;
BEGIN
   EnterGraphics;
   FeigenbaumIteration;
   ExitGraphics;
END;
```

The new or modified parts of the program are shown in bold type. If we type in and run this progam then it gives a print-out as in Figure 2.2.-2. It shows for the 'interesting' range $k > 1.5$ a piece of the so-called *Feigenbaum diagram*.[1]

This program and picture will keep us busy for a while.

[1] *Translator's note*: This is more commonly known as a *bifurcation diagram*.

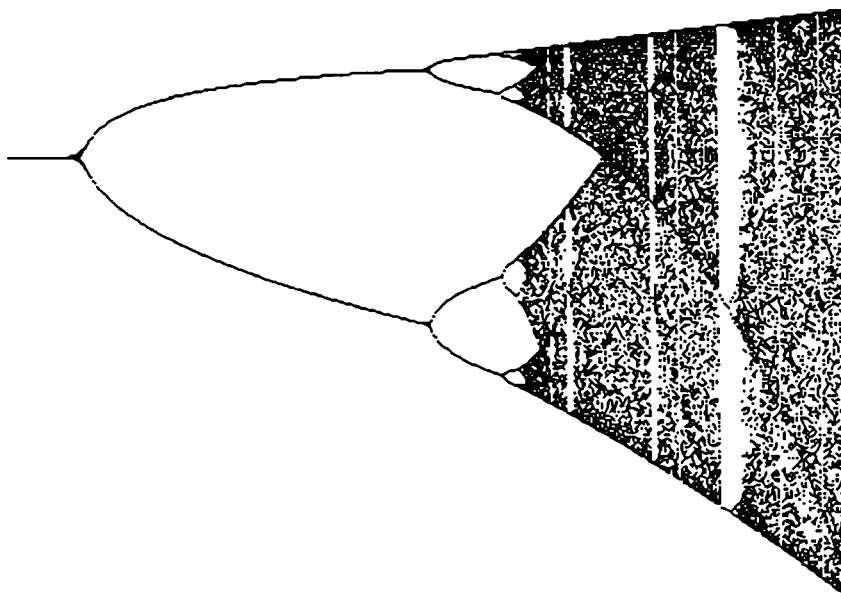


Figure 2.2-2 Print-out from Program 2.2-1.

- The name 'Feigenbaum' is in honour of the physicist Mitchell Feigenbaum, [2] who carried out the pioneering research described in this chapter. We shall call any picture like Figure 2.2-1 a *Feigenbaum diagram.*
- In the program fragment we introduce two new variables `Invisible` and `Visible`, which in the example are given the value 50.

The results show a certain independence of the initial value for p, provided we do not start with $p = 0$ or $p = 1$. You will probably have discovered that already. What interests us here is just the results of a large number of iterations. To stop the picture looking unsightly, the first 50 iterations run 'in the dark'. That is, we do not plot the results k,p. After that, a further 50 (or 100 or 200) iterations are made visible.

In order to facilitate comparison with Figure 2.2-1, you can set the variables in Program 2.2-1 as follows:

```
Invisible := 0; Visible := 10;
```

As regards the working of the program, the following remarks should be made:

Input data are read from the keyboard and assigned to the corresponding variables in the procedure `Initialise`. It is then easy to set up arbitary values from the keyboard. However, the program must then be initialised on each run. The type of input

[2] *Translator's note*: It is also German for ' fig-tree' , hence the section title.

procedure used depends on the purpose of the program. With keyboard input, typing errors are possible. Sometimes it is useful to insert fixed values into the program.

To draw Figure 2.2-2 on the screen on an 8-bit home computer takes about 5-10 minutes. With more efficient machines (Macintosh, IBM, VAX, SUN, etc.) it is quicker.

It is harder to describe these astonishing pictures than it is to produce them slowly on the screen. What for small k converges so regularly to the number 1, cannot continue to do so for larger values of k because of the increased growth-rate. The curve splits into two branches, then 4, then 8, 16, and so on. We have discovered this effect of 2, 4, or more branches (limiting values) by graphical iteration. This phenomenon is called a *period-doubling cascade*, (Peitgen and Richter 1986, p.7).

When $k > 2.570$ we see behaviour that can only be described by a new concept: *chaos*. There are unpredictable 'flashes' of points on the screen, and no visible regularity.

As we develop our investigations we will show that we have not blundered into chaos by accident. We have witnessed the crossing of a frontier. Up to the point $k = 2$ our mathematical world is still ordered. But if we work with the same formula and without rounding errors, for higher values of k it is virtually impossible to predict the outcome of the computation. A series of iterations beginning with the value $p = 0.1$, and one beginning with $p = 0.11$, can after a few iterations become completely independent, exhibiting totally different behaviour. A small change in the initial state can have unexpected consequences. 'Small cause, large effect': this statement moreover holds in a noticeably large region. For our Feigenbaum formula the value $k = 2.57$ divides 'order and chaos' from each other. On the right-hand-side of Figure 2.2-2 there is no order to be found. But this chaos is rather interesting - it contains structure!

Figure 2.2-2 appears to have been drawn by accident. As an example, let us consider the neighbourhood of the k-value 2.84. Here there is a region in which points are very densely packed. On the other hand, there are also places nearby with hardly any points at all. By looking carefully we can discover interesting structures, in which branching again plays a role.

In order to search for finer detail, we must 'magnify' the picture. On a computer this means that we want to display a *window*, or *section*, from the full picture 2.2-2 on the screen.[3] To do this we give suitable values to the variables `Right`, `Left`, `Bottom`, and `Top`. The program user can input values from the keyboard. In that way it is possible to change the window at will, to investigate interesting regions. If the picture is expanded a large amount in the y-direction it becomes very 'thin', because the majority of points lead outside the window. It then makes sense, by changing the variable `Visible`, to increase the total number of points plotted.

We now investigate the precise construction of the Feigenbaum diagram, with the aid of a new program. It is derived by a small modification of Program Fragment 2.2-1.

[3]In the choice of a window there is often a problem, to find out the values for the edges. As a simple aid we construct a transparent grid, which divides the screen into ten parts in each direction.

Program Fragment 2.2-2

```
.....
deltaxPerPixel := (Right - Left) / Xscreen;
FOR range := 0 to Xscreen DO
   BEGIN
      Feedback := Left + range * deltaxPerPixel;
      DisplayFeedback (Feedback);
      population := 0.3;
.....
```

Elsewhere we will introduce a procedure `DisplayFeedback`, and thereby enlarge our experimental possibilities. `DisplayFeedback` displays the current value of k in the lower left corner of the screen. It will be useful later, to establish more accurately the boundaries of interesting regions in the Feigenbaum diagram. To display text on the graphics screen some computers (such as the Apple II) require a special procedure. Other computers have the ability to display numbers of type `real` directly on the graphics screen, or to display text and graphics simultaneously in different windows.

The procedure `DisplayFeedback` can also be omitted if it is not desired to display numerical values on the screen. In this case `DisplayFeedback` must be deleted from the initial part of the main program, and also in the procedure `FeigenbaumIteration` which calls it.

When the program runs correctly, you should use it to draw sections of the Feigenbaum diagram. By choosing the boundaries of the windows suitably you can plot pictures whose fine detail can scarcely be distinguished in the full diagram. A tiny part of the picture can already contain the form of the whole. This astonishing property of the Feigenbaum diagram, containing itself, is called *self-similarity*. Look for yourself for further examples of self-similarity in the Feigenbaum diagram.

We should describe how the above program works in practice. Instructions appear on the screen for the input of the necessary data. The data are always input by using the <RETURN> key. The dialogue might, for example, go like this:

```
Start:     <hit RETURN key
Left         (>= 1.8)      >2.5
Right        (<= 3)        >2.8
Bottom       (>= 0)        >0.9
Top          (<= 1.5)      >1.4
Invisible    (>= 50)       >50
Visible      (>= 50)       >50
```

The picture that arises from this choice of input data is shown in Figure 2.2-3.

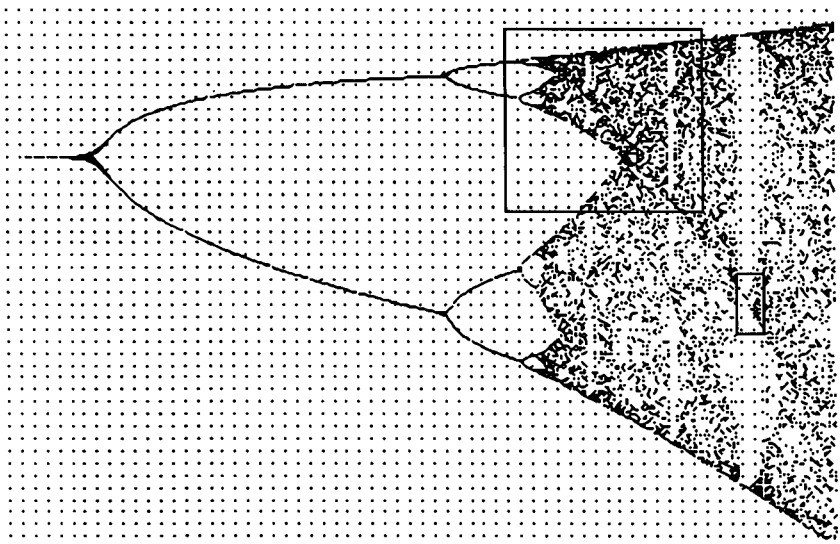


Figure 2.2-3 Section from the Feigenbaum diagram (see the following figures).

Figures 2.2-4 and 2.2-5 represent such sections from the Feigenbaum diagram, as drawn in Figure 2.2-3.

We also suggest that you try equations other than the Feigenbaum equation. Surprisingly, you will find that quite similar pictures appear! In many cases we find that the picture again begins with a line, and splits into 2, 4, 8,... twigs. There is also another common feature, which we do not wish to discuss further at this stage.

The stated values in Figures 2.2-4 and 2.2-5 are just examples of possible inputs. Try to find other interesting places for yourself.

Computer Graphics Experiments and Exercises for §2.2

Exercise 2.2-1

Implement Program 2.2-1 on your computer. Experiment with different values of the variables `Visible` and `Invisible`.

Exercise 2.2-2

Extend the program to include a procedure `DisplayFeedback`, which during the running of the program can 'measure' the k-values.

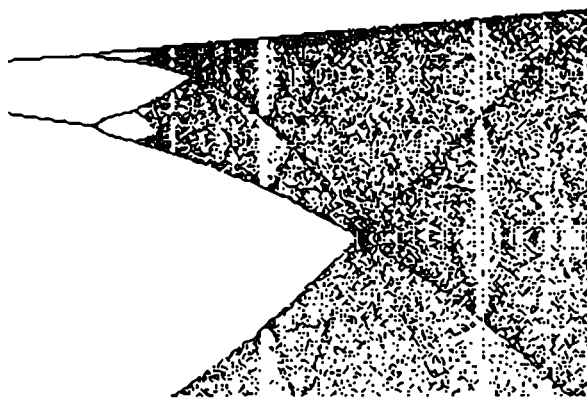


Figure 2.2–4 Fig-tree with data: 2.5, 2.8, 0.9, 1.4, 50, 100.

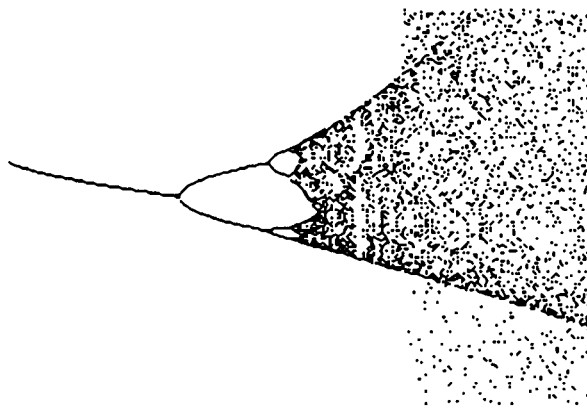


Figure 2.2–5 Fig-tree with data: 2.83, 2.87, 0.5, 0.8, 50, 100.

Exercise 2.2–3

Find regions in the Feigenbaum diagram around $k = 2.8$, where self-similarity can be found.

Exercise 2.2–4

Try to discover 'hidden structure', when you increase the iteration number in interesting regions. Think about taking small regions (and magnifying them).

Exercise 2.2-5

As regards the chaotic phenomena of Feigenbaum iteration, much more can be said.

'The initial value leads to unpredictable behaviour, but on average there is a definite result.' Test this hypothesis by displaying the average value of a large number of results as a graph, for k-values in the chaotic region.

See if you can confirm this hypothesis, or perhaps the contrary: 'Chaos is so fundamental that even the averages for k-values taken close together get spread out.'

Exercises 2.2-6

That after these explanations our investigation of the 'fig-tree' has not revealed all its secrets, is shown by the following consideration:

Why must the result of the function f always depend only on the previous p-value?

It is possible to imagine that the progenitors of this value 'have a say in the matter'. The value f_n for the nth iteration would then depend not only on f_{n-1}, but also on f_{n-2}, etc. It would be sensible if 'older generations' had somewhat less effect. In a program you can, for example, store the most recent value as `pn`, the previous one as `pnMinus1`, and the one before that as `pnMinus2`. The function f can then be viewed as follows. We give two examples in Pascal notation.

```
f(pn)   := pn + 1/2*k*(3*pn*(1-pn)-
                pnMinus1*(1-pnMinus1));
```

or

```
f(pn)   := pn +1/2*k*(3*pnMinus1-pnMinus2)*
                (1-3*pnMinus1-pnMinus2 );
```

To start, `pn`, `pnMinus1`, etc. should be given sensible values such as 0.3, and at each stage they should obviously be given their new values. The k-values must lie in a rather different range than previously. Try it out!

In the above print-out it goes without saying that other factors such as -1 and 3 and other summands are possible. The equations under consideration no longer have anything to do with the original 'measles' problem. They are not entirely unknown to mathematicians: they appear in a similar form in approximation methods for the solution of differential equations.

Exercise 2.2-7

In summary we might say that we always obtain a Feigenbaum diagram if the recursion equation is *nonlinear*. In other words, the underlying graph must be curved.

The diagrams appear especially unusual, if more generations of values are made visible. This gives rise to a new set of functions to investigate, for which we can change the series, in which we 'worry about the important bend in the curve' - which happens to be the term `expression* (1 - expression)` into which we substitute the previous value:

```
f(pn) := pn + 1/2*k*(3*pnMinus1*(1-pnMinus1)-
         pnMinus2*(1-pnMinus2));
```

Exercise 2.2-8

Investigate at which k_i-values branches occur.

2.2.1 Bifurcation Scenario - the Magic Number 'Delta'

The splittings in the Feigenbaum diagram, which by now you have seen repeatedly in your experiments, are called *bifurcations*. In the Feigenbaum diagram illustrated above, some points, the *branch points*, play a special role. We use the notation k_j for these: k_1, k_2, and so on. We can read off from the figures that $k_1 = 2$, $k_2 = 2.45$, and $k_3 = 2.544$. You can obtain these results with some effort from Exercise 2.2-8.

It was the great discovery of Mitchell Feigenbaum to have found a connection between these numbers. He realised that the sequence

$$\frac{k_n - k_{n-1}}{k_{n+1} - k_n}, \text{ for } n = 2, 3, \ldots$$

converges to a constant value δ (the Greek letter 'delta') when n tends to ∞. Its decimal expansion begins $\delta = 4.669 \ldots$.

We have formulated a series of interesting exercises about this number δ (Exercises 2.2.1-1ff. at the end of this section). They are particularly recommended if you enjoy number games and are interested in 'magic numbers'. Incidentally, you will then have shown that δ is a genuinely significant mathematical constant, which appears in several contexts. This same number arises in many different processes involving dynamical systems. For bifurcation problems it is as characteristic as the number π is for the area

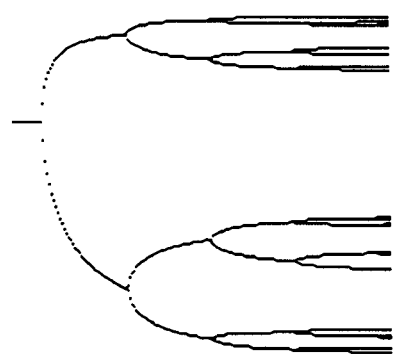


Figure 2.2.1-1 Logarithmic representation from $k = 1.6$ to $k = 2.569$.

and circumference of a circle. We call this number the *Feigenbaum number.* Mitchell Feigenbaum demonstrated its universality in many computer experiments. [4]

The higher symmetry and proportion that lies behind the above is especially significant if we do not choose a linear scale on the k-axis. Once the limiting value k_∞ of the sequence $k_1, k_2, k_3, \ldots$ is known, a logarithmic scale is preferable.

Computer Graphics Experiments and Exercises for §2.2.1

Exercise 2.2.1-1

The Feigenbaum constant δ has proved to be a natural constant, which occurs in situations other than that in which Feigenbaum first discovered it. Compute this natural constant as accurately as possible:

$$\delta = \lim_{n\to\infty} \frac{k_n - k_{n-1}}{k_{n+1} - k_n} .$$

In order to work out δ, the values k_i must be calculated as accurately as possible. Using Program Fragments 2.2-1 and 2.2-2 you can look at the interesting intervals of k and p, and pursue the branching of the lines. By repeatedly magnifying windows taken from the diagram you can compute the k-values.

Near the branch-points, convergence is very bad. It can happen that even after 100 iterations we cannot decide whether branching has taken place.

We should henceforth make tiny changes to the program

- to make the point being worked on flash, and
- to avoid choosing a fixed iteration number at the start.

It is easy to make a point flash by changing its colour repeatedly from black to white and back again.

We can change the iteration number by using a different loop construction. Instead of

```
FOR counter := 1 to Visible DO
```

we introduce a construction of the form

```
REPEAT UNTIL Button;
```
[5]

Exercise 2.2.1-2

Change the Feigenbaum program so that it uses a logarithmic scale for the k-axis instead of a linear one. Positions k should be replaced by $-\ln(k_\infty - k)$ measured from the right.

For the usual Feigenbaum diagram the limit k_∞ of the sequence $k_1, k_2, k_3, \ldots$ has the value 2.570. If, for example, we divide each decade (interval between successive

[4] The universality was proved mathematically by Pierre Collet, Jean-Pierre Eckmann, and Oscar Lanford (1980).

[5] In Turbo Pascal you must use `REPEAT UNTIL Keypressed;`

powers of 10) into 60 parts, and draw three decades on the screen, there will be a figure 180 points wide.

If you also expand the scale in the vertical direction, you will have a good measuring instrument to develop the computation of the k_i-values.

Exercise 2.2.1-3

With a lot of patience you can set loose the 'order within chaos' - investigate the region around $k = 2.84$. Convince yourself that δ has the same value as in the range $k < 2.57$.

Exercise 2.2.1-4

Develop a program to search for the k_i-values automatically, which works not graphically, but numerically. Bear in mind that numerical calculations in Pascal rapidly run into limitations. The internal binary representation for a floating-point number uses 23 bits, which corresponds to about 6 or 7 decimal places.

This restriction clearly did not put Feigenbaum off - he evaluated the aforementioned constant δ as 4.669 201 660 910 299 097 ... .

On some computers it is possible to represent numbers more accurately. Look it up in the manual.

Exercise 2.2.1-5

Feigenbaum's amazing constant arises not only when we follow the branching from left to right (small k-values to large ones). The 'bands of chaos', which are densely filled with points, also split when we go from large k-values to small ones. A single connected band splits into 2, then 4, then 8, ... Compute the k-values where this occurs.

Show that the constant δ appears here too.

2.2.2 Attractors and Frontiers

The mathematical equation which lies at the basis of our first experiment was formulated by Verhulst as early as 1845. He studied the growth of a group of animals, for which a restricted living space is available. In this interpretation it becomes clear what a value $p > 1$ means. $p = 100\%$ means that every animal has the optimum living space available. More than 100% corresponds to overpopulation. The simple calculations we have performed for the measles problem already show how the population then develops. For normal values of k the population is cut back until the value 1 is reached. However, the behaviour is different if we start with negative or large numbers. Even after many steps the population no longer manages to reach 1.

Mathematicians, like other scientists, habitually develop new ideas in order to attack new and interesting phenomena. This takes us a little way into the imposing framework of technical jargon. With clearly defined concepts it is possible to describe clearly

defined circumstances. We will now encounter one such concept.

In the absence of anything better, mathematicians have developed a concept to capture the behaviour of the numbers in the Feigenbaum scenario. The final value $p = 1$ is called an *attractor*, because it 'pulls the solutions of the equations' towards itself.

This can be clearly seen on the left-hand side of Figure 2.1.1-2. However many times we feed back the results p_n into the Feigenbaum equation, all the results tend towards the magic final value 1. The p-values are drawn towards the attractor 1. What you may perhaps have noticed already in your experiments is another attractor, $-\infty$ (minus infinity). At higher values ($k > 2$) of the feedback constant, the finite attractor is not just the value 1. Consider the picture of the Feigenbaum diagram: *the whole figure* is the attractor!

Each sequence of p-values which starts near the attractor invariably ends with a sequence of numbers that belong to the attractor, that is, the entire figure. An example will clarify this. In the program `MeaslesNumber` start with $p = 0.1$ and $k = 2.5$. After about 30 iterations the program stops. From the 20th iteration on we see these numbers over and over again: ... 1.2250, 0.5359, 1.1577, 0.7012, 1.2250, 0.5359, 1.1577, 0.7012, ... etc. It is certainly not easy to understand why this happens, but from the definition it is undeniable that these four successive values determine the attractor for $k = 2.5$. The attractor is thus the set of those function values which emerge after a sufficiently large number of iterations. A set like that illustrated in Figure 2.2-2 is called a *strange attractor*.

In the region $k > 3$ there is just the attractor $-\infty$.

Whenever a function has several attractors, new questions are raised:

- Which regions of the k,p-plane belong to which attractor? That is, with which value p must I start, so that I am certain to reach a given objective - such as landing on the attractor 1?
- With which values should I start, if I do not wish to end at $-\infty$?

Because each sequence of numbers is uniquely determined, this question has a unique answer. Thus the k,p-plane is divided into clearly distinguished regions, whose boundaries are of considerable interest.

For the Feigenbaum diagram this problem can be solved in a relatively simple and clear fashion. But other cases, which we will encounter later, lead to surprising results.

For the above function

$$f(p) = p+k*p*(1-p)$$

we can calculate the boundaries mathematically. Experimenting with the program `MeaslesNumber` should make it apparent that it is best to take negative p-values. Only then is there a chance of landing on the attractor.

This means that $f(p)$ must be > 0. From the equation (see Exercise 2.2.2-1 at the end of this section) this condition holds when

$$p < (k+1)/k.$$

Thus near $p = 0$ we have found the two boundaries for the catchment area, or *basin of*

attraction, of the strange attractor. We will see that these boundaries do not always take such a smooth and simple form. And they cannot always, as in the Feigenbaum diagram, be described by simple equations. The problem of the boundaries between attracting regions, and how to draw these boundaries, will concern us in the next chapter.

In Figure 2.2.2-1 we again show the Feigenbaum diagram for the first quadrant of the coordinate system. We have superimposed the basin of attraction of the attractor.

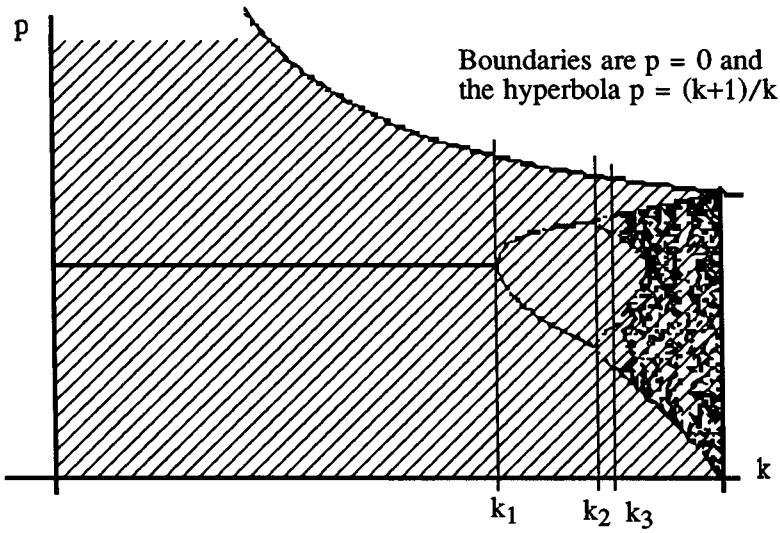


Figure 2.2.2-1 Basin of attraction for the Feigenbaum diagram.

If you are interested in how the attractor looks and what its boundaries are when k is less than 0, try Exercise 2.2.2-2 at the end of this section.

Computer Graphics Experiments and Exercises for §2.2.2

Exercise 2.2.2-1

Show that $p+k*p*(1-p) > 0$, $p \neq 0$, $k \neq 0$ implies that $p < (k+1)/k$.

Exercise 2.2.2-2

So far we have described all phenomena in the case $k > 0$. What happens for $k \leq 0$ the reader/experimentalist must determine. To that end, three types of problem must be analysed:

- In which k,p-regions do we find stable solutions (that is, solutions not tending to $-\infty$)?

- What form does the attractor have?
- Where are the boundaries of the basins of attraction?

Exercise 2.2.2-3

We obtain a further extension of the regions to be examined, and hence extra questions to be answered, if we work with a different equation from that of Verhulst. One possibility is that we simplify the previous formula for $f(p)$ to

$$f(p) = k*p*(p-1).$$

This is just the term that describes the change from one generation to the next in the Verhulst equation. Investigate, for this example in particular, the basins of attraction and the value of δ.

Exercise 2.2.2-4

With enough courage, you can now try other equations. The results cannot be anticipated in advance, but they tend to be startling. Examples which provide attractive pictures and interesting insights are:

- $f(p) = k*p*p*(1-p)$ in the region $4 \leq k \leq 7$,
- $f(p) = k*p*(1-p*p)$ and other powers,
- $f(p) = k*\sin(p)*\cos(p)$ or square (cube, nth) root functions,
- $f(p) = k*(1 - 2*|\, p-0.5\,|)$ where $|\, x\,|$ means the absolute value of x.

2.2.3 Feigenbaum Landscapes

Even the simple fig-tree poses several puzzles. In the 'chaotic regime' we can see zones where points lie more thickly than in others. You can get a nice overview by representing the frequency with which the p-values fall inside a given interval. By putting the results together for different k-values, you will get a *Feigenbaum landscape* (Figures 2.2.3-1 and 2.2.3-2).

These Feigenbaum landscapes can be made to reveal further interesting structure. We suggest you experiment for yourself. It is naturally best if you develop the program yourself too, or at least try it out with your own parameter values. To help you in this task, we now provide some tips for the development of a Feigenbaum landscape program.

The appropriate range of values from 0 to 1.4 for $f(p)$ must be divided into a certain number of intervals. This number of course depends on the size of the screen display, which we have set using the constants `Xscreen` and `Yscreen`. In the program, for example, we have 280 'boxes', one for each interval.

For a given k-value the Feigenbaum program begins as usual. When a value of $f(p)$ falls within a given interval, this is noted in the corresponding box. After a sufficiently large number of iterations we stop the computation. Finally the results are displayed.

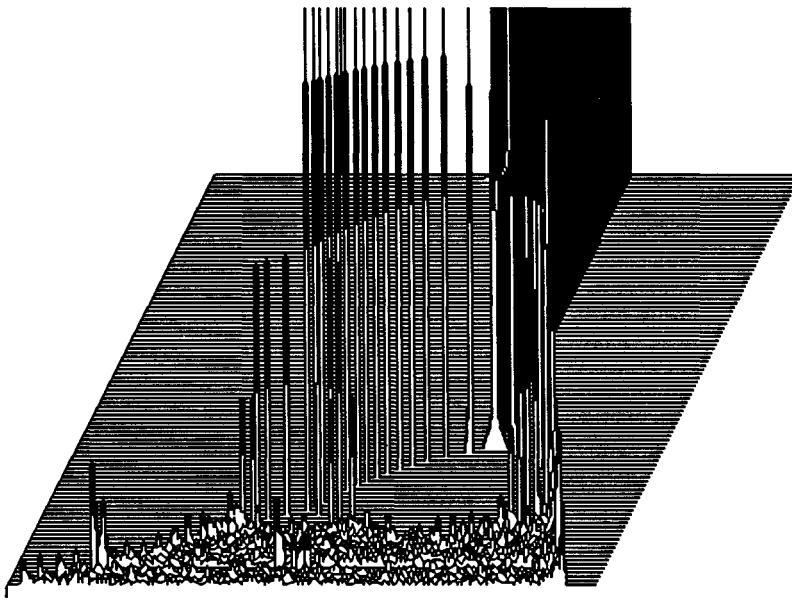


Figure 2.2.3-1 Feigenbaum landscape with the data 0, 3, 0, 1.4, 50, 500.

The box-number is counted from the right and the contents are drawn upwards, joining neighbouring values by a line. The result is a curve resembling a mountain range, which describes the distribution of p-values for a given k-value.

To draw a picture with several k-values, we combine several such curves in one picture. Each successive curve is displaced two pixels upwards and one pixel to the right. In this way we obtain a 'pseudo-three-dimensional' effect. Of course the horizontal displacement can be to the left instead. In Figure 2.2.3-3 ten such curves are drawn.

To improve the visibility, the individual curves must be combined into a unified picture. To achieve this, we do not draw the curves straight away. Instead, for each horizontal screen coordinate (x-axis) we record in another field (in Pascal it is an *array*, just like the 'boxes') whichever of the previous vertical y-coordinates has the largest value. Only this maximal value is actually drawn.

With these hints you should be in a position to develop the program yourself.

A solution is of course given in §11.3.

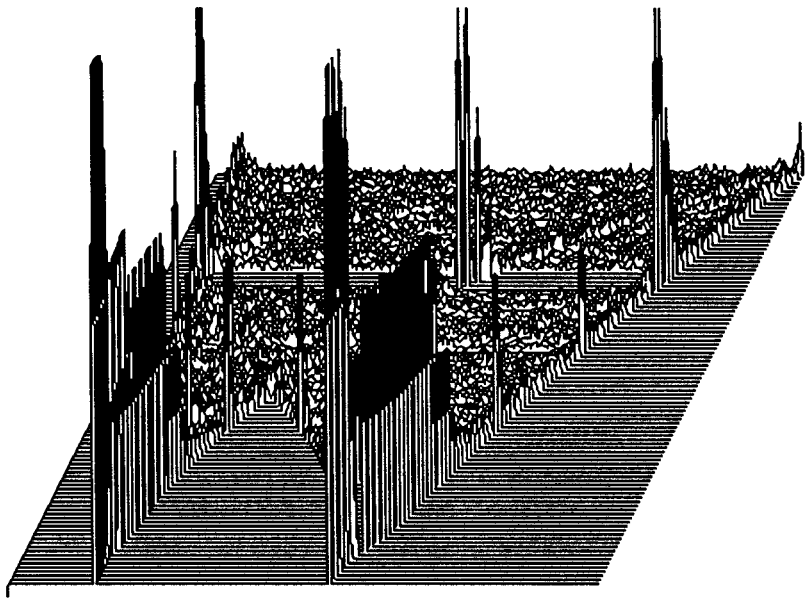


Figure 2.2.3-2 Feigenbaum landscape with the data 3, 2.4, 1.4, 0, 50, 500.

Computer Graphics Experiments and Exercises for §2.2.3

Exercise 2.2.3-1

Develop a program to draw Feigenbaum landscapes. Use the resulting 'three-dimensional measuring instrument' to investigate interesting sections of the Feigenbaum diagram. We have already given hints for the main steps above.

Exercise 2.2.3-2

Generalise the pseudo-three-dimensional landscape method, so that other formulas can be represented in the same way.

2.3 Chaos - Two Sides to the Same Coin

In the previous chapter you were confronted with many new concepts. Furthermore, your own experiments will certainly have given you more to think about, so that the basic idea of the first chapter may have been somewhat obscured. We will therefore discuss the initial consequences of our investigations, before we embark on new adventures. What have we discovered?

- The interesting cases are those in which the results of our computations do not tend to ∞ or $-\infty$.
- The set of all results that can be obtained after sufficiently many iterations is called an *attractor*.
- The graphical representation of attractors leads to pictures, which contain smaller copies of themselves.
- Three new concepts - *attractor*, *frontier*, and *bifurcation* - are connected with these mathematical features.

We began with the central idea of 'experimental mathematics':

- These important concepts in the theory of dynamical systems are based on taking an arbitrary mathematical equation and 'feeding it back' its own results again and again.

By choosing different starting values we repeatedly find the same results upon iteration. With the same initial values we always obtain the same results. There are however some deep and remarkable facts to be observed:

In the Feigenbaum diagram we can distinguish three regions:

- $k \leq 2$ (*Order*);
- $2 < k < 2.57$ (*Period doubling cascade*: $0 \leq p \leq 1.5$);
- $k \geq 2.57$ (*Chaos*).

Under certain conditions, moreover, we cannot predict what will happen at all. Insignificant differences in the initial value lead to totally different behaviour, giving virtually unpredictable results. This 'breakdown of computability' happens around $k = 2.57$. This is the 'point of no return', dividing the region of order from that of chaos.

To avoid misunderstandings, we must again emphasise that the above remarks refer to a completely deterministic system. But - from a practical point of view - the chaos effect produces the bitter aftertaste of indeterminacy.

Mathematicians try to find models that can describe the 'long-term behaviour' of a system. The Feigenbaum scenario exemplifies the behaviour of the simplest nonlinear system. The message is that any nonlinear system may exhibit similar phenomena. Complex systems, depending on many parameters, can under certain conditions switch from stable conditions to instability. We speak of chaos.

Of course we want to keep on the track of this essentially philosophical question. It seems that there is some deep connection between order and chaos, which we cannot yet makeexplicit.

One thing is certain.

As a result of our previous investigations, order and chaos are two sides to the same coin - a parameter-sensitive classification.

Enough theory!

In the following chapter we will return to the question. But now you must try out some exercises for yourself. Good luck with your experiments!