Lesson 7: Iteration and Stability

```
> restart;
  with(plots):
```

Iteration

We were looking at the logistic map, with various values of the parameter r.

```
> g:= x -> r*(x - x^2);
```

$$g := x \to r\left(x - x^2\right) \qquad \textbf{(1.1)}$$

```
> r:= 2.5;
```

$$r := 2.5 \qquad \textbf{(1.2)}$$

Iteration starts out with a particular x value x_0 and repeatedly applies the function g to it, so we get a sequence x_0, $x_1 = g(x_0)$, $x_2 = g(x_1)$, ...

Fixed points, attractors and repellers

If the sequence has a limit, that limit must be a **fixed point** of g: a value p such that $g(p) = p$. But not all fixed points are easy to attain this way. The ones that are, are **attractors**.

• A fixed point p of g is **stable** if for every $\varepsilon > 0$ there is $\delta > 0$ such that whenever $|x_0 - p| < \delta$, all $|x_n - p| < \varepsilon$. That is, we can guarantee that x_n always stays close to p by taking the starting point x_0 sufficiently close to p.

• A fixed point that is not stable is **unstable**.

• A stable fixed point p is an **attractor** if $x_n \to p$ as $n \to \infty$ whenever x_0 is sufficiently close to p.

• An unstable fixed point p is a **repeller** if there exist positive numbers δ and ε such that, whenever $0 < |x_0 - p| < \delta$, there is some n for which $|x_n - p| > \varepsilon$. That is, whenever you start close enough to p (but not exactly at p), eventually you move a certain distance away from p.

The main way to tell whether a fixed point p is an attractor or a repeller is by looking at $g'(p)$.

Theorem:

• A fixed point p of a differentiable function g is an attractor if $|g'(p)| < 1$.

• It is a repeller if $|g'(p)| > 1$.

• If $|g'(p)| = 1$, it could be an attractor or a repeller or neither; we need more information to decide which.

In our example:

```
> D(g)(0);
```

$$2.5 \qquad \textbf{(2.1)}$$

```
> D(g)(0.6);
```

-0.50 (2.2)

This confirms that 0 is a repeller and 0.6 is an attractor.

Cobwebs

A **cobweb diagram** (or **staircase diagram**) is a way of visualizing what's going on here. We start with the graphs of $y = g(x)$ and $y = x$, which intersect at the fixed points. Then, given any x_0 and n, I plot lines joining the points $[x_0, x_0]$, $[x_0, x_1]$, $[x_1, x_1]$, $[x_1, x_2]$, ..., $[x_{n-1}, x_n]$.
Each step of this staircase goes from a point $[x, x]$ on the diagonal $y = x$ vertically to $[x, g(x)]$ on the curve $y = g(x)$, then horizontally to $[g(x), g(x)]$ on the diagonal.
I'll also plot the first point of the staircase in green and the last point in red, so you can tell which direction it's going.
The basic building block is a function I'll call **stair** that takes x and produces the sequence of points $[x, x]$, $[x, g(x)]$.

```
> stair:= x -> ([x,x],[x,g(x)]);
```

$stair := x \rightarrow ([x, x], [x, g(x)])$ (3.1)

Here's the main function **staircase** for making the picture.

```
> staircase:= proc(x0, a, b, n)
    local x, count, Curves, Staircase,Dots;
    uses plots;
    Curves:= plot([x,g(x)],x=a..b, colour=[red,blue]);
    x[0]:= x0;
    for count from 1 to n-1 do
      x[count]:= g(x[count-1])
    end do;
    Staircase:= plot([seq(stair(x[j]),j=0..n-1)],colour=black);
    Dots:= plot([[x[0],x[0]]],style=point,symbol=solidcircle,
  colour=green),
           plot([[x[n-1],g(x[n-1])]],style=point,symbol=
  solidcircle,colour=red);
    display([Curves, Staircase,Dots]);
  end proc;
```

staircase := **proc**(*x0*, *a*, *b*, *n*) (3.2)

local *x*, *count*, *Curves*, *Staircase*, *Dots*;

Curves := *plot*([*x*, *g*(*x*)], *x* = *a* ..*b*, *colour* = [*red*, *blue*]);

x[0] := *x0*;

for *count* **to** *n* − 1 **do** *x*[*count*] := *g*(*x*[*count* − 1]) **end do**;

Staircase := *plot*([*seq*(*stair*(*x*[*j*]), *j* = 0 ..*n* − 1)], *colour* = *black*);

Dots := *plot*([[*x*[0], *x*[0]]], *style* = *point*, *symbol* = *solidcircle*, *colour* = *green*), *plot*([[*x*

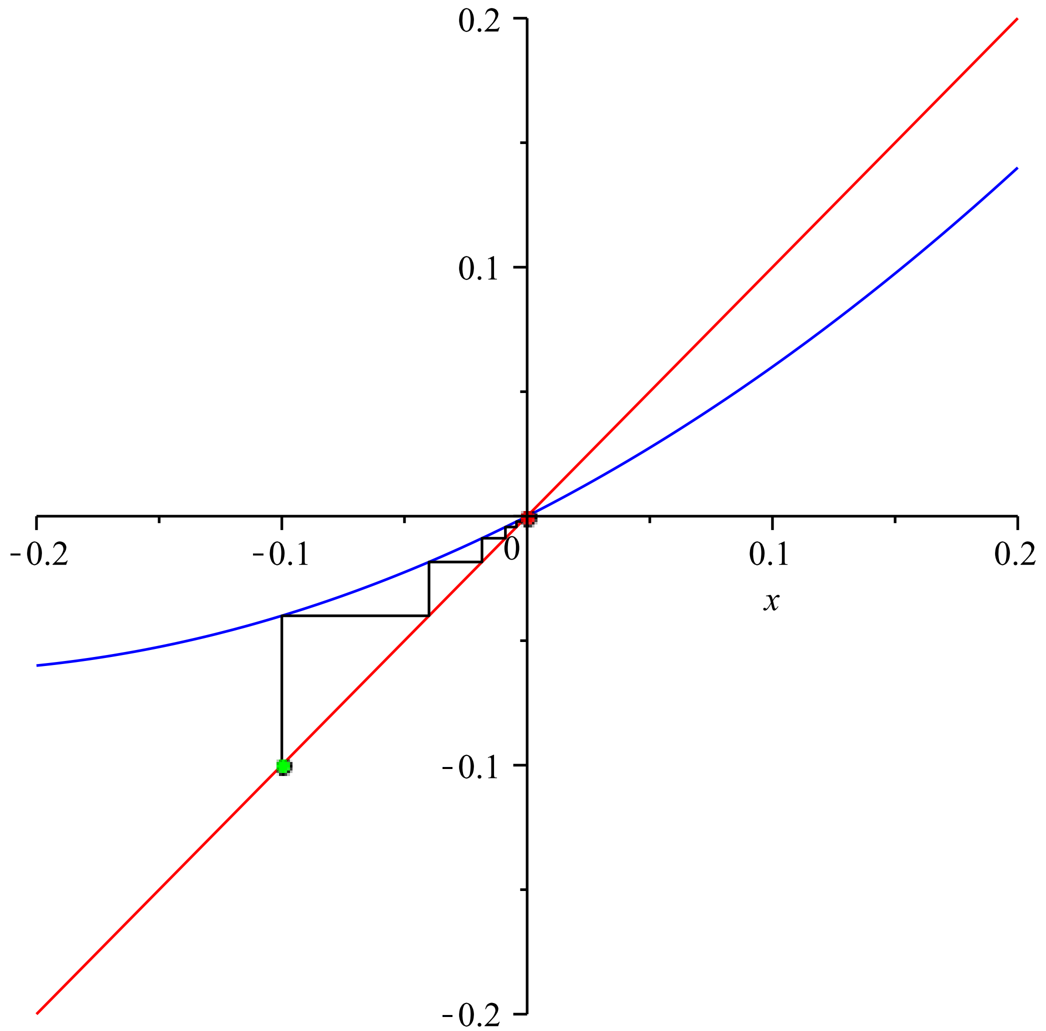
```
    [n − 1], g(x[n − 1]) ]], style = point, symbol = solidcircle, colour = red);
    plots:-display([Curves, Staircase, Dots])
end proc
```

- The procedure definition starts with **proc**. Previously we used **->** to define functions, but when there will be more than one Maple statement in the procedure we need to use **proc**.
- After **proc** come the names of the inputs to the procedure, within parentheses. These will be **x0** (the starting value), **a** and **b** (the endpoints of the interval on which we'll plot the curves) and **n**.
- The next line declares local variables **x**, **count**, **Curves**, **Staircase**: these will belong to the procedure, and will not interfere with variables of the same name outside the procedure.
- The **uses** statement in the next line is a replacement for **with**, which doesn't work well inside procedures.
- The first statement in the main body of the procedure plots the curves $y = x$ and $y = g(x)$ for x from a to b.
- Next we start with $x_0 = x0$ and compute x_1 to x_{n-1} using a **for** loop.
- We use those values to plot the "staircase".
- We use **plot** with **style=point** and **symbol=solidcircle** to plot the first and last points in green and red.
- Finally, we use **display** to combine the plots of the curves, the staircase and the dots.
- The result of the procedure will be the result of its last statement (the **display** command).
- The procedure definition is ended with **end proc**.

If you start near an attractor p where $0 < g'(p) < 1$, the staircase looks like this, with a staircase approaching the attractor. In this case $p = 0$.

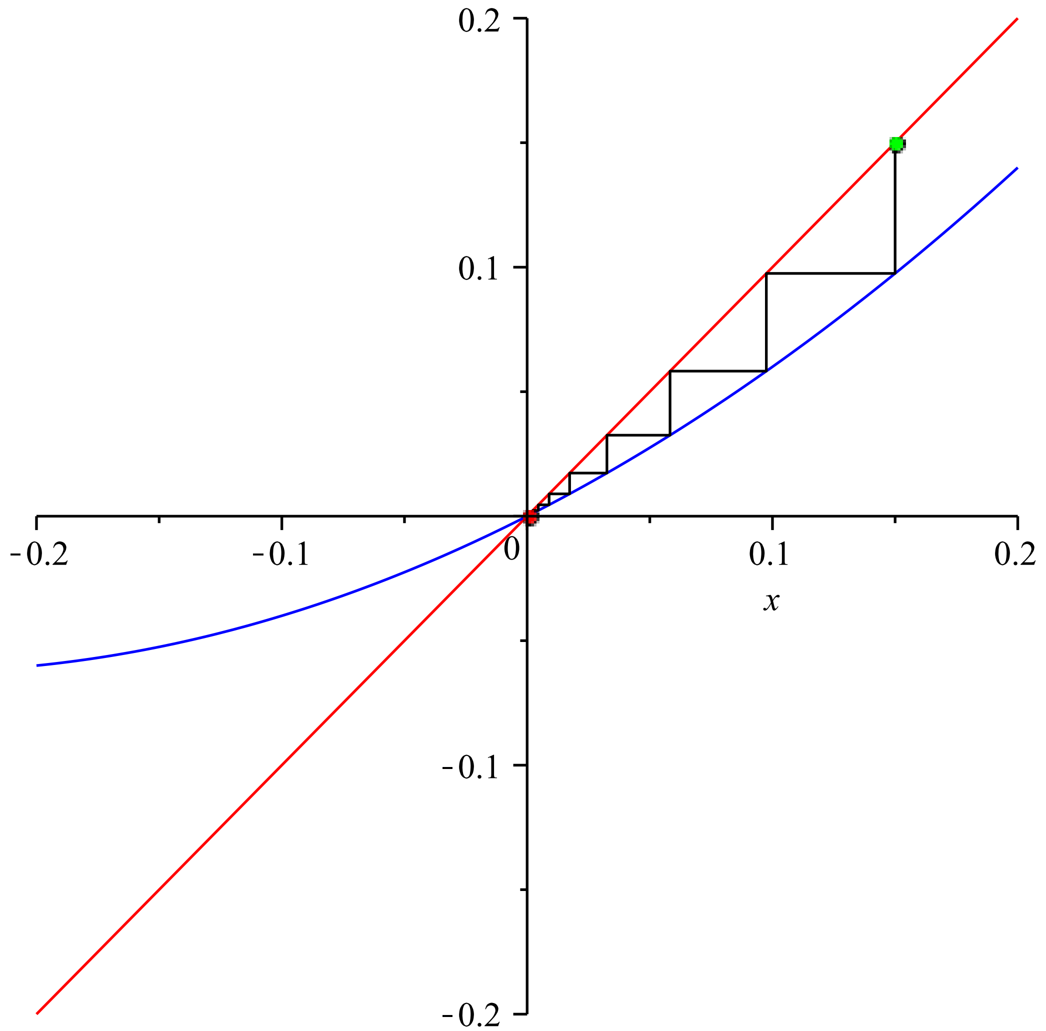
```
> g:= x -> x^2 + x/2;
  staircase(-0.1, -0.2, 0.2, 10);
```

$$g := x \rightarrow x^2 + \frac{1}{2}\,x$$



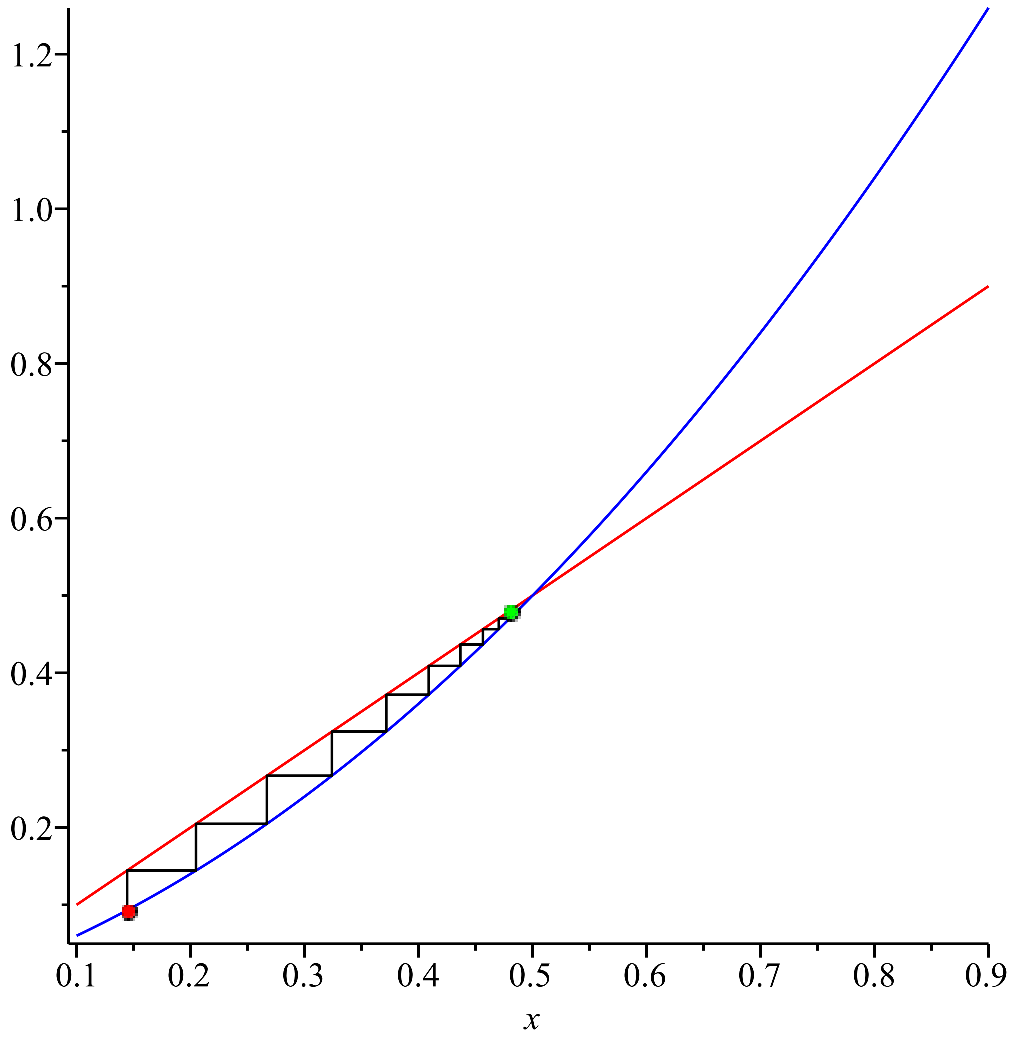
Or starting on the right side of the attractor:

```
> staircase(0.15,-0.2,0.2,10);
```



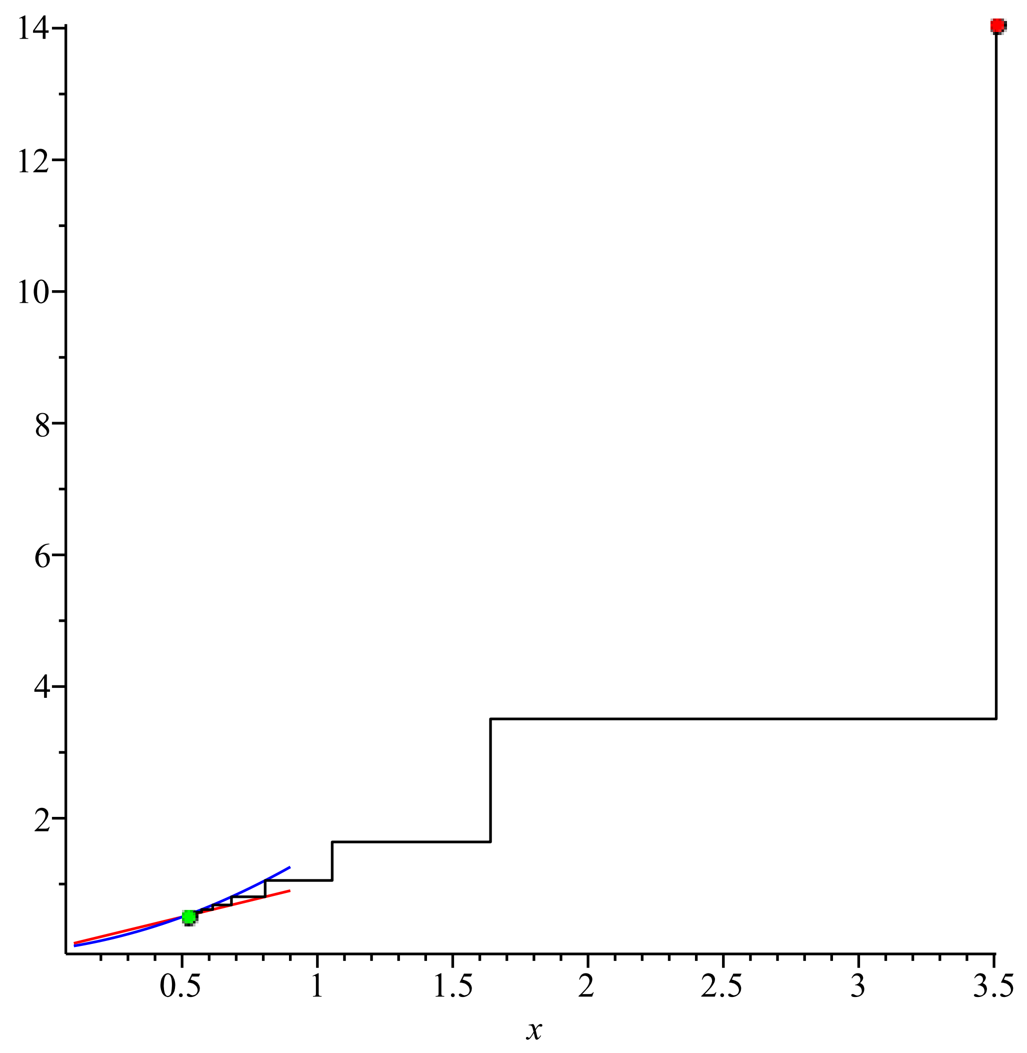
If you start near a repeller with $g'(p) > 1$, it looks like this, with the staircase moving away from the repeller. In this case $p = 0.5$.

```
> staircase(0.48, 0.1, 0.9, 10);
```



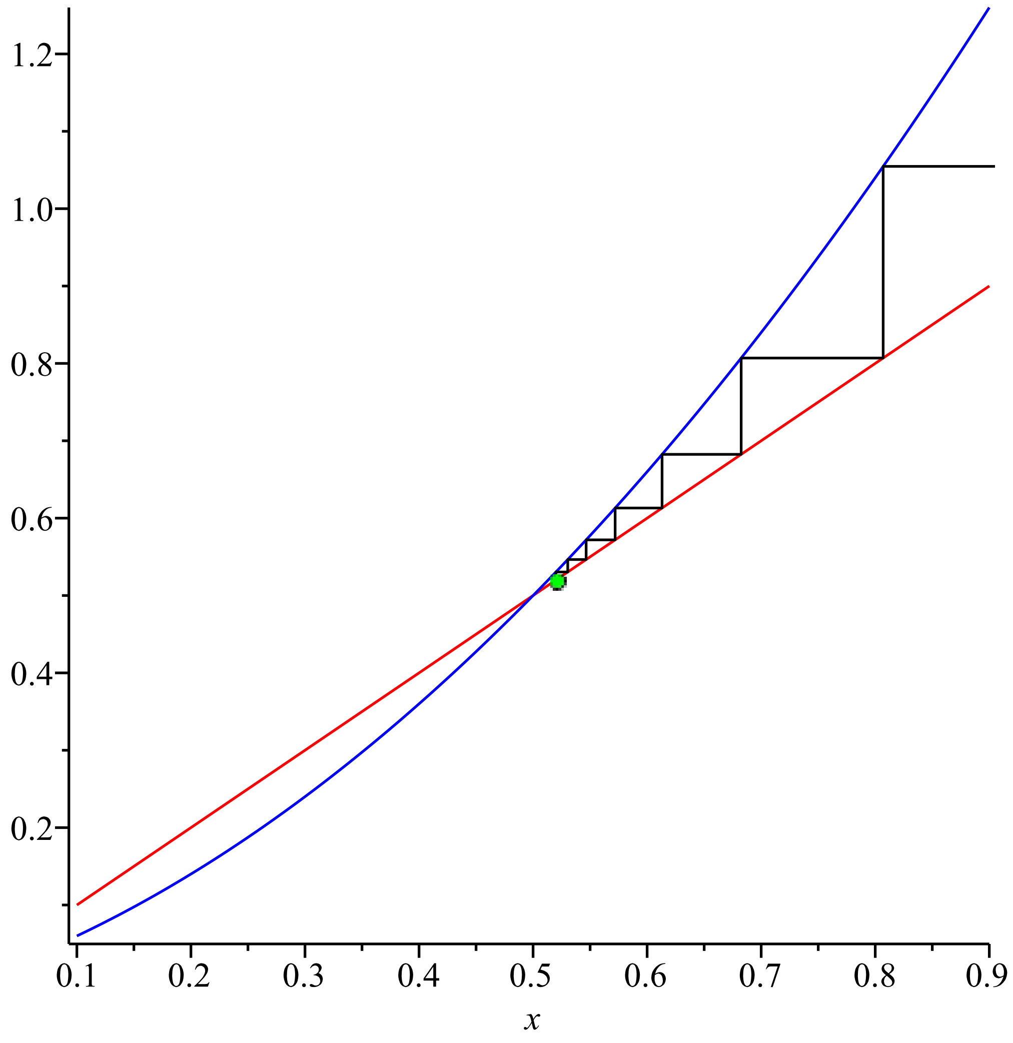
Or on the other side:

```
> staircase(0.52, 0.1, 0.9, 10);
```



Oops, the staircase went too far away. We might have used the **view** option in the **display** command to prevent this.

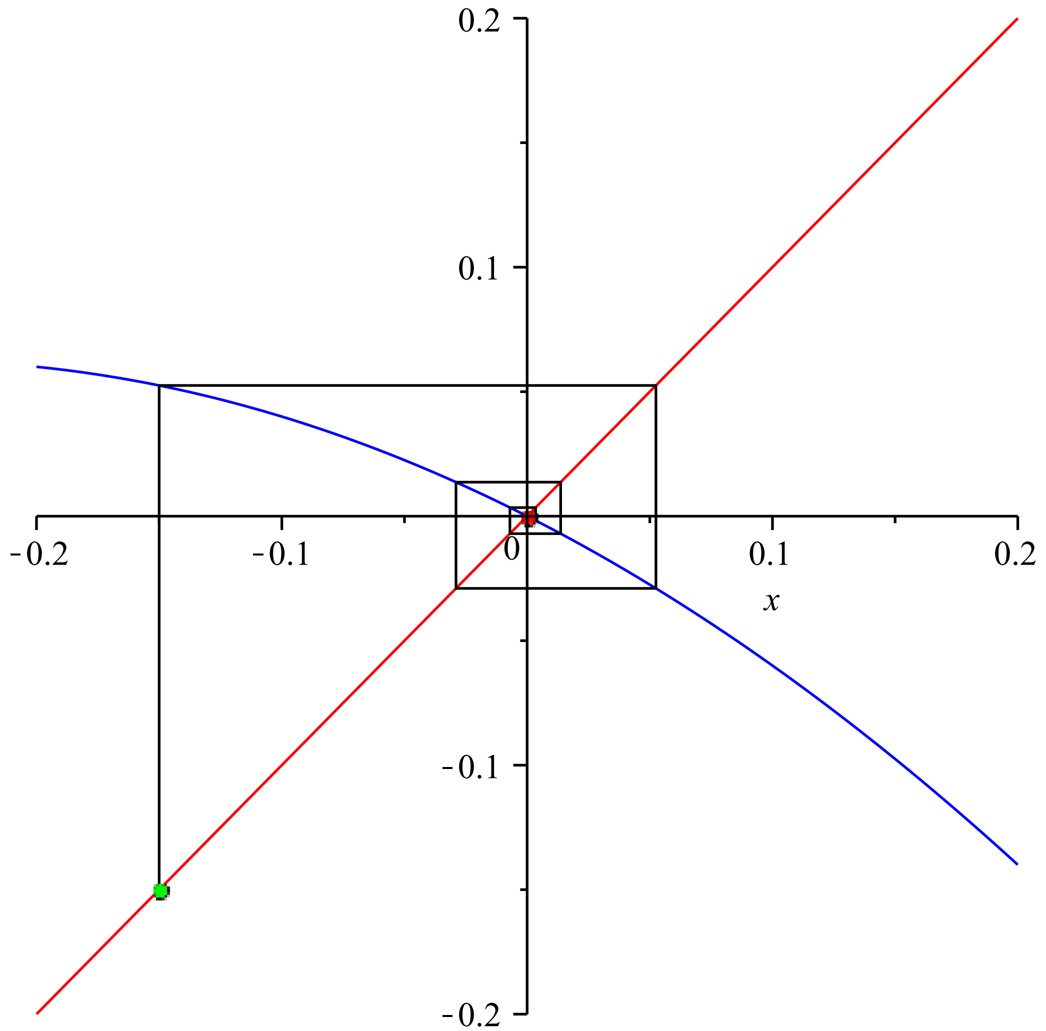
```
> display(%, view=[0.1 .. 0.9, g(0.1) .. g(0.9)]);
```



In the case of an attractor with $-1 < g'(p) < 0$, the staircase becomes a spiral (this is where the term "cobweb" comes in).

```
> g:= x -> -x^2 - x/2;
  staircase(-0.15, -0.2, 0.2, 10);
```

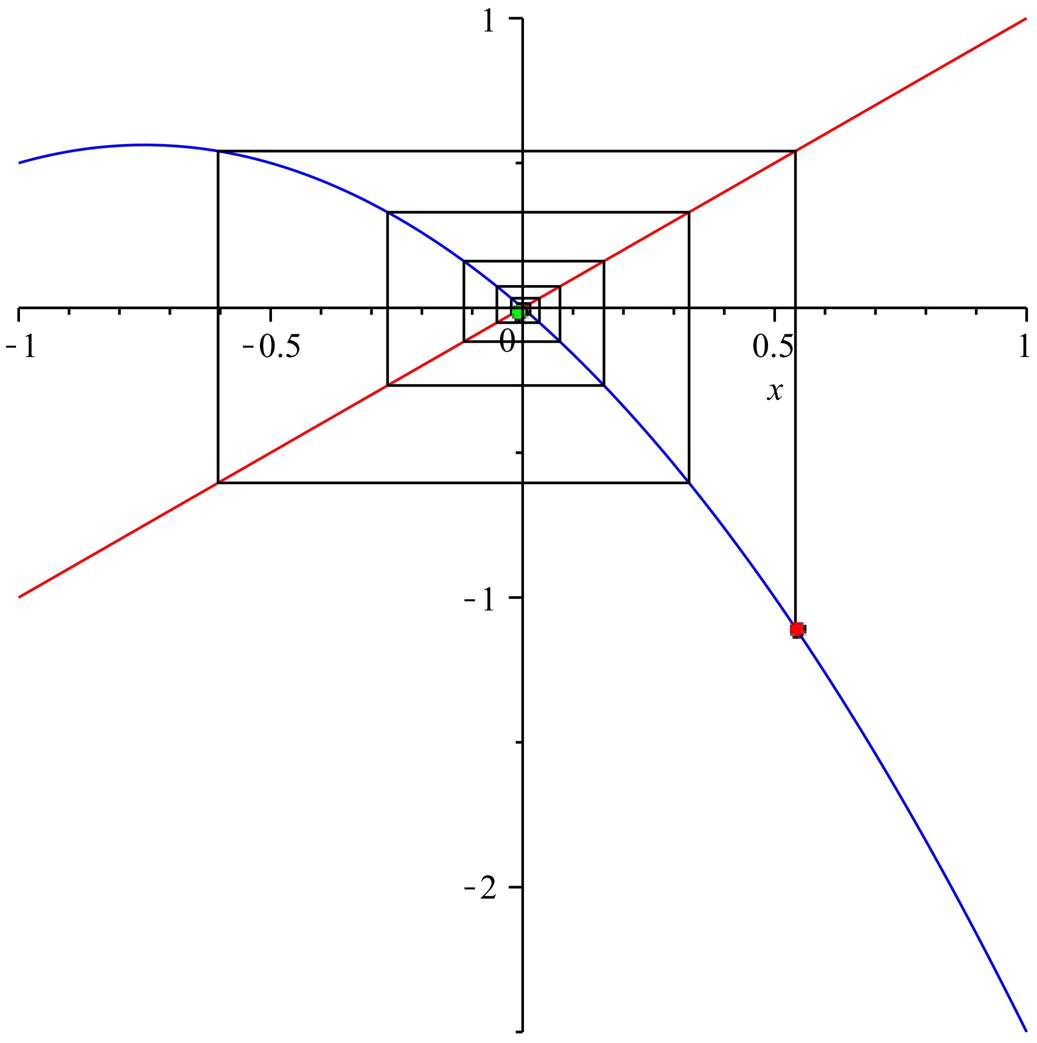
$$g := x \rightarrow -x^2 - \frac{1}{2}\,x$$



Similarly for a repeller with $g'(p) < -1$, it spirals outwards.

```
> g:= x -> -x^2 - 3/2*x;
  staircase(-.01,-1,1,12);
```

$$g := x \rightarrow -x^2 - \frac{3}{2}\,x$$

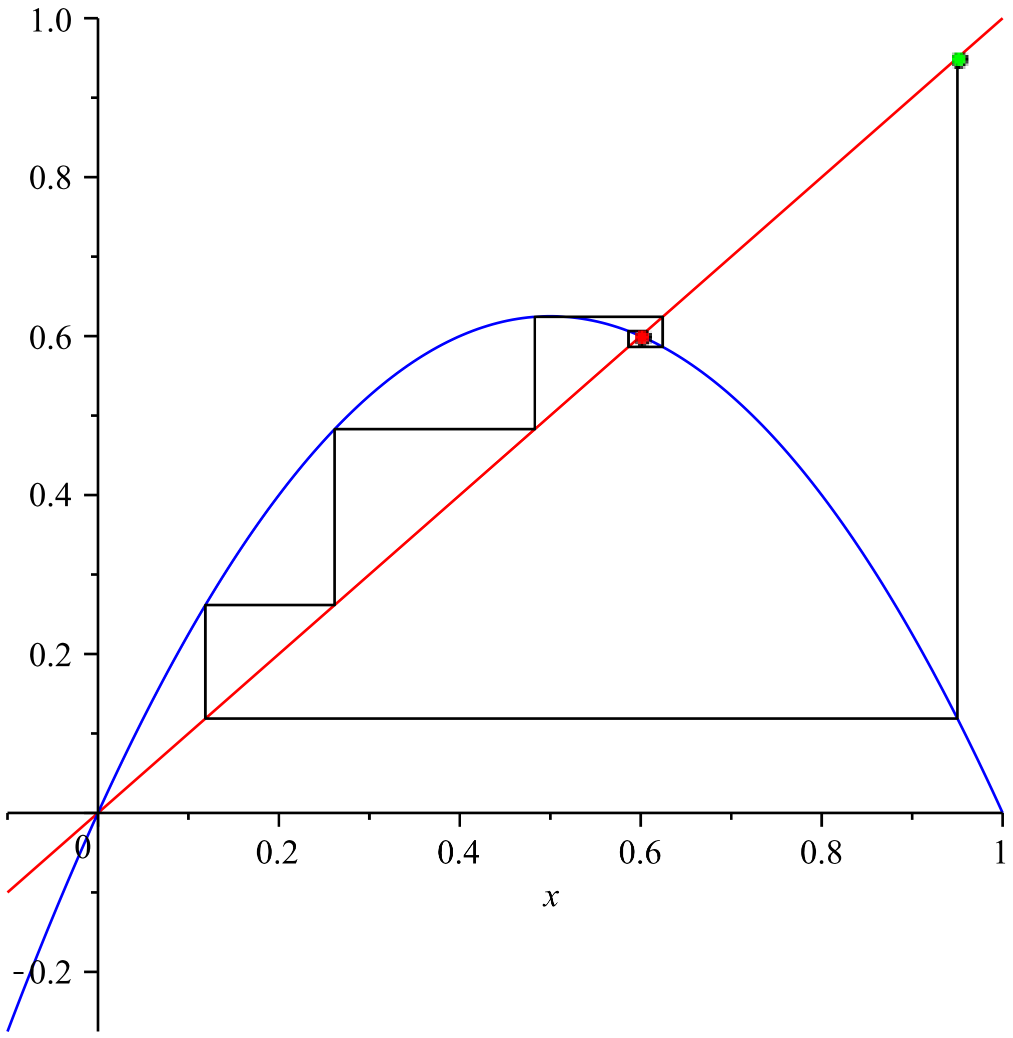


Back to our g: we can get an idea of why every initial point in the interval $(0, 1)$ is attracted to the fixed point 0.6.

```
> g:= x -> r*(x - x^2);
  r:= 2.5;
  staircase(0.95, -0.1, 1, 20);
```

$$g := x \rightarrow r\left(x - x^2\right)$$

$$r := 2.5$$



An attracting 2-cycle

Next we tried another value of the parameter *r*.

```
> r := 3.2;
```

$r := 3.2$ **(4.1)**

```
> solve(g(x)=x);
```

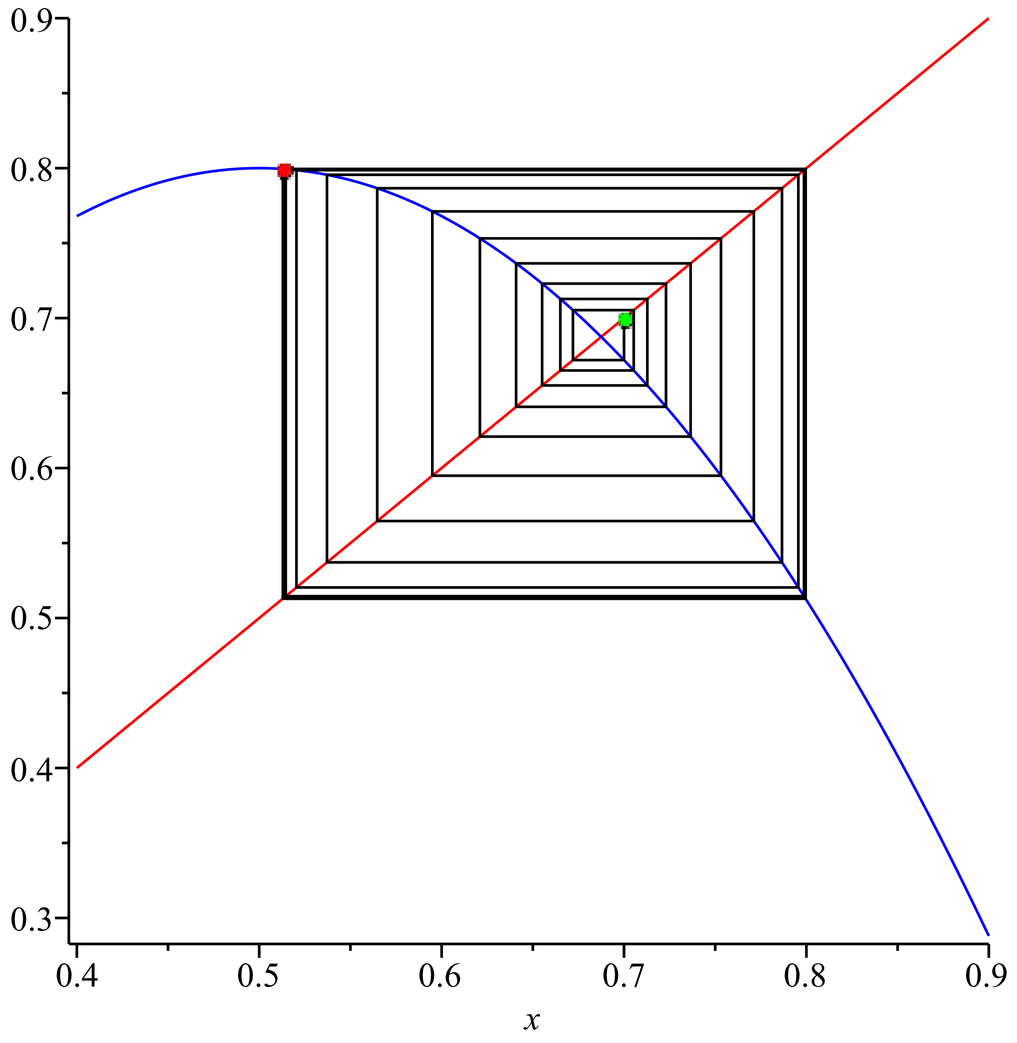
0., 0.6875000000 **(4.2)**

```
> D(g)(0), D(g)(0.6875);
```

3.2, −1.20000 **(4.3)**

This time both fixed points 0 and 0.6875 are repellers. So what will happen when we iterate? I'll start near 0.6875 and draw the cobweb diagram with 50 steps.

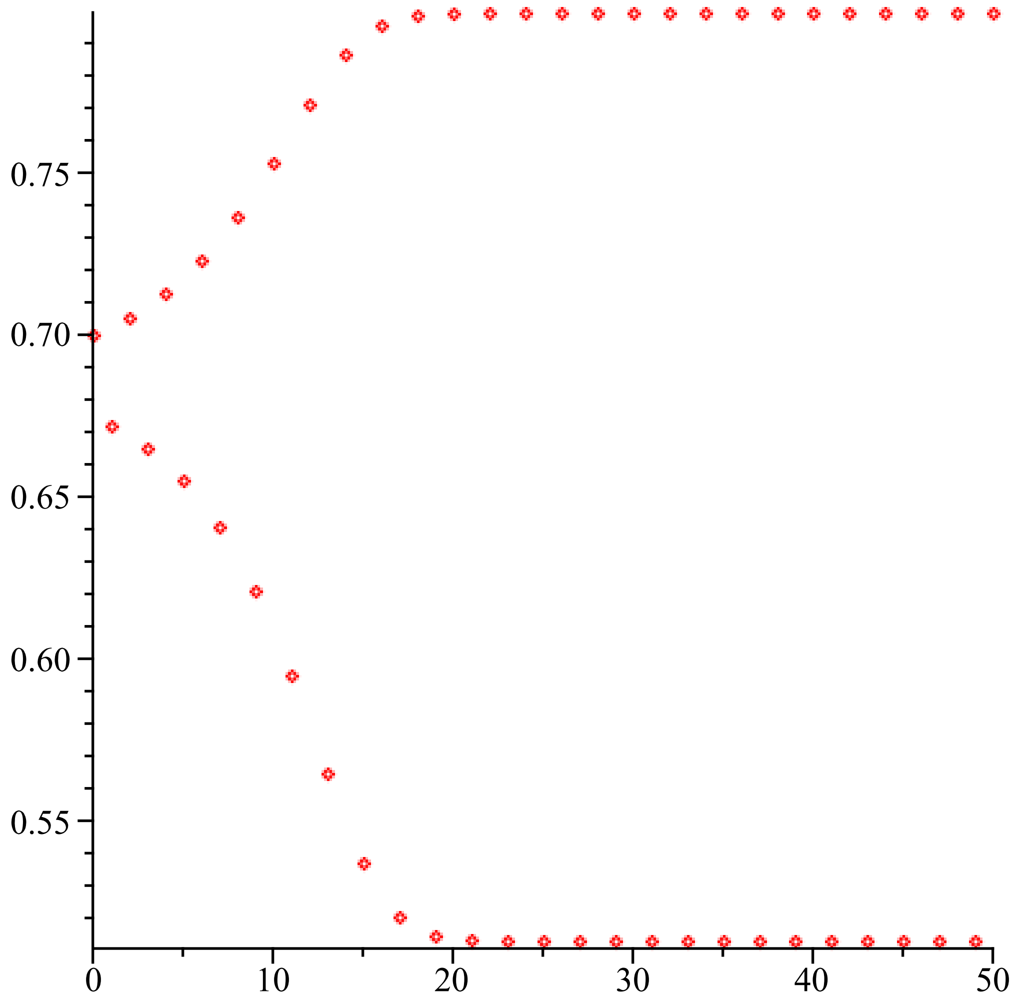
```
> staircase(0.7,0.4,0.9,50);
```



The cobweb seems to be approaching a rectangle. The x values at the two sides of the rectangle, say b and c, have the property that $g(b) = c$ and $g(c) = b$. This constitutes a **cycle** of period 2. If you started the iteration at one of the points of the cycle, x_n would alternate between these forever: say $x_0 = b$, $x_1 = c$, $x_3 = b$, $x_4 = c$, etc.

We also tried plotting the points $[i, x_i]$

```
> X[0]:= 0.7:
  for count from 1 to 50 do
    X[count] := g(X[count-1])
  end do:
  plot([seq([i,X[i]],i=0..50)], style=point);
```



To find b and c, we could solve the two equations $g(b) = c$ and $g(c) = b$ for b and c:

```
> solve({g(b)=c, g(c)=b},{b,c});
```

$\{b = 0., c = 0.\}, \{b = 0.6875000000, c = 0.6875000000\}, \{b = 0.5130445095, c = 0.7994554905\}, \{b = 0.7994554905, c = 0.5130445095\}$ **(4.4)**

Notice that two of the solutions have b and c both equal to a fixed point, the other two have b and c interchanged.
Another way to do it would be to solve the one equation $g(g(x)) = x$.

```
> s:= fsolve(g(g(x))=x);
```

$s := 0., 0.5130445095, 0.6875000000, 0.7994554905$ **(4.5)**

The two fixed points and the points b and c are all fixed points of the composition of the function with itself, which in mathematics might be written as $g \circ g$, or in Maple input as `g @ g`.

```
> b := s[2]; c:= s[4];
```

$b := 0.5130445095$

$c := 0.7994554905$ **(4.6)**

As fixed points of $g \circ g$, are they attractors or repellers?

```
> D(g@g)(b), D(g@g)(c);
```

$0.1599999996, 0.1599999996$ **(4.7)**

The absolute values are less than 1, so these are attractors.
Is it a coincidence that those two derivatives are the same? Think of the chain rule (or ask Maple):

```
> D(G @ G)(x);
```

$$\mathrm{D}(G)(G(x))\ \mathrm{D}(G)(x) \qquad (4.8)$$

```
> eval(%, {x = B, G(x) = C});
```

$$\mathrm{D}(G)(C)\ \mathrm{D}(G)(B) \qquad (4.9)$$

```
> eval(%%, {x = C, G(x) = B});
```

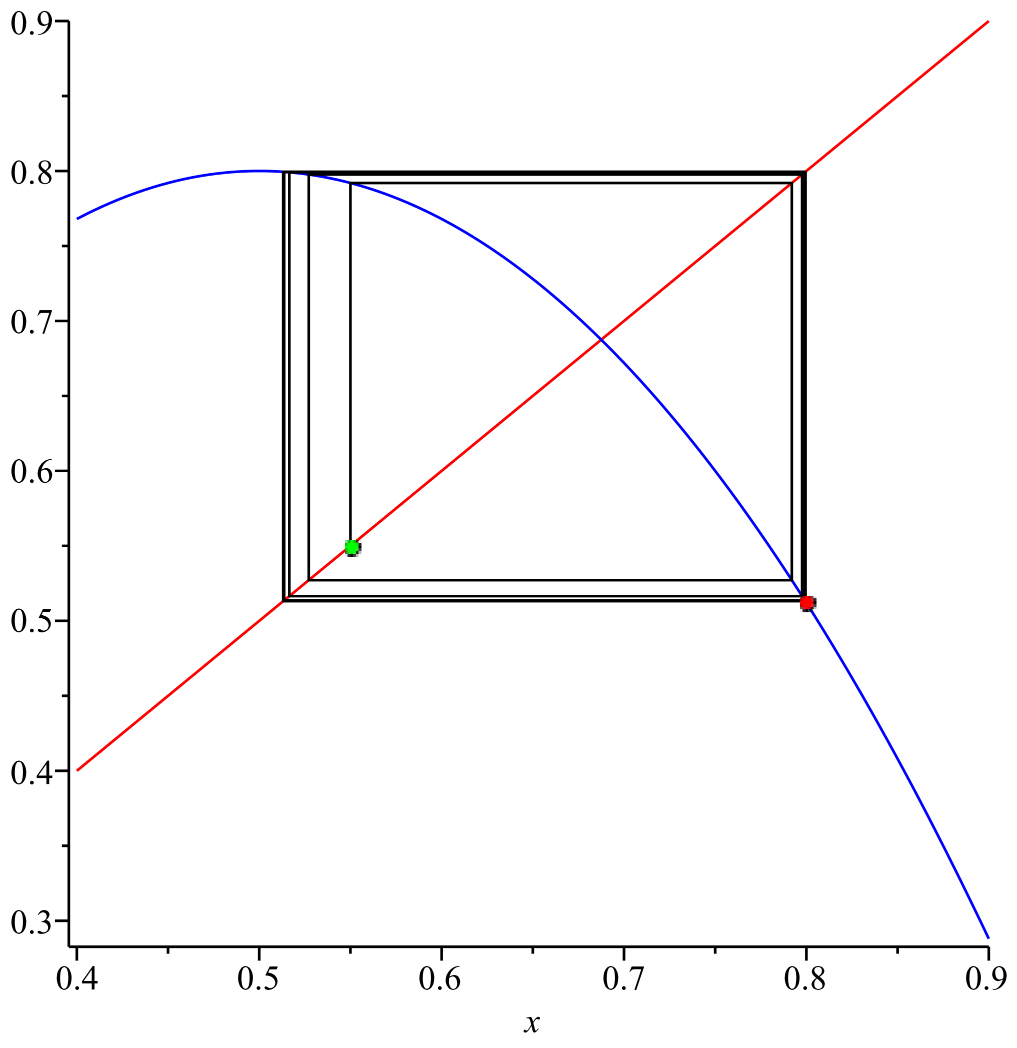
$$\mathrm{D}(G)(C)\ \mathrm{D}(G)(B) \qquad (4.10)$$

```
> D(g@g)(b) = D(g)(b)*D(g)(c);
```

$$0.1599999996 = 0.1599999996 \qquad (4.11)$$

If x_0 is close enough to one of the points on the cycle (say b), then as $n \to \infty$ the even-numbered x_n approach b and the odd-numbered ones approach c.

```
> staircase(0.55, 0.4, 0.9, 10);
```



We say that g has an attracting 2-cycle. A 2-cycle (b, c) is an attractor if $|g'(b)\ g'(c)| < 1$, a repeller if $|g'(b)\ g'(c)| > 1$.

More cycles

More generally, an ***n*-cycle** is a sequence of n different numbers $b_1, b_2, \ldots, b_n$ such that $g(b_1) = b_2$, $g(b_2) = b_3, \ldots, g(b_n) = b_1$. Each member of an n-cycle is said to be a **periodic point of period *n***. These points will be fixed points of $g \circ \ldots \circ g$ (with n g's), so we might find them by solving $g(g(\ldots(g(x))\ldots)) = x$. In Maple input, $g \circ \ldots \circ g$ can be written as `g @@ n`, which in Maple output would be

```
> g@@n;
```

$$g^{(n)} \tag{5.1}$$

To test whether the n-cycle is an attractor or a repeller, we can check $(g^{(n)})'(b_j)$ for any of the b_j, which will be the product of all the $g'(b_j)$ in the cycle. If the absolute value of the result is < 1, it's an attractor; if > 1, it's a repeller.
For example, try

```
> r:= 3.5;
```

$$r := 3.5 \tag{5.2}$$

```
> s:= fsolve((g@@2)(x)=x);
```

$$s := 0., 0.4285714286, 0.7142857143, 0.8571428571 \tag{5.3}$$

```
> g(s[2]);
```

$$0.8571428572 \tag{5.4}$$

```
> g(s[4]);
```

$$0.4285714286 \tag{5.5}$$

```
> g(s[3]);
```

$$0.7142857141 \tag{5.6}$$

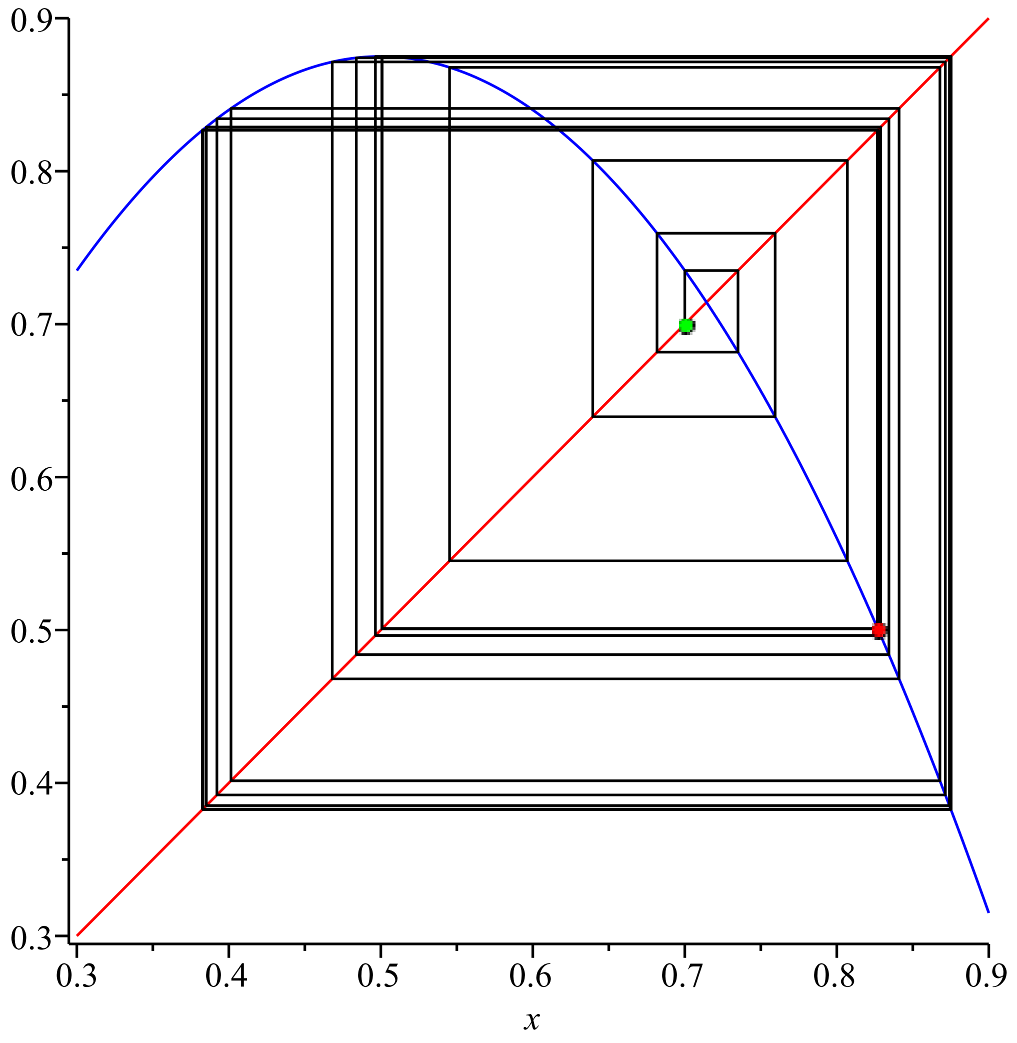
So the 2-cycle is $[s_2, s_4]$, while $s_1 = 0$ and s_3 are fixed points.

```
> D(g@g)(s[2]);
  D(g)(s[1]);
  D(g)(s[3]);
```

$$-1.249999999$$
$$3.5$$
$$-1.500000002 \tag{5.7}$$

So the 2-cycle and fixed points here are repellers. What will the cobweb diagram look like?

```
> staircase(0.7, 0.3, 0.9, 50);
```



It looks like the attractor here is a 4-cycle. To find it, it's best to increase Digits, because $g^{(4)}$ is a rather complicated function.

```
> Digits:= 15;
  s:= fsolve((g@@4)(x)=x);
```

$Digits := 15$

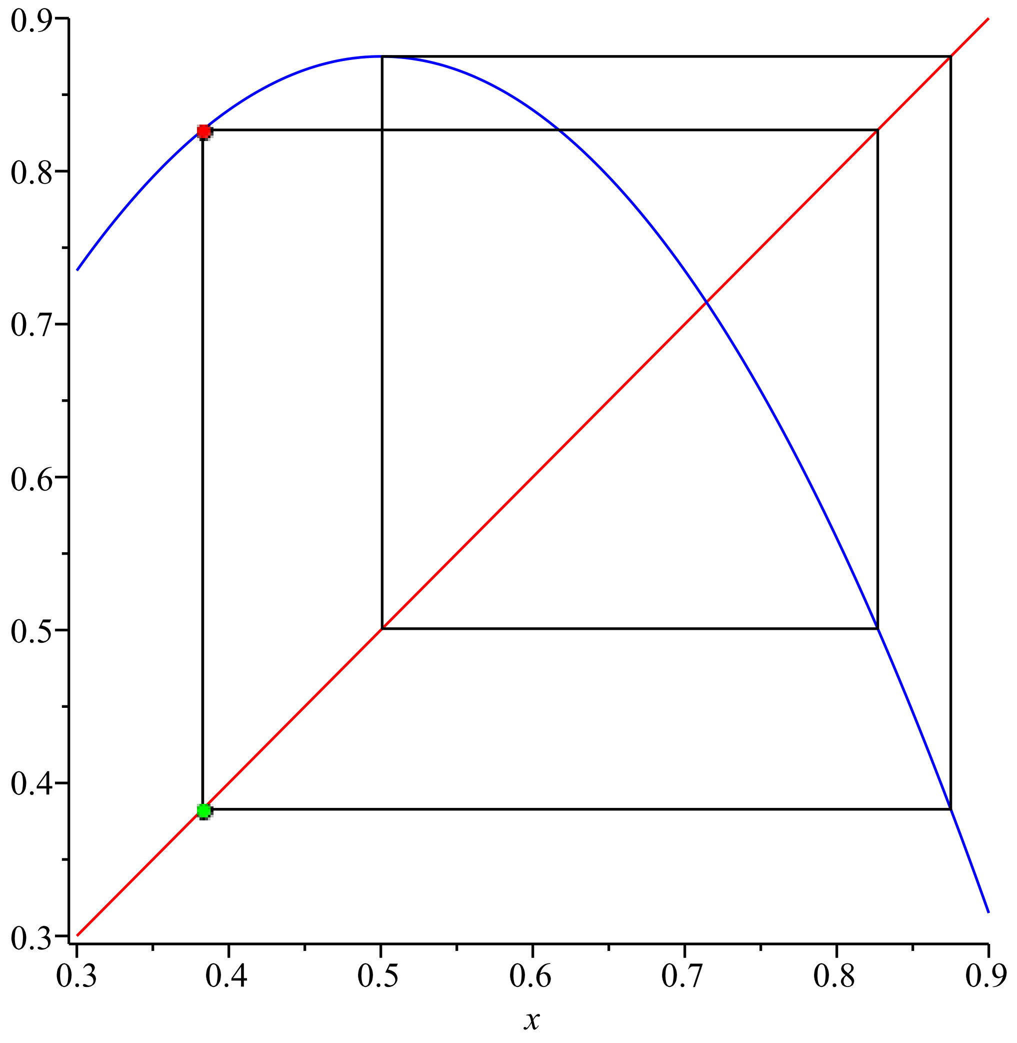
$s :=$ 0., 0.382819678330056, 0.428571456052699, 0.500884136389258, 0.714286476436312, 0.826926060420077, 0.857182725329268, 0.874969913430325 **(5.8)**

```
> seq((g@@j)(s[2]),j=1..4);
```

0.826940702746648, 0.500884219106348, 0.874997263548002, 0.382819683160285 **(5.9)**

So we have a 4-cycle: $[s_2, s_6, s_4, s_8]$.

```
> staircase(s[2],0.3,0.9,5);
```



```
> D(g@@4)(s[2]);
```

```
                         -0.0305002803480967                              (5.10)
```

And it is an attractor.

Counting with nops

It's sometimes tedious to count the number of items in an expression sequence by hand (it may be difficult sometimes to spot the commas if the items are complicated, or there may be lots of items). The **nops** command can help. It returns the number of items in a list or set (or more generally, number of operands in any structure). Here I'll put s into a list (because nops wouldn't work on an expression sequence) and count how many members it has.

```
> nops([s]);
```

```
                                  8                                        (6.1)
```

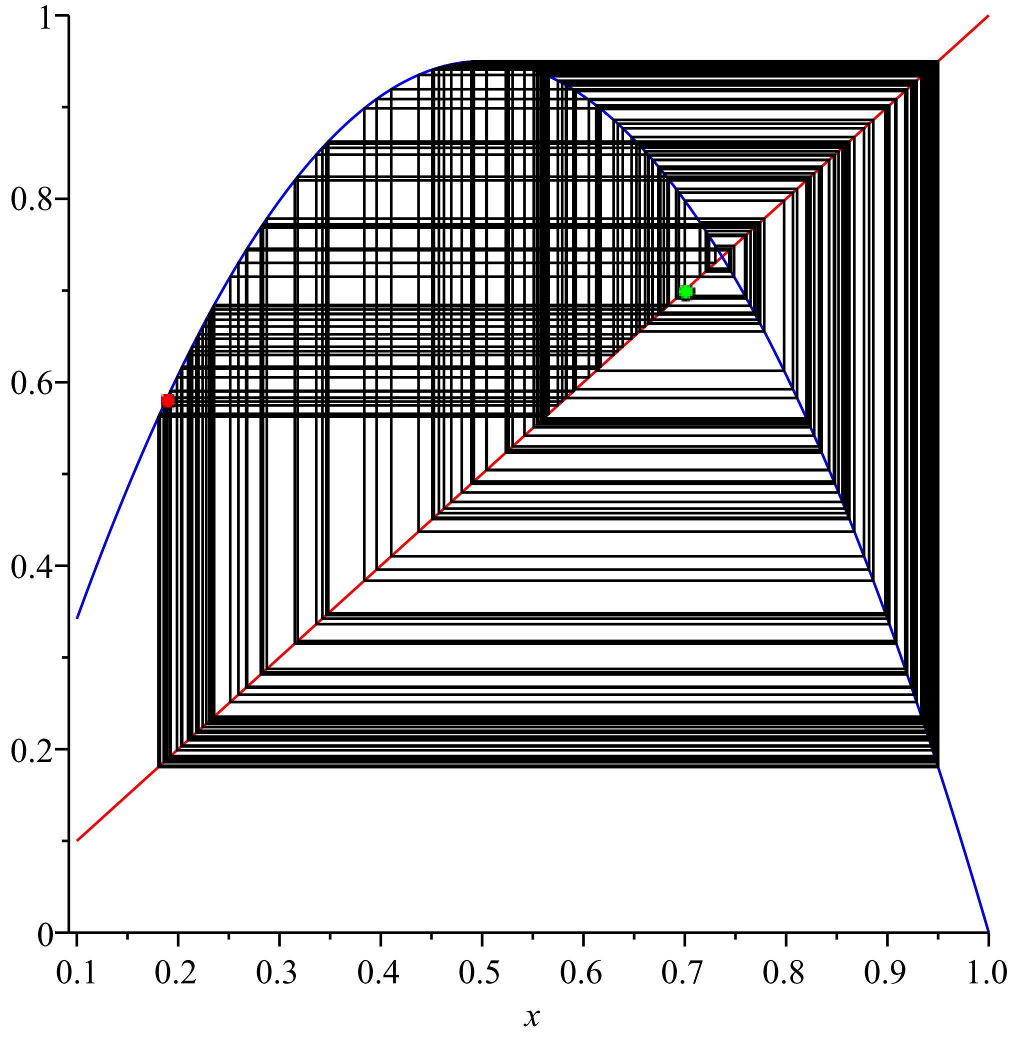
Chaos

One more value of *r*:

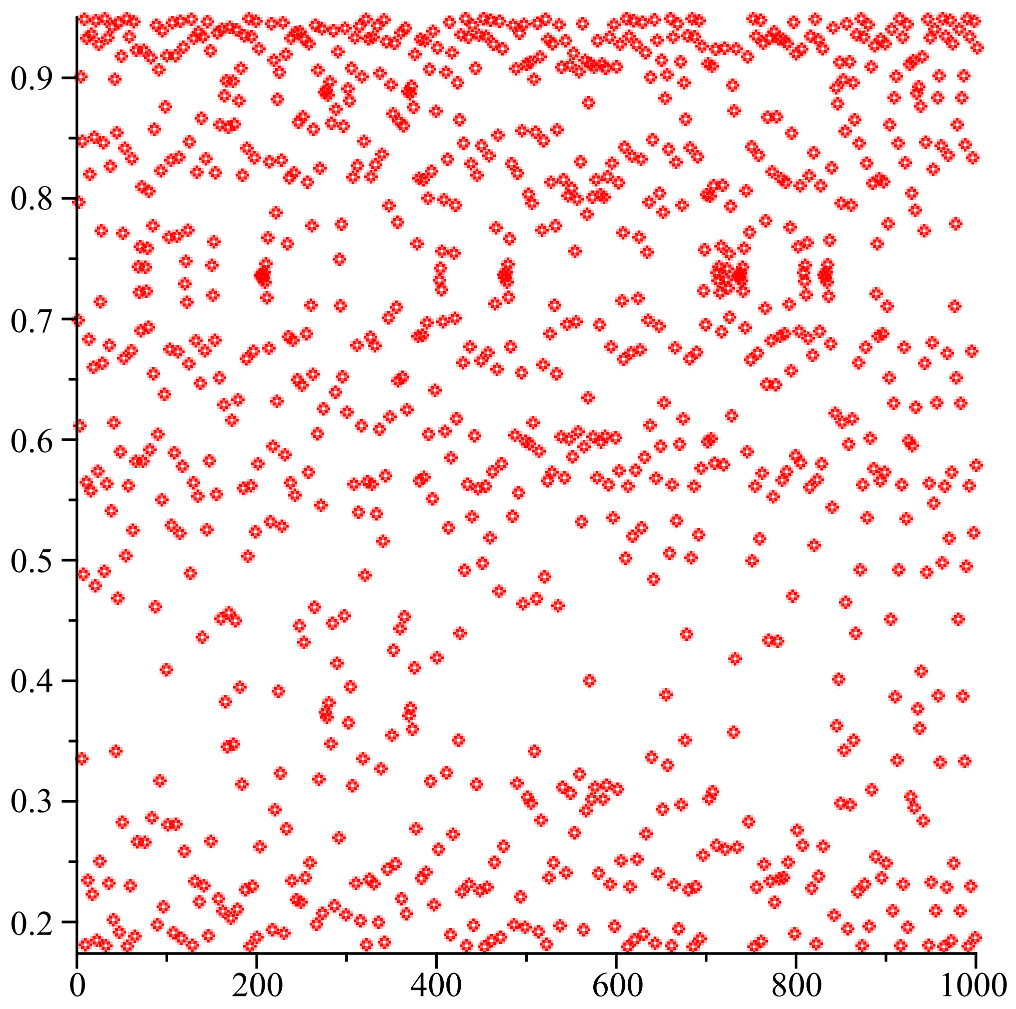
```
> r := 3.8;
```

$$r := 3.8 \tag{7.1}$$

```
> staircase(0.7,0.1,1.0,200);
```



```
> X[0]:= 0.7:
  for count from 1 to 1000 do
    X[count] := g(X[count-1])
  end do:
> plot([seq([i,X[i]],i=0..1000)],style=point);
```



There's no sign of a periodic pattern. This is an example of **chaos**.
There are cycles for this function, but they are repellers.
Let's try finding a 5-cycle.

```
> s:= fsolve((g@@5)(x)=x, x = 0 .. 1);
```

$s := 0., 0.219034136109526, 0.236700447977394$ **(7.2)**

This is a strange result: Maple just found 3 solutions with Digits = 15 when I tried it on my computer: your results may vary. There should be two fixed points, and any more solutions would have to come in fives. The culprit may be roundoff error. I'll try increasing Digits.

```
> Digits:= 30:
  s:= fsolve((g@@5)(x)=x, x = 0 .. 1);
```

$s := 0., 0.219034304822227170265192579168, 0.236705914221693236095343258944,$ **(7.3)**
$0.445198814083808991935840554760, 0.566384184067048757798178059770,$
$0.650021456907103227782809711421, 0.686569652695769856328007584106,$
$0.736842105267469093579000190400, 0.817728705775205959463691415157,$
$0.864475537511362873615618839703, 0.933253932104562119065861736722,$
$0.938587954384671010833225012849$

```
> nops([s]);
                                   12                                   (7.4)
```

This time there are 12 solutions: two fixed points and, presumably, two 5-cycles. If I calculate the derivative of $g^{(5)}$ at each of these points, it should not only tell us whether the cycles are attractors or repellers, it should also help us to see which are the points of the 5-cycles, because the derivative should be the same at each point on a 5-cycle.

```
> seq(D(g@@5)(s[j]),j=1..12);
792.35168, -9.3624536119340963827446370480 4, 11.381870948677382353730113955 1,      (7.5)
    -9.3624536119341259776259078803 9, 11.381870948673433749326779932 8,
    -9.3624536117101370716016706439 7, 11.381870948047163702781032279 8,
    -18.895680002444006870161265029 5, 11.381870985244342708340702596 7,
    -9.3624537271671330106372839623 6, 11.381872068427890351515027114 7,
    -9.3624547345440920449346433124 3
```

But which is which? It might help to see the index j together with the value.

```
> seq([j, D(g@@5)(s[j])],j=1..12);
[1, 792.35168], [2, -9.36245361193409638274463704804], [3,                          (7.6)
    11.38187094867738235373011395 51], [4, -9.36245361193412597762590788039],
    [5, 11.381870948673433749326779932 8], [6,
    -9.3624536117101370716016706439 7], [7, 11.381870948047163702781032279 8],
    [8, -18.895680002444006870161265029 5], [9,
    11.381870985244342708340702596 7], [10, -9.3624537271671330106372839623 6],
    [11, 11.381872068427890351515027114 7], [12,
    -9.3624547345440920449346433124 3]
```

All the numbers have absolute values greater than 1, so the fixed points and cycles are repellers. It looks like s_1 and s_8 (with derivative values about 792 and -19) are the fixed points; the 5-cycles are s_2, s_4, s_6, s_{10}, s_{12} (with derivative values about -9) and s_3, s_5, s_7, s_9, s_{11} (with derivative values about 11). To check:

```
> g(s[2]);
                    0.650021456906429198274875991424                    (7.7)
```

That's approximately s_6.

```
> s[6] - %;
                    6.74029507933719997 10^-13                          (7.8)
```

```
> g(s[6]);
                    0.864475537376386643068718778198                    (7.9)
```

That's approximately s_{10}.

```
> s[10] - %;
                    1.3497623054690006150 5 10^-10                      (7.10)
```

```
> g(s[10]);
```

```
0.445198813712051836288170301422                    (7.11)
```

That's approximately s_4.

```
> s[4] - %;
```

$$3.7175715564767025333\,8\ 10^{-10} \quad (7.12)$$

```
> g(s[4]);
0.938587954084280459555477588047                    (7.13)
```

That's approximately s_{12}.

```
> s[12] - %;
```

$$3.0039055127774742480\,2\ 10^{-10} \quad (7.14)$$

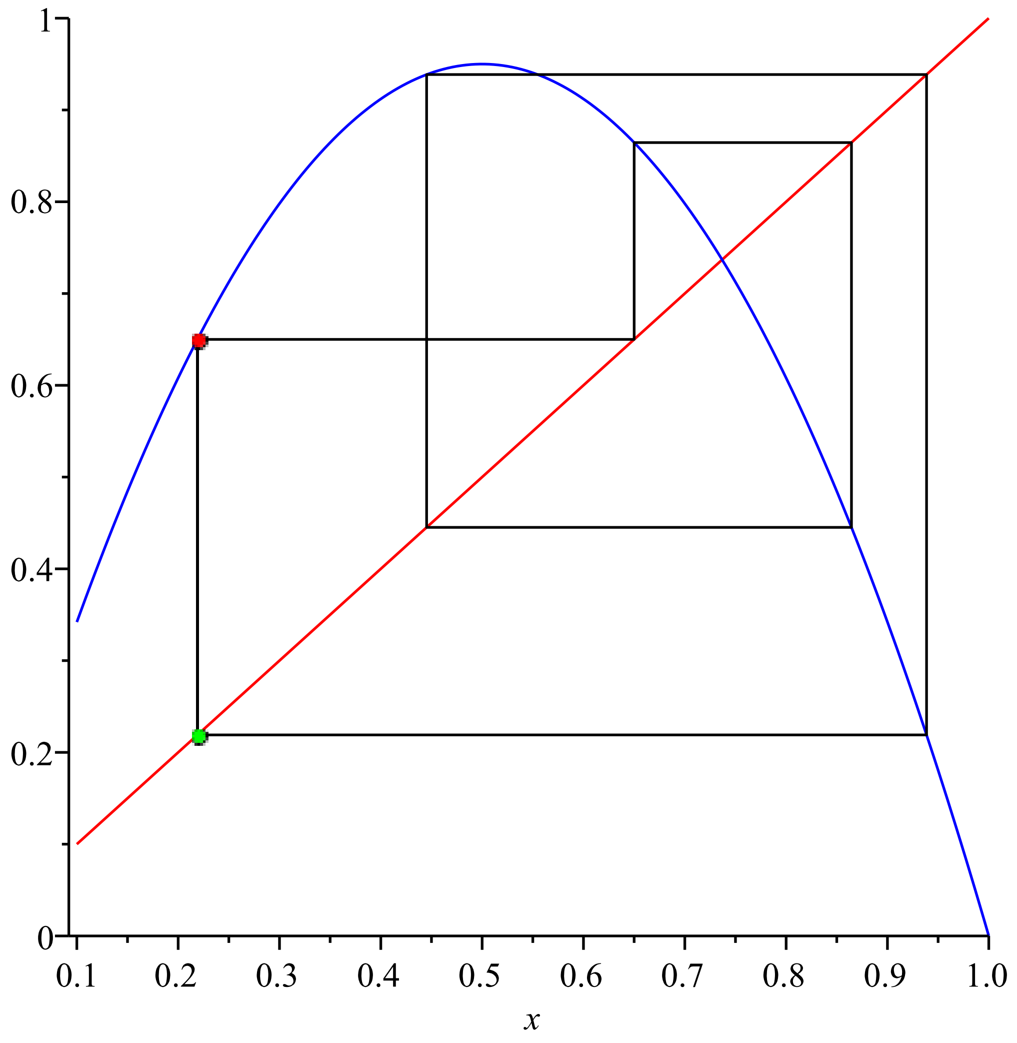
```
> g(s[12]);
0.219034303820945014443841659598                    (7.15)
```

And this is back to s_2.

```
> s[2] - %;
```

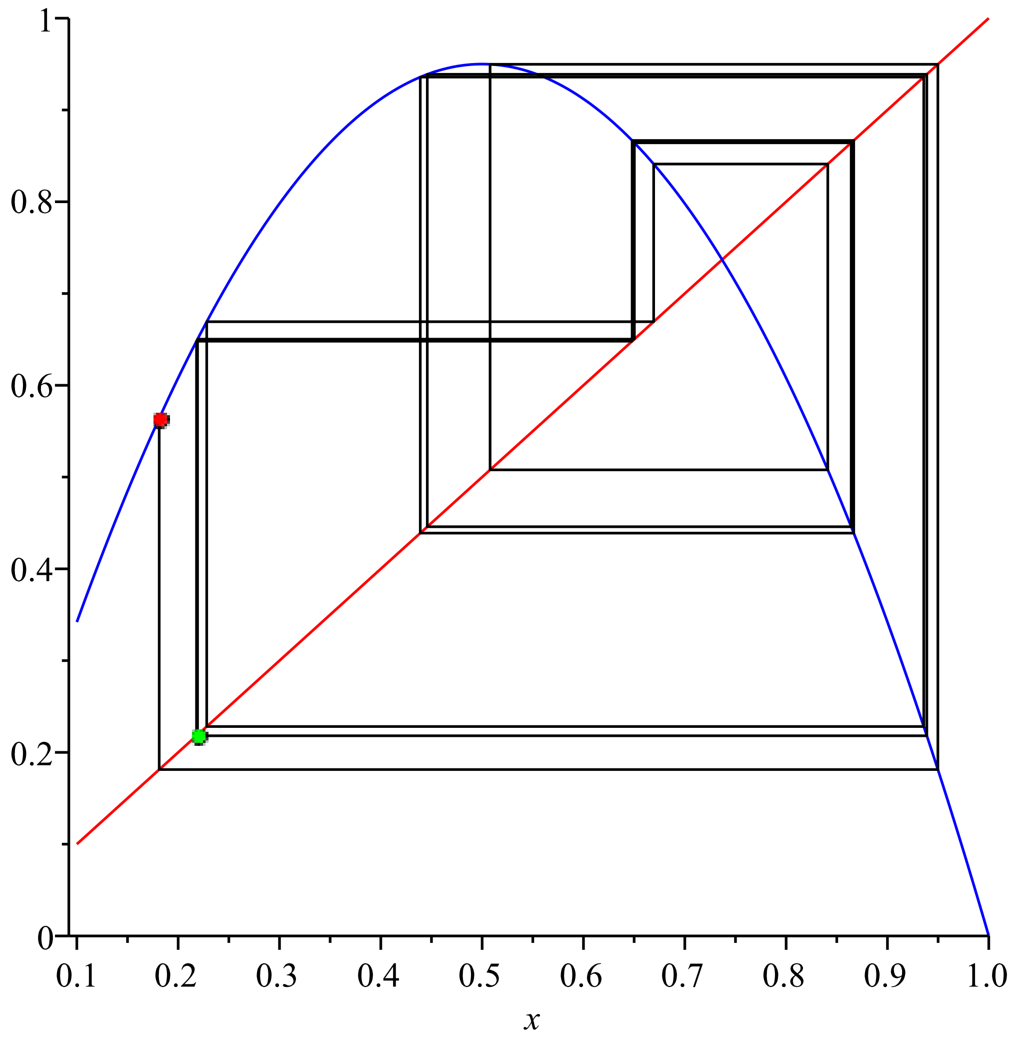
$$1.0012821558213509195\,70\ 10^{-9} \quad (7.16)$$

```
> staircase(s[2],0.1, 1.0, 6);
```



But the cycle is a repeller, so if you start close to s_2 you would move away eventually.

```
> staircase(s[2]+0.0001, 0.1, 1.0, 16);
```



Maple objects introduced in this lesson

view (option for plotting commands)
proc ... end proc
local
uses
style=point
@
@@
nops