Lesson 8: Iteration: cycles and basins

```
> restart;
```

▼ @@ and $

The **@@** that we used for iterating a function can also be used for higher derivatives in the D style. Just as **(g@@3)(x)** is **g(g(g(x)))**, **(D@@3)(f)** is **D(D(D(f)))**, i.e. the third derivative of the function f. For example:

```
> f:= x -> exp(3*x);
```
$$f := x \rightarrow e^{3x} \tag{1.1}$$

```
> (D@@3)(f);
```
$$x \rightarrow 27\, e^{3x} \tag{1.2}$$

Or, for higher partial derivatives of a function of several variables:

```
> H:= (x,y,z) -> x^2*y^3*z^4;
  (D[3]@@2)(H);
```
$$H := (x, y, z) \rightarrow x^2\, y^3\, z^4$$
$$(x, y, z) \rightarrow 12\, x^2\, y^3\, z^2 \tag{1.3}$$

We can also do higher derivatives, including mixed partial derivatives, in the diff style. We can put whatever variables we want to differentiate with in the diff command.

```
> diff(F(x,y),x,x,y);
```
$$\frac{\partial^3}{\partial y\, \partial x^2}\, F(x, y) \tag{1.4}$$

```
> diff(f(x),x,x,x);
```
$$27\, e^{3x} \tag{1.5}$$

To avoid the bother of typing the same variable several times, we can use the **$** operator. This is a shortcut for constructing sequences:
a $ n , where n is a nonnegative integer, means **a** repeated n times. So:

```
> x + y $ 5;
```
$$x + y, x + y, x + y, x + y, x + y \tag{1.6}$$

Thus you often see higher derivatives written this way:

```
> diff(f(x), x$3);
```
$$27\, e^{3x} \tag{1.7}$$

The **$** operator can be used in other ways too. For example, you could use this

```
> f(i) $ i=1..3;
```
$$e^3, e^6, e^9 \tag{1.8}$$

instead of

```
> seq(f(i), i=1..3);
```
$$e^3, e^6, e^9 \tag{1.9}$$

There are some technical differences between **$** and **seq**, having to do with evaluation rules, and in general it's considered better Maple style to use **seq** except in a few stereotypical situations. One of these is in derivatives, as above; another is

```
> $2..6;
```
$$2, 3, 4, 5, 6 \tag{1.10}$$

to obtain a sequence of consecutive integers. You can also have

```
> $"b".."g";
```
$$\text{"b", "c", "d", "e", "f", "g"} \tag{1.11}$$

to obtain a sequence of consecutive letters.

▼ A cycle made to order

Back to iteration of the function g.

```
> g:= x -> r*(x - x^2);
```
$$g := x \rightarrow r\left(x - x^2\right) \tag{2.1}$$

Question:

Find some r for which there is an attracting 3-cycle.

What do we need for some point p to be on an attracting 3-cycle?

For it to be on a 3-cycle: $g^{(3)}(p) = p$ (but $g(p) \neq p$).

For the 3-cycle to be an attractor: $\left|\left(g^{(3)}\right)'(p)\right| = |g'(p)\, g'(g(p))\, g'(g(g(p)))| < 1.$

It looks like a complicated equation and an even more complicated inequality. Actually there's a trick: one way to make sure a product of things is less than 1 is to make one of them 0. And where is $g'(p) = 0$?

```
> solve(D(g)(p)=0, p);
```
$$\frac{1}{2} \tag{2.2}$$

So all we need to do is make 1/2 be a periodic point of period 3.

```
> fsolve((g@@3)(1/2)=1/2, r);
```
$$-1.831874055, 2., 3.831874055 \tag{2.3}$$

I'll try the third of these.

```
> r:= %[3];
```
$$r := 3.831874055 \tag{2.4}$$

Here's what the 3-cycle looks like. I need my **stair** and **staircase** procedures.

```
> stair:= x -> ([x,x],[x,g(x)]);
```
$$stair := x \rightarrow ([x, x], [x, g(x)]) \tag{2.5}$$

```
> 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;
```

$staircase := \mathbf{proc}(x0, a, b, n)$ **(2.6)**

    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**;

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

    $Dots := plot([[x[0], x[0]]], style = point, symbol = solidcircle, colour = green), plot([[x[n - 1], g(x[n - 1])]], style = point, symbol = solidcircle, colour = red);$

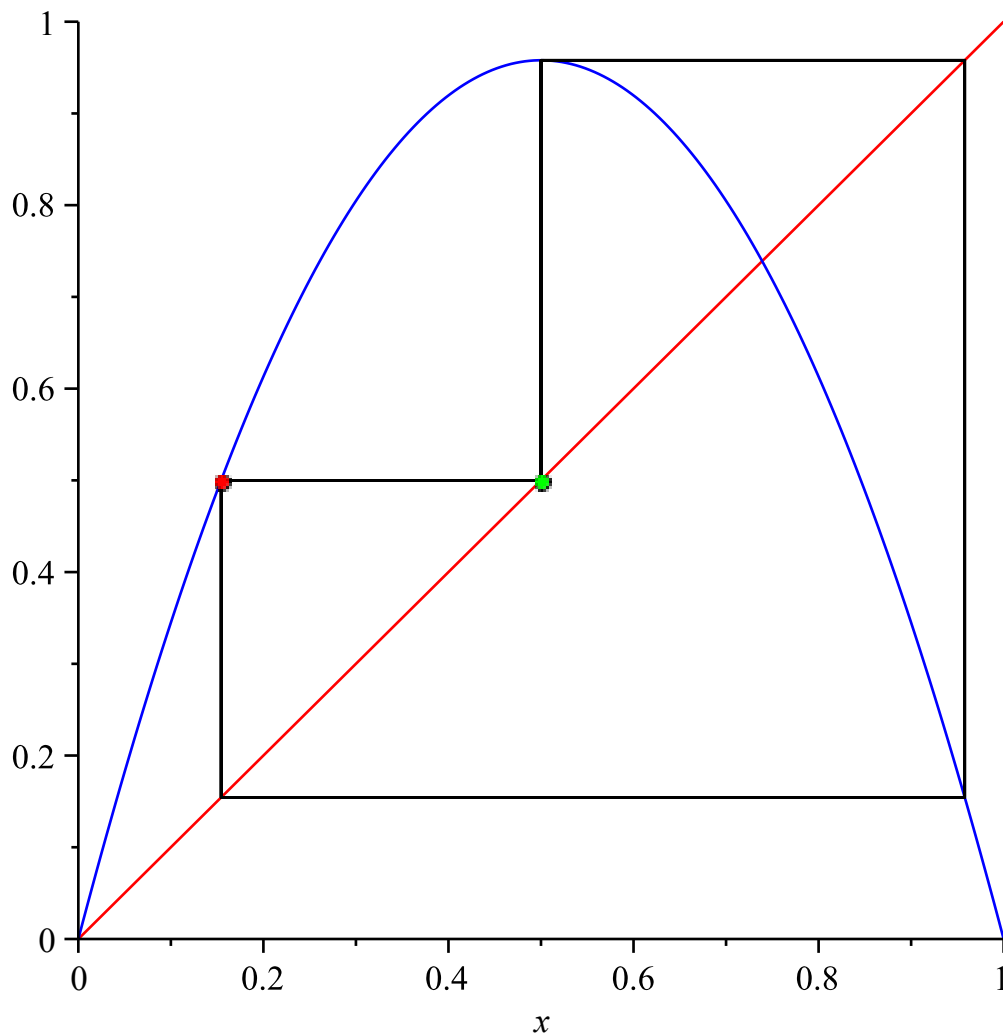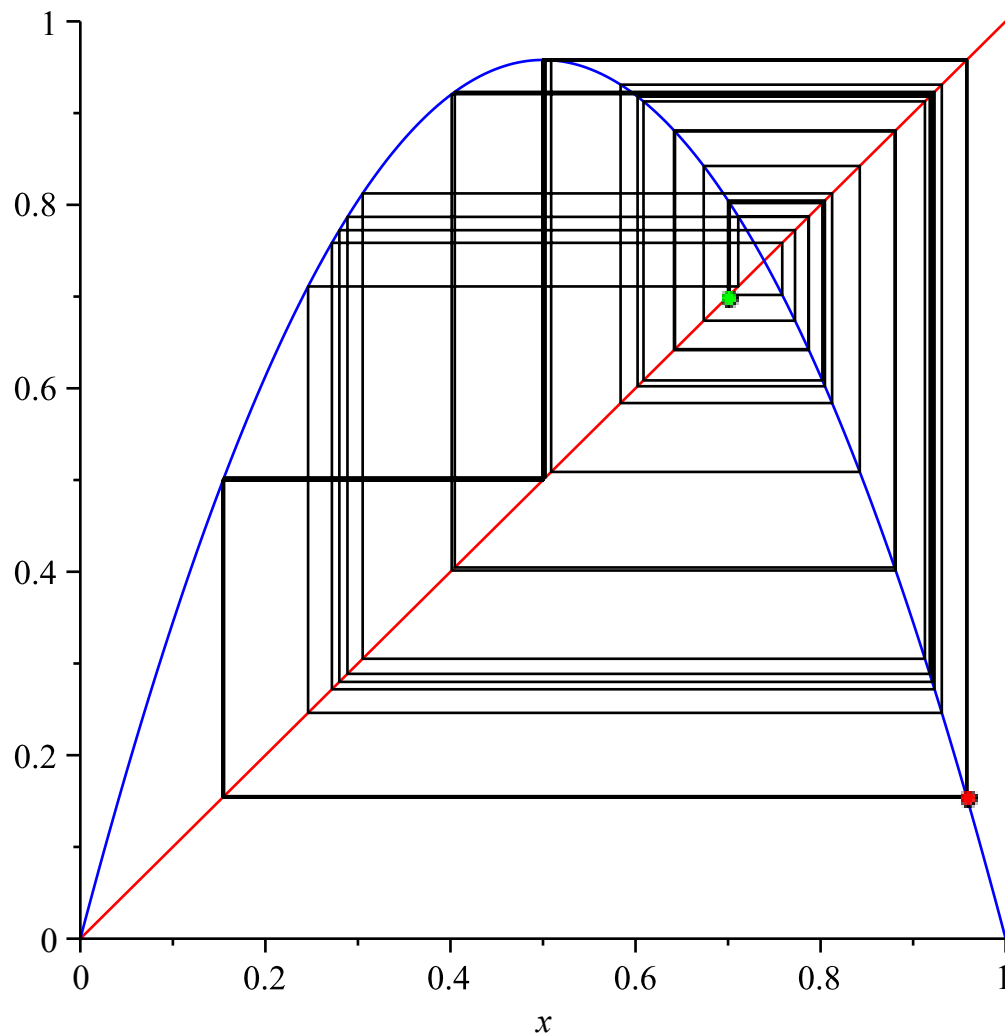 $plots\text{:-}display([Curves, Staircase, Dots])$

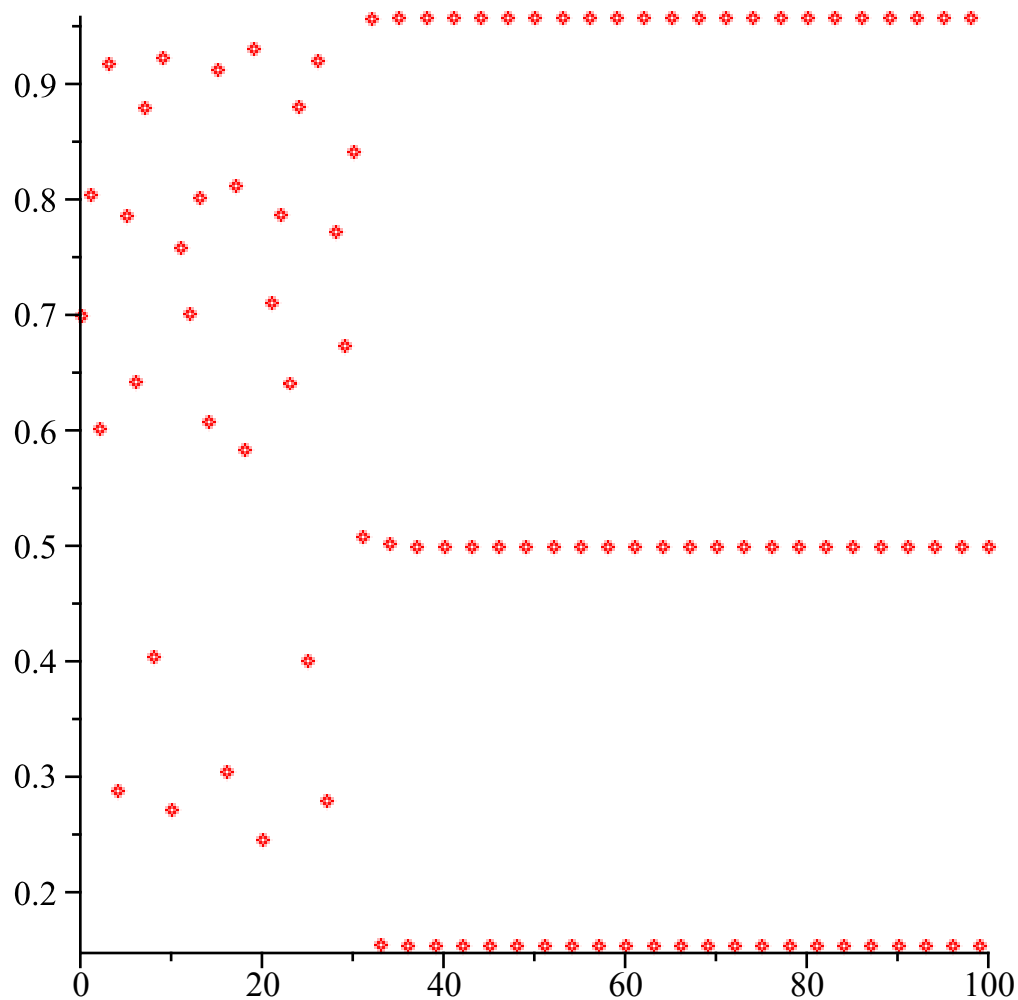**end proc**

```
> staircase(1/2, 0, 1, 9);
```

What about if we start someplace else? Will it be attracted to the 3-cycle?

```
> staircase(0.7,0,1,60);
```


It may be a bit hard to see this way. Try plotting the points.

```
> X[0]:= 0.7:
  for count from 1 to 100 do
     X[count]:= g(X[count-1]);
  end do:
  plot([seq([i,X[i]], i=0..100)], style=point);
```

It looks like the points wander around more or less randomly for a while, then one happens to end up close enough to a point on the 3-cycle, and it's trapped.

Another way to do this is with an animation of the cobweb diagram. An animation consists of a number of "frames" that are shown one after the other, creating the illusion of motion. The **plots** package contains a command **animate** that can produce these. You give **animate** the following inputs:

1) the name of the command that you want to animate (in this case **staircase**: it could be either one of Maple's own plotting commands, or, as in this case, one that you wrote yourself).

2) A list of the inputs that will be passed to that command. This will generally contain an "animation variable" (in this case i) that will be given a different value in each frame, so each frame will be different. In this case I'd want **staircase(X[i], 0, 1, 4)**, to produce a cobweb diagram starting at x_i with four stairs.

3) The name and interval for the animation variable, in the form **i = a .. b**.

4) Various possible options, including **frames = n** which specifies how many frames you want.

The first frame is at $i = a$, and then i is increased by $\dfrac{b-a}{n-1}$ in each subsequent frame, until $i = b$ in the n'th frame. I'll use **i = 0 .. 35** with **frames = 36**, so we'll get $i = 0, 1, 2, ..., 35$.

There's only one problem: **animate** uses **evalf** at some point, giving floating-point values to the variable i. That's usually OK, but it would be bad here: Maple knows what **X[3]** is, but would have trouble with **X[3.0]**.

```
> X[3];
```
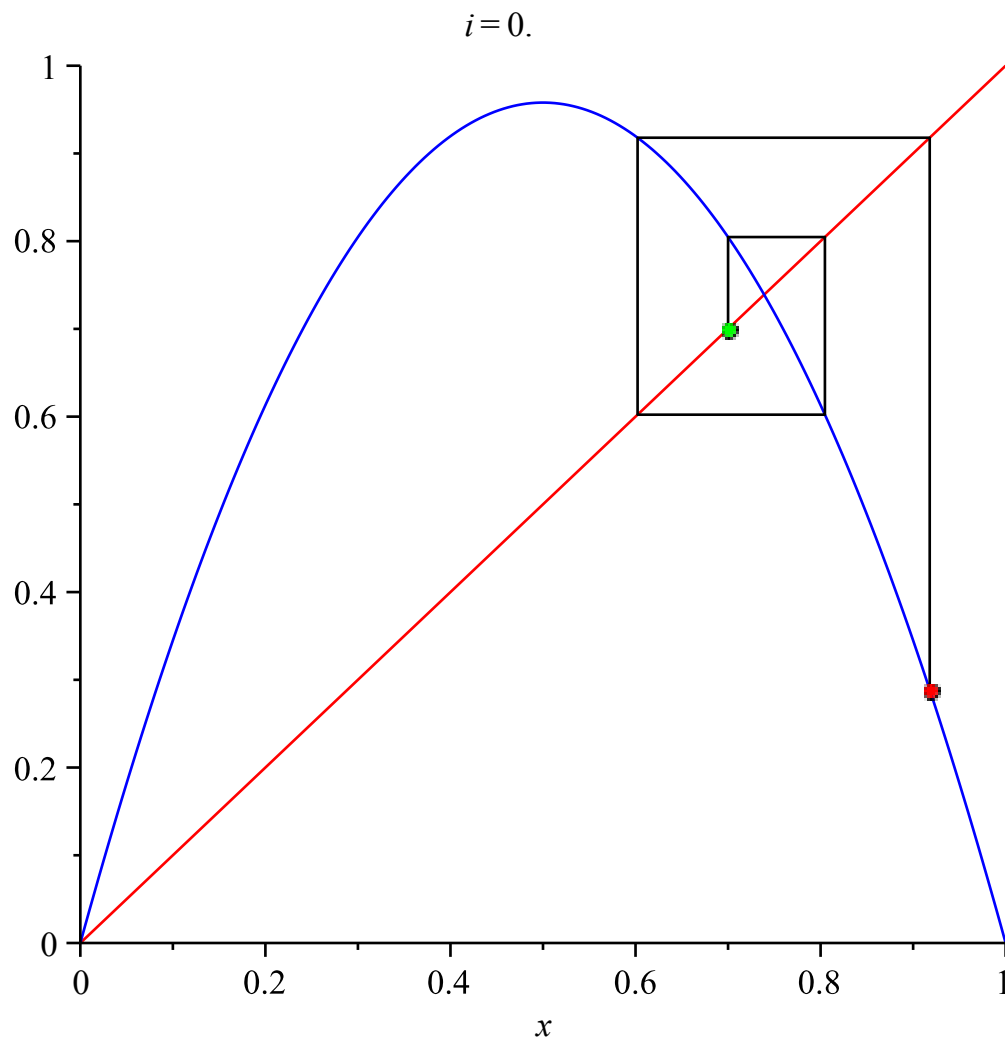$$0.9179261081 \tag{2.7}$$

```
> X[3.0];
```
$$X_{3.0} \tag{2.8}$$

To get around this, I'll use **X[round(i)]** rather than **X[i]**. The **round** function takes a number and returns the nearest integer.

```
> round(3.4), round(3.5),round(3.6);
```
$$3, 4, 4 \tag{2.9}$$

```
> with(plots):
```

```
> animate(staircase,[X[round(i)], 0, 1, 4],i=0 .. 35, frames =
  36);
```

$$i = 0.$$

▼ Basins of attraction

When a fixed point p is an attractor, from any starting point x_0 close enough to p the x_n will converge to p as $n \to \infty$. But how close is "close enough"?

The **basin of attraction** of the attracting fixed point p is the set of all real numbers x such that when $x_0 = x$, $x_n \to p$ as $n \to \infty$. This might be a very complicated set, but it contains an interval around p.

The **immediate basin of attraction** of p is the largest interval containing p that is in this basin of attraction. It is an open interval $(a,b) = \{\, x : a < x < b \,\}$. The endpoints are not in the basin of attraction (the basin of attraction is an **open set**, which means it contains an interval around any member of it). They may be infinite, i.e. the basin of attraction might go all the way from p to ∞ or $-\infty$. But we'll be mainly concerned with cases where they are finite. So if we start with $x_0 = a$ or b, something else has to happen to x_n as $n \to \infty$. What is that something else?

To see what can happen, I'm going to look at some examples with a new visualization tool: embedded components. I'll take some different functions **g**. The embedded component below is a Plot. By clicking on it you can move the starting point for a "staircase".

```
> g:= x -> 1.7*x^2 - x/2;
```

$$g := x \to 1.7\, x^2 - \frac{1}{2}\, x \qquad\qquad \textbf{(3.1)}$$

$x_0 = 0.25130737140941606$
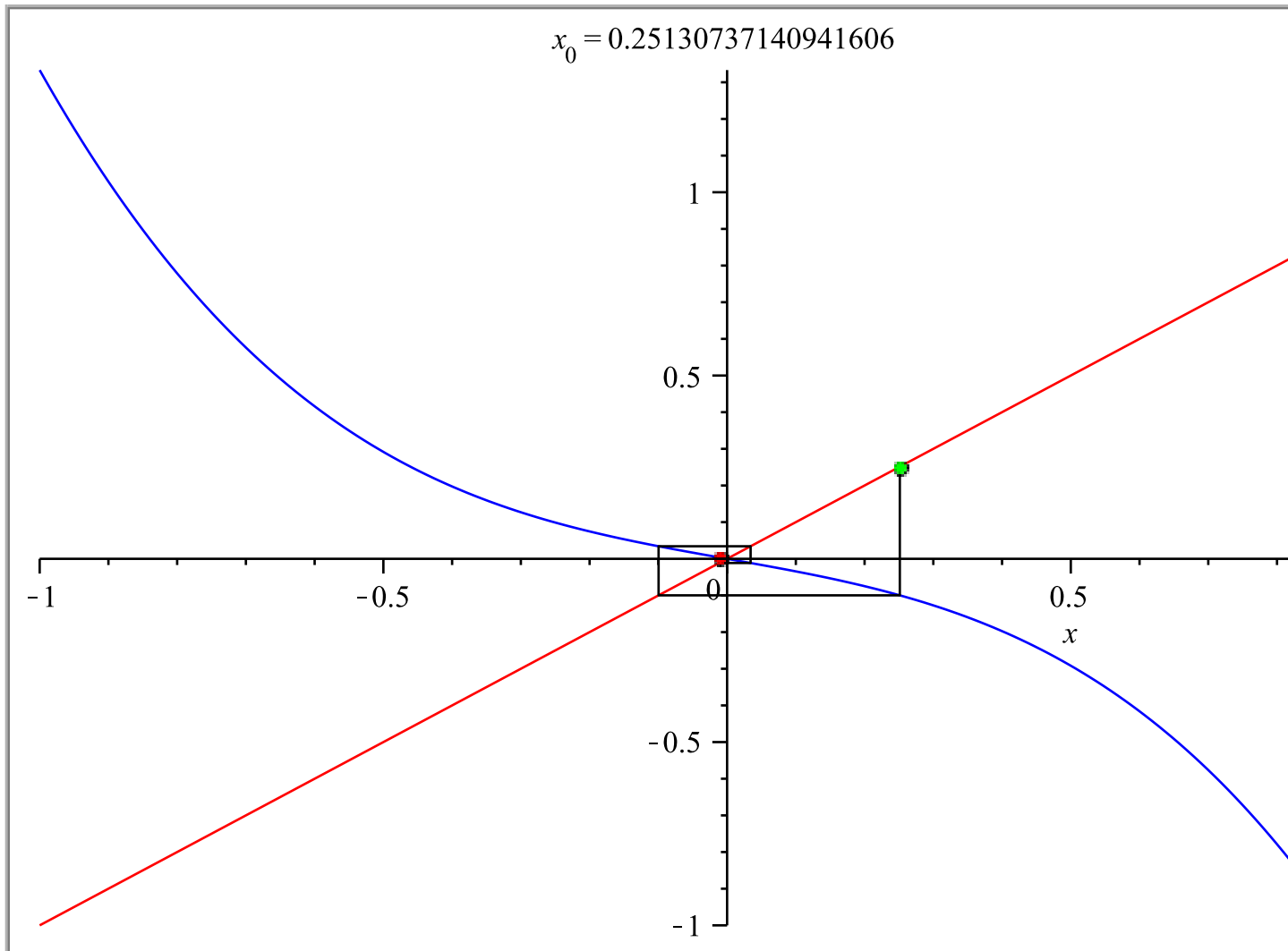
Here we see that one endpoint, b, is a repelling fixed point, while $g(a) = b$.

```
> fsolve(g(x)=x);
```
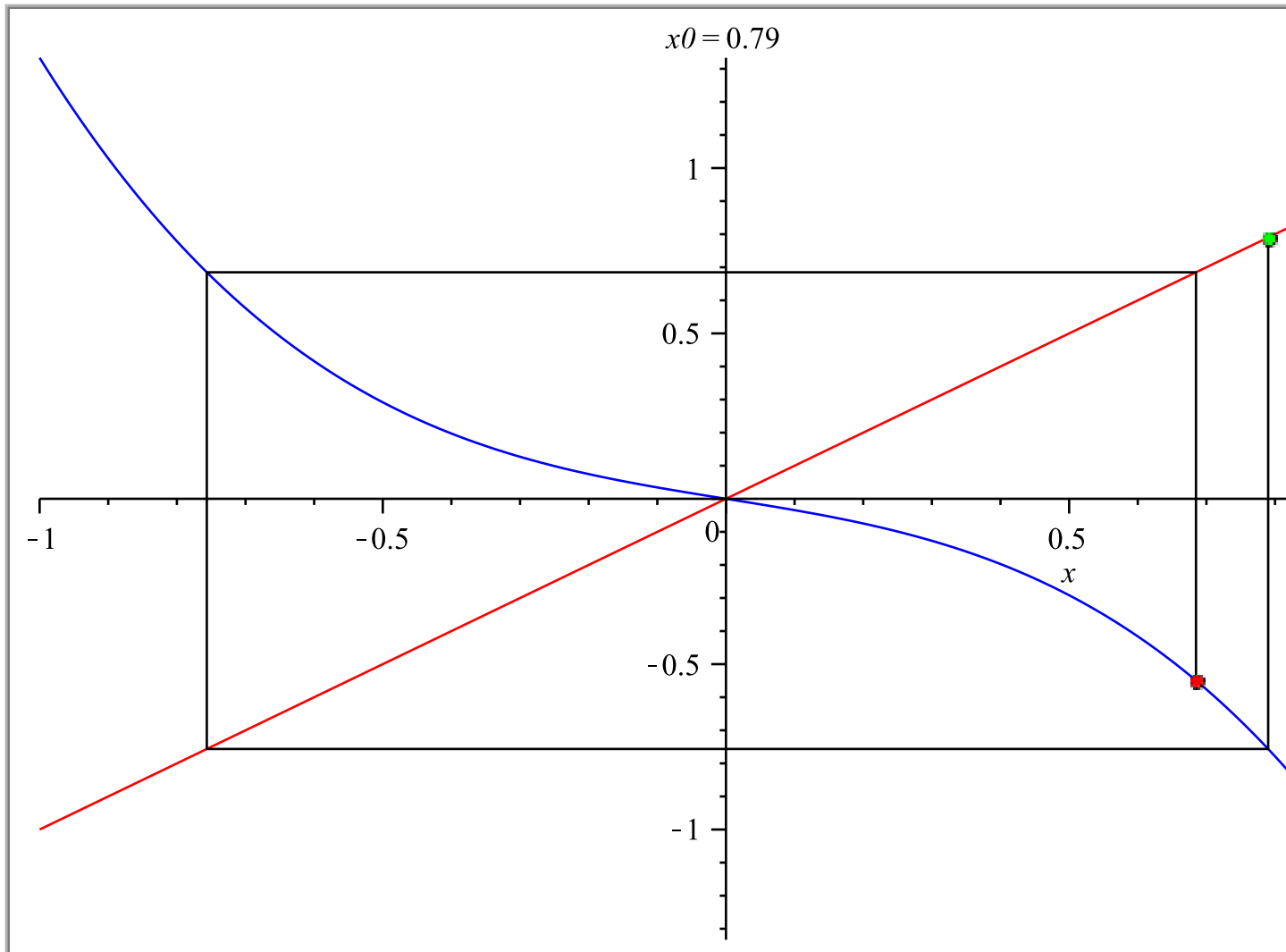$$0., 0.8823529412 \tag{3.2}$$

```
> b := %[2];
```
$$b := 0.8823529412 \tag{3.3}$$

```
> fsolve(g(x)=b);
```
$$-0.5882352941, 0.8823529412 \tag{3.4}$$

```
> a := %[1];
```
$$a := -0.5882352941 \tag{3.5}$$

Now for a new function.

```
> g:= x -> - x/3 - x^3 ;
```
$$g := x \rightarrow -\frac{1}{3}\, x - x^3 \tag{3.6}$$

$x0 = 0.79$

Here a and b form a repelling 2-cycle.

```
> fsolve(g(g(x))=x);
```
$$-0.8164965809, 0., 0.8164965809 \tag{3.7}$$

```
> a := %[1]; b := %%[3];
```
$$a := -0.8164965809$$
$$b := 0.8164965809 \tag{3.8}$$

How did I get those components? From the menu, choose View, Palettes, Expand Docks. In the Components palette, insert a Plot component.
Then you can Collapse Docks.
Right click on the Plot component, choose Component Properties. Notice the Name (Plot0, Plot1 etc).
You can adjust the Width and Height. For Plot Expression, enter a plotting command that will produce the plot you want visible initially, e.g.
plot([x, g(x)], x= -1..1, colour=[red, blue]);
Now click on the "Edit" after "Action when clicked".
Here you have to tell Maple what to do when you click on the Plot component. In this case we want to change what the Plot component shows. You can do that with
Do (%Plot0 = (some plot command))
(if the Plot component's name is Plot0).
That plot command can depend on the x and y components of the point where you click, which you get by %Plot0(clickx) and %Plot0(clicky). What I used was

Do (%Plot0 = display(staircase(%Plot0(clickx), -1, 1, 3),
    view=[-1..1, -1 .. g(-1)],
    title = (x[0] = %Plot0(clickx))));

This makes a cobweb diagram where the starting point is the x coordinate of the point clicked. I also wanted to add a title, which staircase doesn't provide, so I put that in a **display** command with the **title** option; also the **view** option. The **display** command is in the **plots** package, so I inserted **plots** into the **uses** statement.

Back in "Plot Properties", select "Make "execute code" the default manipulator".

▼ Maple objects introduced in this lesson

```
@
@@
nops
$
```
animate (in **plots** package)
frames (option for **animate**)
round
Plot component
title (option for plot commands)
view (option for plot commands)