Lesson 6: Iteration

```
> restart;
```

Iteration

Newton's method is a particular case of an iteration method. In general, iteration deals with a sequence defined by $x_{n+1} = g(x_n)$ for some function g. The study of iterations, also called **discrete dynamical systems**, is a very active and important area of modern mathematics, related to fractals and chaos.

Suppose this sequence x_n converges to some value c as $n \to \infty$, and g is continuous at c. Then we can take limits on both sides of the equation and get $c = g(c)$. A solution of $c = g(c)$ is called a **fixed point** of the function g.

This suggests that you might use an iteration scheme to find a fixed point of g: if the sequence x_n converges, the limit is a fixed point. Newton's method is an example of this, with $g(x) = x - \frac{f(x)}{f'(x)}$, because $g(c) = c$ is equivalent to $f(c) = 0$ (assuming $f'(c) \neq 0$). Now for this scheme to work, it must happen that if x_0 is close to the fixed point c, the sequence x_n will converge to c. That may or may not happen, depending on g.

Here is a very simple function, called the logistic map, which is the most famous iteration. It depends on a parameter r.

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

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

I'll use several different values of the parameter and starting points, starting with $r = 2.5$ and $x_0 = 0.43$. I'll use a **for** loop to iterate 20 times. Note that since my function contains a floating point value and no symbolic constants such as π or functions such as **sin**, I don't have to worry about getting complicated symbolic values (in which case I'd need to put in **evalf**).

```
> r:= 2.5:
  X[0]:= 0.43:
  for count from 1 to 20 do
    X[count] := g(X[count-1])
  end do:
```

Here are the last few values:

```
> X[18], X[19], X[20];
```

$$0.5999998990, 0.6000000505, 0.5999999748 \quad \textbf{(1.2)}$$

We seem to be converging to 0.6.
I'd like to see all the values, but I don't want to type individually **X[1], X[2]**, etc. Instead, I can use the **seq** command, which makes a sequence of expressions, one for each value of an index variable. Its syntax is

seq(expression, index_variable = start_value .. end_value)

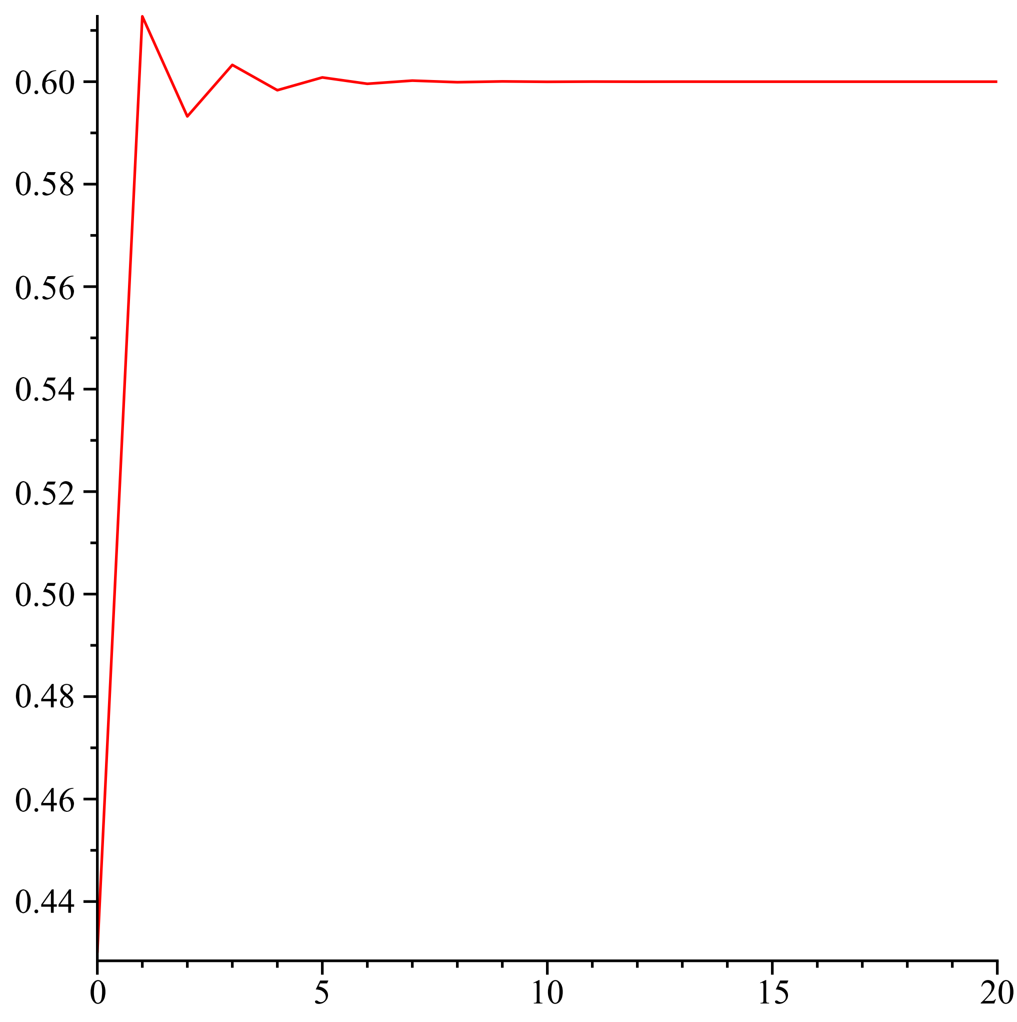
where **index_variable** is a name and **start_value** and **end_value** are integers. It starts with **index_variable** = **start_value**, and goes through all the integers from there to **end_value** one by one, evaluating **expression** with that value of **index_variable,** and returns the expression sequence of all the values obtained. In this case we'll get **X[0], X[1], ..., X[20].**

```
> seq(X[i], i=0..20);
```

0.43, 0.61275, 0.5932185938, 0.6032757345, 0.5983353068, 0.6008254185, 0.5995855875, 0.6002067770, 0.5998965045, 0.6000517210, 0.5999741328, 0.6000129320, 0.5999935335, 0.6000032332, 0.5999983835, 0.6000008082, 0.5999995960, 0.6000002020, 0.5999998990, 0.6000000505, 0.5999999748 **(1.3)**

Now let's plot these results. As we've seen, **plot** can plot a list of points. In this case the points I want are **[i, X[i]]**. I can use **seq** again to make a sequence of these points, and enclose the result in square brackets to make a list.

```
> plot([seq([i, X[i]], i = 0 .. 20)]);
```



It looks very much like 0.6 will be the limit of the sequence (although we haven't gone far enough to get X**[n+1]** = **X[n]**). Is that a fixed point of *g*?

```
> g(0.6) = 0.6;
```

$$0.600 = 0.6 \quad \textbf{(1.4)}$$

There was nothing very special about $x_0 = 0.43$. If you try other starting points, you'll see that the sequence converges to this same fixed point as long as $0 < x_0 < 1$. On the other hand, if $x_0 < 0$ or $x_0 > 1$ the limit is $-\infty$. For example:

```
> w[0]:= 1.4:
  for count from 1 to 20 do
    w[count] := g(w[count-1])
  end do:
> seq(w[i],i=0..20);
```

$$1.4, -1.400, -8.4000000, -197.4000000, -97910.40000, -2.396636084\ 10^{10}, -1.435966130\ 10^{21}, -5.154996818\ 10^{42}, -6.643498048\ 10^{85}, -1.103401658\ 10^{172}, -3.043738048\ 10^{344}, -2.316085326\ 10^{689}, -1.341062809\ 10^{1379}, -4.496123645\ 10^{2758}, -5.053781958\ 10^{5517}, -6.385178020\ 10^{11035}, -1.019262459\ 10^{22072}, -2.597239900\ 10^{44144}, -1.686413774\ 10^{88289}, -7.109978542\ 10^{176578}, -1.263794872\ 10^{353158} \quad \textbf{(1.5)}$$

Those are some huge negative numbers. What's the largest floating-point number that Maple can handle?

```
> Maple_floats(MAX_FLOAT);
```

$$1.\ 10^{2147483646} \quad \textbf{(1.6)}$$

```
> for count from 1 to 50 do
    w[count] := g(w[count-1])
  end do:
> seq([i,w[i]],i=21..50);
```

$$[21, -3.992943695\ 10^{706316}], [22, -3.985899838\ 10^{1412633}], [23, -3.971849380\ 10^{2825267}], [24, -3.943896875\ 10^{5650535}], [25, -3.888580640\ 10^{11301071}], [26, -3.780264848\ 10^{22602143}], [27, -3.572600580\ 10^{45204287}], [28, -3.190868725\ 10^{90408575}], [29, -2.545410805\ 10^{180817151}], [30, -1.619779042\ 10^{361634303}], [31, -6.559210362\ 10^{723268606}], [32, -1.075581014\ 10^{1446537214}], [33, -Float(\infty)], [34, -Float(\infty)], [35, -Float(\infty)], [36, -Float(\infty)], [37, -Float(\infty)], [38, -Float(\infty)], [39, -Float(\infty)], [40, -Float(\infty)], [41, -Float(\infty)], [42, -Float(\infty)], [43, -Float(\infty)], [44, -Float(\infty)], [45, -Float(\infty)], [46, -Float(\infty)], [47, -Float(\infty)], [48, -Float(\infty)], [49, -Float(\infty)], [50, -Float(\infty)] \quad \textbf{(1.7)}$$

Of course those are not really $-\infty$, it's just that they're larger than any number Maple can handle.

The fixed points

You don't really need Maple to see that if $x_0 = 0$ they stay at 0, i.e. 0 is a fixed point, while if $x_0 = 1$ we have $x_1 = 0$ and then all later $x_n = 0$ too.

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

$$0. \tag{2.1}$$

```
> g(1);
```

$$0. \tag{2.2}$$

It turns out that these are the only ways to get x_n to converge to 0.
By the way, how do we know 0 and 0.6 are the only fixed points?
The equation $g(x) = x$ is a quadratic equation, and a quadratic equation has at most two solutions.
Both 0.6 and 0 are fixed points, but they have very different properties as far as the iteration is concerned.
If x_0 is close to, but not exactly, 0, x_n will be close to 0 for a while, but eventually moves away toward either 0.6 or $-\infty$.
If x_0 is close to, but not exactly, 0.6, $x_n \to 0.6$ as $n \to \infty$.
Of the two fixed points, 0.6 is said to be an **attractor**, while 0 is a **repeller**.

Attractors and repellers

Here are the formal definitions:

- 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.

I don't want to get bogged down in complications. For us the main classifications of fixed points are attractors and repellers. We won't consider any examples that are neither one nor the other.

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);
```

$$2.5 \tag{3.1}$$

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

$$-0.50 \tag{3.2}$$

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

In the case of Newton's method:

```
> newt:= x -> x - f(x)/D(f)(x);
```

$$newt := x \to x - \frac{f(x)}{\mathrm{D}(f)(x)} \tag{3.3}$$

```
> D(newt)(p);
```

$$\frac{f(p)\,\mathrm{D}^{(2)}(f)(p)}{\mathrm{D}(f)(p)^2} \tag{3.4}$$

The fixed points of **newt** are the solutions of $f(p) = 0$. Assuming $f'(p) \neq 0$, this says that $newt'(p) = 0$. So this is an attractor; in fact, since the derivative is not just small but 0, it is a **superattractor**.

Proof of the theorem (for those interested)

First suppose $|g'(p)| < 1$. Take some number c so $|g'(p)| < c < 1$. Consider the secant line joining the points $[p, g(p)] = [p, p]$ and $[x_0, g(x_0)] = [x_0, x_1]$. It has slope $\dfrac{g(x_0) - g(p)}{x_0 - p} = \dfrac{x_1 - p}{x_0 - p}$.

By the definition of the derivative, $g'(p)$ is the limit of this slope as $x_0 \to p$.

Thus there is some $\delta > 0$ such that if $|x_0 - p| < \delta$, $\left|\dfrac{x_1 - p}{x_0 - p}\right| \le c$.

Since $0 < c < 1$, we have $|x_1 - p| \le c\,|x_0 - p| < c\,\delta < \delta$.

By mathematical induction we get $|x_n - p| < c^n\,\delta$ for all positive integers n, and in particular $x_n \to p$ as $n \to \infty$.

So p is an attractor in this case.

Now suppose $|g'(p)| > 1$. Take some number c so $|g'(p)| > c > 1$.

There is some $\varepsilon > 0$ such that if $|x_0 - p| < \varepsilon$, $\left|\dfrac{x_1 - p}{x_0 - p}\right| \ge c$, and so $|x_1 - p| \ge c\,|x_0 - p|$.

If N is an integer large enough that $c^N\,|x_0 - p| > \varepsilon$, there must be some $n \le N$ with $|x_n - p| \ge \varepsilon$.

This implies that p is a repeller.

In the case $|g'(p)| = 1$, consider these examples, which all have a fixed point at 0 and derivative 1 there:

$g_1(x) = \sin(x)$

$g_2(x) = \sinh(x) = \dfrac{e^x - e^{-x}}{2}$

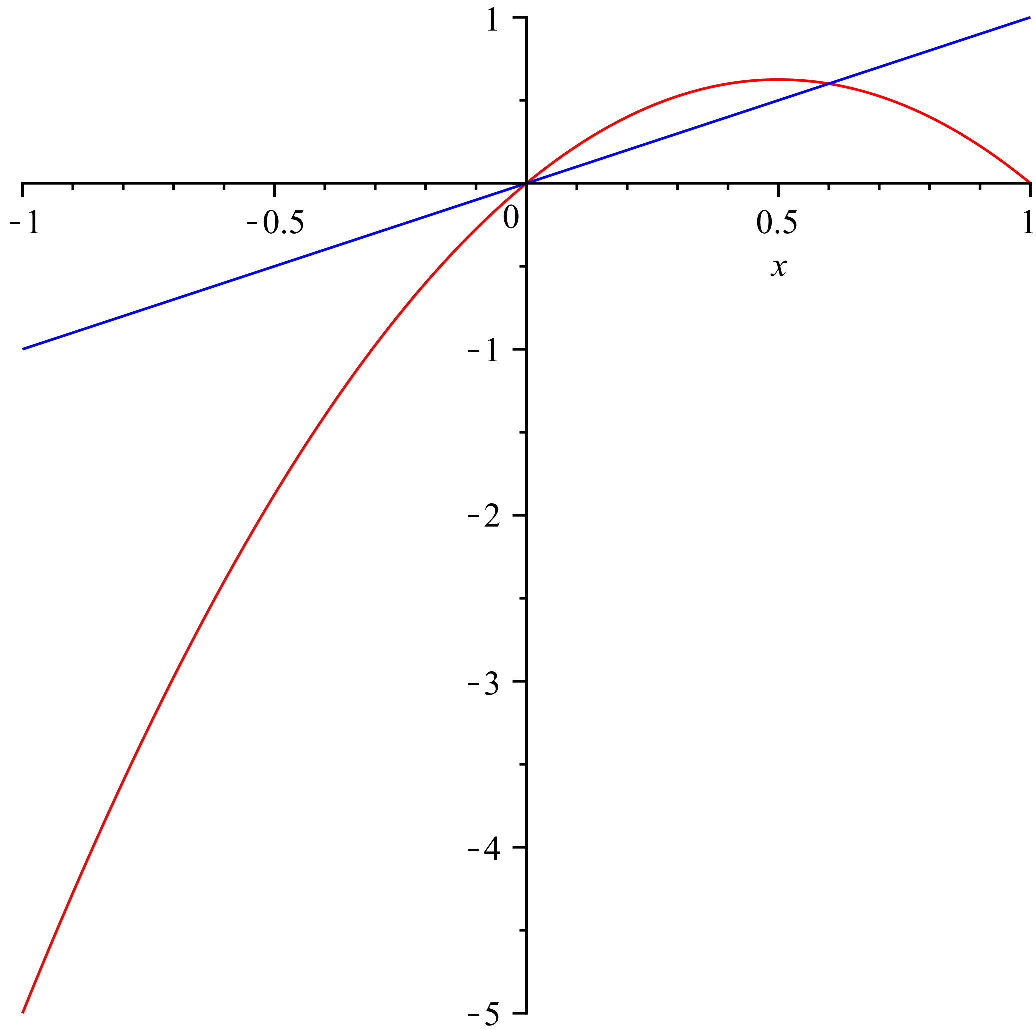
$g_3(x) = x$

It can be shown that 0 is an attractor for g_1, a repeller for g_2, and neither for g_3.

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 $y = x$. These intersect at the fixed points.

```
> Curves:= plot([g(x),x], x=-1..1, colour=[red,blue]):
  Curves;
```



Now, given any x_0, we'll plot the following sequence of points: $[x_0, x_0]$, $[x_0, x_1]$, $[x_1, x_1]$, $[x_1, x_2]$, ...
This starts on the diagonal, then moves up or down to the curve $y = g(x)$, then right or left to the diagonal, and continues in that way. 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)]);
```

$$stair := x \rightarrow ([x, x], [x, g(x)]) \quad \textbf{(5.1)}$$

Then using seq, we can put together as many of these units as we want.

```
> X[0]:= 0.43:
```

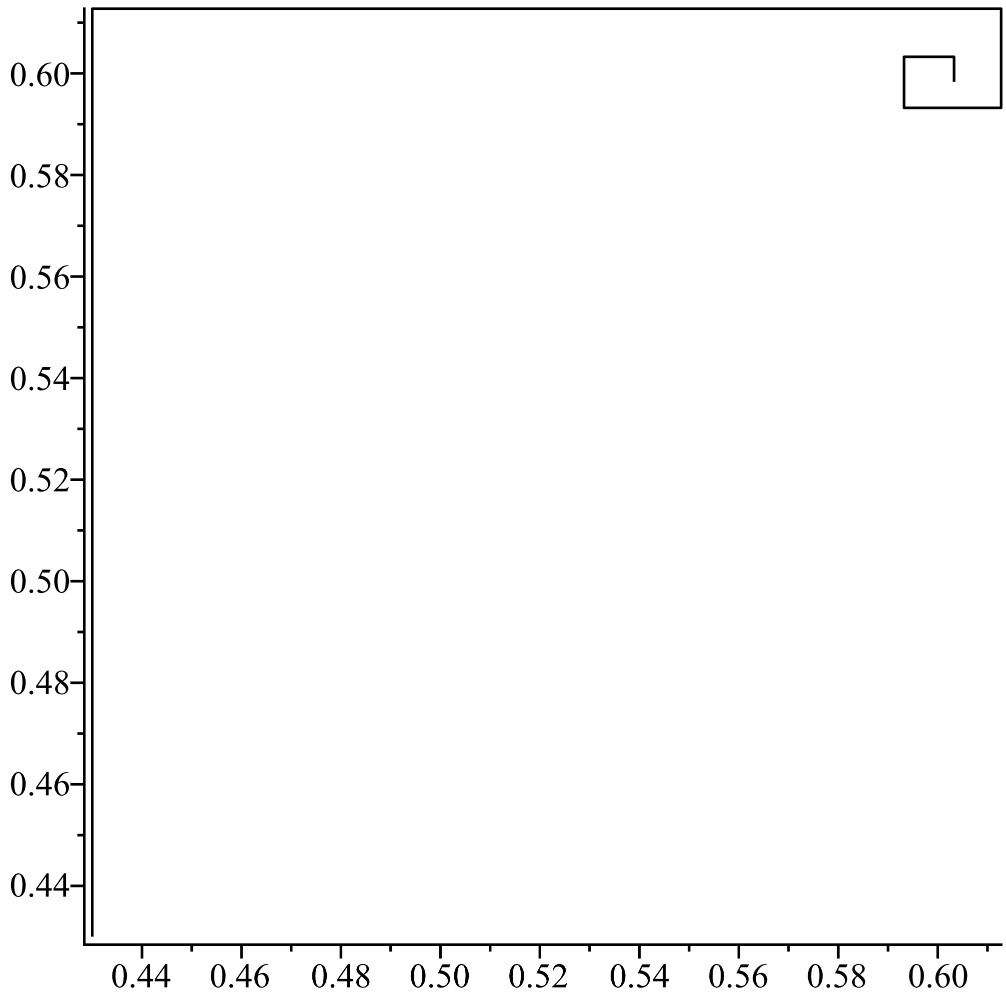
```
for count from 1 to 5 do
  X[count] := g(X[count-1])
end do:
```

```
> seq(stair(X[n]),n=0..3);
```

[0.43, 0.43], [0.43, 0.61275], [0.61275, 0.61275], [0.61275, 0.5932185938], [0.5932185938, 0.5932185938], [0.5932185938, 0.6032757345], [0.6032757345, 0.6032757345], [0.6032757345, 0.5983353068] **(5.2)**

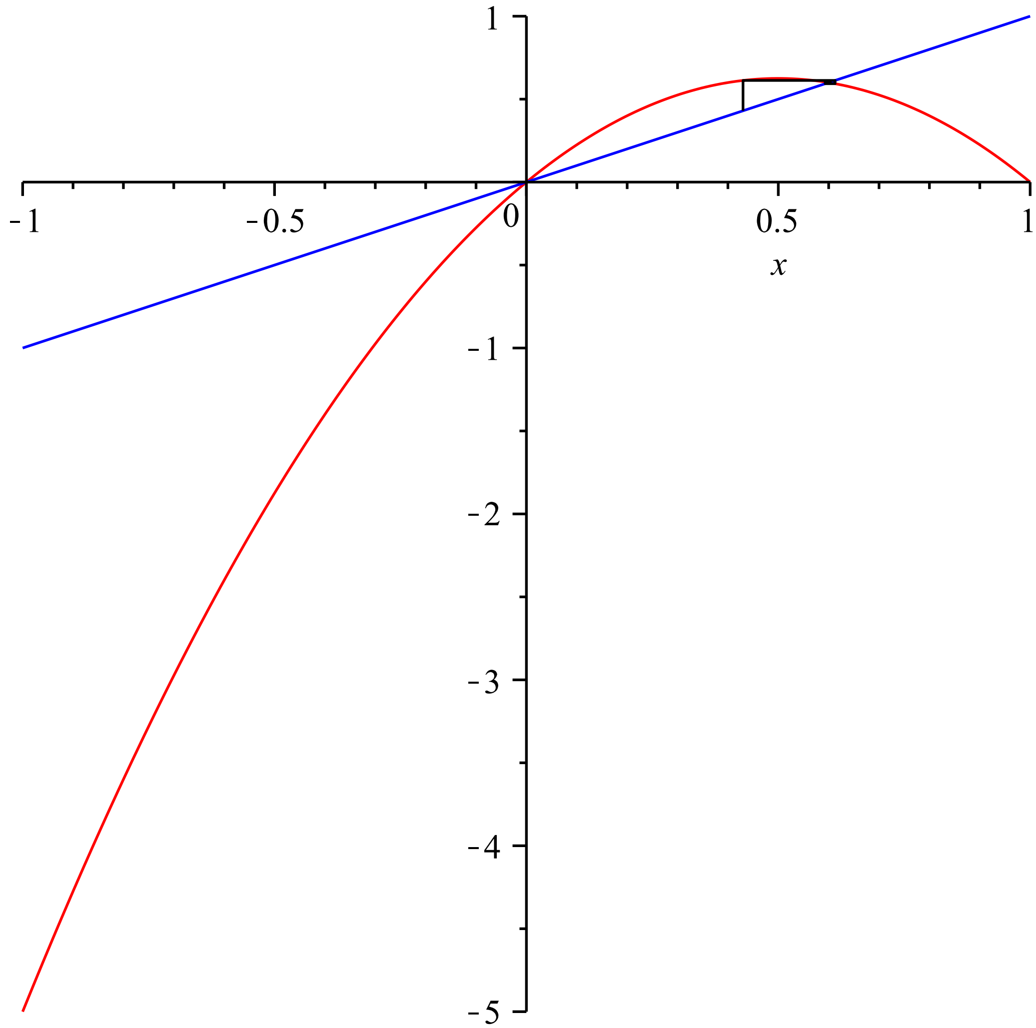
For the purpose of plotting, we make a list by putting square brackets around the result.

```
> Staircase:= plot([%], colour = black):
  Staircase;
```



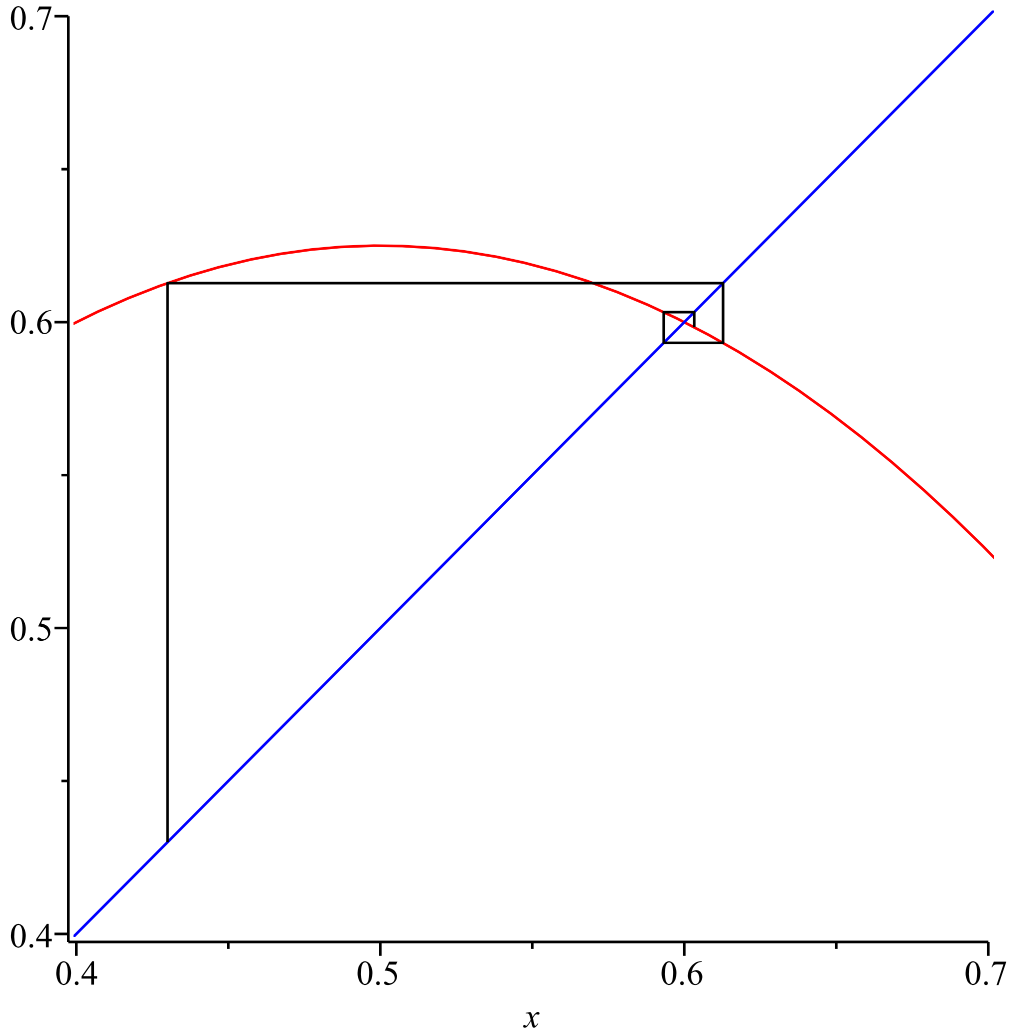
Now combine the two plots using **display**.

```
> with(plots):
  display([Curves,Staircase]);
```



It would be better to "zoom in" to get a better look at this.

```
> display([Curves,Staircase],view=[0.4 .. 0.7, 0.4 .. 0.7]);
```



Here's a more elaborate function for making the picture. Since it will involve several statements rather than just one expression, I can't use →; instead, I use **proc** (which stands for procedure).

- The inputs to the procedure 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'll also put dots at the first and last points computed (green for the first, red for the last). This can be done using plot with style=point and symbol=solidcircle.
- 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**.

```
> 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*) **(5.3)**

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*);

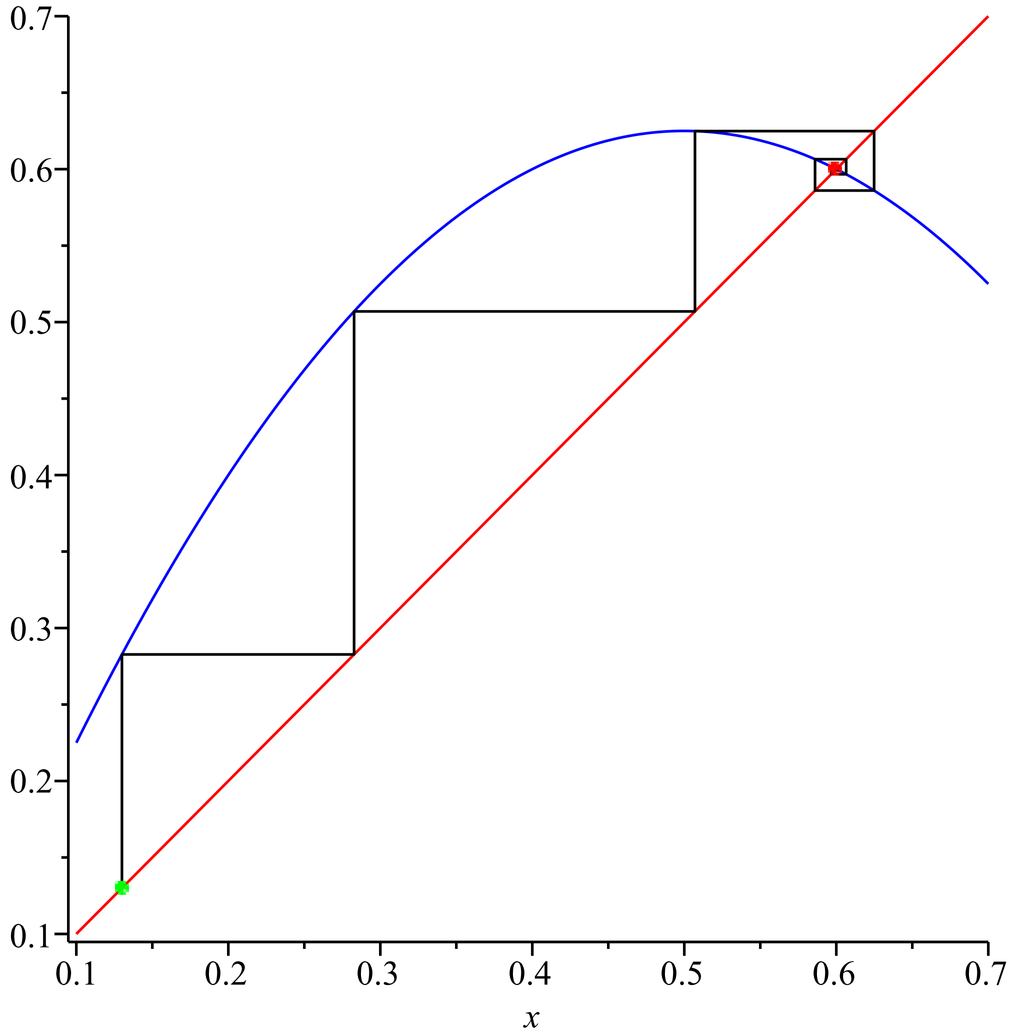
Dots := *plot*([[*x*[0], *x*[0]]], *style* = *point*, *symbol* = *solidcircle*, *colour* = *green*), *plot*([[*x*[*n* − 1], *g*(*x*[*n* − 1])]], *style* = *point*, *symbol* = *solidcircle*, *colour* = *red*);

plots:-*display*([*Curves*, *Staircase*, *Dots*])

end proc

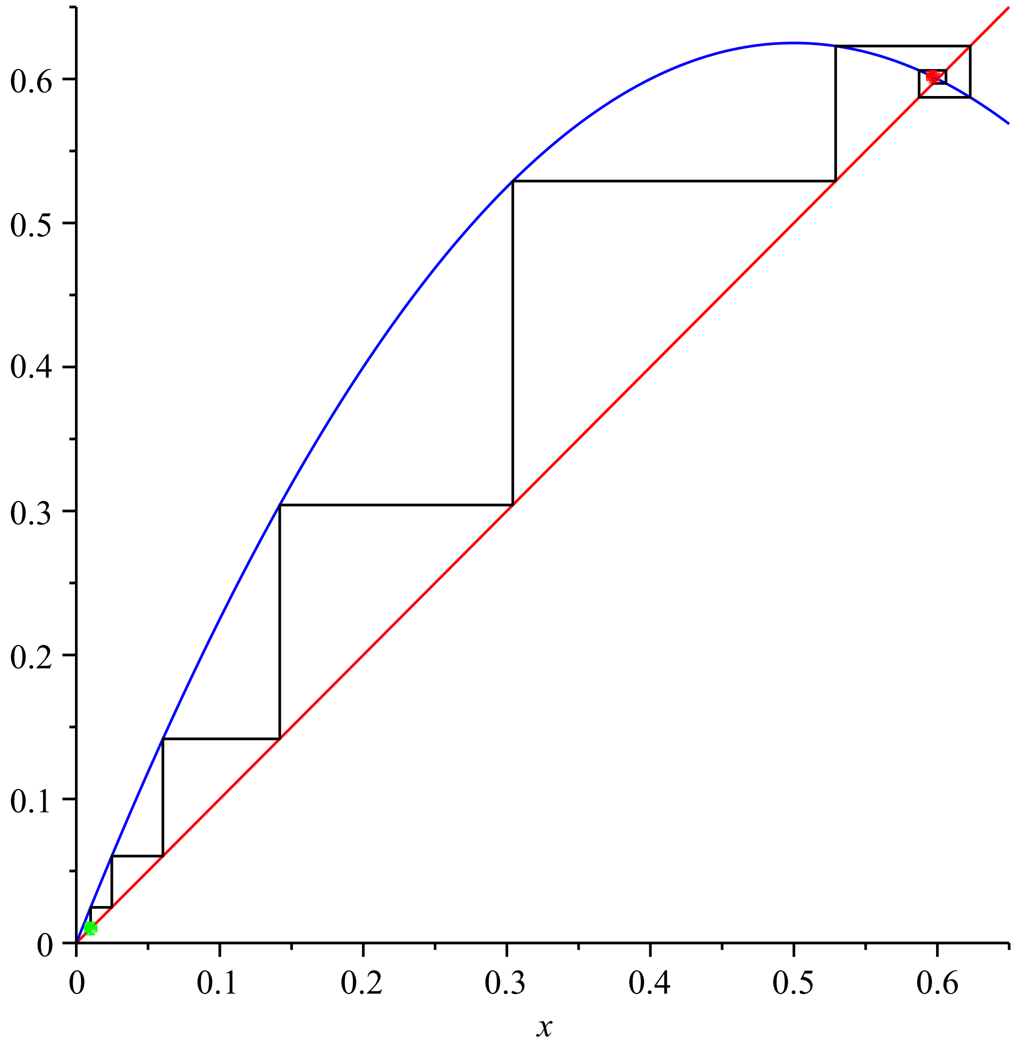
Here's the result, starting at $x = 0.13$. We see the staircase spiralling in to the fixed point at $x = 0.6$.

```
> staircase(0.13,0.1,0.7,9);
```



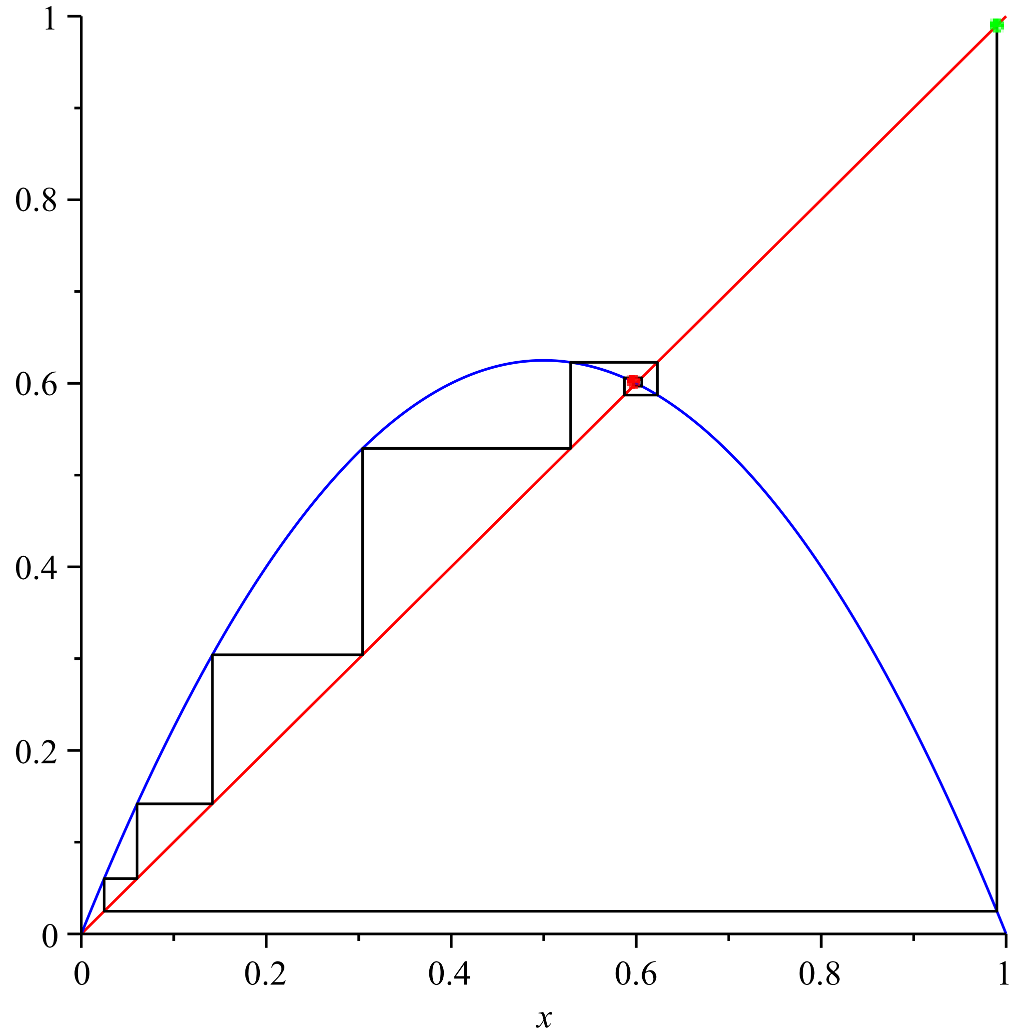
And here it is starting near the repeller $x = 0$, and moving away.

```
> staircase(0.01,0.0,0.65,10);
```



And here it is starting near (but slightly below) $x = 1$.

```
> staircase(0.99,0.0,1.0,10);
```



By looking at these, we can get an idea of why every initial point in the interval $(0, 1)$ is attracted to the fixed point 0.6.

An attracting 2-cycle

Now let's try another value of the parameter *r*.

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

$$r := 3.2 \quad (6.1)$$

What are the fixed points, and are they attractors or repellers?

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

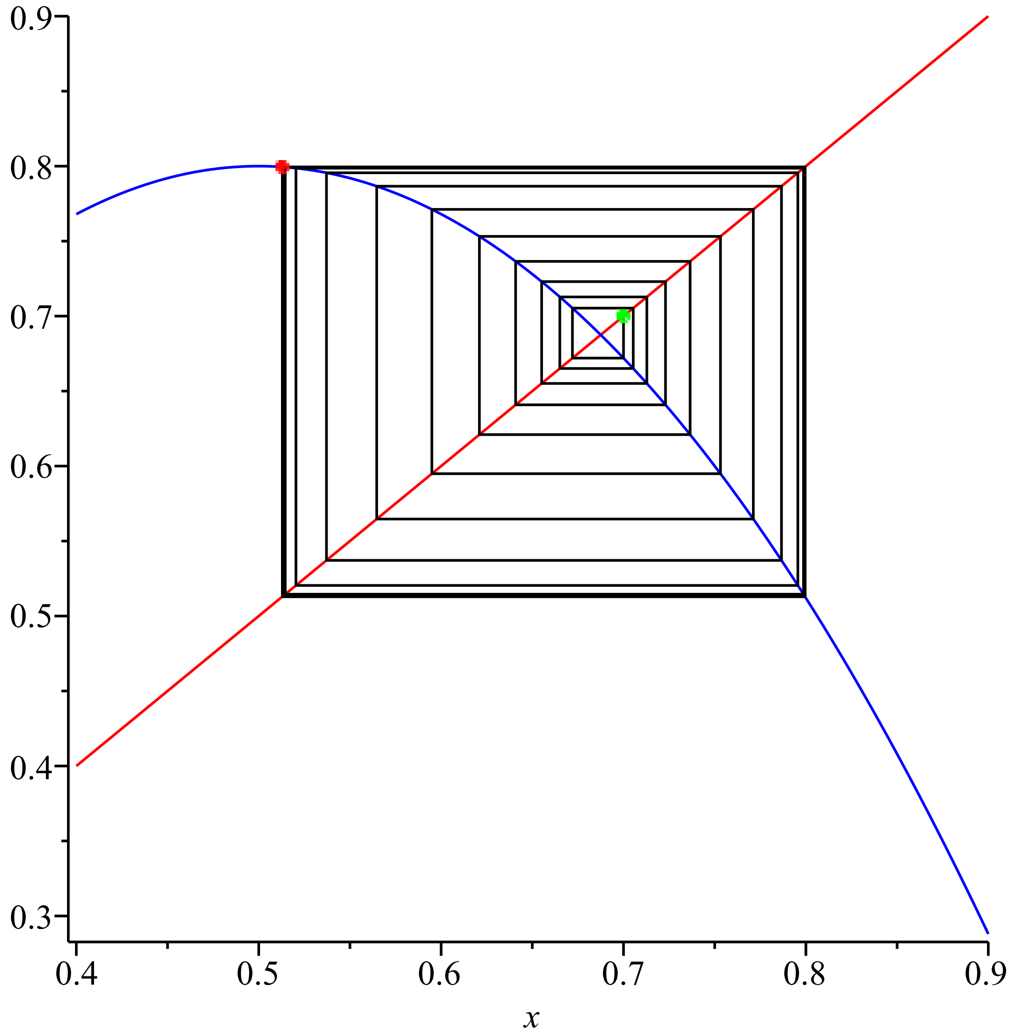
$$0., 0.6875000000 \quad (6.2)$$

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

$$3.2, \ -1.20000 \quad (6.3)$$

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

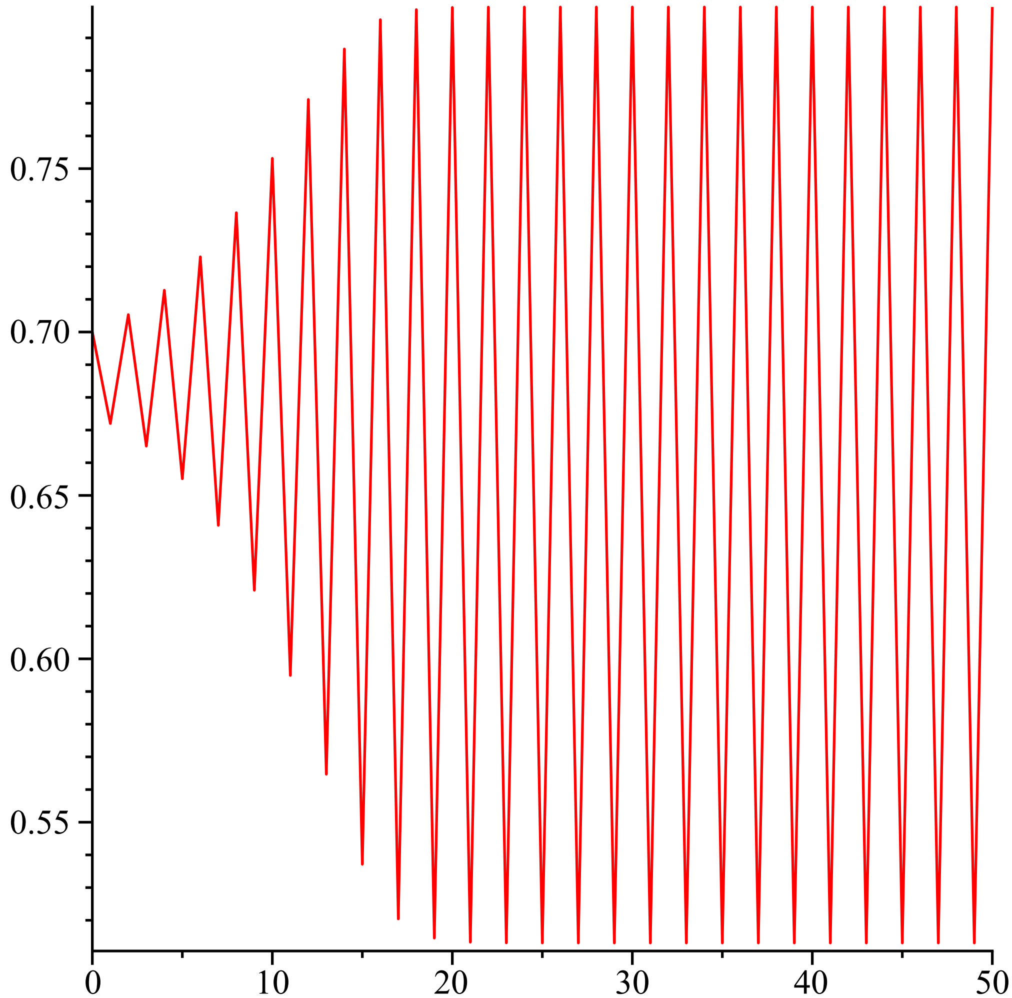
```
> staircase(0.7,0.4,0.9,50);
```



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.

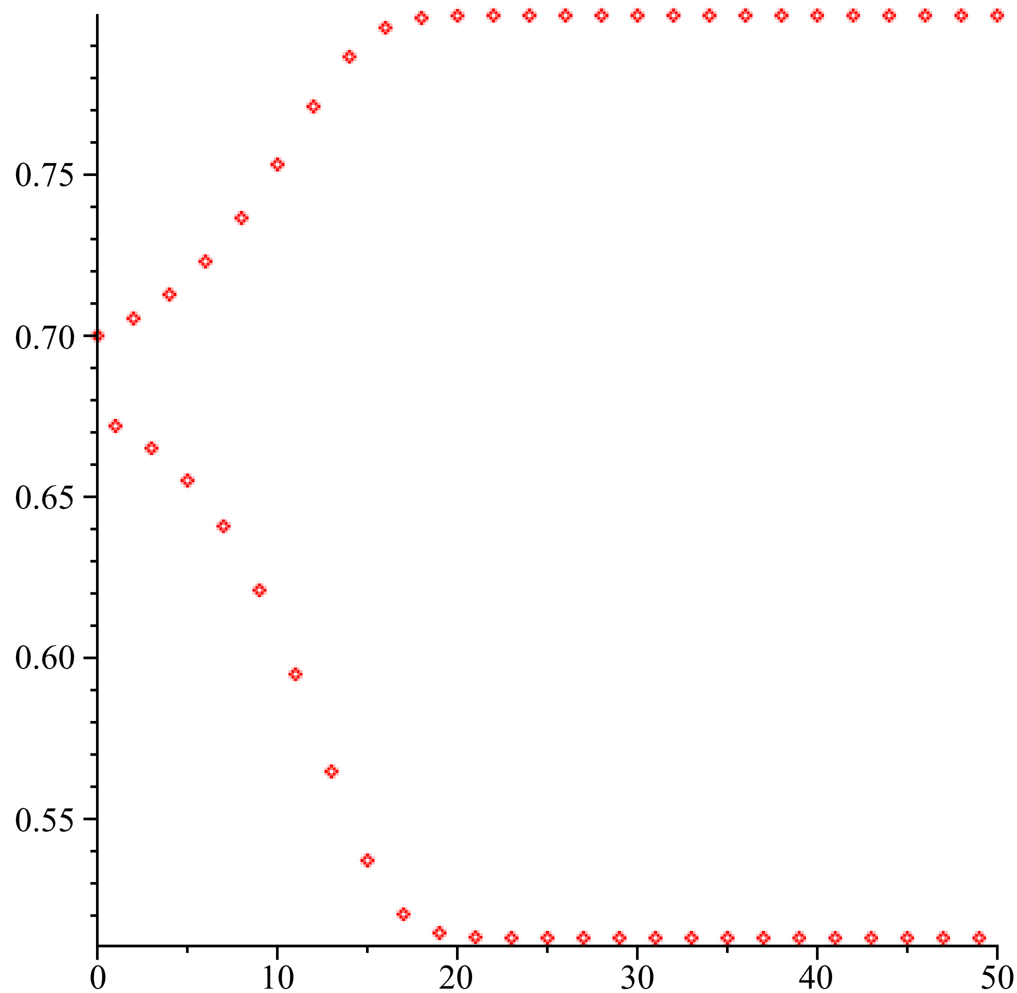
What if we plot the points $[i, x_i]$?

```
> 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)]);
```



It might be better to plot just the points, without the connecting lines. We can do that with the option **style=point**.

```
> plot([seq([i,X[i]],i=0..50)], style=point);
```



How can we 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});
```

$\{b = 0., c = 0.\}, \{b = 0.6875000000, c = 0.6875000000\}, \{b = 0.5130445095, c = 0.7994554905\}, \{b = 0.7994554905, c = 0.5130445095\}$ **(6.4)**

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);
```

$s := 0., 0.5130445095, 0.6875000000, 0.7994554905$ **(6.5)**

Again, two of the solutions are the fixed points, the other two are b and c. We can think of these all as fixed points of the composition of the function with itself, what we might write as $g \circ g$. In Maple input the composition sign is @, so this function could be written as `g @ g`.

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

$b := 0.5130445095$

$c := 0.7994554905$ **(6.6)**

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

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

$$0.1599999996, 0.1599999996 \qquad \textbf{(6.7)}$$

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);
```

$$D(G)(G(x))\, D(G)(x) \qquad \textbf{(6.8)}$$

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

$$D(G)(C)\, D(G)(B) \qquad \textbf{(6.9)}$$

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

$$D(G)(C)\, D(G)(B) \qquad \textbf{(6.10)}$$

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

$$0.1599999996 = 0.1599999996 \qquad \textbf{(6.11)}$$

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.
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 $|g'(b)\, g'(c)| > 1$.

Maple objects introduced in this lesson

plot (for list of points)
plots package
with
display
isolate
expand
factor
seq
Maple_floats(MAX_FLOAT)
Float(-infinity)