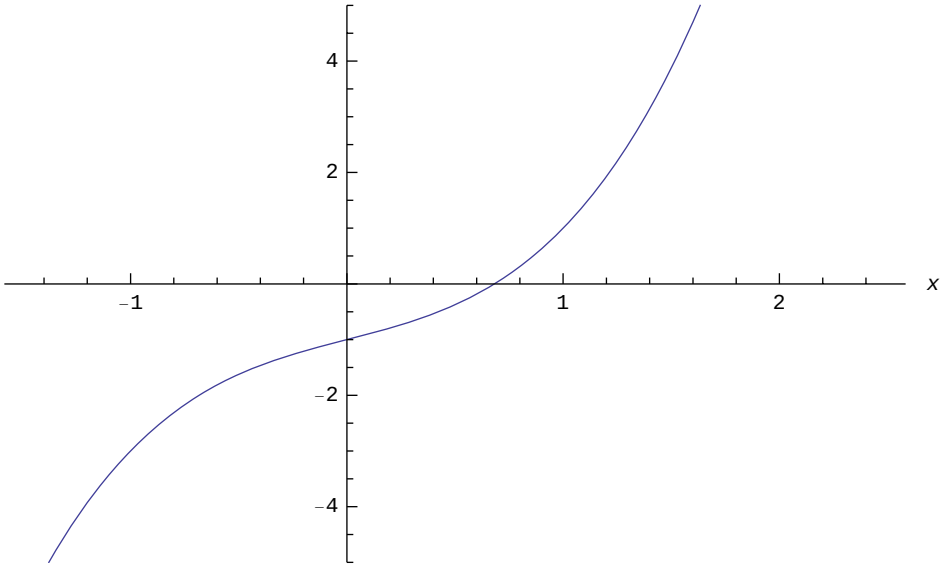
Fixed Point Iteration

We investigate the rate of convergence of various fixed point iteration schemes and try to discover what controls this rate of convergence and how we can improve it. We will discuss the following polynomial

```
f[x_] := x^3 + x - 1
```

Here is the graph of f

```
Plot[f[x], {x, -1.5, 2.5}, AxesLabel → {x}, PlotRange → 5]
```



So we see there is a root near 0.7. Here it is to lots of decimal places...

```
x0 = x /. FindRoot[f[x], {x, 0.7}]
```

```
0.6823278038280223
```

To solve $f(x) = 0$ iteratively we rearrange the equation in the form $x = g(x)$ and choose an initial estimate for x. If this x is a solution then applying g leaves it unchanged. If however applying g changes the value of x we apply g to this new value and continue this process until (hopefully) the value no longer changes, i.e. we find a fixed point. To illustrate this let's solve $x^3 + x - 1 = 0$ for the linear x term, $x = 1 - x^3$, and call the right side of this g

```
g[x_] := 1 - x^3
```

Starting with the initial estimate $x = 1$ we apply g to obtain

```
g[1]
```

```
0
```

Hmmm, maybe that wasn't such a good choice. Applying g again to this gives $g(0) = 1$ and we are going around in circles. Let's start a little closer to the actual root, say $x = 0.7$ ...

```
g[0.7]
```

```
0.657
```

```
g[0.657]
```

```
0.7164066069999999
```

Mathematica has a shortcut - we can use % for the last value ...

```
g[%]
```

```
0.6323126002509658
```

```
g[%]
```

```
0.7471892665664612
```

```
g[%]
```

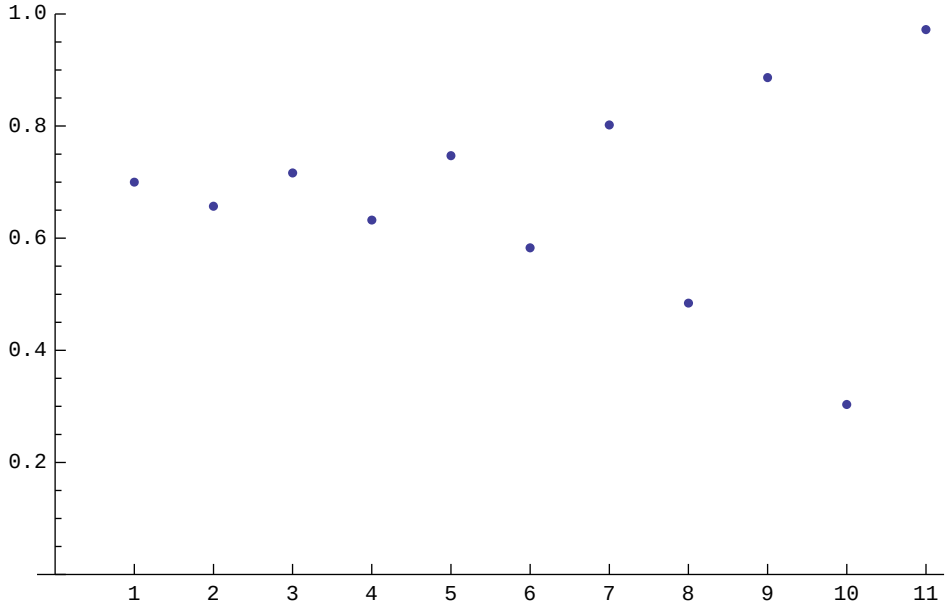
```
0.5828503593740386
```

This is getting tiresome. We can do a bunch of them at once ...

```
s = NestList[g, 0.7, 10]
```

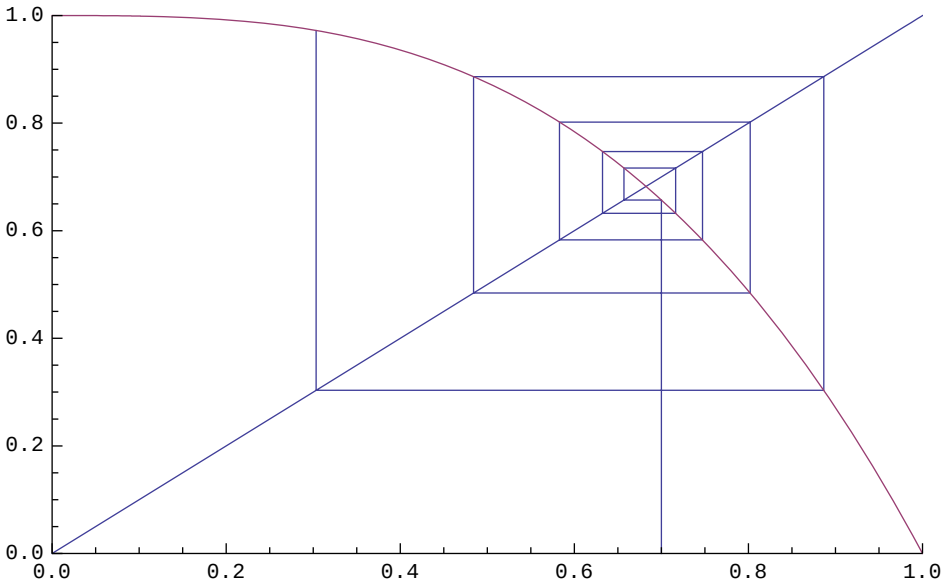
```
{0.7, 0.657, 0.7164066069999999, 0.6323126002509658,
 0.7471892665664612, 0.5828503593740386,
 0.8019972574473406, 0.4841556840444252,
 0.8865106510388125, 0.3032902736862523, 0.9721018471736975}
```

```
ListPlot[s, AxesOrigin → {0, 0}, PlotRange → {0, 1},
 Ticks → {Range[11], Automatic}, PlotStyle → PointSize[0.01]]
```



Here's another way to visualize the data - it's called a *Web diagram* (for obvious reasons) ... The graph shows $y = x$ and $y = g(x)$ and the iterations displayed in a special way

```
z =
  Flatten[Table[{{s[[k]], s[[k]]}, {s[[k]], s[[k + 1]]}}, {k, 10}], 1];
z[[1, 2]] = 0;
p = Plot[{x, g[x]}, {x, 0, 1}];
l = ListPlot[z, AxesOrigin → {0, 0},
   PlotRange → {{0, 1}, {0, 1}}, PlotJoined → True];
Show[
 l,
 p]
```



We seem to be moving away from the answer. Let's try instead solving for the first x in $x^3 + x - 1 = 0$, i.e. $x = g(x) = \sqrt[3]{1 - x}$. So we redefine g

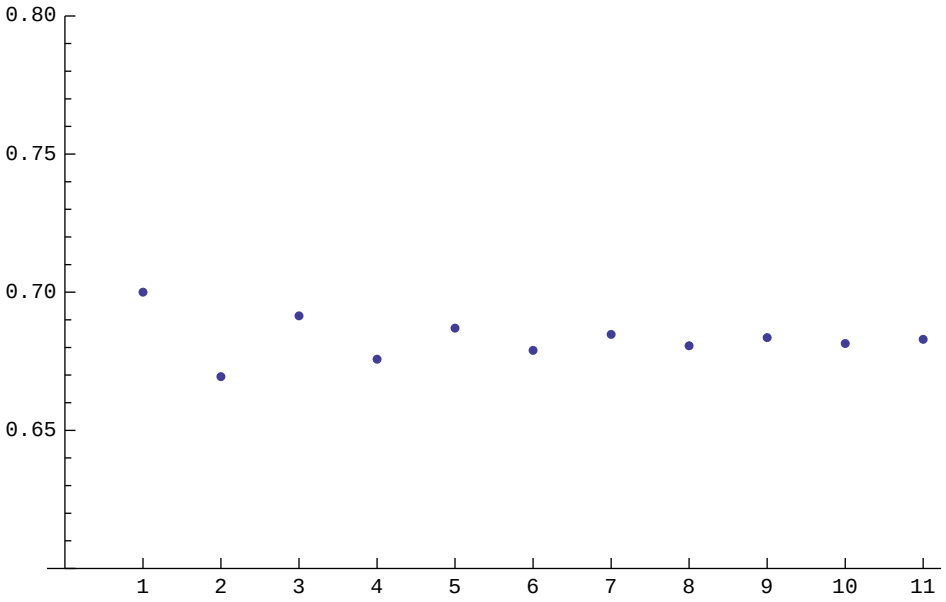
```
g[x_] := (1 - x) ^ (1 / 3)
```

and reevaluate the list of iterations...

```
s = NestList[g, 0.7, 10]
```

```
{0.7, 0.6694329500821695, 0.691437910132843,
 0.6757419111722766, 0.687010866239357, 0.678958276461744,
 0.6847317923031716, 0.6806022689687362,
 0.6835609985763513, 0.6814437329226418, 0.6829601825758527}
```

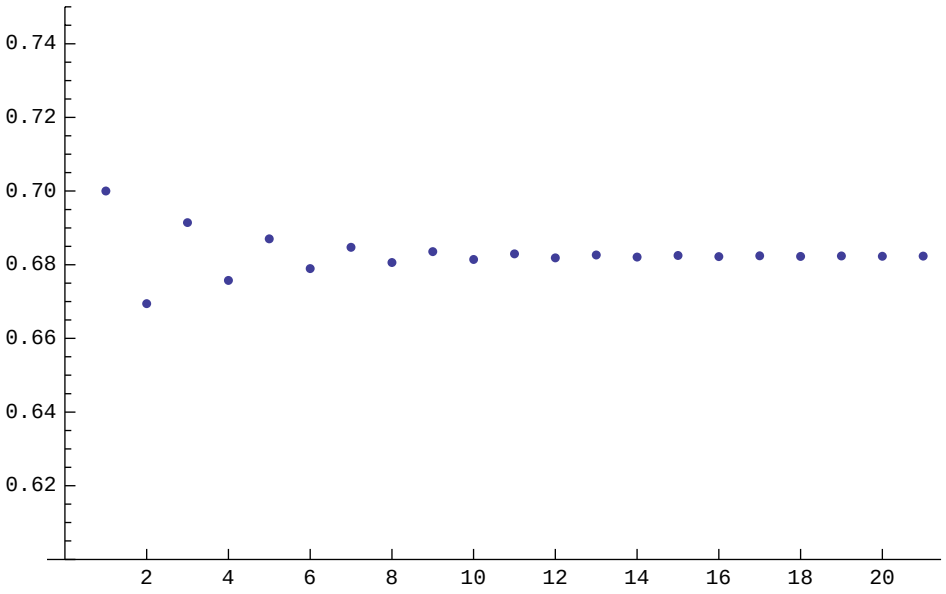
```
ListPlot[s, AxesOrigin → {0, 0.6}, PlotRange → {0.6, 0.8},
 Ticks → {Range[11], Automatic}, PlotStyle → PointSize[0.01]]
```



This is much better - the sequence of iterations really does seem to beconverging to something and that something must be a fixed point of g, i.e. a solution of our problem. Let's try 20 iterations ...

```
s = NestList[g, 0.7, 20];
```

```
ListPlot[s, AxesOrigin → {0, 0.6}, PlotRange → {0.6, 0.75},
 PlotStyle → PointSize[0.01], Ticks → {Range[2, 21, 2], Automatic}]
```



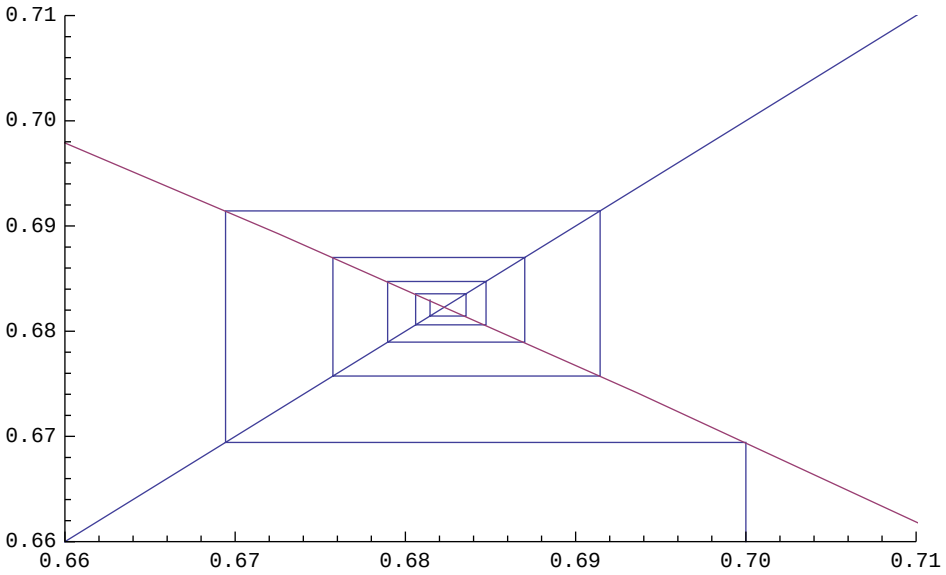
and the values of x and x_0 are

```
{s[[20]], x0} // TableForm
```

```
0.68229652955095
0.6823278038280223
```

so we have roughly four decimal places correct. Here's the web diagram... Notice how it "spirals in" this time, in contrast with the previous example

```
z =
  Flatten[Table[{{s[[k]], s[[k]]}, {s[[k]], s[[k + 1]]}}, {k, 10}], 1];
z[[1, 2]] = 0;
p = Plot[{x, g[x]}, {x, 0, 1}];
l = ListPlot[z, AxesOrigin → {0.66, 0.66},
   PlotRange → {{0.66, 0.71}, {0.66, 0.71}}, PlotJoined → True];
Show[l, p, DisplayFunction → $DisplayFunction]
```



Let's iterate until the sequence stops changing...

```
Short[FixedPointList[g, 0.7], 5]
```

```
{0.7, 0.6694329500821695, 0.691437910132843, 0.6757419111722766,
 0.687010866239357, 0.678958276461744, 0.6847317923031716,
 0.6806022689687362, <<85>>, 0.6823278038280187, 0.6823278038280198,
 0.6823278038280189, 0.6823278038280196, 0.6823278038280192,
 0.6823278038280195, 0.6823278038280193, 0.6823278038280194}
```

which took a total of 100 iterations (the <<85>> indicates 85 iterates *Mathematica* did not bother to display). So it worked, but it would be nice if it worked a little faster. For a single computation this is really not a big deal, but often in practice this is only a part of an algorithm which is executed many times so speed can be an issue. Let's try one more...

```
g[x_] := (1 - x) / x^2
```

```
s = NestList[g, 0.7, 10]
```

```
{0.7, 0.6122448979591839, 1.034444444444443, -0.03218880406478722,
 996.2067072892856, -0.001002800106598092, 995420.4287951194,
 -1.004599630941654 × 10^-6, 9.908648163279457 × 10^11,
 -1.009219404625647 × 10^-12, 9.818130842395861 × 10^23}
```

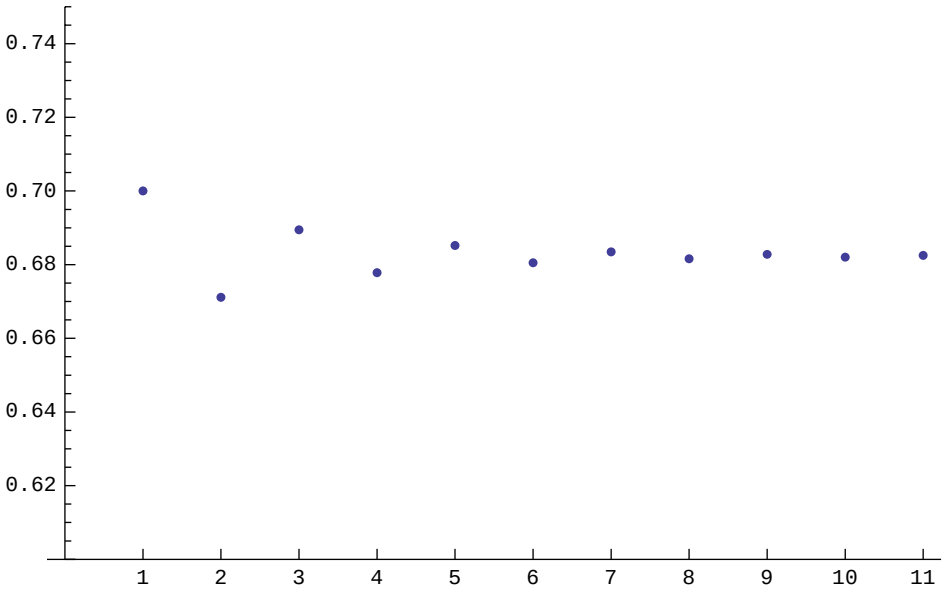
Ouch! How about...

```
g[x_] := 1 / (x^2 + 1)
```

```
s = NestList[g, 0.7, 10]
```

```
{0.7, 0.6711409395973155, 0.6894506381789385,
 0.677808858088373, 0.6852014348664703, 0.680503106621582,
 0.6834875326432596, 0.6815911466209261,
 0.6827959031162862, 0.6820304272167625, 0.68251675152682}
```

```
ListPlot[s, AxesOrigin → {0, 0.6}, PlotRange → {0.6, 0.75},
 PlotStyle → PointSize[0.01], Ticks → {Range[11], Automatic}]
```



That seems to be working so let's run it all the way...

```
Short[FixedPointList[g, 0.7], 5]
```

```
{0.7, 0.6711409395973155, 0.6894506381789385, 0.677808858088373,
 0.6852014348664703, 0.680503106621582, 0.6834875326432596,
 0.6815911466209261, <<59>>, 0.6823278038280182, 0.68232780382802,
 0.6823278038280188, 0.6823278038280197, 0.682327803828019,
 0.6823278038280195, 0.6823278038280193, 0.6823278038280194}
```

and this is a little better taking only 74 iterations.

This picking an iteration scheme and hoping for the best leaves a lot to be desired. Here is a class of iteration schemes for f

```
g[x_] := x - λ f[x]
```

Provided $\lambda \neq 0$ the fixed points of g correspond to the roots of f. With $\lambda = 1$ we get the sequence

```
λ = 1; s = NestList[g, 0.7, 10]
```

```
{0.7, 0.657, 0.7164066069999999, 0.6323126002509658,
 0.7471892665664612, 0.5828503593740386,
 0.8019972574473406, 0.4841556840444252,
 0.8865106510388125, 0.3032902736862524, 0.9721018471736975}
```

which is not working. What about

```
λ = 2; s = NestList[g, 0.7, 10]
```

```
{0.7, 0.6140000000000001, 0.9230489119999996, -0.495959876556741,
 2.739948527293546, -41.8792779519042, 146945.8262029948,
 -6.346024738845897 × 10^15, 5.111345971499598 × 10^47,
 -2.670765944041049 × 10^143, 3.810109743020192 × 10^430}
```

Ok, that's worse. What about

```
λ = 1 / 2; s = NestList[g, 0.7, 10]
```

```
{0.7, 0.6785, 0.6830721066875001, 0.6821795990519857,
 0.6823571787812093, 0.6823219762217286,
 0.6823289597388748, 0.6823275745438182,
 0.6823278493080569, 0.6823277948067403, 0.6823278056174518}
```

Actually that's not bad - we have 7 decimal places after just 10 iterations and we go all the way

```
Short[FixedPointList[g, 0.7], 4]
```

```
{0.7, 0.6785, 0.6830721066875001, 0.6821795990519857,
 0.6823571787812093, 0.6823219762217286, 0.6823289597388748, <<10>>,
 0.6823278038279977, 0.6823278038280236, 0.6823278038280185,
 0.6823278038280195, 0.6823278038280193, 0.6823278038280194}
```

in 22 steps, a big improvement over the previous choices. Now look at this one...

```
λ = 0.417238; FixedPointList[g, 0.7] // TableForm
```

```
0.7
0.682058766
0.682327742024517
0.6823278038280178
0.6823278038280193
0.6823278038280193
```

Now that's really impressive, convergence to 16 decimal places after just 4 iterations! The question is - what's up with 0.417238 that made this work so well? Setting $g'(x_0) = 0$ gives $\lambda = 1/f'(x_0) = 1/\left(3\,x_0{}^2 + 1\right)$ which in this example gives the value of λ used above. Unfortunately in practice we don't know what x_0 is (it's what we're trying to find!) so we use our current guess instead (hopefully we're not too far away anyway) - which means we look for the fixed points of the function

```
g[x_] := x - f[x] / f'[x]
```

Here is what the iteration does now...

```
FixedPointList[g, 0.7] // TableForm
```

```
0.7
0.6825910931174088
0.6823278630224631
0.6823278038280224
0.6823278038280193
0.6823278038280193
```

So the convergence is much the same. This is *Newton's method*.