1 Fixed Point Theorem

Given a function $G(x)$, we say that, $p$ is a fixed point of $G(x)$ when $G(p) = p$. It therefore follows that, given another function, $F(x)$, where $F(p) = 0$, we can write:

\[
\begin{align*}
G(p) & = p \\
F(p) & = 0 \\
G(p) & = p - F(p) \\
\Rightarrow G(x) & = x - F(x),
\end{align*}
\]  

written for a general point, $x$. The last equation in (1), is used for fixed point iteration.

1.1 Fixed Point Iteration

Since we know what a fixed point is, it is then a matter of writing the problem in a form of an iterative algorithm to solve for the root of the equation, $G(x)$. To illustrate, we will use the equations of the form:

\[
\begin{align*}
G(x) & = x^3 + x - 1, \\
G_1(x) & = x^3 - 1, \\
G_2(x) & = x,
\end{align*}
\]  

where we have split $G(x)$ into two functions, $G_1(x)$ and $G_2(x)$, so we form an equation of the form of (1). We now can write this in the form:

\[
x_{n+1} = \frac{1}{x_n^2 + 1}.
\]  

We implement finding the root using the difference the iteration calculate is away from zero, the error. This works as an stopping criterion. The final iteration displayed is therefore the first calculated root within the error tolerance stated.

```
clear;clc;format('long','g')
i = 1;
x(i) = 2;
error(i) = 1000;
while error(i) \geq 0.0001
    x(i+1) = (1 / ((x(i))^2 + 1));
    error(i+1) = ((abs((x(i+1) - x(i)))/abs((x(i+1)))))*100;
    i = i + 1;
end;
disp([x', error'])
```

To start the algorithm, a initial condition is set, $x(1) = 2$. This is an initial guess at the root, which allows the iteration to begin, similarly with the error(1) = 1000, which is arbitrarily larger than the expected tolerance. The algorithm ends when the iteration $i+1$ has an error less than a defined tolerance, here it is 0.0001, and will only depend on the level of precision required. Furthermore, the computation can be displayed graphically by plotting the trajectories, iteratively, on a graph of the functions $G_1(x)$ and $G_2(x)$. Note, to plot multiple trajectories, as has been done in the figure 1, a for loop is usually needed, see the MatLab code made available.
FIGURE 1: Example of plotting the solution graphically:
\[ g_2 = \frac{1}{x^2 + 1}; \]
\[ fplot(g_2, [0 \ (x(1) + 1)]); \]
hold on
straightLine = @(x) x;
fplot(straightLine, [0 \ (x(1) + 1)], 'g')
legend('g(x)=(x^2+1)^{-1}', 'f(x) = x', 'Location', 'NorthWest');
grid on
z(1) = x(1);
y(1) = z(1);
z(2) = z(1);
y(2) = g_2(z(2));
plot(z,y, 'r', z(1), y(1), 'r*')

1.2 Creating Movies in MatLab

Creating movies in MatLab uses a collection of single frames or ‘photos’, compiles them together, and cycles through them at a high frames per second (typically 12) which gives the effect of a seamless video. To do this, we collect a frame using the getframe command. Collecting frames in a for loop will yield many frames which can then be appended together into a matrix of frames using the writeVideo(writerObj, getframe) command. The only thing left to do is start the function with writerObj = VideoWriter('file_name'); open(writerObj); and close the function with close(writerObj). For example, lets have some physics.

Consider two waves, \(y_1, y_2\), which interfere with each other such that they result in a wave, \(y\), given by:

\[
\begin{align*}
y_1 &= \frac{1}{2} \sin (2\pi n x - t), \\
y_2 &= \frac{1}{2} \sin (2\pi n x + t), \\
y &= y_1 + y_2.
\end{align*}
\]

Iterating over series of time points, \(t\), over different harmonics, \(n \in \mathbb{Z} : n \geq 1\), we produce a series of images, one of which is shown in figure 2. The default output is a .avi video file, now you have a movie, yay!

FIGURE 2: The \(n = 2\) harmonic of a standing wave. Example MatLab code:

```matlab
writerObj = VideoWriter('StandingWaves');
open(writerObj);
x=0:0.01:0.5;
for n=1:1:5
    for t=0:0.05:2*pi
        y1=0.5*sin(2*pi*n*x-1*t);
        y2=0.5*sin(2*pi*n*x+1*t);
        y = y1+y2;
        plot(x,y,'b');
        a = getframe;
        writeVideo(writerObj,a);
    end;
end;
close(writerObj);
```