Chapter 1

Root Finding Methods

We begin by considering numerical solutions to the problem

\[ f(x) = 0 \quad (1.1) \]

Although the problem above is simple to state it is not always easy to solve analytically. This is usually the case for instance when nonlinear functions are involved such as \( f(x) = \exp(x) - x \).

Furthermore, there are many other types of mathematical problems which can be formulated in such a way so that they fall under this same “root-finding” category. Systems of equations \( AX = b \), where \( A \) denotes a matrix and \( b \) a vector, for instance can be re-formulated so that they can also be considered to be of the form (1.1). As a result therefore the numerical methods which we will consider in this chapter are useful for a variety of mathematical problems.

There are a number of techniques for solving (1.1) some of which will be presented here. Note however that not all the methods presented to solve (1.1) will produce a solution 100% of the time. The success or failure of the method used to solve a particular problem will greatly depend on information we know regarding the function \( f(x) \) as well as limitations of the specific method used. It is important therefore that we have a good understanding of both the type of problem we wish to solve as well as the numerical methods available for that type of problem.

The schemes which we will shortly examine fall under the following two basic categories:

- Two-point methods
- One-point methods

The distinction simple refers to the number of initial data that we must provide to the scheme in order to start the iterations and obtain an approximation to the root.

There are many other characterizations differentiating each of the methods we will consider here. The reader is prompted to pay attention to terms such as the “speed of convergence” or terms like “method is encompassing” while reading about the advantages or disadvantages of each new method introduced.

1.1 Bisection method

As mentioned previously the difference between two-point and one-point methods is how many initial values we provide to the method (two or one) so that we can start the iterations. In this section we examine numerical schemes which require two different starting guesses.
Many different root finding schemes fall under the category of two-point methods. Among them are the so called *enclosure* or *bracketing* methods. These methods are unique among general two-point methods since they require that the two initial guesses, say \( x_0 \) and \( x_1 \), satisfy the following condition:

\[
f(x_0)f(x_1) \leq 0
\]

i.e. \( f(x_0) \) and \( f(x_1) \) have opposite sign. The *bisection method* which we consider next is such a two-point enclosure method.

This method therefore falls under the category of two-point enclosure methods. The bisection method requires two starting guesses, \( x_0 \) and \( x_1 \) as well as the condition that \( f(x_0)f(x_1) \leq 0 \) in order to obtain the desired roots. Note that it is this later condition \( f(x_0)f(x_1) \leq 0 \) which guarantees that the method is encompassing. The method is implemented as follows:

1. Obtain two starting guesses, \( x_i \) and \( x_j \) such that \( f(x_i)f(x_j) \leq 0 \).
2. Produce the next iterate \( x_k \) by obtaining the mid-point
   \[
x_k = \frac{x_i + x_j}{2}
\]
3. If \( f(x_k) = 0 \) or required tolerance has been reached then stop. Problem solved. The root is \( x_k \).
4. Otherwise continue as follows:
   - If \( f(x_k) \) has the same sign as \( f(x_i) \) redo step 2 above by assigning \( x_i = x_k \).
   - If \( f(x_k) \) has the same sign as \( f(x_j) \) redo step 2 above by assigning \( x_j = x_k \).

Note that in step 3 above the algorithm may stop even though \( f(x_k) \neq 0 \) just because the required tolerance has been reached. In general it is a good advice to always have a safe guard in your algorithms which will eventually stop the program from running for ever. Such safe guards normally consist of a maximum number of iterations after which the code should stop and a minimum tolerance value. The tolerance in the example above would normally be a measure of how close your solution \( f(x_k) \) is close to zero. If close enough then the code should stop.

*Example:*
Find the root of the following function \( f(x) = x^3 - 3x + 1 \) within an accuracy of \( 10^{-7} \) in the interval \([0, 1]\).

*Solution:*
Based on the prescribed tolerance our algorithm must stop once we have produced an iterate \( x_n \) for which

\[
|f(x_n)| < 10^{-7}
\]

The true root can be easily calculated with the matlab `fsolve` function to be approximately \( 0.347296353164248 \). Note that this type of information is not always available in reality. Knowing the root however will help us calculate the exact error of our approximation as well.

Applying the bisection algorithm above we obtain the following table of results:
Therefore the root of $f(x) = x^3 - 3x + 1$ is approximated to be at $x = .3472967$ with an error of approximately $0.000000362$.

We can obtain the following result on the rate of convergence of the bisection method:

**Theorem 1.1.1.** Suppose $f(x)$ is continuous on $[a, b]$ and $f(b)f(a) \leq 0$. Then after $n$ steps the computed root will insure a maximum possible error of $\frac{(b-a)}{2^{n+1}}$.

Note that the theorem implies that at the very worst case the bisection method converges to the root at the same rate as the sequence $\{2^{-n}\}$ converges to zero. Depending on the given problem the method may converge even faster.

The bisection method will work under most circumstances but its convergence rate may be slow. As a result when programming the method, and any other method which you might program for that matter, you should allow a maximum number of iterates after which the code should terminate anyway even though the required tolerance has not been reached yet. This will prevent the algorithm from getting into infinite loops as well.

Let us now provide a quick overview of advantages and possible disadvantages of the bisection method. Note that the list below and other similar lists throughout the text are not intended to be complete and in some cases may require further arguing.

**Advantages:**

- Always converges to the root (this is not the case for some other methods which we will encounter)!
- Simple and easy to implement.

**Disadvantages:**

- It may converge slowly to the root (especially if we use complicated functions or large starting intervals).
- An early iteration which is actually quite close to the root may be easily discarded.

### 1.2 Newton Method

This is probably the most well known method for finding function roots. The method can produce faster convergence by cleverly implementing some information about the function $f$. Unlike the bisection method, Newton’s method requires only one starting value and does not need to satisfy any other serious conditions (except maybe one!). This is a one-point method.
Let us start by presenting one of the ways that we can actually obtain such a numerical root finding scheme. We start by writing the Taylor polynomial for \( f(x) \) of first order for \( x \) near \( x^* \):

\[
f(x) = f(x^*) + (x - x^*)f'(x^*) + \frac{(x - x^*)^2}{2}f''(\xi(x))
\]

where as usual \( \xi(x) \) lies between \( x \) and \( x^* \). Let us denote the root we are trying to find by \( r \). Thus \( f(r) = 0 \). Therefore,

\[
0 = f(x^*) + (r - x^*)f'(x^*) + \frac{(r - x^*)^2}{2}f''(\xi(r)) \quad (1.2)
\]

Assuming now that we take \( x^* \) close enough to the true root \( r \) then the term \( (r - x^*) \) is very small and further \( (r - x^*)^2 \) is almost zero! Thus we can disregard the last term in (1.2). As a result, under this assumption, we obtain the following scheme,

\[
0 = f(x^*) + (r - x^*)f'(x^*).
\]

Therefore rewriting the above and solving for the root \( r \) we obtain the famous Newton method,

\[
r = x^* - \frac{f(x^*)}{f'(x^*)}
\]

This gives the idea of how we can iterate an initial guess in order to obtain the root of \( f \):

\[
x_n = x_{n-1} - \frac{f(x_{n-1})}{f'(x_{n-1})}
\]

The following theorem, without proof, gives a little bit more insight on the conditions under which we expect the method to converge to the root of the function \( f(x) \).

**Theorem 1.2.1.** Suppose \( f \in C^2([a,b]) \). If the root \( r \) is in \([a,b]\) and \( f'(r) \leq 0 \) then there exists \( \delta > 0 \) such that the Newton method produces a sequence of iterates \( \{x_n\}_{n=1}^\infty \) which converges to \( r \) for any initial approximation \( x_0 \) which is near the root \([r - \delta, r + \delta]\).

That is if we choose the initial guess \( x_0 \) close enough to the root of the function \( f(x) \) we are essentially guaranteed that we will find the true root!

In short the method is implemented as follows:

1. Obtain a starting guess \( x_0 \) and find a formula for the derivative \( f' \) of \( f \).

2. Produce the next iterate \( x_n \) through the following equation:

\[
x_n = x_{n-1} - \frac{f(x_{n-1})}{f'(x_{n-1})}
\]

3. Stop if any of the following occurs: required tolerance is reached, maximum iterates exceeded, \( f'(x_{n-1}) = 0 \).

**Advantages:**

- Fast convergence!
Disadvantages:

- Calculating the required derivative itself $f'(x_{n-1})$ for every iteration may be a costly task for some functions $f$.
- May not produce a root unless the starting value $x_0$ is close to the actual root of the function.
- May not produce a root if for instance the iterations get to a point $x_{n-1}$ such that $f'(x_{n-1}) = 0$. Then the method fails!

1.3 Regula Falsi Method

This is another two-point encompassing method. However it differs from the bisection method since it uses some information from the function $f(x)$ in order to faster predict its root much like Newton’s method. In essence this is a clever remake of the bisection and Newton’s method we just encountered. As you might recall for instance the bisection method always divides in half the given interval in order to obtain the next iterate closer to the root. Instead regular falsi tries to cleverly divide the interval closer to the root!

Let’s take an example in order to outline the thinking behind this new method: We let for instance $x_0 = 2$ and $x_1 = 10$ for the function $f(x) = x^2 - 6$ and we wish to approximate its root in that interval $[2, 10]$. Then we have that $f(2) = -2$ while $f(10) = 94$ and note that the condition $f(x_0)f(x_1) < 0$ is satisfied. Here the bisection method, which does not use any information from the function $f$, would produce the next iterate at $x_2 = 6$. Whereas just looking at the two values for $f$ we would expect the root to be much closer to $x_0 = 2$. The regula falsi method instead takes into account precisely that information about the function at the end points in order to make an educated guess as to where it is most likely that the root ought to be closer to. Regula falsi weighs that information in the calculation and produces the next iterate much closer to $x_0 = 2$. In gives that $x_2 = 2.16$. Note the difference between the two methods in just one iteration:

- Bisection method $x_2 = 6$,
- Regula Falsi method $x_2 = 2.16$,
- True root $x = 2.4496$

As you can see regula falsi got a lot closer to the true solution in just one iteration.

Let’s us then describe in detail how the method is implemented. In essence the algorithm which we provide below is the same as the bisection method except for the main formula. The pseudo-code follows:

1. Obtain two starting guesses, $x_i$ and $x_j$ such that $f(x_i)f(x_j) \leq 0$.
2. Produce the next iterate $x_k$ from the following formula

$$x_k = x_i - f(x_i) \frac{(x_j - x_i)}{f(x_j) - f(x_i)}$$

3. If required tolerance or maximum iterates have been reached then stop.
4. Otherwise continue as follows:
   If \( f(x_k) \) has the same sign as \( f(x_i) \) redo step 2 above by assigning \( x_i = x_k \).
   If \( f(x_k) \) has the same sign as \( f(x_j) \) redo step 2 above by assigning \( x_j = x_k \).

Although this method is more clever than the bisection method it also have its drawbacks. In fact it converges very slowly for certain functions which display high curvature. Therefore there exists a modification which allows a small speed up for such type of functions. We will present this method in the next section.

Advantages:

• No need to calculate a complicated derivative (as in Newton’s method).

Disadvantages:

• May converge slowly for functions with big curvatures.
• Newton’s method may be still faster if we can apply it.

1.4 Tolerance criteria

Let us also discuss briefly how we should measure tolerance limits. Measuring when a given iterate is “close enough” to the true root is important in its own right since it influences significantly the number of times a given method must be performed.

There are a number of different ways to measure how close an iterate \( x_n \) is to a root \( r \) for a given function \( f(x) \). We may wish for instance to just measure how close \( f(x_n) \) is to the true root \( f(r) \). Note that since \( f(r) = 0 \) then we can simply measure

\[
|f(x_n) - f(r)| = |f(x_n)| \leq \text{TOL}
\]

where TOL denotes a given predefined tolerance. This is the typical way to measure absolute error.

Similarly we might instead wish to measure whether the iterates themselves are converging sufficiently. This is another form of absolute error,

\[
|x_n - x_{n-1}| \leq \text{TOL}
\]

Another possibility is to have a bit more impartial way of measuring differences by using something called the relative error

\[
\frac{|x_n - x_{n-1}|}{|x_n|} \leq \text{TOL}
\]

Thus, in this later formulation, we try to counter the size of the values of the iterates by “normalizing” them. In essence therefore the above fraction should be less than one when the iteration is converging.

In all cases, as also mentioned before, we should not simply rely on the tolerance computation provided above in order to terminate an iteration. We should always include a secondary mechanism in our iterations which terminates the method once a maximum number of iterations has been performed.
1.5 Modified Regula Falsi Method

As mentioned above the problem of slow convergence rates appears when the function in question \( f \) has high concavity. In this case the usual regula falsi method produces iterates only on one side of the actual root at a slow rate. So to speed things up the modified regula falsi method tries to force the scheme to produce iterates on the other side of the root. This is achieved by changing the value of \( f(x) \) to half: \( f(x)/2 \). In more detail the method is applied as follows:

1. Obtain two starting guesses, \( x_i \) and \( x_j \) such that \( f(x_i)f(x_j) \leq 0 \).

2. Produce the next iterate \( x_k \) from the following formula

\[
x_k = x_i - f(x_i) \frac{(x_j - x_i)}{f(x_j) - f(x_i)}
\]

and let \( f_i = f(x_i) \), \( f_j = f(x_j) \) and \( f_k = f(x_k) \). Let us also denotes the previous iterate at \( x_{k-1} \) as \( f_{k-1} \).

3. If required tolerance or maximum iterates have been reached then stop.

4. If \( f_i f_k > 0 \) then go to step 6

5. (Divide \( f_i \) in half):
   - Let \( x_j = x_k \) and \( f_j = f_k \).
   - If \( f_{k-1} \cdot f_k > 0 \) then \( x_{k-1} = x_k, f_{k-1} = f_k \) and \( f_i = f_i/2 \). Go to step 2.

6. (Divide \( f_j \) in half):
   - Let \( x_i = x_k \) and \( f_i = f_k \).
   - If \( f_{k-1} \cdot f_k > 0 \) then \( x_{k-1} = x_k, f_{k-1} = f_k \) and \( f_j = f_j/2 \). Go to step 2.

Let us now give an example of how the usual and the modified regula falsi method fairs in practice:

**Example:**
Suppose \( f(x) = x^2 - 6 \) and we wish to obtain the root of the function in the interval \([2, 3]\) with an error not to exceed \(10^{-5}\).

**Solution:**
We start as usual by setting \( x_0 = 2 \) and \( x_1 = 3 \). We first implement, for comparison purposes the usual Regula Falsi method:

\[
\begin{array}{|c|c|c|c|c|c|}
\hline
n & a & b & c & f(c) & \text{Rel. Error} \\
\hline
1 & 2.000000 & 3.000000 & 2.400000 & -0.240000 & 0.166667 \\
2 & 2.400000 & 3.000000 & 2.444444 & -0.024691 & 0.018182 \\
3 & 2.444444 & 3.000000 & 2.448980 & -0.002499 & 0.001852 \\
4 & 2.448980 & 3.000000 & 2.449438 & -0.000252 & 0.000187 \\
5 & 2.449438 & 3.000000 & 2.449485 & -0.000026 & 0.000019 \\
6 & 2.449485 & 3.000000 & 2.449489 & -0.000003 & 0.000002 \\
\hline
\end{array}
\]
Now we present the implementation of the modified version of the above for the exact same problem:

Modified Regula Falsi

<table>
<thead>
<tr>
<th>n</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>f(c)</th>
<th>Rel. Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2.000000</td>
<td>3.000000</td>
<td>2.400000</td>
<td>-0.240000</td>
<td>0.166667</td>
</tr>
<tr>
<td>2</td>
<td>2.400000</td>
<td>3.000000</td>
<td>2.482759</td>
<td>0.164090</td>
<td>0.033333</td>
</tr>
<tr>
<td>3</td>
<td>2.400000</td>
<td>2.482759</td>
<td>2.449153</td>
<td>-0.001652</td>
<td>0.013722</td>
</tr>
<tr>
<td>4</td>
<td>2.449153</td>
<td>2.482759</td>
<td>2.449487</td>
<td>-0.000011</td>
<td>0.000137</td>
</tr>
<tr>
<td>5</td>
<td>2.449487</td>
<td>2.482759</td>
<td>2.449492</td>
<td>0.000011</td>
<td>0.000002</td>
</tr>
</tbody>
</table>

Note that in fact the modified regular falsi did slightly better. Also note how the values of a and b changed in the modified regula falsi whereas b did not change throughout in the usual regula falsi.

Advantages:

- No need to calculate a complicated derivative.
- Increased convergence when compared to the usual regula falsi method.

Disadvantages:

- Not many, but Newton’s method may be still faster if we can apply it.

### 1.6 Secant Method

The Secant method is another two point scheme which avoids the problem of calculating the derivative of the function \( f \) at each iteration (as Newton’s method does) but still obtains improved convergence to the root.

The method uses the same formulation as Regula Falsi or the Modified Regula Falsi to obtain the next iterate \( x_n \),

\[
x_n = x_{n-1} - \frac{f(x_{n-1})}{f(x_{n-1}) - f(x_{n-2})} (x_{n-1} - x_{n-2})
\]

Thus it has a much improved convergence over the bisection method since it uses cleverly information about the function \( f(x) \). However this is not an encompassing scheme. As such it does not require that the iterates \( x_{n-1} \) and \( x_{n-2} \) contain the solution.

In short the pseudocode for the Secant method goes as follows:

1. Obtain two starting guesses, \( x_{n-1} \) and \( x_{n-2} \).
2. Produce the next iterate \( x_n \) through the formula,

\[
x_n = x_{n-1} - \frac{f(x_{n-1})}{f(x_{n-1}) - f(x_{n-2})} (x_{n-1} - x_{n-2})
\]

3. If required tolerance has been reached or maximum number of iterates exceeded then stop with appropriate message.
4. Otherwise continue from step 2 above.
As we also showed earlier we expect that in general the convergence of the secant method will be good but probably not as good as Newton’s method is. In general the Secant method is used in conjunction with some other method (Bisection) in order to refine the solution and the convergence. In other words we start our code with the Bisection scheme and then once a certain first tolerance has been satisfied we pass the result to the Secant (or some other method) for fast and more accurate resolution of the root.

Example:
Find the root of the function \( f(x) = x^2 - 6 \) (as usual) in the interval \([2, 10]\) with an accuracy of \(10^{-6}\).

Solution:
Lets suppose that we take \(x_0 = 2\) and \(x_1 = 10\) so that we can compare with results from previous examples (for that reason we denote \(a = x_0 = 2\) and \(b = x_1 = 10\)).

<table>
<thead>
<tr>
<th>(n)</th>
<th>(a)</th>
<th>(b)</th>
<th>(c)</th>
<th>(f(c))</th>
<th>Rel. Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2.000000</td>
<td>10.00000</td>
<td>2.166667</td>
<td>-1.305556</td>
<td>3.615385</td>
</tr>
<tr>
<td>2</td>
<td>10.00000</td>
<td>2.166667</td>
<td>2.273973</td>
<td>-0.829049</td>
<td>0.047189</td>
</tr>
<tr>
<td>3</td>
<td>2.166667</td>
<td>2.273973</td>
<td>2.460668</td>
<td>0.054889</td>
<td>0.075872</td>
</tr>
<tr>
<td>4</td>
<td>2.166667</td>
<td>2.460668</td>
<td>2.449075</td>
<td>-0.002030</td>
<td>0.004734</td>
</tr>
<tr>
<td>5</td>
<td>2.460668</td>
<td>2.449075</td>
<td>2.449489</td>
<td>-0.000005</td>
<td>0.000169</td>
</tr>
<tr>
<td>6</td>
<td>2.449075</td>
<td>2.449489</td>
<td>2.449490</td>
<td>-0.000000</td>
<td>0.000000</td>
</tr>
</tbody>
</table>

Check these results with the equivalent ones from the regula falsi and the modified regula falsi methods. What do you think?

Advantages:
- No need to calculate a complicated derivative.
- Increased convergence when compared to the usual Bisection method.
- Not encompassing, thus less things to worry about, easier to program.

Disadvantages:
- Not many, but Newton-Raphson may be still faster.
1.7 Fixed Point Method

This method is altogether very different in form from the usual root finding schemes we have discussed thus far. In particular the fixed point method does not solve directly \( f(x) = 0 \) but instead it produces the solution for the problem

\[
g(x) = x.
\]

The solution \( x \) is called the “fixed point” of \( g(x) \). It is not hard to see the reason this problem is considered as equivalent to our usual \( f(x) = 0 \). Simply let \( f(x) = g(x) - x \) and use any of the methods we discussed so far in order to find the root of \( f \). Once you solve the problem \( f(x) = 0 \) then you automatically have also found the solution to \( g(x) = x \). Take a minute to convince yourself about that fact. Some examples later below will further convince you of that.

It is important to note that not all continuous functions \( g(x) \) have fixed points! In that respect the following theorem gives some very simple guidelines for functions with fixed points,

**Theorem 1.7.1.** If \( g \in C[a,b] \) and \( g(x) \in [a,b] \) for all \( x \in [a,b] \) then \( g \) has a fixed point in \([a,b]\). Furthermore assuming \( g'(x) \) exists on \((a,b)\) and that there exists a positive constant \( c < 1 \) such that,

\[
|g'(x)| \leq c < 1 \quad \text{for all} \quad x \in (a,b)
\]

then the fixed point is unique.

**Proof:** Let's start by first showing that a fixed point does exist for \( g \) as specified in the theorem. Naturally if \( g(a) = a \) or \( g(b) = b \) then we are done. Then by contradiction we suppose this is not the case: i.e. instead we must have that \( g(a) > a \) and \( g(b) < b \). Let us now define

\[
h(x) = g(x) - x.
\]

Note that since \( g(x) \) and \( x \) are both continuous functions then \( h(x) \) must also be continuous on \([a,b]\) and the following must hold:

\[
h(a) = g(a) - a > 0 \quad \text{and} \quad h(b) = g(b) - b < 0
\]

Therefore by the intermediate value theorem there must exist a \( r \in (a,b) \) such that \( h(r) = 0 \). Thus, \( g(r) - r = 0 \) or \( g(r) = r \) in \((a,b)\). That shows existence.

For uniqueness we assume again by contradiction that there exist two different fixed points, say \( p \) and \( q \) in \([a,b]\) such that \( g(p) = p \) and \( g(q) = q \) but \( p \neq q \). Then by the mean value theorem there must exist a number \( \xi \in (p,q) \) such that

\[
g'(\xi) = \frac{g(p) - g(q)}{p - q} \quad (1.3)
\]

Taking absolute value of (1.3) and rewriting we get,

\[
|g(p) - g(q)| = |p - q||g'(\xi)| \quad (1.4)
\]

Note however that in the statement of the theorem we are given that \( |g'(x)| < c < 1 \) for all \( x \in (a,b) \). Thus (1.4) becomes,

\[
|g(p) - g(q)| = |p - q||g'(\xi)| \leq |p - q|c < |p - q|
\]
The left hand side of the above however is also equal to $|p - q|$. Therefore we showed that $|p - q| < |p - q|!$. Not possible and therefore by contradiction we showed there must exists a single fixed point for $g$.

It is very important before you go on solving any problem to first check that a fixed point exists by applying the theorem above or some other type of logical argument. Now to find such a fixed point numerically we only need to take a single initial guess $x_0$ and generate iterations by substituting it into the function $g(x)$ itself. After several such iterations the solution (or an approximation to the solution) should emerge. The following simple algorithm makes this clear:

1. Obtain a starting guess, $x_0$.
2. Produce the next iterate $x_n$ through the formula,
   \[ x_n = g(x_{n-1}) \]
3. If required tolerance has been reached or maximum number of iterates exceeded then stop with appropriate message. Otherwise continue from step 2 above.

How is it possible that this simple scheme works? Isn’t it surprising that you just plug in the previous iterate into the function and there comes the next iterate which is closer to the true solution?

Let us give an example to further shed light into the idea. In order to make it easier to compare we will take our usual example $f(x) = x^2 - 6 = 0$ in the interval $[2, 10]$ which we know has a unique root and arrange it appropriately in order to become a fixed point problem. That is quite easy simply (but not optimally) done by adding $x$ on both sides

\[ x^2 - 6 + x = x \]

So now we have a fixed point problem to solve for which we are guaranteed (why?) to have a solution in the interval $[2, 10]$:

**Example:**
Find the fixed point of the following function $g(x) = x^2 - 6 + x$ in the interval $[2, 10]$.

**Solution:**
The solution can now be easily found by applying our pseudocode from above:

<table>
<thead>
<tr>
<th>$n$</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>...</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_n$</td>
<td>2</td>
<td>0</td>
<td>-6</td>
<td>24</td>
<td>594</td>
<td>...</td>
<td>$\infty$</td>
</tr>
<tr>
<td>Error</td>
<td>0</td>
<td>-6</td>
<td>24</td>
<td>594</td>
<td>353424</td>
<td>...</td>
<td>$\infty$</td>
</tr>
</tbody>
</table>

What happened? We were supposed to obtain a solution to our fixed point problem near 2.4 not $\infty$! *The problem is the derivative of $g$.* The fixed point iteration method we outlined above failed in this case. Why? Because in this problem

\[ |g'(x)| > 1 \]

in the interval $[2, 10]$.

The next theorem gives some guidelines for convergence or divergence of the fixed point iteration scheme.
Theorem 1.7.2. Let, as usual, \( g \in C[a,b] \) and suppose that \( g(x) \in [a,b] \) for all \( x \in [a,b] \). Further suppose that the derivative exist on \((a,b)\) and

\[
g'(x) \leq c < 1 \quad \text{for all} \quad x \in (a,b)
\]

If we take any initial value \( x_0 \in (a,b) \) then the sequence arising from

\[
x_n = g(x_{n-1})
\]

converges to the unique fixed point \( r \in [a,b] \).

**Proof:** We already know that there must exist a unique fixed point in \( [a,b] \) from our previous theorem. Note now that from the assumption that \( g(x) \in [a,b] \) for all \( x \in [a,b] \) we must also have that all the \( x_n \)'s are also in \( [a,b] \). Using (1.3) and the mean value theorem we then must have,

\[
|x_n - r| = |g(x_{n-1}) - g(r)| = |g'(\xi)||x_{n-1} - r| \leq c|x_{n-1} - r|
\]

for some \( \xi \in (a,b) \). We now keep reapplying this inequality on the above and obtain,

\[
|x_n - r| \leq c|x_{n-1} - r| \leq c^2|x_{n-2} - r| < \cdots \leq c^{n-1}|x_1 - r| \leq c^n|x_0 - r| \tag{1.5}
\]

However, \( c < 1 \). Thus,

\[
\lim_{n \to \infty} |x_n - r| \leq \lim_{n \to \infty} c^n|x_0 - r| = 0
\]

That is the sequence of \( \{x_n\}_{n=1}^\infty \) converges to the true solution \( r \).

### 1.8 Error estimates for fixed point iteration

We can now use this theorem to obtain some very useful results about the error of our approximations regarding the fixed point algorithm. Both of the inequalities below are a-priori type estimates.

**Corollary 1.8.1.** Suppose \( g \) as in Theorem (1.7.2) above. Then the error of our approximating the fixed point \( r \) of \( g \) can be estimated by either of the following two formulas:

\[
|x_n - r| \leq c^n \max\{x_0 - a, b - x_0\} \tag{1.6}
\]

\[
|x_n - r| \leq \frac{c^n}{1 - c}|x_1 - x_0| \tag{1.7}
\]

for all \( n \geq 1 \).

**Proof:** Since \( r \) is in \( [a,b] \) the first formula (1.6) is clear from inequality (2.9).

To prove the second formula (1.7) we need some more work. Note that, as in Theorem (1.7.2), the following is true,

\[
|x_{n+1} - x_n| = |g(x_n) - g(x_{n-1})| \leq c|x_n - x_{n-1}| \leq \cdots \leq c^n|x_1 - x_0|
\]

Therefore for any \( m > n \geq 1 \) by adding and subtracting similar terms the following must be true,

\[
|x_m - x_n| = |x_m - x_{m-1} + x_{m-1} - \cdots - x_{n+1} + x_{n+1} - x_n|
\]
But the right hand side of this is clearly less than,
\[ \leq |x_m - x_{m-1}| + |x_{m-1} - x_{m-2}| + \cdots + |x_{n+1} - x_n| \]
Or further,
\[ \leq c^{m-1}|x_1 - x_0| + c^{m-2}|x_1 - x_0| + \cdots + c^n|x_1 - x_0| \]
and finally,
\[ = c^n(1 + c + c^2 + \cdots + c^{m-n-1})|x_1 - x_0| = c^n|x_1 - x_0| \sum_{j=0}^{m-n-1} c^j \]
Thus we have that,
\[ |x_m - x_n| \leq c^n(1 + c + c^2 + \cdots + c^{m-n-1})|x_1 - x_0| \]
Note however that from Theorem (1.7.2) \( \lim_{n \to \infty} x^n = r \) where \( r \) denotes the fixed point of \( g \). So,
\[ |r - x_n| = \lim_{m \to \infty} |x_m - x_n| \leq c^n|x_1 - x_0| \sum_{j=0}^{\infty} c^j \]
However \( \sum_{j=0}^{\infty} c^j \) is a geometric series with \( c < 1 \) so we can easily sum it:
\[ \sum_{j=0}^{\infty} c^j = \frac{1}{1-c} \]
Thus
\[ |r - x_n| \leq \frac{c^n}{1-c}|x_1 - x_0| \]
Note that for both of the inequalities (1.6) and (1.7) the rate of convergence depends on the value of \( c \). Thus the smaller the \( c \) the faster the convergence.

This is what went wrong with our previous example. The derivative of \( g(x) = x^2 - 6 + x \) is \( g'(x) = 2x + 1 \) which is definitely not less than 1 in the interval \([2, 10]\). In fact it is greater than 1. It is not surprising therefore that the method diverged from, instead of converged to, the fixed point.

Similarly we can obtain an a-posteriori type estimate and we therefore list it here without proof.

**Corollary 1.8.2.** Suppose \( g \) as in Theorem (1.7.2) above. Then the error of our approximating the fixed point \( r \) of \( g \) can be estimated by
\[ |x_n - r| \leq \frac{c}{1-c}|x_n - x_{n-1}| \]

Let us now look at another example where we pay some more attention to the derivative of \( g(x) \).

**Example:**
Solve the equation \( f(x) = x^3 - 2x - 5 = 0 \) in the interval \([2, 3]\). Let us first rewrite this in the form of a fixed point in two different ways:
\[
x = g_1(x) = \frac{x^3 - 5}{2} \\
x = g_2(x) = (2x + 5)^{1/3}
\]
Solution:
The solution can now be easily found by applying the pseudocode,

<table>
<thead>
<tr>
<th>n</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>g₁(x)</td>
<td>1.50000</td>
<td>-8125000</td>
<td>-2.7681885</td>
<td>-13.1061307</td>
<td>-1128.1243667</td>
</tr>
<tr>
<td>g₂(x)</td>
<td>2.0800838</td>
<td>2.0923507</td>
<td>2.0942170</td>
<td>2.0945007</td>
<td>2.0945438</td>
</tr>
</tbody>
</table>

Thus clearly from the above $g₆(x)$ was a good function to use in order to approximate the root while $g₁(x)$ is not! The reason is really the theorem we just proved. Note that the respective derivatives are

$$g₁'(x) = \frac{3}{2}x² > 1 \text{ in the interval } [2, 3]$$
$$g₂'(x) = \frac{2}{3\sqrt{(2x+5)²}} << 1 \text{ in the interval } [2, 3]$$

which explains why $g₂(x)$ produced a solution and not $g₁(x)$ using the fixed point iteration.

1.9 Convergence and higher order methods

The question of convergence is of great importance for any numerical method which we will encounter. As such we need to devote more time in understanding how to find the convergence rates of some of the schemes which we have seen so far.

We start by defining a very important quantity: the rate of convergence. In some texts this is also called the speed of convergence.

**Definition 1.9.1.** We say that \( \{x_n\} \) \( n=0 \) converges to \( x^* \) with order \( \alpha \) if there exist a constant \( \mu \in [0, 1] \) such that

$$\lim_{n \to \infty} \frac{|x_{n+1} - x^*|}{|x_n - x^*|^{\alpha}} = \mu$$

We call \( \mu \) the rate of convergence and for \( \alpha = 1 \) must be a finite number in \( [0, 1] \). For other \( \alpha \) values it is sufficient that \( \mu < \infty \) for convergence.

Clearly if \( \mu > 1 \) then the sequence \( \{x_n\} \) \( n=0 \) diverges. Furthermore, we distinguish convergence as follows:

- If \( \alpha = 1 \) then the convergence is linear.
- If \( \alpha = 2 \) then the convergence is quadratic, etc.
- If the sequence is linearly convergent and \( \mu = 0 \) then in fact the convergence is superlinear.
- If instead \( \mu = 1 \) then the convergence is sublinear.
- Finally if the sequence is linearly convergent and furthermore

$$\lim_{n \to \infty} \frac{|x_{n+1} - x_n|}{|x_n - x_{n-1}|} = 1$$

then the sequence \( \{x_n\} \) \( n=0 \) converges logarithmically to \( x^* \). Alternatively you could plot \( \lim_{n \to \infty} |x_{n+1} - x^*| \). If the resulting plot is a straight line then the sequence converges logarithmically.
An example of a linearly converging sequence for instance would be
\[ \{1, 1/2, 1/4, 1/8, 1/16, \ldots \} \]

An example of a superlinearly converging sequence would be
\[ \{1/2, 1/4, 1/16, 1/256, \ldots \} \]

Check the sequence above yourself to see why it is superlinearly convergent. In fact you could compute the order of convergence, \( \alpha \), for this sequence to be 2 which shows that it converges quadratically.

As a final example we also provide a sequence which is logarithmically convergent,
\[ \{1/\log(n)\}_{n=1}^{\infty} \]

Now we are in position to find out the order of convergence for some of the methods which we have seen so far. We start with the fixed point iteration.

**Theorem 1.9.2.** The fixed point iteration method

\[ x_n = g(x_{n-1}) \]

starting with an arbitrary \( x_0 \) converges linearly to the unique fixed point \( x \) under the assumption

\[ 0 \neq |g'(x)| \leq c < 1 \]

with asymptotic error constant \( |g'(x)| \).

**Proof:** Since the fixed point is \( x \) then \( g(x) = x \). Then the following holds:

\[ x - x_n = g(x) - g(x_{n-1}) = g'(\xi)(x - x_{n-1}) \]

by the mean value theorem for some \( \xi \in (x, x_{n-1}) \). If we rewrite the above and take the limit as \( n \to \infty \)

\[ \lim_{n \to \infty} \frac{|x - x_n|}{|x - x_{n-1}|} = \lim_{n \to \infty} |g'(\xi)| \]

Remember however that for the mean value theorem \( \xi \in (x, x_{n-1}) \). Since \( x_n \to x \) then also \( \xi \to x \). As a result,

\[ \lim_{n \to \infty} |g'(\xi)| = |g'(x)| \]

And therefore the result,

\[ \lim_{n \to \infty} \frac{|x - x_n|}{|x - x_{n-1}|} = |g'(x)| \]

Note that the above indicates the convergence rate for the fixed point iteration to be linear (since here \( \alpha = 1 \)) with asymptotic constant \( |g'(x)| \). In fact, as we will see, many of the methods which we will learn converge linearly.

Once again, we must emphasize our earlier remark, based on the Corollary, which states that for the fixed point iteration convergence is guaranteed if \( |g'(x)| \leq c < 1 \). In particular the smaller the \( c \) is the faster the convergence!