Some equations cannot be solved exactly, but it is possible to approximate their solution. A simple example is:

\[ \cos x = x \]

Often when working with equations it is better to put everything on one side:

\[ x - \cos x = 0 \]

The first thing you should do with a difficult equation is graph it to get an idea of where the solutions might lie:

It looks like there is a solution close \( x = 1 \); let’s try it: \( 1 - \cos 1 = 0.45969769413186 \). Let’s define \( f(x) = x - \cos x \). We have seen that \( f(1) = 0.46 > 0 \). From the graph we can see that the solution is less than 1. Here are some random tries to find the solution:

<table>
<thead>
<tr>
<th>( x )</th>
<th>1</th>
<th>0.75</th>
<th>0.7</th>
<th>0.73</th>
<th>0.74</th>
</tr>
</thead>
<tbody>
<tr>
<td>( f(x) )</td>
<td>0.46</td>
<td>0.018</td>
<td>-0.065</td>
<td>-0.015</td>
<td>0.0015</td>
</tr>
</tbody>
</table>

The best answer so far is 0.74. Could you write a computer program to search for a solution and find one up to several digits accuracy? Yes, but there’s a better way, in fact several better ways. That’s what we will undertake to study for the next few days.

1. **Bisection Method**

We will continue to use the example \( f(x) = x - \cos x = 0 \). Everything we say about this example would apply to any other equation.

You know that there is a solution of \( f(x) \) between \( x = 0 \) and \( x = 1 \). (In fact you know that there is a solution of \( f(x) \) close to \( x = 0.74 \), but forget that for
Your goal is to find two numbers \( a \) and \( b \) very close together that bracket the solution. That is, you want to find two numbers \( a \) and \( b \) very close together such that the solution is in the interval \([a, b]\).

Here is the important observation, since \( f \) is a continuous function it suffices to find numbers \( a \) and \( b \) close together such that \( f(a) \) and \( f(b) \) have different signs; one is positive and one is negative. Equivalently, find \( a \) and \( b \) close together such that \( f(a)f(b) \leq 0 \). Then if you take \( x = \frac{a + b}{2} \) as your estimate of the solution to \( f(x) = 0 \), and if \( r \) is the true solution, you know that \( |x - r| < \frac{b - a}{2} \).

Here’s an example: for \( f(x) = x - \cos x \) you know \( f(0.73) < 0 \) and \( f(0.74) > 0 \) (see table above). Thus there is a solution \( r \) such that

\[
0.73 < r < 0.74 \\
r \in (0.73, 0.74)
\]

Thus:

\[
|r - 0.735| < 0.005
\]

\( x = 0.735 \) may not be an exact solution, but it is within 0.005 of the exact solution. That’s pretty good information.

The bisection method is a regular way of finding numbers \( a \) and \( b \) as close together as you want (within the limits of machine accuracy) bracketing the solution of a given equation \( f(x) = 0 \). Once you have \( a \) and \( b \) you can say that \( \frac{a + b}{2} \) is the approximate solution to the equation with an error of no more than \( \frac{b - a}{2} \).

The idea is simple. Let \( r \) be the true (unknown) solution. If you have numbers \( a \) and \( b \) such that \( a \leq r \leq b \), you can find new numbers \( a' \leq r \leq b' \) such that \( b' - a' = \frac{b - a}{2} \). Here’s how you do it.

```plaintext
// assume a < b and f(a)f(b) <= 0
c = (a + b) / 2
if f(a)*f(c) < 0 then
    a' = a
    b' = c
else
    a' = c
    b' = b
endif
// now a' < b' and f(a')f(b') <= 0 and b'-a' = (b-a)/2
```

Suppose you want to approximate the solution of an equation \( f(x) = 0 \) within a tolerance \( \text{tol} \). Start with two numbers \( a \) and \( b \) bracketing the solution, and bring them closer and closer together, continuing to bracket the solution, until \( b - a < \text{tol} \).
// assume \( a < b \) and \( f(a)f(b) \leq 0 \)
while \( b - a > 2 \times \text{tol} \) do
    \( c = (a + b) / 2 \)
    if \( f(a)f(c) < 0 \) then
        \( b = c \)
    else
        \( a = c \)
    endif
endwhile
// now \( a < b \) and \( f(a)f(b) \leq 0 \) and \( b - a \leq 2 \times \text{tol} \)

Here's some sample Mathematica output for the bisection routine. You can see how you start with the interval \([0, 1]\) and end with \([0.739014, 0.739136]\) bracketing the solution. The approximate solution is \( 0.739014 + \frac{0.739136}{2} = 0.739075 \), and the maximum error is \( \frac{0.739136 - 0.739014}{2} = 0.000061 \)

\[
\text{In}[28]:= f[x_] := x - \cos[x];
\text{r} = \text{bisect}[f, 0., 1., 0.0001]
\text{f}[\text{r}]
\]

\[
\{0.5, 1.\}
\{0.5, 0.75\}
\{0.625, 0.75\}
\{0.6875, 0.75\}
\{0.71875, 0.75\}
\{0.734375, 0.75\}
\{0.734375, 0.742188\}
\{0.738281, 0.742188\}
\{0.738281, 0.740234\}
\{0.738281, 0.739258\}
\{0.73877, 0.739258\}
\{0.739014, 0.739258\}
\{0.739014, 0.739136\}
\]

\[
\text{Out}[29]= 0.739075
\]

\[
\text{Out}[30]= -0.0000174493
\]
Whenever you have a method for solving equations, there are several questions that need to be kept in mind, together with their answers for the bisection method:

1. What kinds of equations is the method good for?
   (a) \( f(x) = 0 \) where \( f \) is continuous and the function changes sign around the solution.

2. What information is needed to start the method?
   (a) an interval \([a, b]\) on which \( f \) is continuous and where \( f(a)f(b) \leq 0 \)

3. Does the method always converge to an approximate solution? If not, how can we tell if we are converging on an exact solution?
   (a) The bisection method always converges to an exact solution.

4. How close is an approximate solution to the exact solution?
   (a) For the bisection method we always have two numbers, \( a \) and \( b \), that bracket the solution. The error between the approximate solution \( \frac{a+b}{2} \) and the exact solution is always less than \( \frac{b-a}{2} \).

5. How many steps does it take to get an approximate solution within \( \text{tol} \) of the exact solution, and how much work is necessary for each step?
   (a) the number of steps is the smallest non-negative integer larger than \( \log_2 \frac{b-a}{\text{tol}} - 1 \)
   (b) each step requires two evaluations of \( f(x) \) and a few arithmetic operations and comparisons (but see below for an algorithm that only requires one evaluation)
   (c) In our example: \( \log_2 \frac{1-0}{0.0001} - 1 = 12.29 \), so we should have, and do have, 13 steps.

Here is another version of the algorithm that makes only one function evaluation each time through the main loop (and possibly one more assignment). Assume you start with \( a \) and \( b \) and you have already calculated \( f(a) = f(a) \). This version is twice as fast as the previous version, so it is the one you should always use for the bisection method.

```c
// assume a < b and f(a)f(b) <= 0
while b-a > 2*\text{tol} do
  c = (a + b) / 2
  fc = f(c)
  if fa*fc < 0 then
    b = c
  else
    a = c
    fa=fc
  endif
endwhile
// now a < b and f(a)f(b) <= 0 and b-a <= 2*\text{tol}
```

2. **Newton-Raphson Method**

When you are trying to solve an equation \( f(x) = 0 \) and (a) \( f \) is differentiable and (b) the derivative of \( f \) is not too expensive to compute and (c) you know a
number $a$ not too far from the actual solution, then you can use a method due initially to Newton. Even if condition (c) is not satisfied, sometimes you can use Newton’s method to quickly improve the result you get from a slower but more robust solving method (like bisection).

Let’s start from a very general, surprisingly interesting idea (it leads to the whole discipline of dynamical systems): given a function $F(x)$ and a starting point $a_1$, what happens if you create a sequence:

$$a_1, a_2 = F(a_1), a_3 = F(a_2), \ldots, a_{i+1} = F(a_i), \ldots$$

Class try on calculators for $F(x) = \cos x$. Start with 1 and just keep pressing the cosine key. You get a number $x = 0.7390\ldots$ that satisfies $\cos x = x$.

Class give examples of sequences that go to infinity or oscillate.

In general, if the sequence $a_i$ converges to some number $a$ then you have solved the equation $F(a) = a$ or $F(a) - a = 0$. The point $a$ is called a fixed point of $F$. The fixed point of $\cos x$ is $x = 0.7390\ldots$

But when does the sequence converge? A sequence is guaranteed to converge under the following circumstances. Let $r$ be the point of convergence and $c = |r - a_1|$. If $|F'(x)| < 1$ for $x \in [r - c, r + c]$, then the sequence stays inside this interval and converges to $r$. Of course, if all you know is $F$ and $a_1$ you cannot use this test for convergence because you don’t know $r$. But if $|F'(x)| < 1$ on a large region around $a_1$ you have hope that the sequence converges.

In the example $F(x) = \cos x$, $|F'(x)| = |−\sin x| < 1$ around the starting point $a_1 = 1$.

Let’s examine the sequence $1, \cos 1, \cos (\cos 1), \ldots$ more closely:

```
Out[8]= (1., 0.540302, 0.857553, 0.65429, 0.79348, 0.701369, 0.76396, 0.722102, 0.750418, 0.731404, 0.744237, 0.735605, 0.741425, 0.737507, 0.740147, 0.738369, 0.739567, 0.739018, 0.739055, 0.739071, 0.739082, 0.739087)
```

I’ve taken these numbers and calculated their differences:

```
Out[10]= (-0.459698, 0.317251, -0.203263, 0.139191, -0.092116, 0.0625909, -0.0418573, 0.0283153, -0.0190137, 0.0128333, -0.00863261, 0.00582035, -0.0039182, 0.00264045, -0.00177813, 0.001198, -0.000806882, 0.000543573, -0.000366136, 0.000246643, -0.000166137, 0.000111914, -0.0000753858, 0.0000507812, -0.0000342066, 0.0000230421, -0.0000155214, 0.0000104554, -7.04288 \times 10^{-6}, 4.74417 \times 10^{-6})
```

Then I calculated the ratios of the differences:
The ratios of the differences of the sequence all came out almost the same, very close to \(-F'(a) = -\sin 0.7391 = -0.6736\). Let’s see why this is true:

The last table calculates the values:

\[
\frac{a_{i+2} - a_{i+1}}{a_{i+1} - a_{i}} = \frac{F(a_{i+1}) - F(a_{i})}{a_{i+1} - a_{i}}
\]

By the Mean Value Theorem

\[
\frac{F(a_{i+1}) - F(a_{i})}{a_{i+1} - a_{i}} = F'(c)
\]

for some number \(c\) between \(a_{i+1}\) and \(a_{i}\). In our example all the numbers \(a_{i}\) are near 0.74 and \(F'(0.74) = (\cos x)'|_{x=0.74} = -\sin x|_{x=0.74} = -0.67\), which agrees with the third table.

Let’s do some analysis that proves if \(|F'(x)| < e < 1\) for all the \(x\)’s encountered, then the sequence \(a_{i}\) converges to some value \(a\).

\[
\left| \frac{a_{i+1} - a_{i}}{a_{i+1} - a_{i}} \right| = \left| \frac{a_{i+1} - a_{i}}{a_{i+1} - a_{i-1}} \right| \left| \frac{a_{i} - a_{i-1}}{a_{i-1} - a_{i-2}} \right| \cdots \left| \frac{a_{3} - a_{2}}{a_{2} - a_{1}} \right| < e^{i-2}
\]

If \(i > j\) we have:

\[
|a_{i} - a_{j}| < |a_{i} - a_{i-1}| + |a_{i-1} - a_{i-2}| + \cdots + |a_{j+1} - a_{j}|
\]

\[
< |a_{2} - a_{1}| (e^{i-3} + e^{i-4} + \cdots + e^{j-2})
\]

\[
= |a_{2} - a_{1}| e^{i-2} (1 + e + e^{2} + \cdots + e^{i-j-1})
\]

\[
= |a_{2} - a_{1}| e^{i-2} \frac{1 - e^{i-j}}{1 - e}
\]

\[
< \frac{|a_{2} - a_{1}|}{1 - e} e^{i-j-2}
\]

Thus the sequence \(a_{1}, a_{2}, a_{3}, \ldots\) is a Cauchy sequence and converges to some fixed point \(a\) for the function \(F(x)\). (A sequence \(a_{i}\) is a Cauchy sequence if for all \(\varepsilon > 0\) there exists \(j\) such that, for all \(i > j\), \(|a_{i} - a_{j}| < \varepsilon\). All Cauchy sequences converge. Cauchy sequences will not be on the final exam.)

If the function \(F(x)\) has a very small derivative, then the sequence \(a_{i}\) converges rapidly to a fixed point \(a\) for \(F(x)\). Newton’s method of equation solving takes advantage of this fact.

Suppose we have an equation \(f(x) = 0\), for example \(x - \cos x = 0\). Let

\[
F(x) = x - \frac{f(x)}{f'(x)}
\]
Notice that \( F(a) = a \) if and only if \( f(a) = 0 \), so if we start with some value \( a_1 \) and the sequence \( a_{i+1} = F(a_i) \) converges to \( a \), then \( f(a) = 0 \) and the equation is solved. A solution to \( f(x) = 0 \) is a fixed point for \( F(x) \). Moreover:

\[
F'(x) = \frac{f(x) f''(x)}{[f'(x)]^2}
\]

If \( a_1 \) is close to \( a \) then \( F'(a_1) \) is close to 0 and \( a_1, a_2, \ldots \) can be expected to converge rapidly to \( a \) (that is, a few terms gives a lot of correct digits for \( a \)).

**Newton’s Method** of solving an equation \( f(x) = 0 \) consists of taking a first guess \( a_1 \) and generating a sequence of (hopefully) improves estimates \( a_{i+1} = a_i - \frac{f(a_i)}{f'(a_i)} \) converging to the solution of the equation.

Let’s try it with \( f(x) = x - \cos x \). In this case \( F(x) = x - \frac{x - \cos x}{1 + \sin x} = \frac{x \sin x + \cos x}{1 + \sin x} \). If we start with \( a_1 = 1 \) we get the sequence of approximate solutions:

\[
\begin{align*}
\text{Out[20]} &= \{1., 0.750364, 0.739113, 0.739085, 0.739085, 0.739085, \\
&\quad 0.739085, 0.739085, 0.739085, 0.739085, 0.739085, \\
&\quad 0.739085, 0.739085, 0.739085, 0.739085, 0.739085, \\
&\quad 0.739085, 0.739085, 0.739085, 0.739085, 0.739085\}\n\end{align*}
\]

Only four terms are required to get to six-digit accuracy. The function \( x - \frac{x - \cos x}{(x - \cos x)} \) has the same fixed point as \( \cos x \), but the first function converges much more rapidly then the second.

We can give a geometric interpretation to Newton’s method. Given an equation \( f(x) = 0 \) and an estimate \( a_i \) for the solution, here’s how to find the next estimate \( a_{i+1} \) geometrically. Draw the tangent line to the graph of \( f(x) \) at the point \((a_i, f(a_i))\). The next estimate \( a_{i+1} \) is the point on the \( x \)-axis where the tangent line crosses. In the example below, start with \( a_1 = 1 \), draw the tangent line at \((1, 1 - \cos 1) = (1, 0.46)\), and see that the tangent line crosses at \( a_2 = 0.75 \).
When Newton’s method works, it usually works fast. But it doesn’t always work. In fact the times it doesn’t work very well has led to some beautiful mathematics including fractals, Mandelbrot sets and Julia sets. Try googling “Mandelbrot set”, “Julia set”, and “fractal”.

Now let’s look at the four questions about equation solving methods.
(1) What kinds of equations is the method good for?
   (a) $f(x) = 0$ where $f$ is differentiable and $f'(x) \neq 0$ at the point $x$ where $f(x) = 0$.

(2) What information is needed to start the method?
   (a) it is usually a good idea to pick a starting point $a$ not too far from the solution.

(3) Does the method always converge to an approximate solution? If not, how can we tell if we are converging on an exact solution?
   (a) Newton’s method does not always converge to a solution of the equation. Consider the equation $(x - 1)^{1/3} = 0$, whose solution is $x = 1$.

   If $f(x) = (x - 1)^{1/3}$ then $f'(x) = \frac{1}{3(x-1)^{2/3}}$ and $x - f(x) = x - 3(x-1) = 3 - 2x$. Even if we pick a starting point very close to the actual solution, Newton’s method diverges. Here are the successive values generated by Newton’s method starting at 1.0001. Instead of getting closer and closer to the correct solution 1, the approximations move away from 1.

   \[
   \begin{align*}
   \text{Out}[35] &= \{1.0001, 0.9998, 1.0004, 0.9992, 1.0016, \\
   & \quad 0.9968, 1.0064, 0.9872, 1.0256, 0.9488, 1.1024, \\
   & \quad 0.7952, 1.4096, 0.1808, 2.6384, -2.2768, \\
   & \quad 7.5536, -12.1072, 27.2144, -51.4288, 105.8586\}\n   \end{align*}
   \]

   (b) Here’s another bad example: $f(x) = x^5 + 5x^4 + 10x^3 + 10x^2 + 5x + 1$.

   An exact solution is $f(-1) = 0$. Newton’s method $x \to x - \frac{f(x)}{f'(x)}$ starting at $x = 0$ yields:

   \[
   \begin{align*}
   \text{Out}[30] &= \{0., -0.2, -0.36, -0.488, -0.5904, -0.67232, -0.737856, \\
   & \quad -0.790285, -0.832228, -0.865782, -0.892626, -0.914101, \\
   & \quad -0.931281, -0.945024, -0.95602, -0.964816, -0.971853, \\
   & \quad -0.977482, -0.981986, -0.985588, -0.988471, -0.990777, \\
   & \quad -0.992621, -0.994097, -0.995277, -0.996222, -0.996977, \\
   & \quad -0.997585, -0.99807, -0.998467, -0.99882, -0.999095, \\
   & \quad -0.999566, -0.998775, -0.999012, -0.999199, \\
   & \quad -0.99963, -0.99963, -0.99963, -0.99963, -0.99963, \\
   & \quad -0.99963, -0.99963, -0.99963, -0.99963, -0.99963\}\n   \end{align*}
   \]

   This is not rapid convergence we saw above.

   (c) Let $r$ be the exact solution to $f(x) = 0$. If $f'(r) \neq 0$ and if $a_1$ is close enough to $r$, then the sequence of approximate solutions $a_{i+1} = a_i - \frac{f(a_i)}{f'(a_i)}$ converges quadratically to $r$. That means $|a_{i+1} - r| \approx c |a_i - r|^2$ for some constant $c$. 

(i) Example: solving $x - \cos x = 0$:

```
{1., 0.7503638678402439, 0.7391128909113617, 0.7390851332151607, 0.7390851332151607, 0.7390851332151607, 0.7390851332151607, 0.7390851332151607, 0.7390851332151607, 0.7390851332151607, 0.7390851332151607, 0.7390851332151607, 0.7390851332151607, 0.7390851332151607, 0.7390851332151607, 0.7390851332151607, 0.7390851332151607, 0.7390851332151607}
```

You can see the number of correct digits is 1, 4, 9, 15 for the first, second, third and remaining estimates. Quadratic convergence means that the number of correct digits approximately doubles at every step.

(ii) You can see why this is true if you assume that the solution is 0 and $f'(0) = a_1 \neq 0$. Then

\[ f(x) = a_1 x + a_2 x^2 + \cdots \]

\[ f'(x) = \frac{1}{a_1 + 2a_2 x + \cdots} = a_1^{-1} - 2a_1^{-2} a_2 + \cdots. \]

\[ x - \frac{f(x)}{f'(x)} = -a_1^{-1} a_2 x^2 + \cdots \]

so if the error in one estimate is $x$ then the error in the next estimate is a constant multiple of $x^2$.

(d) The moral is that in "good" cases Newton’s method converges rapidly, but there are bad cases.

(4) How close is an approximate solution to the exact solution?

(a) For Newton’s method you can’t be sure, but when it’s working right the convergence is rapid.

(b) If Newton’s method arrives at a fixed point, you usually have a solution (but test it); if the successive approximate solutions keep changing in more than the last digit you probably have failure of convergence and you shouldn’t trust partial answers.

(5) The amount of work required for each step is one evaluation of the function and one evaluation of its derivative (and a couple of arithmetic operations).

A bad way to program Newton’s method to solve $f(x) = 0$ is to search for a value of $x$ that either satisfies $f(x) = 0$ or $f(x)$ is very small. If the graph of $f$ is very steep near the solution, the values of $f(x)$ may be very large for values of $x$ close to a solution. The best approach for programming Newton’s method (or any other solving method that is not as predictable as the bisection method) is to keep generating approximate solutions until (a) you have two successive estimates that
are almost the same; or (b) you have exceeded some fixed number of estimates and you decide the method isn’t converging. For example, to solve \( f(x) = 0 \):

```plaintext
// Assume a function f(x) and its derivative df(x)
n = 0
next_guess = your_first_guess
Repeat
    last_guess = next_guess
    next_guess = last_guess - f(last_guess) / df(last_guess)
    n = n + 1
Until n > some_big_number or
    abs(next_guess - last_guess) < 1e-15 * max(1, abs(next_guess))
Return(next_guess)
```

3. Secant Method

Sometimes you may have an equation \( f(x) = 0 \), and you may not want to use Newton’s method because either you cannot differentiate \( f(x) \) or the derivative is difficult or time-consuming to evaluate accurately. In that case you can use the **secant method**, which is just Newton’s method with a twist. The twist is that instead of using the derivative \( f'(x) \) you use the approximate derivative \( \frac{f(x_2) - f(x_1)}{x_2 - x_1} \). More specifically, instead of starting with one point, like Newton’s method, you start the secant method with two points, call them \( x_1 \) and \( x_2 \). Then you define a sequence:

\[
x_{i+1} = x_i - \frac{f(x_i)}{f(x_i) - f(x_{i-1})}
\]

\[
= x_i - \frac{(x_i - x_{i-1}) f(x_i)}{f(x_i) - f(x_{i-1})}
\]

It would appear that you could simplify this further to \( \frac{x_{i-1} f(x_i) - x_i f(x_{i-1})}{f(x_i) - f(x_{i-1})} \); however this version is prone to errors of destructive subtraction near the end of convergence when \( x_i \approx x_{i-1} \), so the last displayed version is better to use.

From a geometric point of view, \( x_{i+1} \) is the place where the \( x \)-axis meets the line through the points \( (x_{i-1}, f(x_{i-1})) \) and \( (x_i, f(x_i)) \). Because this intersection point with the secant line is likely to be closer to a solution the the midpoint \( \frac{x_i + x_{i-1}}{2} \), the secant method usually converges faster than the midpoint method.

Ideally the points you start with, \( x_1 \) and \( x_2 \), should be close to a solution. The method does not converge quite as rapidly as Newton’s method, but it may require less calculation per step (when \( f'(x) \) is expensive to compute).

Let’s do the same example as we did with the other solution methods: \( x - \cos x = 0 \). We start with \( x_1 = 0 \) and \( x_2 = 1 \). We get the sequence:
Convergence of the secant method required seven steps instead of the four required for Newton’s method, but the same answer was achieved. Here’s the code I used. Note the efficiency that reduces to one function evaluation of \( f(x) \) per cycle through the Repeat..Until loop.

```plaintext
// Assume a function f(x)
n = 0
last_guess = your_first_guess
next_guess = your_second_guess
f_last_guess = f(last_guess)
f_next_guess = f(next_guess)
Repeat
    previous_guess = last_guess
    f_previous_guess = f_last_guess
    last_guess = next_guess
    f_last_guess = f_next_guess
    next_guess
        = last_guess-(last_guess-previous_guess)*f_last_guess/(f_last_guess-f_previous_guess)
    f_next_guess = f(next_guess)
n = n+1
Until n > some_big_number or abs(next_guess-last_guess) < 1e-15*max(1,abs(next_guess))
Return(next_guess)
```

Now let’s answer the five questions for the secant method:

1. What kinds of equations is the method good for?
   (a) \( f(x) = 0 \) where \( f \) is continuous.

2. What information is needed to start the method?
   (a) it is usually a good idea to pick a starting points \( x_1 \) and \( x_2 \) not too far from the solution.

3. Does the method always converge to an approximate solution? If not, how can we tell if we are converging on an exact solution?
   (a) The secant method does not always converge to a solution of the equation. Consider the equation \( (x - 1)^{1/3} = 0 \), whose solution is \( x = 1 \). Newton’s method failed on this one. If \( f(x) = (x - 1)^{1/3} \). Even if we pick two starting point very close to the actual solution, \( x_1 = 0.98 \) and \( x_2 = 1.01 \). Here are the successive values generated by the secant method. Instead of getting closer and closer to the correct solution 1, the approximations wander around without getting much better.
(b) Here’s an example for which Newton’s method converged very slowly:

\[ f(x) = x^5 + 5x^4 + 10x^3 + 10x^2 + 5x + 1. \]

An exact solution is \( f(-1) = 0 \).

The secant method starting with 0 and 1 yields:

\[
\begin{align*}
\text{Out}[43]//\text{InputForm} &= \{0., 1., -0.032258064516129004, -0.06038426717172034, \ldots, -0.9442386092788765)\}
\end{align*}
\]

The secant method is not converging rapidly to any solution at all.

(c) The moral is that in "good" cases the secant method converges rapidly, but there are bad cases.

(4) How close is an approximate solution to the exact solution?

(a) For the secant method you can’t be sure, but when it’s working right the convergence is rapid.

(b) If the secant method arrives at a fixed point, you usually have a solution (but test it); if the successive approximate solutions keep changing in more than the last digit you probably have failure of convergence and you shouldn’t trust partial answers.

(5) The amount of work required for each step is one evaluation of the function (and a couple of arithmetic operations). If the secant method were to converge in no more than twice as many iterations as Newton’s method, it would actually be faster.
4. Homework due April 11

Let \( f(x) = x^5 + 1000x^4 + x^3 + x^2 + x + 1 \).

1. Show that \( f(0) > 0 \).
2. Find a number \( a \) such that \( f(a) < 0 \). (Hint: why does \( a \) have to be negative?)
3. Conclude that there is at least one solution to \( f(x) = 0 \) between \( a \) and 0.
4. Write code for the bisection method, Newton’s method and the secant method to find the solution of \( f(x) = 0 \).
5. For your three solutions (one for each method), find the value of \( f(x) \).
6. For your three solutions, estimate the distance to the true solution. (You may NOT use equation-solving software from Mathematica or other software packages. Create your own estimates.)

5. Regula Falsi or False Position Method

The method of false position combines the secant and bisection method. It is very old, as shown by its Latin name. Like the bisection method, false position starts with two points \( a \) and \( b \) that bracket a solution of your equation \( f(x) = 0 \). That is, we start with two points \( a \) and \( b \) such that \( f(a)f(b) \leq 0 \). The next estimate for the root, \( c \), is calculated exactly like the secant method:

\[
c = b - \frac{(b - a)f(b)}{f(b) - f(a)}
\]

The next step is like the bisection method. Decide which interval, \( (a,c) \) or \( (c,b) \), contains the solution to the equation and use those endpoints for the next step.

Unlike the bisection method, you cannot be sure that the new interval is much smaller than the original interval \( (a,b) \), so the criterion for stopping the algorithm will be more like the criterion used in the secant method: you stop when the new estimates stop changing significantly. Here’s the entire algorithm in pseudo-code:
// assume a continuous function f(x)
// assume you know values bracketing the solution
a = ... // (a,b) brackets the solution
b = ...
fa = f(a)
fb = f(b)
c = b
n = 0
Repeat
    old_c = c
    c = b - (b - a)*fb / ( fb - fa )
    fc = f(c)
    n = n + 1
    If fa*fc < 0 then
        b = c
        fb = fc
    Else
        a = c
        fa = fc
    Endif
Until n > some_big_number or abs(c - old_c) < 1e-15 * max(1,abs(c))
If n > some_big_number then
    return(did not converge)
Else
    return(c)
Endif

Let's see how this algorithm works on \( x - \cos x = 0 \) starting with the interval (0, 1):

```
Out[16]/TableForm=
0.  1.
0.736299  1.
0.738945  1.
0.739078  1.
0.739085  1.
0.739085  1.
0.739085  1.
0.739085  1.
0.739085  1.
0.739085  1.
0.739085  1.
0.739085  1.
0.739085  1.
0.739085  1.
```

Here's the left sides with more accuracy:
For this equation, the right endpoint never moves and the left endpoint moves right to the solution. The final interval is not small (unlike the case with bisection), so the more complex stopping criterion is necessary.

Let’s try false position on our two hard examples, first \((x - 1)^{1/3} = 0\) starting with \((0.98, 1.01)\):

Unlike the Newton and secant methods, the method of false position does seem to converge slowly to a solution of this equation.

Now for \(x^5 + 5x^4 + 10x^3 + 10x^2 + 5x + 1 = 0\), starting with \((-2, 1)\):

These values are moving towards \(-1\), but it would take a long time to get there. False position is not a good approach to this equation.

Let’s take a look at our five questions:
(1) What kinds of equations is the method good for?
   (a) \( f(x) = 0 \) where \( f \) is continuous and the function changes sign around the solution.

(2) What information is needed to start the method?
   (a) numbers \( a \) and \( b \) bracketing the root: \( f(a) f(b) \leq 0 \).

(3) Does the method always converge to an approximate solution? If not, how can we tell if we are converging on an exact solution?
   (a) The method of false position always keeps the root bracketed but may not converge in a reasonable number of iterations to the root.

(4) How close is an approximate solution to the exact solution?
   (a) You cannot be sure because the changing interval \((a, b)\) may not get small even though one end converges rapidly to a solution. As with Newton and secant, the best criterion is that the estimate \( c \) does not change significantly from one iteration to the next.

(5) The amount of work required for each step is one evaluation of the function (and a couple of arithmetic operations).

6. Laguerre’s Method

6.1. Dividing Polynomials. In computers a polynomial is usually represented as an array. In this section I’ll speak C and use 0-based arrays. We will say \( p = (a_0, a_1, \ldots, a_n) \) represents \( a_0 + a_1 X + \cdots + a_n X^n \).

You add polynomials by adding their coefficients. You multiply them as follows:

\[(a_0, \ldots, a_m) (b_0, \ldots, b_n) = (c_0, \ldots, c_{m+n}),\]

where

\[c_i = \sum_{j=\max(0,i-n)}^{\min(i,m)} a_j b_{i-j} .\]

In this section we want to develop an algorithm for dividing a polynomial by a linear polynomial \( b + X = (b, 1) \). That is, we want to find a quotient polynomial \((c_0, \ldots, c_{n-1})\) and a remainder \( r \) such that \((a_0, \ldots, a_n) = (c_0, \ldots, c_{n-1}) (b, 1) + r\).

Here is the algorithm:

**Proposition 1.** The quotient of \((a_0, a_1, \ldots, a_n)\) divided by \((b, 1)\) is \((c_0, \ldots, c_{n-1})\) where

\[c_{n-1} = a_n,\]
\[c_i = a_{i+1} - bc_{i+1} \quad \text{for } 1 < i < n\]

and the remainder is \(a_0 + a_1 (-b) + \cdots + a_n (-b)^n\).

**Proof:** Assume \((c_0, \ldots, c_{n-1})\) is the quotient. We have to show that we have the right formula for the \( c_i \). Let \((d_0, \ldots, d_n) = (c_0, \ldots, c_{n-1}) (b, 1)\). We know that \(d_i = a_i\) for \(1 \leq i \leq n\) and the remainder is \(a_0 - d_0\). Moreover

\[d_n = c_{n-1} = a_n,\]
\[d_i = c_{i-1} + bc_i = a_i \quad \text{for } 1 < i < n\]

The first line tells us that the formula in the Proposition for \( c_{n-1} \) is correct, and the next line tells us that the formula for the remaining \( c_i \) is correct.
The formula for the remainder is classical. If \( X + b \) divides a polynomial \( p(X) \) with quotient \( q(X) \) and remainder \( r \) (which must be a constant), then

\[
\begin{align*}
p(X) &= q(X)(X + b) + r \\
p(-b) &= q(-b)(0) + r \\
&= r
\end{align*}
\]

6.2. **How to find all the roots of a polynomial.** To find the roots of a polynomial \( p(X) \), find one root \( r \) and let \( q(X) = \frac{p(X)}{X - r} \) (since \( r \) is a root, \( X - r \) divides \( p(X) \) exactly). Then find all the roots of \( q(X) \). This process is called “deflation”; the idea is to reduce finding all the roots of a degree \( n \) polynomial to finding one root, then finding all the roots of a degree \( n - 1 \) polynomial. Of course next you will find one root of \( q(X) \), deflate again, find a root of the quotient, deflate again, find a root of the next quotient, etc., until all the roots are found.

6.3. **How to find one root of a polynomial—Laguerre’s Method.** Doing this in full generality requires using complex numbers and complex arithmetic, but we’ll try to stick to real examples.

Let \( p(X) \) be a polynomial of degree \( n \). If \( a \) is an approximate root of \( p(X) \), you can get a better approximation replacing \( a \) by

\[
a \rightarrow a - \frac{n \ p(a)}{\max \left(p'(a) \pm \sqrt{(n-1)^2 p'(a)^2 - n(n-1) p(a) p''(a)}\right)}
\]

As with all approximation methods, you do this until \( a \) stops changing significantly. Let’s try it on \( p(X) = -2 + 6X + X^2 - 5X^3 + X^5 \)

![Graph of the polynomial](image)

We always start with a guess of 0. Laguerre’s method isn’t too fussy about starting points. Starting almost anywhere will give converge to a root.
Deflate the polynomial by dividing by $X - 0.3479$...

$$\text{Out}[99]= 5.75877 - 0.694593X - 4.87939X^2 + 0.347296X^3 + X^4$$

Now use Laguerre’s method to solve this polynomial starting at 0.

$$\text{Out}[999]= \{0., 0.8386105648268041, 1.2677645336837067, 1.3968685221497246, 1.4140977388585094, 1.4142135623289842, 1.4142135623730954, 1.4142135623730954\}$$

Deflate by removing the root 1.414...(Divide by $X - 1.414$...)

$$\text{Out}[1002]= -4.07207 - 2.38823X + 1.76151X^2 + X^3$$

and solve the deflated polynomial.

$$\text{Out}[1007]= \{0., -0.9324615752509507, -1.3567635921382315, -1.4139090647771568, -1.4142135623730954\}$$

Deflate again, removing the root $-1.414$...

$$\text{Out}[1010]= -2.87939 + 0.347296X + X^2$$

At this point we could use the quadratic formula to find the remaining two roots, but let’s apply Laguerre once more starting at 0.

$$\text{Out}[1019]= \{0., 1.2890290865940943, 1.531564413545749, 1.53208886233306, 1.532088862379558, 1.53208886237956\}$$

Deflate by removing the root 1.532..., and a linear polynomial remains.

$$\text{Out}[1018]= 1.87939 + X$$

The root of this polynomial is $-1.87939$...

Collecting our results, the five roots are:

$$\text{Out}[1024]= \{0.347296, 1.41421, -1.41421, 1.53209, -1.87939\}$$

Putting all five roots into the original polynomial, here are the resulting values.
It appears that we have found the five roots.

6.4. Derivation of Laguerre’s Method. Suppose we have a monic polynomial 
\[ p(X) = a_0 + a_1X + \cdots + a_{n-1}X^{n-1} + X^n \]
with roots \( x_1, \ldots, x_n \). Even if we don’t know what the roots are, we do know something about them. For example, we know that we can factor our polynomial:
\[ p(X) = (X - x_1) \cdots (X - x_n). \]

Note:
\[ \log(p(X)) = \log(X - x_1) + \cdots + \log(X - x_n) \]

Define:
\[
\begin{align*}
g(X) &= -\frac{d}{dX} \log(p(X)) \\
&= \frac{p'(X)}{p(X)} \\
&= \frac{1}{X - x_1} + \cdots + \frac{1}{X - x_n}
\end{align*}
\]

\[
\begin{align*}
h(X) &= \frac{d}{dX}g(X) \\
&= \left[ \frac{p'(X)}{p(X)} - \frac{p''(X)}{p(X)} \right] \\
&= \frac{1}{(X - x_1)^2} + \cdots + \frac{1}{(X - x_n)^2}
\end{align*}
\]

Suppose \( a \) is an approximation to the root \( x_1 \). Define \( \alpha = a - x_1 \). Now for the bold step. Assume that all the other roots are close together, and you can replace \( a - x_i \) with a single number. Let \( \beta \approx a - x_i \) for \( i > 1 \). Then:
\[
\begin{align*}
g(a) &= \frac{1}{\alpha} + \frac{n-1}{\beta} \\
h(a) &= \frac{1}{\alpha^2} + \frac{n-1}{\beta^2}
\end{align*}
\]

Since we know the polynomial \( p(X) \) from its coefficients, and we can calculate the numbers \( g(a) \) and \( h(a) \). Then we can solve for the unknown \( \alpha \) (and \( \beta \), which we don’t need). There are two solutions:
\[
\alpha = \frac{n}{g(a) \pm \sqrt{(n-1)(nh(a) - g(a)^2)}}
\]

We choose the largest denominator:
\[
\alpha = \frac{n}{\max \left[ g(a) \pm \sqrt{(n-1)(nh(a) - g(a)^2)} \right]}
\]

\[
= \frac{n p(a)}{\max \left[ p'(a) \pm \sqrt{(n-1)^2 p'(a)^2 - n(n-1) p(a) p''(a)} \right]}
\]
Of course everything here is an approximation, but we can expect that $a - \alpha$ is a better approximation of $x_1$ than $a$ alone. So if $a$ is our first guess at a root of $p(X)$, the next guess will be:

$$a \rightarrow a - \frac{n \cdot p(a)}{\max \left[ p'(a) \pm \sqrt{(n-1)^2 p'(a)^2 - n \cdot (n-1) \cdot p(a) \cdot p''(a)} \right]}$$

Finally, let’s see how Laguerre’s method does with our five questions.

1. What kinds of equations is the method good for?
   (a) polynomial equations
2. What information is needed to start the method?
   (a) any number, but it’s hard to guess which root will be found
3. Does the method always converge to an approximate solution? If not, how can we tell if we are converging on an exact solution?
   (a) Laguerre’s method usually converges to a solution (provided you don’t encounter a point were $p'(a) = p''(a) = 0$ and $p(a) \neq 0$). However convergence is slow near multiple zeros. There are also accuracy issues. If you try to solve $(X - 1)^5 (X - 2) = 0$ starting with $a = 0$ using double-precision numbers in Mathematica you get:

   ![Mathematica output](image)

   If you solve the expanded version of the polynomial $-2 + 9X - 16X^2 + 14X^3 - 6X^4 + X^5$.starting with 0 you get:
The slight decrease in precision computing the second version compared to the first is enough to spoil the algorithm.

(4) How close is an approximate solution to the exact solution?
(a) As with Newton and secant, the best criterion that an estimate $a$ is close to an exact solution is that the estimate does not change significantly from one iteration to the next.

(5) The amount of work required for each step is one evaluation of the function and its derivative and second derivative (and a couple of arithmetic operations).
7. Homework Due Monday 4/18

(1) Use the method of false position to find a root of \(x^5 + 1000x^4 + x^3 + x^2 + x + 1 = 0\)

(2) Write programs to (a) implement polynomial division; and (b) implement Laguerre’s method, and use them to find all the roots of \(x^5 - 9x^4 + 17x^3 + 29x^2 - 60x - 50 = 0\).