Bisection Method

- Recall that we start with an interval $[a, b]$ where $f(a)f(b) < 0$; the IVT guarantees that a zero or root of $f(x)$ lies in (a, b).
- We then take the midpoint of the interval $[a, b]$ as our approximation and check to see if the root is in $[a, \frac{a+b}{2}]$ or in $[\frac{a+b}{2}, b]$.
  - To do this we simply evaluate $f(\frac{a+b}{2})$ and compare the sign with that of $f(a)$ and $f(b)$.
  - We choose the interval which has function values at the endpoints of opposite signs.
- This process is repeated and we are guaranteed that we can get as close to a root as we want by continually halving the interval.
- We saw that the actual error at the nth step ($|x^* - x^n|$, where $f(x^*) = 0$)

is no larger than

$$\frac{b-a}{2^n}$$

where $[a, b]$ is the initial interval bracketing the root.

Implementing the Bisection Method

Strategy:

- We will have a function routine which as input has an x value and as output the function value.
- In our algorithm, we will start with an interval, say $[a, b]$ containing the root.
- At the end of each iteration we will have a new interval of half the length of the previous interval, either $[a, \dfrac{a+b}{2}]$ or $[\dfrac{a+b}{2}, b]$.
- We will still call this interval $[a, b]$; we will simply modify a or b to be the midpoint.
- Once we have a midpoint of a new interval we will check the error $\frac{b-a}{2^n}$ to see if it is less than the tolerance.

- To begin with we need an interval, say $[a, b]$, which brackets the root.
  - The values for a and b should be input by the user
  - The fact that $[a, b]$ brackets the root should be verified by the program; if it is not satisfied then an error message should be output and execution terminated.
- We will also need a maximum number of steps to do and a tolerance to check for convergence of the method; for now we can "hard wire" these.
- Evaluate function at initial a and b, call them `f_at_a, f_at_b`
- We will have an iteration loop which must accomplish the following
  - Evaluate midpoint, call it `midpoint`
  - Check for convergence; i.e., if $\dfrac{b-a}{2}$ is less than tolerance (where $[a, b]$ is the current interval, not the initial one); we will also check the residual error $f(\frac{a+b}{2})$.
  - Evaluate f at midpoint, call it `f_at_midpoint`

- Evaluate (for example) $f(a)f(\frac{a+b}{2})$ call this number `value`
- `if ( value ) == 0` terminate because root has been found
- `if ( value ) < 0` then we know that the root is in $\left(a, \frac{a+b}{2}\right)$ so move the endpoint b to the midpoint. Also, since we have already evaluated f at the midpoint we might as well use it; i.e.,

  ```
  b= midpoint
  f_at_b = f_at_midpoint
  ```

- `if ( value ) > 0` then we know that the root is in $\left(\frac{a+b}{2}, b\right)$ so move the endpoint a to the midpoint; i.e.,

  ```
  a= midpoint
  f_at_a = f_at_midpoint
  ```

Code for the Bisection Method

Now let's look at the actual code `bisection.f90` for the bisection method for our example of $f(x) = x^2 - 5$ on $[2,3]$ and the results we get when we run it.

We generate the following sequence of approximations and their errors. The errors we provide are

- the relative (actual) error which is given by $\dfrac{|\sqrt{5} - \text{approximation}|}{\sqrt{5}}$;
- the error bound given by $\dfrac{b-a}{2^n}$ where $[a,b]$ is the original interval
- the residual error which is given by $|f(x)|$ evaluated at our approximation, the midpoint.

approximation	relative error	$\dfrac{b-a}{2^n}$	residual error
2.5	-0.118034	0.5	1.25
2.25	-0.62306 $\times 10^{-2}$	0.25	6.25$\times 10^{-2}$
2.125	0.496711 $\times 10^{-1}$	0.125	0.484
2.1875	0.217203 $\times 10^{-1}$	0.625 $\times 10^{-1}$	0.215
2.21875	0.774483 $\times 10^{-2}$	0.3125$\times 10^{-1}$	7.715$\times 10^{-2}$
2.23438	0.757123 $\times 10^{-3}$	0.15625$\times 10^{-1}$	7.568$\times 10^{-3}$
2.24219	0.273673 $\times 10^{-3}$	0.78125$\times 10^{-2}$	2.740$\times 10^{-2}$

We note that the error oscillates; it is not monotonically decreasing. Also the actual error is always less than the estimate of the error.

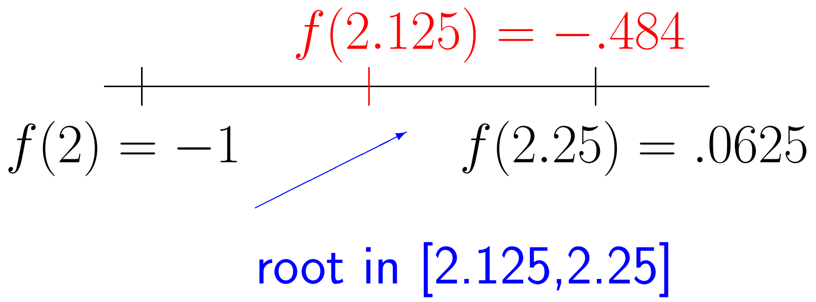
Other Methods for Finding a Root of a Single Nonlinear Equation

- The Bisection Method had several disadvantages.
  - It requires an interval bracketing the root which may not be easily determined.
  - The convergence is slow and not monotonic.
  - It is not really feasible in higher dimensions, i.e, for solving more than one nonlinear equation.
- The main advantage of the Bisection Method is that it will always converge to a root.
- However this advantage does not outweigh the disadvantages.
- We will first look at a straightforward way to improve the speed of convergence of the Bisection Method but this method still requires an interval bracketing the root which is still a very strenuous requirement.

- Newton's Method (which you probably studied in Calculus I) is probably the most widely used method for determining the root of a nonlinear equation. When it converges, the convergence is fast. A possible shortcoming is that it requires the calculation of the derivative of the function.
- The Secant Method is also widely used; it has the disadvantage that it is slightly slower than Newton's Method to converge but the advantage that it does not require the derivative of the function.
- Both Newton's and the Secant method may converge or may not converge depending on the initial guess. Neither requires knowing an interval bracketing the root.

The Method of False Position - Regula Falsi

- We noticed in the Bisection Method that the actual error can oscillate.
- This is because the midpoint of the interval can be very close to the root but the next iteration takes us farther away. This was the case in our example.



- Regula Falsi addresses this problem by replacing the midpoint $\dfrac{a+b}{2}$ with a weighted average of a and b. Note that in the Bisection Methods the weights are equal – both are $1/2$.
- How should we choose the weights?
  - Since we are looking for a point x such that $f(x) = 0$ we should give

more weight to the endpoint whose function value is smallest.

– We can simply choose the weights in a linear fashion. That is, if we have the two points $(a, f(a)$ and $(b, f(b))$ we can write the equation of the line through these points and where it crosses the x-axis is our guess.

– For the equation of the line through $(a, f(a))$ and $(b, f(b))$ (using the point-slope formula) we have

$$y - f(a) = \frac{f(b) - f(a)}{b - a}(x - a)$$

By setting $y = 0$ and solving for x we get

$$x = \frac{f(b)a - f(a)b}{f(b) - f(a)}$$

– Note that the weight for a is just $\dfrac{f(b)}{f(b) - f(a)}$ and for b is just $\dfrac{f(a)}{f(b) - f(a)}$

– So if $|f(b)| = |f(a)|$ we simply choose the midpoint.

– If $|f(b)| > |f(a)|$ then we weight the point a more heavily than b and choose our next guess closer to a.

- It should be a simply task to modify our code for the Bisection Method to incorporate this change to have the Regula Falsi implemented.
- Of course our estimate for the error $\frac{b-a}{2^n}$ is no longer valid.
- The reason for the name of this method is that one endpoint will remain fixed for all iterations.
- Because an endpoint remains fixed, this method can be slow to converge. Modifications to this method can make it more useful but Newton's Method or the Secant Method is usually preferred.
- An Historical Note - Wikipedia says that the oldest surviving document demonstrating knowledge and proficiency in this method is an Indian mathematical text dating from the third century BC but it was only applied to linear problems. Fibonacci quotes the method in his 1202 AD book.

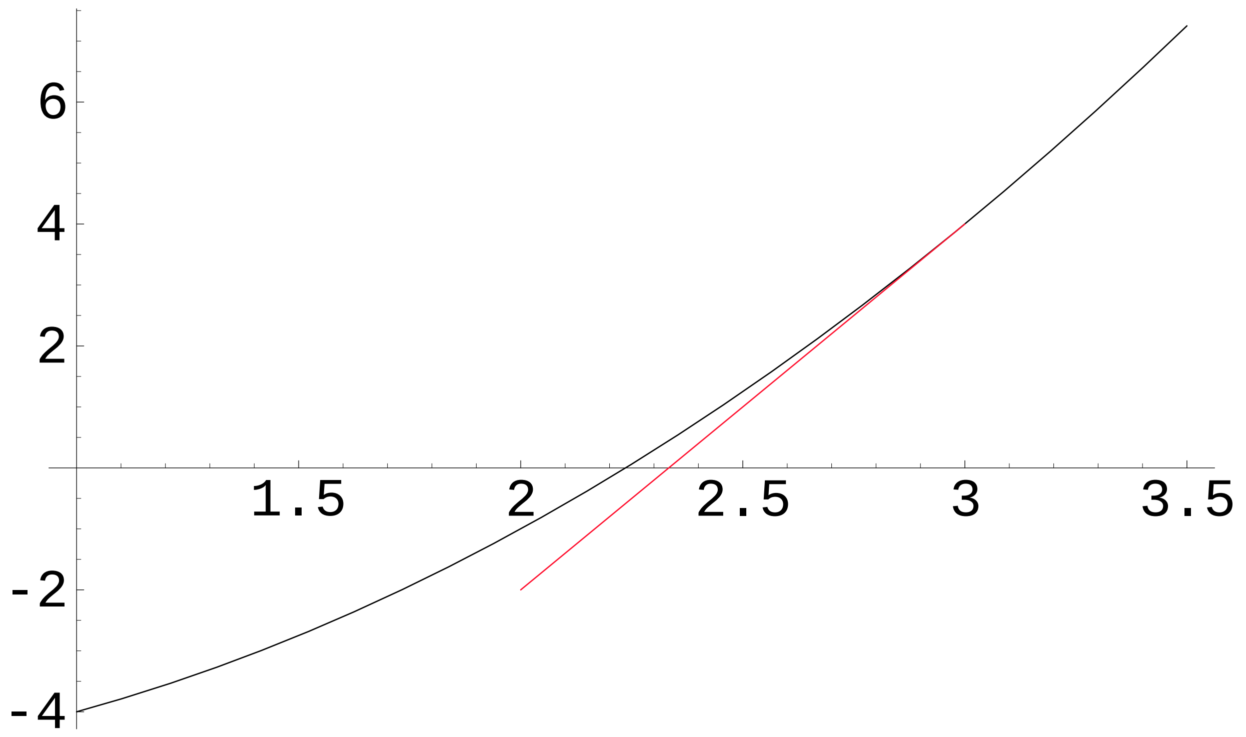
Sample Results for $f(x) = x^2 - 5$

	bisection	**regula falsi**
1	2.5	2.2
2	2.25	2.23077
3	2.125	2.235294
4	2.1875	2.235955
5	2.21875	2.236052
6	2.24219	2.236066

$\sqrt{5} = 2.236067977$

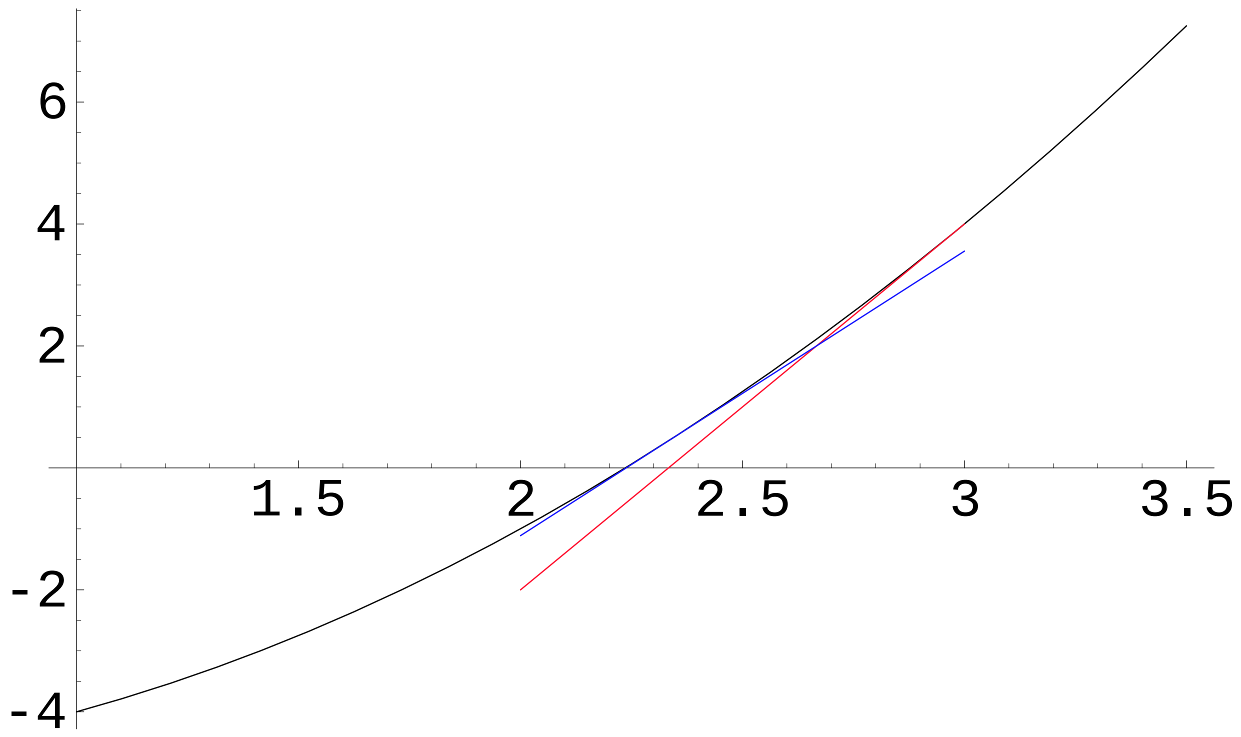
Newton's Method

- Recall that we want a formula for generating a sequence of iterates $x^0, x^1, x^2, \ldots$ which approach our root where x^0 is our initial guess.
- Newton's Method has a very simple geometric interpretation for approximating the root of a differentiable function $f(x)$.
- Start with an initial guess x^0 to the root.
- At the point $(x^0, f(x^0))$ draw the tangent line (that is, the line which passes through this point with slope $f'(x^0)$).
- Your next approximation, x^1, will be the point where this tangent line crosses the $x-$axis.



Here we have the function $x^2 - 5$ and set $x^0 = 3$.

The tangent line at (3,4) (indicated by the red line) crosses the axis at $x = 2\frac{1}{3}$ which is our next approximation, x^1.



To get x^2, we write the equation of the tangent line to x^2-5 at $(7/3, f(7/3)) = (7/3, 4/9)$ (indicated by the blue line).

Where it crosses the x-axis is our next approximation $x^2 = 2.2381$. Note that the convergence seems very fast since $\sqrt{5} = 2.23607$

- We now want to use this geometric interpretation to get a formula for determining the point where the tangent line crosses the x-axis.
- Suppose we have a point $(\tilde{x}, f(\tilde{x}))$ on the curve $y = f(x)$. Then the tangent line at this point (using the point-slope formula) is just

$$y - f(\tilde{x}) = f'(\tilde{x})(x - \tilde{x})$$

- Now this line crosses the x-axis when $y = 0$ so if we solve this for x we get

$$0 - f(\tilde{x}) = f'(\tilde{x})(x - \tilde{x}) \implies x = \tilde{x} - \frac{f(\tilde{x})}{f'(\tilde{x})}$$

- We now have a formula for determining our iterates. For example, if we have $\tilde{x} = x^0$ then

$$x^1 = x^0 - \frac{f(x^0)}{f'(x^0)}$$

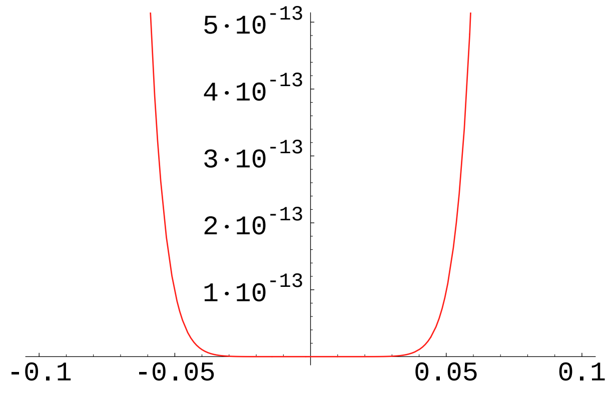
and x^1 will be the point where the tangent line at $(x^0, f(x^0))$ crosses the x-axis.

Algorithm for Newton's Method

```
Given $x^0$
For $k = 0, 1, 2, \ldots,$ max_iterations
    compute
        $x^{k+1} = x^k - \frac{f(x^k)}{f'(x^k)}$
```

How do we decide when to stop this iteration?

- Remember that we always incorporate a maximum number of iterations but the method may have reached a satisfactory answer before this.
- In the Bisection Method it was easy to estimate the error, but in general this is not possible.
- Since we are attempting to find a zero of the function, i.e., when $f(x) = 0$ one possibile stopping criteria would be to check the function value at each iteration, $f(x^k)$ and when it is less than some tolerance, stop. The problem with this criteria alone is indicated by the following graph where the function is very "flat" near the root. The function has a root at $x = 0$ but the function value is already very small at values such as 0.04.



- Another possible stopping criteria is to see if the iterates are getting close enough together, i.e.,

$$|x^{k+1} - x^k| < \text{tolerance}$$

- To see the problem with this test, consider the following two sets of consecutive approximations

$$100.01, 100.005 \rightarrow 100$$

  and

$$0.001, 0.0005 \rightarrow 0.0$$

- In the first set the absolute difference in the two iterates is 0.005 and the second one is 0.0005 so if our tolerance was say 0.001 then we would stop the second iteration but not the first.

- If we look at the relative difference we see that in the first case

$$\frac{|x^{k+1} - x^k|}{|x^{k+1}|} = \frac{|100.01 - 100.005|}{100.01} = 0.00004999 \implies .005\%$$

whereas in the second case

$$\frac{|0.001 - 0.0005|}{0.0005} = 1.0 \implies 100\%$$

so we see that the first one is better than the second! The problem is that in the second set the numbers are already small.

- The moral of this example is that we should always use relative differences or relative errors (if we know the solution).
- Often we will test our codes on a problem whose solution is known. In this case we may want to calculate a relative actual error at the kth step

$$\frac{|\text{exact solution} - x^k|}{|\text{exact solution}|}$$

- A good strategy for a problem we don't have the solution to is to check both the relative difference and the residual error.

Algorithm for Newton's Method with stopping criteria

Given x^0, a maximum number of iterations, `max_iterations` and a tolerance, `tol`

```
For $k = 0, 1, 2, \ldots,$ max_iterations
    compute
        $x^{k+1} = x^k - \frac{f(x^k)}{f'(x^k)}$
    if $\frac{|x^{k+1} - x^k|}{|x^{k+1}|} \leq$ tol  and  $f(x^{k+1}) \leq$ tol  stop
```

Implementing Newton's Method

- One difference from the Bisection Method is that we need two function routines – one for the function $f(x)$ and one for its derivative $f'(x)$.
- To implement this formula we do not really need to have variables called `x0, x1, x2, x3, ...`.
- Really all we need at any time are two iterates x^{k+1} and x^k so we could, e.g., call them `xk` and `xkp1`. Once we have checked to see if the convergence tolerance is met, then we simply replace `xk` with `xkp1`.
- Pseudocode for Newton's Method could be written as follows.

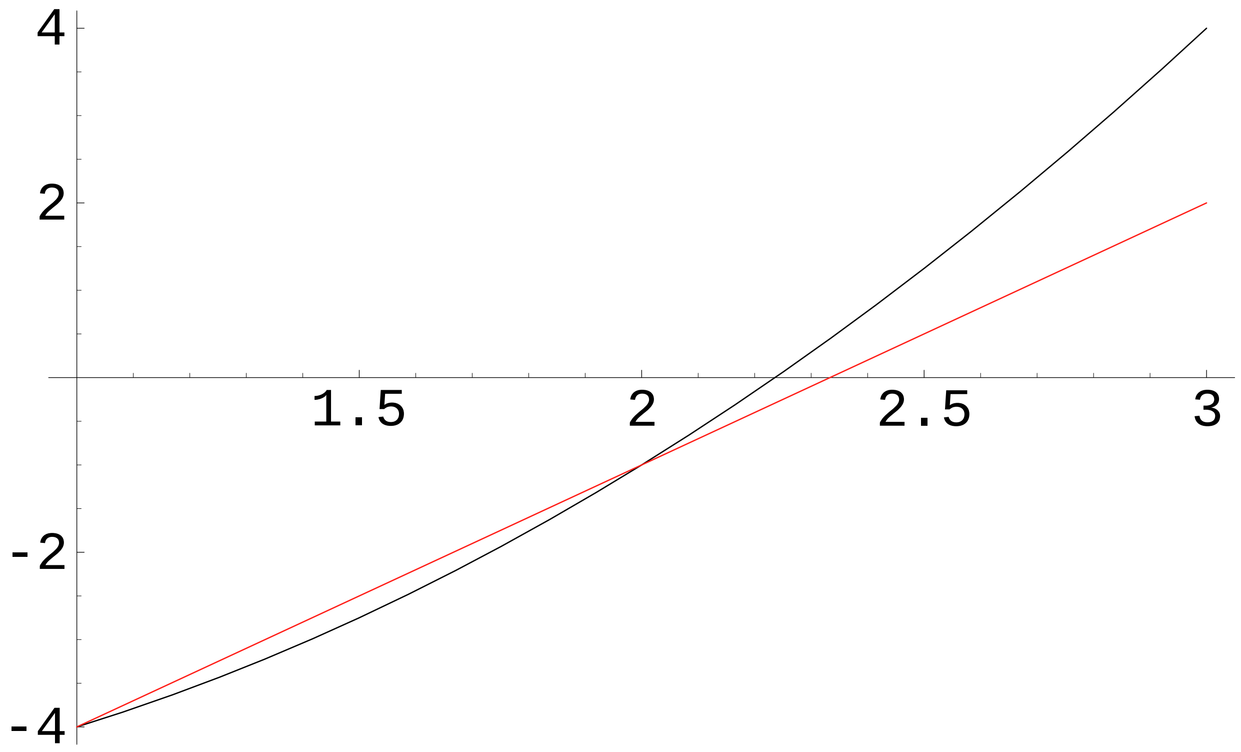
```
input   xk, max_iterations, tol

do k = 1, max_iterations

    xkp1 = xk - f ( xk ) / f' (xk)

    if ( | xkp1 - xk |) / |xkp1 | $\leq$ tol ) and if ( f(xkp1) $\leq$ tol)   stop

    xk = xkp1

end do
```

All we need to do is add routines for calculating the function and its derivative.

Let's look at the code newton.f90 to see how this is done.

Secant Method

- The Secant Method also has a very simple geometric interpretation.
- Here we begin with two initial guesses x^0 and x^1. However, they do not have to bracket the root like in the Bisection Method.
- We draw the secant line joining the points $(x^0, f(x^0))$ and $(x^1, f(x^1))$ and where it crosses the x-axis is our next approximation.
- In our example of $f(x) = x^2 - 5$ with $x^0 = 1$ and $x^1 = 2$ (which do not bracket the root) we get the following graph. Here the secant line crosses the x axis at $x = 2.3333$ which becomes our next approximation x^2.



- As in Newton's Method, we can derive a formula for the point where the secant line crosses the x axis to get a formula for our iterates.
- However, there is an easy way to remember the formula for the secant method. Recall from Calculus I that we used the secant line between a point say $(a, f(a))$ and $(b, f(b))$ to approximate the derivative at say $x = a$. As the distance between a and b shrinks, the secant line becomes a better

approximation. So we can view the secant line as an approximation to the tangent line. The Secant Method can be obtained by replacing the derivative term $f'(x^k)$ in Newton's Method and with the slope of the secant line $(f(x^k) - f(x^{k-1})/(x^k - x^{k-1})$.

So we have

$$x^{k+1} = x^k - \frac{f(x^k)}{\Big(f(x^k) - f(x^{k-1})\Big)/(x^k - x^{k-1})} = x^k - \frac{f(x^k)(x^k - x^{k-1})}{f(x^k) - f(x^{k-1})}$$

Algorithm for Secant Method

Given x^0, x^1, a maximum number of iterations, `max_iterations` and a tolerance, `tol`

```
For $k = 1, 2, \ldots,$ max_iterations
    compute
        $$x^{k+1} = x^k - \frac{f(x^k)(x^k - x^{k-1})}{f(x^k) - f(x^{k-1})}$$
    if $\frac{|x^{k+1} - x^k|}{|x^{k+1}|} <$ tol and $f(x^{k+1}) \leq$ tol    stop
```

- The important difference in the Secant Method is that it does not require the derivative.
- It is also a little slower converging than Newton's Method.
- It is imperative that you remember that both methods may fail to converge for a given choice of initial conditions. This is illustrated in the following examples.

Computational Examples for Newton & Secant Methods

$f(x) = x^2 - 5$

	Newton	Secant
0	6.0	4.0, 6.0
1	3.416667	2.90
2	2.440041	2.516854
3	2.244593	2.238086
4	2.236084	2.236084
5	2.236068	2.236068

$f(x) = x^7 - e^x + 1$

	Newton	Newton	Secant
0	2.0	0.5	.5, 2
1	1.723996	8.365×10^{-2}	0.50786
2	1.495815	0.340285×10^{-3}	0.158335
3	1.3116	5.7681×10^{-6}	8.450058×10^{-2}
4	1.190171		
5	1.124234		
8	1.105785		8.92896×10^{-9}

With different initial guesses the iteration converges to different roots or it may even fail to converge.

Implementing the Secant Method

- For the Secant Method we only need one function for evaluating $f(x)$.
- Unlike Newton's Method we need to have 3 iterations at a single time.
  - We read in say `xkm1, xk`
  - We compute `xkp1` from `xkm1, xk` and their function values.
  - We then check for convergence and if it we need to do more iterations, we replace `xkm1` with `xk` and `xk` with `xkp1`.
- You will be asked to make a copy of the program implementing Newton's Method and modify it to implement the Secant Method.

Classwork

- Make a copy of the code `bisection.f90` and convert it to carry out the Regula Falsi method. Reproduce the computational results in the notes to make sure your code is working properly.
- Make a copy of the code `newton.f90` and convert it to implement the Secant method. Reproduce the computational results in the notes to make sure your code is working properly.