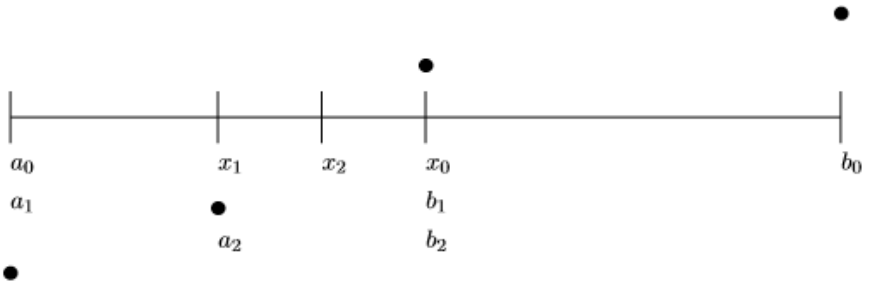
1 Bisection method

Recall the bisection method. Suppose that $c = f(a) < 0$ and $d = f(b) > 0$. If f is continuous, then obviously it must be zero at some x* between a and b. The bisection method then consists of looking half way between a and b for the zero of f, i.e. let $x = (a + b)/2$ and evaluate $y = f(x)$. Unless this is zero, then from the signs of c, d and y we can decide which new interval to subdivide. In particular, if c and y have the same sign, then $[x, b]$ should be the new interval, but if c and y have different signs, then $[a, x]$ should be the new interval. (See Figure 1)

Figure 1: Bisection method



Deciding to do different things in different situations in a program is called flow control. The most common way to do this is the $if...else...end$ statement which is an extension of the $if...end$ statement we have used already.

1.1 Bounding the Error

One good thing about the bisection method, that we dont have with Newtons method, is that we always know that the actual solution x is inside the current interval $[a, b]$, since $f(a)$ and $f(b)$ have different signs. This allows us to be sure about what the maximum error can be. Precisely, the error is always less than half of the length of the current interval $[a, b]$, i.e.

$$Absoluteerror = |x - x*| < (b - a)/2 \tag{1}$$

where x is the center point between the current a and b.

The following function program (also available on the internet) does n iterations of the bisection method and returns not only the final value, but also the maximum possible error:

```
function [x, e] = mybisect(f,a,b,n)
    % function [x e] = mybisect(f,a,b,n)
    % Does n iterations of the bisection method for a
        function f
    % Inputs: f -- a function
    %         a,b -- left and right edges of the interval
    %         n -- the number of bisections to do.
    % Outputs: x -- the estimated solution of f(x) = 0
    %          e -- an upper bound on the error

    % evaluate at the ends and make sure there is a sign
        change
    c = f(a); d = f(b);
    if c*d > 0.0
        error(Function has same sign at both endpoints.
            )
    end

    disp('  x  y')
    for i = 1:n
        % find the middle and evaluate there
        x = (a + b)/2;
        y = f(x);
        disp([ x y])
        if y == 0.0 % solved the equation exactly
            e = 0;
            break % jumps out of the for loop
        end
        % decide which half to keep , so that the signs
            at the ends differ
        if c*y < 0
            b=x;
        else
            a=x;
        end
    end % set the best estimate for x and the error bound
    x = (a + b)/2;
    e = (b-a)/2;
end
```

Another important aspect of bisection is that it always works. We saw that Newton's method can fail to converge to x* if x_0 is not close enough to x*. In contrast, the current interval $[a, b]$ in bisection will always get decreased by a factor of 2 at each step and so it will always eventually shrink down as small as you want it.

1.2 Locating a root

The bisection method and Newtons method are both used to obtain closer and closer approximations of a solution, but both require starting places. The bisection method requires two points a and b that have a root between them, and Newtons method requires one point x_0 which is reasonably close to a root. How do you come up with these starting points? It depends. If you are solving an equation once, then the best thing to do first is to just graph it. From an accurate graph you can see approximately where the graph crosses zero.
There are other situations where you are not just solving an equation once, but have to solve the same equation many times, but with different coefficients. This happens often when you are developing software for a specific application. In this situation the first thing you want to take advantage of is the natural domain of the problem, i.e. on what interval is a solution physically reasonable. If that is known, then it is easy to get close to the root by simply checking the sign of the function at a fixed number of points inside the interval. Whenever the sign changes from one point to the next, there is a root between those points. The following program will look for the roots of a function f on a specified interval $[a_0, b_0]$.

```
function [a,b] = myrootfind(f,a0,b0) s
    % function [a,b] = myrootfind(f,a0,b0)
    % Looks for subintervals where the function changes
      sign
    % Inputs: f -- a function
    %         a0 -- the left edge of the domain
    %         b0 -- the right edge of the domain
    % Outputs: a -- an array , giving the left edges of
      subintervals
    %                on which f changes sign
    %          b -- an array , giving the right edges of
      the subintervals
    n = 1001; % number of test points to use
    a = []; % start empty array
    b = [];
    % split the interval into n-1 intervals and evaluate
      at the break points
    x = linspace(a0,b0,n);
    y = f(x);
    % loop through the intervals
    for i = 1:(n-1)
        if y(i)*y(i+1) < 0 % The sign changed , record it
            a = [a x(i)];
            b = [b x(i+1)];
        end
    end
    if size(a,1) == 0
```

```
24            warning('no roots were found')
25        end
26    end
```

The final situation is writing a program that will look for roots with no given information. This is a difficult problem and one that is not often encountered in engineering applications.
Once a root has been located on an interval [a,b], these a and b can serve as the beginning points for the bisection and secant methods (see the next section). For Newtons method one would want to choose x0 between a and b. One obvious choice would be to let x_0 be the bisector of a and b, i.e. $x_0 = (a+b)/2$. An even better choice would be to use the secant method to choose x_0.

2 Secant method

In this section, we introduce two additional methods to find numerical solutions of the equation $f(x) = 0$. Both of these methods are based on approximating the function by secant lines just as Newtons method was based on approximating the function by tangent lines.

The secant method requires two initial approximations x_0 and x_1, preferably both reasonably close to the solution x. From x_0 and x_1 we can determine that the points $(x_0, y_0 = f(x_0))and(x_1, y_1 = f(x_1))$ both lie on the graph of f. Connecting these points gives the (secant) line.

$$y - y_1 = \frac{y_1 - y_0}{x_1 - x_0}(x - x_1) \tag{2}$$

Since we want $f(x) = 0$, we set $y = 0$, solve for x, and use that as our next approximation. Repeating this process gives us the iteration

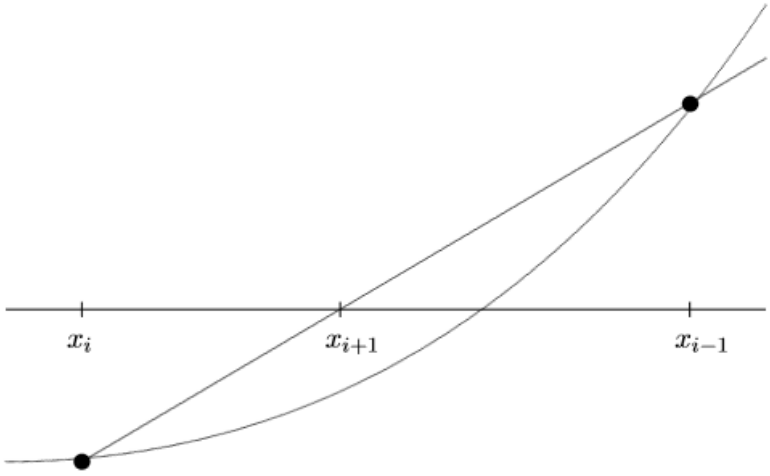
$$x_{i+1} = x_i - \frac{x_i - x_{i-1}}{y_i - y_{i-1}}y_i \tag{3}$$

with $y_i = f(x_i)$. See Figure 2 for an illustration.
For example, suppose $f(x) = x^4 - 5$, which has a solution x* $= 4\sqrt{5} \approx 1.5$. Choose $x_0 = 1$ and $x_1 = 2$ as initial approximations. Next we have that $y_0 = f(1) = -4and y_1 = f(2) = 11$. We may then calculate x_2 from the formula 3.

$$x_2 = 2 - \frac{2-1}{11-(-4)}11 = \frac{19}{15} \approx 1.2666... \tag{4}$$

Plugging $x_2 = 19/15$ into $f(x)$ we obtain $y_2 = f(19/15) \approx -2.425758....$ In the next step, we would use $x_1 = 2$ and $x_2 = 19/15$ in the formula 3 to find x_3 and so on.
Below is a program for the secant method. Notice that it requires two input guesses x_0 and x_1, but it does not require the derivative to be input.

Figure 2: Secant method where root is bracketed




```
function x = mysecant(f,x0,x1,n)
    % Solves f(x) = 0 by doing n steps of the secant
       method
    %   starting with x0 and x1.
    % Inputs: f -- the function
    %         x0 -- starting guess , a number
    %         x1 -- second starting guess
    %         n -- the number of steps to do
    % Output: x -- the approximate solution
    y0 = f(x0);
    y1 = f(x1);
    for i = 1:n      % Do n times
        x = x1 - (x1-x0)*y1/(y1-y0) % secant formula.
        y=f(x) % y value at the new approximate solution.
        % Move numbers to get ready for the next step
        x0=x1;
        y0=y1;
        x1=x;
        y1=y;
    end
end
```

3 The *Regula Falsi* (False Position) Method

The *Regula Falsi* method is a combination of the secant method and bisection method. As in the bisection method, we have to start with two approximations a and b for which $f(a)$ and $f(b)$ have different signs. As in the secant method, we follow the secant line to get a new approximation, which gives a formula similar to 3.

$$x = b - \frac{b-a}{f(b)-f(a)} f(b) \tag{5}$$

.

Then, as in the bisection method, we check the sign of $f(x)$; if it is the same as the sign of $f(a)$ then x becomes the new a and otherwise let x becomes the new b. Note that in general either $a \rightarrow$x* or $b \rightarrow$x* but not both, so $b - a \not\rightarrow 0$. For example, for the function in Figure 1, $a \rightarrow$x* but b would never move.

3.1 Convergence

If we can begin with a good choice x_0, then Newtons method will converge to x* rapidly. The secant method is a little slower than Newtons method and the *Regula Falsi* method is slightly slower than that. However, both are still much faster than the bisection method.
If we do not have a good starting point or interval, then the secant method, just like Newtons method, can fail altogether. The *Regula Falsi* method, just like the bisection method, always works because it keeps the solution inside a definite interval.

3.2 Simulations and Experiments

Although Newtons method converges faster than any other method, there are contexts when it is not convenient, or even impossible. One obvious situation is when it is difficult to calculate a formula for $f'(x)$ even though one knows the formula for $f(x)$. This is often the case when $f(x)$ is not defined explicitly, but implicitly. There are other situations, which are very common in engineering and science, where even a formula for $f(x)$ is not known. This happens when $f(x)$ is the result of experiment or simulation rather than a formula. In such situations, the secant method is usually the best choice.

4 Questions (To be completed in lab)

1. Modify mybisect to solve until the absolute error is bounded by a given tolerance. Use a while loop to do this. Run your program on the function $f(x) = 2x^3 + 3x - 1$ with starting interval $[0, 1]$ and a tolerance of $10^(-8)$. How many steps does the program use to achieve this tolerance? (You can count the step by adding 1 to a counting variable i in the loop of the program.) How big is the final residual $f(x)$? Turn in your program and a brief summary of the results.

2. Show that $f(x) = 0$ has a root α between 1.4 and 1.5.

$$f(x) = x^3 - \frac{7}{x} + 2, x > 0 \tag{6}$$

3. Modify mysecant.m to iterate until the absolute value of the residual is less than a given tolerance. (Let tol be an input instead of n). Modify comments accordingly. Test your program with following function $f(x) = x^3 - 4$ with starting points $x_{-1} = 1$ and $x_0 = 3$.

4. Find root of $2cosh(x)sin(x) = 1$, using secant method where two starting values are 0.4 and 0.5.

5. Write MATLAB code for residual falsi method and test it with $f(x) = x^3 - 4$ and interval $[1, 3]$.

5 Homework

1. Consider the equation $230x^4 + 18x^3 + 9x^2 - 22x - 9 = 0$ has two real roots, one in $[-1, 0]$ and the other in the interval $[0, 1]$. Attempt to approximate these roots to within 10^{-6} using (i) Method of false position, (ii) Secant method and (iii) Newton's method. Use the end points of each interval as the initial approximations in (i) and (ii); the midpoints as the initial approximation in (iii).

2. Compute $3^{1/2}$ to 10 digits of accuracy using Newton's method.

3. Using Newton's method in its multivariable form to find a solution of

$$x_1^2 + x_2^2 + x_3^2 = 100 \tag{7}$$

$$x_1 x_2 x_3 = 1 \tag{8}$$

$$x_1 - x_2 - sin(x_3) = 0 \tag{9}$$

4. Consider Newton's method for minimizing F(x):

$$x_{k+1} = x_k - \frac{F'(x_k)}{F''(x_k)} \tag{10}$$

In what follows we'll take $F(x) = 1 + \int_0^x atan(y)dy$.
(a) Show that F is strictly convex, i.e. $F''(x) > 0$. (Strictly convex functions always have a unique minimum.)
(b) Find one value of the starting guess x_0 for which Newton's method converges, and one for which it diverges. (Convexity does not ensure convergence).

5. Compute 25^3 to 10 digits of accuracy using Bisection's method.

6. The function $f(x) = tan(\pi x) - 6$ has a zero at $(1/\pi)arctan6 \approx 0.447431543$. Let $p_0 = 0$ and $p_1 = 0.48$, and use ten iterations of each of the following methods to approximate this root. Which method is most successful and why?
(a) Bisection method
(b) Method of False Position
(c) Secant method

7. The equation $4x^2 - exp(x) - exp(-x) = 0$ has two positive solutions x_1 and x_2. Use Newton's method to approximate the solution to within 10^{-5}.
8. You would like to precisely determine the resistance of an electrical component. The advertised value is $R = 2(Ohms)$. When connecting the resistance to a battery, you measure the voltage and current with a (cheap) multimeter as $V = 2.9V(Volts)$ and $I = 1.4A(Amps)$ respectively. You figure that Ohms law $V = RI$ is not exactly satisfied because there are errors both in the measured values of V, I, and in the advertised value of R. Find the best fit for V, I, and R by finding the minimum value of the function.