

Finding the roots of a function is a common problem in mathematics, science, and engineering. Once a function gets to be more than fifth order a deterministic solution to the factoring problem is no longer possible and we often resort to guessing. Using loops a computer can make millions of guesses a second so we present here some introductory methods of finding roots by making successive guesses. We will limit our roots to real roots and for convenience we will use polynomials for functions.

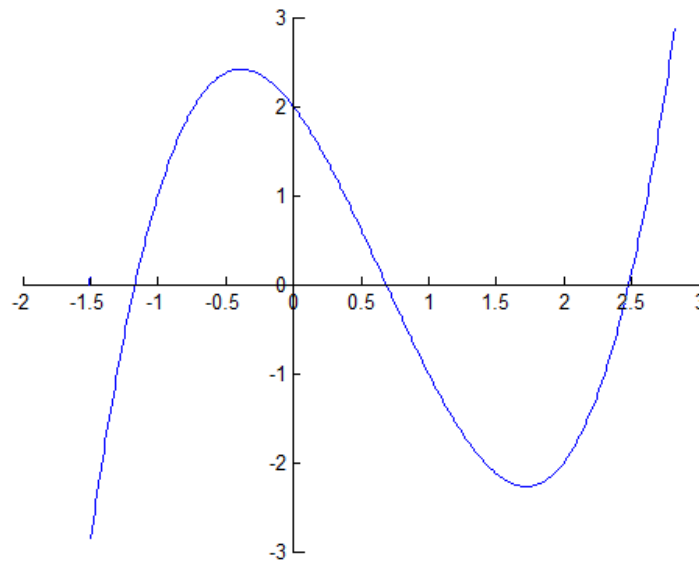


Figure 1

The function $y = x^3 - 2x^2 - 2x + 2$

Consider the function $y = x^3 - 2x^2 - 2x + 2$ shown in Figure 1. This function has three real roots. The first is between -1.5 and -1, the second between .5 and 1.0 and the third between 2 and 2.5. We could write a computer program which would guess at the exact root locations by a trial and error method. Finding the exact roots may take an enormous amount of computing time so generally we attempt to find the roots to within some user specified limits of accuracy. For example, suppose the user specified an accuracy of .1. We could then write a computer program that found the value of the function from say -1.5 to +2.5 in increments of 0.1 and use those values of x which make the function closest to zero. Alternatively, we could look for where the function crosses the x axis. So if at some value n, $f(n)$ is negative and at the next point $f(n+\text{increment})$ the value is positive we know the root lies between n and n + increment. To find zero crossings we need only find the two points where the product $f(n) * f(n+\text{increment})$ is negative. The pseudocode to do this might look like the following:

```

x = start;
while(x <= end)
  {xNew = x + increment
  if(f(x)*f(xNew) < 0)
    root found between x and xNew
  x = xNew
  }

```

Note that when the increment is very small, say 10^{-6} or less there are a tremendous number of calculations to be made.

One way to greatly increase the efficiency of the program would be to adjust the increment as the program runs making it smaller when it is in the vicinity of a root. To do this we could set an initial increment to a coarse value say 0.1. When we find a zero crossing we back up to the point just before the crossing, reduce the increment by a factor of 10 and find the crossing again. We could repeat this procedure until the increment is at or below some user specified value. The pseudocode for this version of the program would look like this:

```

x = start;
while(x < end)
  {xNew = x + increment
  if(f(x)*f(xNew) < 0)
    {divide increment by 10
    xNew = x + increment
    while(increment > UserIncrement)
      {while((f(x)*f(xNew) >= 0) && x < end)
      {x = xNew;
      xNew = x + increment;
      }
      divide increment by 10
      xNew = x + xCoarseIncr;
    }
    Root found a x.
    Reset increment back to its coarse value
  }
  Add increment to x to get past found root.
}

```

There are some potential pitfalls to this technique.

1. If the initial coarse increment is too big it can step past two roots which are adjacent and miss them both.
2. Each time through the loop we calculate $f(x)$ and $f(xNew)$ and yet x becomes $xNew$ in the next loop iteration. For functions that are complex we could save a function evaluation by saving $f(xNew)$ and using it for $f(x)$ in the next iteration.
3. This technique finds only real roots. For complex roots this technique would have to search an area of the complex plane instead of a section of the x axis.
4. Newton's method of using the slope to zero in on the root maybe more efficient in some cases.