Ideal:

1. Pre-requisites can be linked to a video, not the actual material (As there is a choice to watch the video or not)
2. Each coding exercise will be divided into three parts. Basic, intermediate and Advanced. No code will be provided, but hints will be there for certain difficult parts or non-obvious optimization techniques. Each code exercise will be a challenge, a level above the theory we taught. In most cases, there will be sequence of tasks they have to achieve to find the final output. Let it be frustrating, it will teach them resourcefulness.

MAKE IT INTERACTIVE

**Theory:**

Bisection Method is a simple idea to find roots of an equation or zeroes of a function\*  
\*both roots and zeroes are often used interchangeably but an equation has roots and a function has zeroes.

“ Hey! I know how to find the root of a function. It’s called the quadratic formula.”   
Okay, let’s test it. Use quadratic formula to find the root(s) of x2 – 3x + 4.

Now, find the root of x3-7x+6

This was tougher, right?

“Why do I have to learn a new method for finding roots?”  
In non-linear equations and transcendental equations (equations containing non algebraic functions such as 3x + sin(x) - exp(x)), root finding can be pain.   
Some direct methods exist (such as factor method for find roots of polynomial) but there are many equations that cannot be solved directly.  
In this case, we use Approximate Solution Techniques

One way to obtain an approximate solution is to draw the graph of the function and see where it touches the x-axis. Let’s estimate the root of x^3 – x -2  
(plot graph)

We can see the root is somewhere between -2 and 2.   
But it is difficult to pin-point the exact root from this graph.  
Graphical Methods are great for rough estimates but they lack precision.

So, we’ll use a numerical method: an algorithm to find a close enough approximation by repeating the procedure again and again.

The general idea of a numerical method is that we start with an initial guess. And then at each step, we improve our guess till we get close enough to the answer. These steps are also called iterations.

The good news is, we can simply write a program to feed the equation to the computer and then let it run the iterations over and over again and output the final answer to us. This abstractions helps increase our productivity.

There are two kinds of numerical methods for finding roots:  
1. Bracketing Method (when we start with an interval and make it smaller)  
2. Open Method (when we start with a point and move towards the root)  
  
Open Methods are faster but do not always converge (i.e. they diverge or move away from the root sometimes) and Bracketing Methods always converge (move closer to the root) but sometimes at painfully slow speed.

Bisection method is a type of bracketing method.

Bisection Method is based on the idea of a continuity of a function.

Each continuous function follows the Intermediate Value Theorem (but the converse need not be true!).  
 If you need a refresher, watch this video:  
<https://www.youtube.com/watch?v=ufFla_aAHFU>

In Bisection method, we first start with an interval where we know a root exists.   
Now it is important to choose your brackets (or intervals) in such a way that only one root lies in the interval.

Let us consider: x^3 – x -2 again

# By plotting the graph, we can see that a root exists between -2 and 2 Plotting graphs is a great way to figure out your initial guess.

# (P.S. It also has two complex roots but we cannot use bisection method for it. In case you’re curious, we use the Lehmer–Schur algorithm, a generalization of the bisection method for the complex plane)

The value of the polynomial at -2 is negative and at +2 is positive.  
So, we can say that since the function is continuous, by Intermediate Value Theorem, the polynomial would have passed through the value y=0 to get from the negative y to the positive y value at +2.

A quicker way to figure out if a root lies in an interval (or bracket) is to see if the product of values of polynomial (or function) at both end points is less than zero or not. I.e:

If f(a).f(b) < 0 if [a,b] is the interval

You can test it out yourself!  
The formula simply means that if the function has opposite signs in an interval, and if it is continuous, it must have a zero value somewhere in the [a,b] interval

Going back, we know that a root can be anywhere between -2 and +2  
And we want to reduce this interval of uncertainty.

So we divide the interval in the middle. This is where the “bi-“ part comes from. Bisection = Two sections. (AKA Interval halving method)

Now we know that either the root will lie in the left half or in the right half, since we chose the bracket in such a way that there is just one root.   
 Let’s try it out. -2 +2 /2 = 0

So we shall split the interval at 0.  
Will the root be in [-2,0] or in [0.2]?  
Hint: Use f(a).f(b) < 0

Great work!  
We can see that f(0) is not zero. So x=0 is not a root.  
Let’s try the process again  
What will be the mid-point this time?

Okay! Will the root lie in [0,1] or [1,2] ?

Great !

You will see that at each step our interval is getting smaller (by half)  
This means our region of uncertainty is decreasing.

Look at the graph is easy to determine the new interval we shall be focusing on, but since this is an algorithm, we need to use a numerical approach.

Let c be the midpoint of the interval [a, b], i.e.,

c=\(a + b)

and consider the product f(a)f(c).

if f(a).f(c) <0 where does the root lie?  
  
1. [a,c] 2. [c,b] 3. At c 4. Cannot be determined

What if f(a)f(c) = 0?

1. [a,c] 2. [c,b] 3. At c 4. Cannot be determined

Great!  
And finally. If f{a)f(c) > 0 that means the root lies in [c,b]  
you can check this using f(c)f(b) <0 but if you bracketed correctly, this won’t be required.

Also you’ll notice that after each iteration, our function value is getting closer to zero.  
We also use the same algorithm with binary search method. But there is one difference. Can you guess what that is?  
<answer>

The sorted list in binary search has always finite elements. So we always find the number it if exists. But there exist infinitely many points in an interval, so the iterations can go on indefinitely if the number is transcendental (a subset of irrational numbers)

We stop will the procedure when the mid-point we have chosen is a root of the function. But that is seldom the case. So we repeat the whole method till the interval is small enough (or till we find the root).

“What do you mean by small enough?”

An example shall explain:

Let us see the solution for the first 5 iterations

Now we can stop. NO, but surely we must go on. (10 iterations)

You would say, now we can stop. No. LET'S GO FURTHER (15)

"What about now?" EVEN MORE (20)

Okay, let's stop here. (25)

What is small enough? So when do we stop?   
Whenever you’ve found the solution to sufficient decimal places

Sometimes we can get lucky and find the exact root. (At a point when f(c) = 0)  
This is with finite non-repeating roots.   
  
Otherwise, we are satisfied when the solution is correct upto a few decimal places. For infinite decimal numbers, there will be always some error / tolerance limit.   
It will take infinite time to find infinite decimal points so we say after a stage “ This is enough for our work” and the error is too small to be negleceted for practical purposes

The biggest my error can be will be the length of the interval

Ofcourse, the error gets shorter too!

To perform error analysis, go here: https://en.wikipedia.org/wiki/Bisection\_method#Analysis

Want to find all the roots? the whole method can be peformed on all the brackets to find all roots of the polynomial (or zeros of the function)

Binary Chopping has one advantage and disadvantage  
Advantage: It always works. If a root has been bracketed, bisection method will always find the root (As long as the equation is of the form f(x)=0. Note that the equation of the form f(x) = x will stumble the good ol’ bisection method this however can be corrected by using f(x)-x = 0 as the new equation)  
Disadvantage: It’s slow

It’s the one friend you can always depend on but is so slothful that you will only call him is there is no one else around.

In practical terms:

We usually don’t have to use bisection method alone. We using it along with an open method to make it a rather effective tool.

Would you like to revise the theory or move on to the code?

**Code:  
Three levels of code so that the student achieves mastery**

**(Discuss)**

1. Basic: Create a function BisectionMethod() to input a function and initial guesses and find a root

***Programming Hints:*** Note that we have used the formula

*c = a +* 0.5(6 - *a)* (3.4)

to compute c, rather than the more obvious choice,

*c=(a + b)/2.* (3.5)

* The reason is that for very large values of a and *b,* (3.5) can lead to a computational overflow,

Note also that we were very careful in the algorithm to re-use the

function values from iteration to iteration. A careless implementation of bisection will

have two function calls each time through the loop, but it is possible to get by with only

* one function call.

1. Intermediate: Store the value of all iterations, percentage error at each step and then output the final result
2. Advanced: Find all roots. (No initial guess provided)

--- Make the function f(x) = 0

Now we need to start somewhere

Let’s start at zero because it’s the easiest to find.

Then think about a value to the left of zero and to the right of zero that is opposite to the sign of the function.

Since the value is iterative, we can use a loop!

You can think about the value elegantly or just look at all integer values till you find one that satisfies your required stuff. Integers are easier to evaluate.

You can start at 2.653 if you want,

Ideas: Continuous ke liye through.   
And polynomials are continuous

Sometimes, the initial guess will be given to you. But do not simply trust value, check it for yourself

We can be lazy when doing it on paper, we need to only figure out if the value is positive or negative

It is highly advisable to go a step further and *bracket* the root (determine its lower

* and upper bounds) before passing the problem to a root-finding algorithm.
* Identify all brackets

(Look at matlab functions that find the root: fminbnd