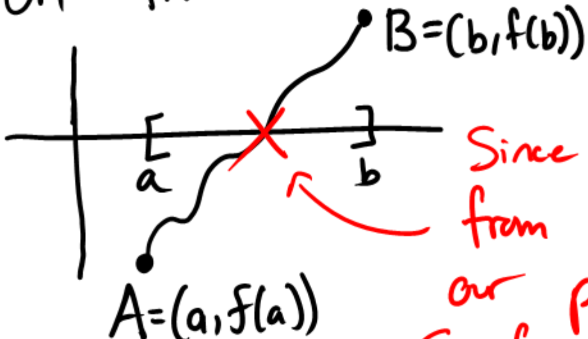
Approximating Roots of Continuous Functions (This won't be on HW or Exam)

__Theorem (Root Theorem)__ Let f be a continuous function. If $f(a)<0$ and $f(b)>0$ (or vice versa), then f has a root on the interval $[a,b]$.

__Why?__



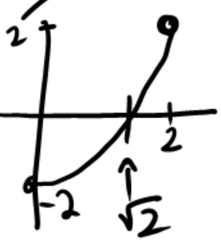
Since f is continuous, we need to get from point A to point B without lifting our pen (ie, without breaking continuity). So f need to cross the x-axis eventually, that will be the location of the root

__Corollary (Intermediate Value Theorem)__ Let f be a continuous function. If $f(a)=c$ and $f(b)=d$, then the interval $[\min(c,d), \max(c,d)]$ is in the range of f. Ie, if y is between c and d, there is an $x\in[a,b]$ where $f(x)=y$.

__Why?__ Apply the root theorem to $g(x)=f(x)-y$

Ex/ Show that $f(x)=x^2-2$ has a root in the interval $[0,2]$

Sol/



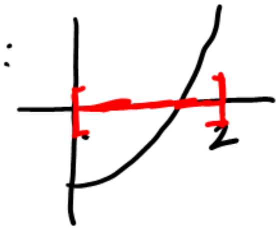
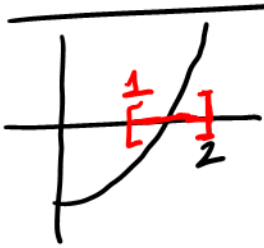
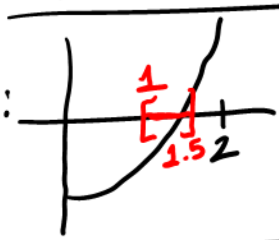
$f(0)=-2<0$ and $f(2)=2^2-2=2>0$. Since f is continuous, there is a root on the interval $[0,2]$

Q: What's an approximate value of the root ?

A: Since the root is in $[0,2]$, taking the midpoint of the interval $\frac{0+2}{2}=1$ is a good guess.

Bisection Method

We can get a better approximation by shrinking our interval.

		Interval	Approximation of root
Stage 0:		[0,2]	$x \approx \frac{2+0}{2} = 1$
Stage 1:		[1,2] ☆ Notice we halved the length	$x \approx \frac{1+2}{2} = 1.5$
Stage 2:		[1,1.5] ☆ Halved the length of the last interval	$x \approx \frac{1+1.5}{2} = 1.25$
Stage 3:		[1.25, 1.5]	$x \approx \frac{1.25+1.5}{2} = 1.375$

Repeat to get an approximation to the precision you want. We bisect (cut in half) each interval to get a smaller interval, so this is called the <u>bisection method</u>.

Algorithm (Bisection Method)

Inputs: ① A continuous function f

② Endpoints a and b which $f(a)$ and $f(b)$ have opposite signs

③ An error $\varepsilon > 0$

Outputs: An approximate root x where $\left| x - \begin{bmatrix}\text{True value} \\ \text{of the root}\end{bmatrix} \right| < \varepsilon$

(Next Page)

START

1. $x \leftarrow \frac{a+b}{2}$
2. if $f(x)=0$ OR $\frac{b-a}{2} < \varepsilon$, then # If a solution has been found
   - 2.1 RETURN x
   - 2.2 STOP
3. if $f(a)$ and $f(c)$ have the same sign
   - 3.1 $a \leftarrow c$
4. Otherwise,
   - 4.1 $b \leftarrow c$
5. Repeat Steps 1 – 4

Python Implementation

```python
def bisection(fcn, start, end, TOLERANCE=1e-5/2 , MAX_NUM_ITERATIONS=1000):
    """
    Parameters
    ----------
    fcn : Callable
        Our continuous function (we called this f).
    start : float
        Our left end point (we called this a).
    end : float
        Our right end point (we called this b).
    TOLERANCE : float, optional
        Our tolerance (this was our epsilon. The default is 1e-5/2, which
        goes until it is accurate to 5 decimal places.
    MAX_NUM_ITERATIONS : int, optional
        The max number of iterations (to prevent an infinite loop).
        The default is 1000.
    Returns
    -------
    root: float or None
    Returns a root of fcn up to the specified level of precision. Returns None
    if the maximum number of iterations was reached or fcn(start) and fcn(end)
    have the same sign.
    """
    # Make sure a root is present
    if (fcn(start)>0 and fcn(end)>0) or  (fcn(start)<0 and fcn(end)<0):
        return None

    # Start bisection
    N = 0
    while N < MAX_NUM_ITERATIONS: # prevents infinite loop
        x = (end + start)/2
        if fcn(x)==0 or (end - start)/2 < TOLERANCE:
            return x

        if (fcn(x)>0 and fcn(start)>0) or  (fcn(x)<0 and fcn(start)<0):
            start = x
        else:
            end = x

        N = N + 1

    # If the maximum number of iterations was reached
    return None

if __name__ == "__main__":

    print("Final Output", bisection(lambda x:x**2 - 2, 0, 2))
```

Output for $x^2 - 2$

```
Iteration 1: 1.0
Iteration 2: 1.5
Iteration 3: 1.25
Iteration 4: 1.375
Iteration 5: 1.4375
Iteration 6: 1.40625
Iteration 7: 1.421875
Iteration 8: 1.4140625
Iteration 9: 1.41796875
Iteration 10: 1.416015625
Iteration 11: 1.4150390625
Iteration 12: 1.41455078125
Iteration 13: 1.414306640625
Iteration 14: 1.4141845703125
Iteration 15: 1.41424560546875
Iteration 16: 1.414215087890625
Iteration 17: 1.4141998291015625
Iteration 18: 1.4142074584960938
Iteration 19: 1.4142112731933594
Final Output 1.4142112731933594

In [9]: sqrt(2)
Out[9]: 1.4142135623730951
```

Bound for the Error

If you start on the interval $[a,b]$, then

$$\left| \text{Difference between the approximate value and the true value} \right| < \frac{b-a}{2^{N}}$$

N ← # of bisections

Ex/ How many rounds of bisection do you need to approximate $\sqrt{c}$ accurate to 5 decimal places.

Sol/ We approximate a root of $x^2 - c$ on $[0, c]$ (since we know the root is on that interval).

$$|(\text{Approximation}) - \sqrt{c}| < \frac{c-0}{2^N} \overset{\text{WANT}}{<} \underbrace{\frac{10^{-5}}{2}}_{\substack{\text{make it accurate} \\ \text{to 5 decimal places}}}$$

Need to solve

$$\frac{c}{2^N} < \frac{10^{-5}}{2}$$

$$\frac{c}{10^{-5}} < 2^{N-1}$$

$$\log c - \log(10^{-5}) < (N-1)\log(2)$$

$$N > \underbrace{\frac{\log(c) + 5}{\log(2)} + 1}$$

N is the integer when we round up this value

Ex/ When $c = 2$, $N = 19$ (which we saw in the Python code)

This method is slower compared to others (e.g., Newton's Method), but we only need continuity to apply this.