Discrete Structures: CMPSC 102

Oliver BONHAM-CARTER

Fall 2019
Week 3

Suppose we have a very complicated equation to solve and we need to find some mathematical way to solve for x.

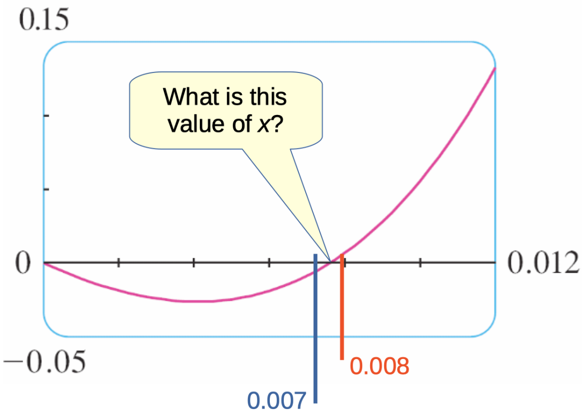
We are given an equation to solve: Find the Roots

$$f(x) = 48x(1+x)^{60}(1+x)^{60} + 1 = 0$$

- An approximate solution
- Let's plot the equation to see where it crosses the x axis
- Ask: for what value of x does this x-axis intersection happen?
- In general, how do we find values of x?

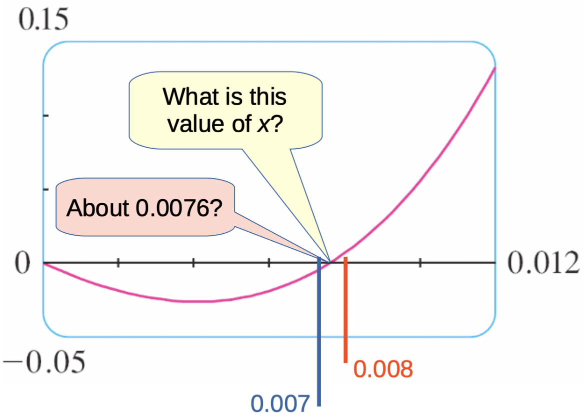
Plot the Equation

Approaching solutions by approximation ...

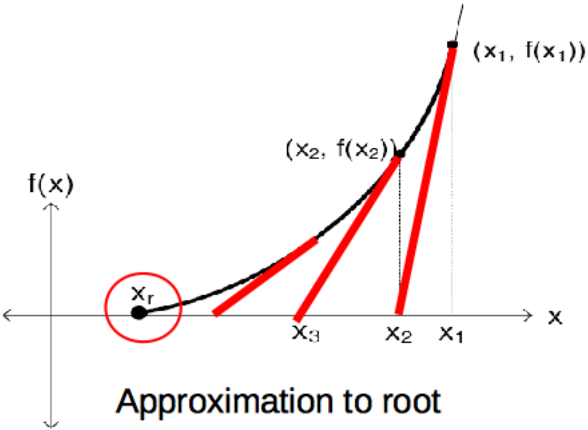


- We know that there are two roots.
- One solution is zero (or near) and the other is between 0.007 and 0.008
- We can estimate the value of one of the points (sort of) ...

Plot the Equation

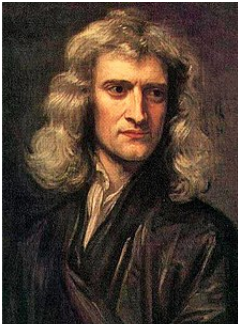


- Estimate a solution between 0.007 and 0.008 (approx 0.0076)
- Want to be able to calculate this value to any level of accuracy
- How to find these *roots* mathematically for any *zoomed-in* value?

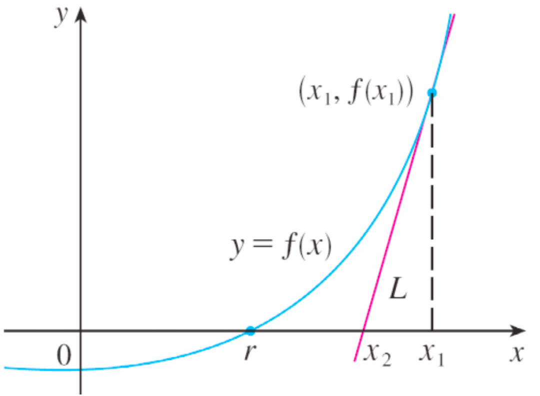


- Approximate the root (x_r) using *Newton's Method*

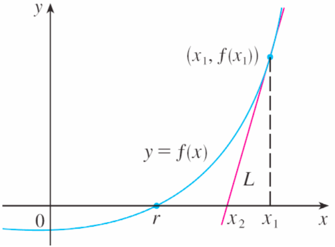
Isaac Newton



- Time line: 25 December 1642 - 20 March 1726 or 1727)
- English mathematician, astronomer, theologian, author and physicist
- One of the most influential scientists of all time
- A key figure in the scientific revolution.



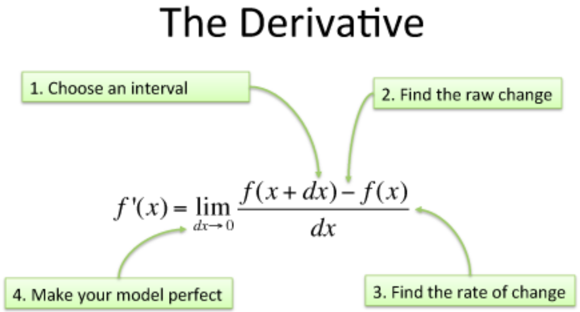
- The relevant root is labeled r
- First approximation for x_1 is a simple *guess* made by understanding the plot



- Consider the tangent line, L to the curve $y = f(x)$ at the point $(x_1, f(x_1))$ and look at the x-intercept of L, labeled x_2.
- Main idea: the tangent line is close to the curve and its x-intercept (an intersection point at x_2), is close to the x-intercept of the curve (the root r).
- This point root r that want to find!

Derivatives

Defined Mathematically (somewhat ...)



Roughly speaking...

- The derivative is an equation extracted from the original $f(x)$ used to find the x values of where the $y = 0$.

Derivatives

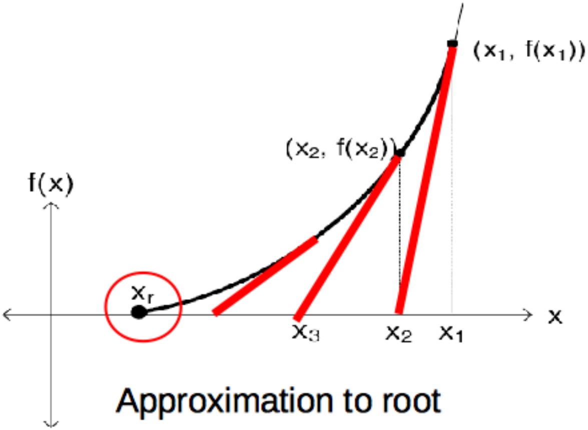
Roughly speaking...

- We want to find where line L passes x-intercept
- Slope of line L: $f'(x)$ (the derivative is tangent to curve)
- Line formula (from algebra): $y = m * x + b$
- To find a formula for x_2 in terms of x_1

$$y - f(x_1) = f'(x_1)(x_2 - x_1) \tag{1}$$

Thinking Recursively

Use derivatives to find lines crossing x-axis, converging on root



We want to insert initial approximation values of x back into the line equation (recursively) to find the next approximation (and converge on the root, x_r). The slope of the tangent line is $f'(x)$, known as the derivative.

Find a General Equation for Finding Roots

We are solving for x_2, or a closer approx. of root!

$$
\begin{aligned}
y - f(x_1) &= f'(x_1)(x_2 - x_1) \\
0 - f(x_1) &= f'(x_1)(x_2 - x_1) \\
f(x_1) &= -f'(x_1)(x_2 - x_1) \\
f(x_1) &= x_1 * f'(x_1) - x_2 * f'(x_1) \\
f(x_1) - x_1 * f'(x_1) = x_2 * f'(x_1) & \\
x_2 * f'(x_1) &= x_1 * f'(x_1) - f(x_1) \\
\frac{x_2 * f'(x_1)}{f'(x_1)} &= \frac{x_1 * f'(x_1)}{f'(x_1)} - \frac{f(x_1)}{f'(x_1)}
\end{aligned}
$$

$$x_2 = x_1 - \frac{f(x_1)}{f'(x_1)}$$

Now what?!

First approximation of root values

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}$$

Second approximation of root values

$$x_2 = x_1 - \frac{f(x_1)}{f'(x_1)}$$

Third approximation of root values

$$x_3 = x_2 - \frac{f(x_2)}{f'(x_2)}$$

General approx. of root vales; x_{n_1} from previous x_n

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(n)}$$

Let's find the **square** root for a number

Recall that Newtons method finds an approximate root of $f(x) = 0$

Define $f(x)$

$$
\begin{aligned}
f(x) &= x^2 - a \\
x^2 &= a \\
x &= \sqrt{a} \qquad \text{(find positive root, } a\text{)} \\
x &= -\sqrt{a}
\end{aligned}
$$

Define the derivative of $f(x)$, $f'(x)$, using calculus

$$f'(x) = 2x$$

Establish the Approximation Equation

Note: a in $f(x)$ is the initial guess!

$f(x) = x^2 - a$
$f'(x) = 2x$

The root to find

$a = x_r$

The initial guess of root (to start the method)

$x_1 = 1.0$

General approx. for root x_{n_1} from approx. root x_n

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(n)}$$

Finding Square Root of a

Knowns

$a = 2$ (want to find $\sqrt{a}$)	$f(x) = x^2 - 2$ (function)
$x_1 = 1.0$ (initial guess)	$f'(x) = 2x$ (derivative)

$$
\begin{aligned}
x_2 &= 1.0 - \frac{f(1.0)}{f'(1.0)} \\
&= 1.0 - \frac{(1.0)^2 - 2}{2 * (1.0)} \\
&= 1.0 - \frac{1.0 - 2}{2} \\
&= 1.0 - \frac{-1.0}{2} \\
&= \frac{3.0}{2} \\
&= 1.5
\end{aligned}
$$

Table of Iterations

Finding square root of a

Calculations

Guess			Approx. root
x_n	$f(x) = x_n^2 - 2$	$f'(x_n) = 2x$	$x_n - \frac{f(x_n)}{f'(x_n)}$
1	-1	2	$1 - \frac{-1}{2} = \frac{3}{2} = 1.5$
$\frac{3}{2}$	$\frac{1}{4} = 0.25$	3.0	$\frac{3}{2} - \frac{(\frac{1}{4})}{3} = \frac{17}{12} = 1.4167$
$\frac{17}{12}$	$\frac{1}{144}$	$\frac{17}{6}$	$\frac{17}{6} - \frac{\frac{1}{144}}{\frac{17}{6}} = \frac{577}{408} = 1.4142$

Python to the rescue

```
>>> math.sqrt(2)
1.4142135623730951
```

Atom newtonsMethod.py

```
n = 2.0 # the number from which to find square root.
guess = 1.0 # initial value for approx

print("  Initial values:  n = ",n, "guess = ",guess)

while abs(n - guess*guess) > .0001:
   #find  x_n - \frac{f(x_n)}{f'(x_n)}
   guess = guess - (guess*guess - n)/(2*guess)
   print("   *Current guess:   ",guess)
root = guess

print("   Result :",root)
```

Put This Script Into a Function

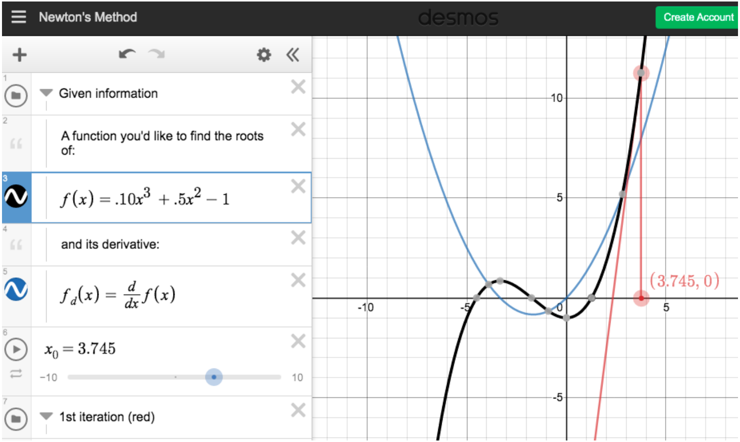
Finding square root of a

Atom newtonsMethodFunction.py

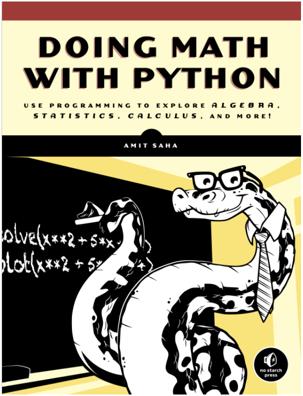
```
def NM(n, guess):
    print("  Initial values:  n = ",n, "guess = ",guess)

    while abs(n - guess*guess) > .0001:
        #find  x_n - \frac{f(x_n)}{f'(x_n)}
        guess = guess - (guess*guess - n)/(2*guess)
        print("   *Current guess:   ",guess)
        root = guess
    return root
#end of NM()

#get parameters to call function NM()
n = 2 # the number from which to find square root.
guess = 1.0 # initial value for approx
print(" Finding root : ",n)
print(" Approx guess : ", guess)
print("  Result : ",NM(n, guess))
```

Desmos Online Calculator

Newton's Method in action



https://www.desmos.com/calculator/kgwfrkiyh8



Chapter 2: Working with Numbers

You Can Use Python as a Calculator

Integers

- 1 + 2
- 200 + 4
- x_int = 1 + 2
- type(x_int) #Is: <class 'int'>

Floats

- 1.0 + 2.2
- 200.001 + 56.05
- x_flt = 123.007 + 0.002
- x_flt = 100 / 4 #Is: <class 'float'>!

Multiplication and Division

Floor Division

- Rounding down to nearest integer
- 3 / 4 = 0.75
- 3 // 4 = 0
- 50 / 6 = 8.33
- 50 // 6 = 8
- No *Ceiling* operator with a single character like this... :-(

Ceiling With Python's Math Library

- import math
- math.floor(5/6)
- math.ceil(5/6)

Powers and Types

Floor Division

- $2^3 = 2**3 =$math.pow(2,3)
- (5 + 5)**5
- $2^(1/2) = ??$

Variable Names

- a1 = 2
- type(a1) #Is: <class 'int'>!
- a2 = 2.0
- type(a2) #Is: <class 'float'>!
- a3 = 3/4
- type(a3) #Is: <class 'float'>!

Type Conversions

Start with an integer, end with a float

- a_int = 3
- b_flt = 0.1415
- print(a_int + b_flt)
- type(a_int + b_flt) #Is: <class 'float'>!

Start with a float, end with an integer

- a_flt = 3.1415
- b_int = int(a_flt) # conversion
- type(b_int) #Is: <class 'int'>!

Start with a string, end with an integer

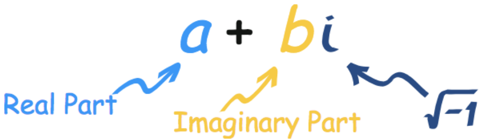
- a_str = ”3.1415”
- b_flt = float(a_str) #Is: <class 'float'>!
- c_int = int(b_flt) #Is: <class 'int'>!

Working With Fractions

- *from fractions import Fraction*
- f = Fraction(3,4)
- f
- print(f)

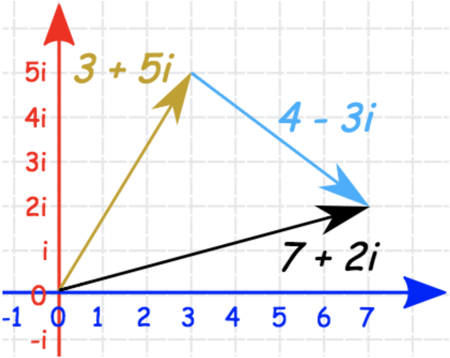
Complex/Imaginary Numbers

- $i = sqrt(-1)$
- $i^2 = -1$



1 + i	39 + 3i	0.8 − 2.2i	−2 + πi	√2 + i/2

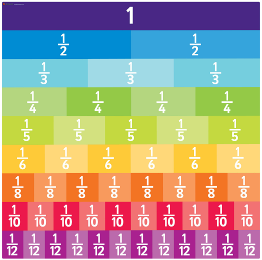
Complex/ Imaginary Numbers



1 + i	39 + 3i	0.8 − 2.2i	−2 + πi	√2 + i/2

- a_cp = 2 + 3j #Is: <class 'complex'>!

Fractions



Requires the library:

```
from fractions import Fraction
```

- `Fraction` provides support for rational number arithmetic
- Creates a Fraction instance from integers, floats, numbers decimals, and strings

Fractions

```
print(Fraction('3.14159265358979323846'))
# returns Fraction(157079632679489661923, 50000000000000000000)

print(Fraction('3.14159265358979323846').limit_denominator(10000))
# returns Fraction(355, 113)

print(Fraction('3.14159265358979323846').limit_denominator(100))
# returns Fraction(311, 99)

print(Fraction('3.14159265358979323846').limit_denominator(10))
# returns Fraction(22, 7)

print(Fraction(125, 50).numerator)
# returns 5

print(Fraction(125, 50).denominator)
# returns 2
```

Fractions


```
print(Fraction(113, 100) + Fraction(25, 18))
# returns Fraction(2267, 900)

print(Fraction(18, 5) / Fraction(18, 10))
# returns Fraction(2, 1)

print(Fraction(18, 5) * Fraction(16, 19))
# returns Fraction(288, 95)

print(Fraction(18, 5) * Fraction(15, 36))
# returns Fraction(3, 2)

print(Fraction(12, 5) ** Fraction(12, 10))
# returns 2.8592589556010197
```

Exceptions

When working with data-types, use exception handling

Wrong data-type for python keyword input()

- Invalid inputs lead to errors:
  - a = float(input()) #enter "Hello"
  - `ValueError: could not convert string to float`
  - float(input()) # was not possible
  - float("hi") # also not possible

Use Exceptions

- `try: ... except: ...`
- Used to detect and prevent errors dealing with data types from crashing code.

Try and Except

Wrong data-type for input()

```
try:
  a = float(input(" Enter a float :  "))
except ValueError:
  print("  Entry invalid...")
```

Use Exceptions

- Used to detect and prevent errors dealing with data types from crashing code.

- Note that this exception handling will not crash the program.
- Can you build another exception handling block to catch strings being converted to integers?
  - Catch `int(input("hello"))`

Accept Integers or Floats, not Strings

sandbox/exceptionHandling

```
print("Exception handling...")
try:
  a_int = int(input("  Enter integer, not string  :"))
except ValueError:
  print("  Cannot convert string to ints or floats...")
```

Catch zeros in denominator of fractions

Wrong data-type for input()

```
from fractions import Fraction # load library
print("Exception handling...")
a_fraction = Fraction(input("Enter a fraction: "))
```

Catch the exception

```
print("Exception handling...")
from fractions import Fraction # load library
try:
   a_fraction = Fraction(input("Enter a fraction: "))
except ZeroDivisionError:
  print("  Cannot divide by zero...")
```

Catch bad complex numbers

Wrong data-type for input()

```
from fractions import Fraction # load library
print("Exception handling...")
z_complex = complex(input("Enter a complex number: "))
```

Catch the exception

```
print("Exception handling...")
from fractions import Fraction # load library
try:
   z = complex(input("Enter string as complex num.: "))
except ValueError:
  print("  This is not a complex number...")
```

Formatting strings


```
item1 = "apples"
item2 = "bananas"
item3 = "grapes"

print("I have: {0} and {1} and {2}".
format(item1, item2, item3))

#note: all on same line
print("I have: {0} and {1} and {2} and {3}".
format(item1, item2, item3))

#fix:
print("I have: {0} and {1} and {2} and {3}".
format(item1, item2, item3, "PINEAPPLES"))
```

Formatting Numbers

Make a formatted multiplication table

sandbox/multiplicationTable.py

```
def multi_table(a):
  for i in range(1, 11):
    print("{0} x {1} = {2}".format(a, i, a*i))

multi_table(4) # begin program by calling function
```

Miles to KM Converter with Formatting, part 1

sandbox/kmConverter.py

```
'''
Unit converter: Miles and Kilometers
'''
def print_menu():
      print('1. Kilometers to Miles')
      print('2. Miles to Kilometers')
#end of print_menu()

def km_miles():
      km = float(input("Enter distance in km: "))
      miles = km / 1.609
      print("Distance in miles: {0}".format(miles))
#end of km_miles()
```

Miles to KM Converter with Formatting, part 2

sandbox/kmConverter.py

```
def miles_km():
      miles = float(input("Enter distance in miles: "))
      km = miles * 1.609
      print("Distance in kilometers: {0}".format(km))
#end of miles_km()

if __name__ == "__main__":
# Note: execute this program when run
# importing this code into another
# script will not prompt a menu

  print_menu()
  choice = input("Choose a conversion : ")
  if choice == "1":
        km_miles()
  if choice == "2":
        miles_km()
```